Intelligent Behavior Simulation Module for Software Process Elements

Rafael O. Chaves Walter A. Da L. Lobato* Emanuel M. Da C. Tavares*
Tales C. Miranda* Sandro R. B. Oliveira* Elói L. Favero

Postgraduate Program in Electrical Engineering – Instituto de Tecnologia
Universidade Federal do Pará (UFPA)

*Postgraduate Program in Computer Science – Instituto de Ciências Exatas e Naturais
Universidade Federal do Pará (UFPA)

Abstract

Componentization is a Software Engineering (SE) strategy successfull in the development of new projects. Mainstream games use components that handle a set of functions relevant to a common aspect of a genre (e.g. real time strategy, first person shooter). However, Artificial Intelligence (AI) components, in spite of their great gameplay weight in current games, few has been researched and published about. This scenario gets worse when it comes to specific components for SE simulation games. This paper shows the results of the development of an AI component called Intelligent Behavior Simulation Module for Software Process Elements (MSCI-ep) for software process simulators. MSCI-ep objectives are to support and encourage development of this genre of educational games.

Keywords: artificial intelligence, software process, simulation, educational games

Authors' contact:
rochaves@ufpa.br, {walter.lobato, emanuelmaues, tales.miranda88}@gmail.com, {sbro, favero}@ufpa.br

1. Introduction

Certain aspects of game development such as physics, networking, audio components, etc., have had support for their specific purposes. These components bring great benefits to the gaming industry: they reduced development time, increased reliability and lowered cost. But, despite AI being a key element to some recent games, few has been researched and published in order to componentize it. Game engines usually treat AI as part of other components [FILHO 2005] or controlled by script languages like Lua or Python [FOLMER 2007].

As quoted in [CHAVES 2010], software process simulation games has made little use of componentization, being a demotivating factor for the development of new games. Also, [CHAVES 2010] proposed a software process simulator machine (SPSM) aiming to bundle the main generic requirements of software process simulation. However, SPSM does not feature a way to simulate the behavior of resources allocated to process. These resources are the process element instances defined in their software architecture (roles, artifacts, tasks and tools). Bearing in mind that the behavior of these elements is important to give more realism to the simulation, thereby software process simulation games become more interesting for educational purposes.

Through the fundamental rules of Software Engineering and Scenarios (both behavioral models), [OH 2006] and [BARROS 2001] present the importance of software process element behavior for educational purposes, and their influence on the overall project behavior. These behaviors are usually difficult to implement computationally, requiring knowledge of a programming language or another specific language for this problem. So, facilitate the modeling, implementation and reuse of these behaviors is an important contribution to research on simulation games software process.

This paper presents an Intelligent Behavior Module Simulation Software Process Elements (MSCI-ep) to describe the individual behavior of software process elements and relations among them, which defines the global behavior of a staff assigned to a software project. The MSCI-ep, through a fully graphical and customizable interface, abstracts the complexity of the programming language needed to describe the simulation logic. The MSCI-ep is based on Data-Driven Systems [DAREMA 2004] techniques and System Dynamics [FORRESTER 1961] and was developed according to a set of criteria compiled by Barros [2001], Dantas [2004] and Oh [2006]. Tests were carried out to verify its ability to create intelligent behavior solely from its graphical interface and its real adherence to criteria in section 3.

2. Motivation and Relevance

To simulate Software Processes parameters are required to establish a temporal relationship of events and states of the elements that make up the process.

In a real software process, although there are precedence relations between tasks [BRENDAOU et. al. 2007], it is known that the time for completion of a task depends mainly on the attributes of roles (e.g. experience, ability to work in a group) and tools (e.g. ease of use) allocated for its implementation. Tasks' nature (e.g. complexity) also influence in their completion time.

Process elements can influence each other (e.g. a tool may decrease the productivity of a role that has no practical for it). Same way each can influence themselves (e.g. role humor can decrease or increase his productivity). Influence is caused by a proportional or inversely proportional relationship between the attributes of the elements, i.e. there is a positive or negative influence, respectively. Thus, the MSCI-ep is concerned with all the attributes of the elements of process simulation software and their relationships. It is also designed to maintain the lowest coupling with other components of game development (e.g. SPSM).

Wangenheim's work [2009] found that the games for teaching software project management [DANTAS 2004; LUDEWIG and DRAPPA 2000] and software process [Oh 2006] did not have a module (component) to deal with specific "intelligence"(behavior) of the software process elements. Consequences: high coupling among different aspects of those games (e.g. flow of process execution and behavior of the resources allocated in the project share the same code), lack of reusable features and a graphical environment that completely abstracts the coding to create a new behavior. OH [2006] implemented a graphicalenvironment that allowed the description of behaviors. However, there were some behaviors that the graphical environment was not flexible enough to describe. These require an advanced knowledge in the language used, demanding workarounds.

The importance of an AI component in game architecture is pointed out by [FOLMER 2007]: it determines the behavior of game objects and allows reuse. An AI component example is the OpenAI Project[1]. It implements tools to support some AI techniques which are specified to operate with great modularity and interoperability with other tools, components and engines (for games) that are compliant to their specifications.

Given the existence of OpenAI, which handles and implements some AI techniques for general purposes, this proves the relevance of an AI module that is specific to simulation games for teaching ES. This genre uses mainly two AI techniques: Dynamic Systems and Data-Driven Systems. MSCI-ep aims to contribute to research and development of serious games that teach ES, which, in conjunction with SPSM [CHAVES 2010], will create a game engine for these games.

3. Requirements for the development of AI modules for SE simulation games

Requirements were compiled from Barros [2001], Dantas [2004], Oh [2006], Neves et. al. [2005] to develop AI modules for SE simulation games. These requirements are references to MCSI-ep development.

3.1 Mirror the necessary programming for the behavior of the software process elements

As in Barros [2001], Dantas et. al. [2004] and partially Oh [2006] it is necessary to know the language to define the behavioral models. This situation increases the complexity and cost of creating these models. The Intelligence Module must be a framework that: offers graphical representations for software process elements (e.g. role, artifact, tool, task, etc.); allow the definition of those attributes; have the most important mathematical functions (e.g. linear, exponential, polynomial) to establish the relationships between the attributes of the process elements.

As a result of the AI module being a fully graphical environment where only arguments and parameters are set, this limits the semantic power and accuracy of behavioral models created within. This situation may be appropriate for models with educational purposes. However when it is necessary that the model be more accurate, the AI module should provide an option for the modeler to have the freedom to plan beyond the constraints of the framework.

3.2 Be responsible for all the attributes to the behavioral model

Behavioral models should be responsible for all the attributes and relationships needed for the simulation in order to look like the real behavior of the software process elements.

3.3 Stratify behavioral patterns in levels of difficulty

The behavioral models should be classified into levels of difficulty, making them similar to phases of non-educational games.

3.4 Being a reusable component

Behavioral models should be independent of other components for game development because one purpose of componentization is to encourage research in ES education through games. That way, AI module can be reused in other game projects. This requirement origins from [NEVES et al. 2005], but the problem of

[1] [1] http://openai.sourceforge.net

reuse was found in [OH 2006; DANTAS 2004; BARROS 2001].

3.5 Eliminate the deterministic effect

Software development is mainly influenced by human behavior and organizational politics, which are very difficult to formalize, measure and predict. Relationships between human attributes are complex and not always stable and deterministic. For example, a developer who is in a bad mood will always have a lower productivity? His work will always be more prone to errors? After a rest period the developer returns exactly to its initial state? Despite the common sense answer "yes" to these questions, a deterministic result is unreal in these situations.

To increase the level of realism of the simulation, relationships and deterministic values of the attributes shall be reduced as much as possible when it involves human and political issues of the software process. Deterministic simulations quickly become boring for the players, after they discover the “trick" to solve a particular challenge. Random variables can minimize this situation. However, depending on the degree of their variance and frequency values, they can still bring absurd results and events far distant from reality. For example, a role that has three humor moods: "good mood", "normal" and "grumpy" if the value "grumpy" and "good mood" have a frequency of 70%, it is not usually consistent with reality.

3.6 Behavioral models need to be simulated and tested independently of other game components

One of the main limitations of behavioral models referenced in this paper is that they do not work independent of other components, hindering their development. To solve this situation, behavioral models should be created and tested independently of any other component.

4. Intelligent Behavior Simulation Module for Software Process Elements Description

MSCI-ep was developed based on a dynamic creation of a set of variables that can be instantiated as needed to create a behavioral model. The dynamically created variables are classified as: continuous, cumulative, dependent and discrete. Each one has specific characteristics of classification and relationship rules with each other.

For the creation and instantiation of the variables it is used graphical user interface that defines the classification and the values of the variables, becoming it transparent and eliminating any coding as shown in examples in Figures 1, 2 and 3.

The figure 1 shows the creation of a discrete variable, inserting the variable name, the number of states and the probability values of each one takes place.

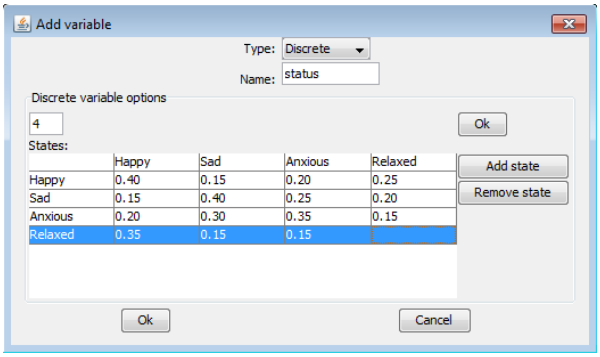

Figure 1 – Creation of the Discrete Variable “Status”

Figure 2 illustrates the creation of a dependent variable, which is inserted the name of the variable, the variable which it depends (need to have been created previously), and the function that rules this dependence.

Finally the figure 3 shows the list of variables that were created and their types. This screen attaches the behavior created to a element of the software process, in this case the element is one role named “role 1”.

By completing this step is automatically generated a class called Intelligence.java that describes the behavioral model with the logic and configuration that will be used in the simulation. A class that represents a software process element that has an aggregation relationship with the class Intelligence.java makes use of the behavioral model described in the class Intelligence. Figure 4 is shown a aggregation between the Role User class and Intelligence class.

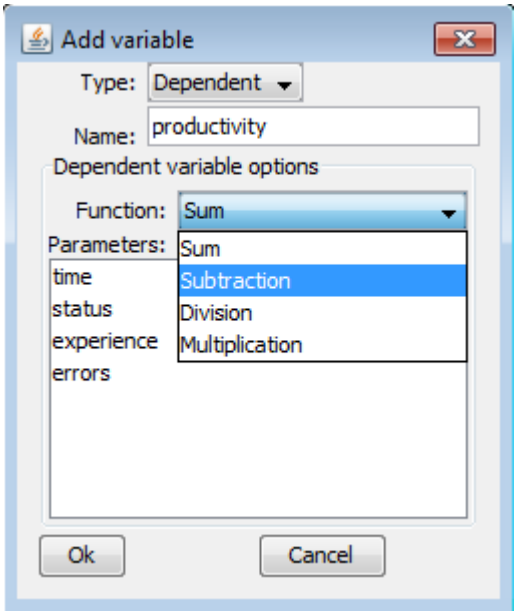

Figure 2 – Creation of the Dependent Variable “Productivity”

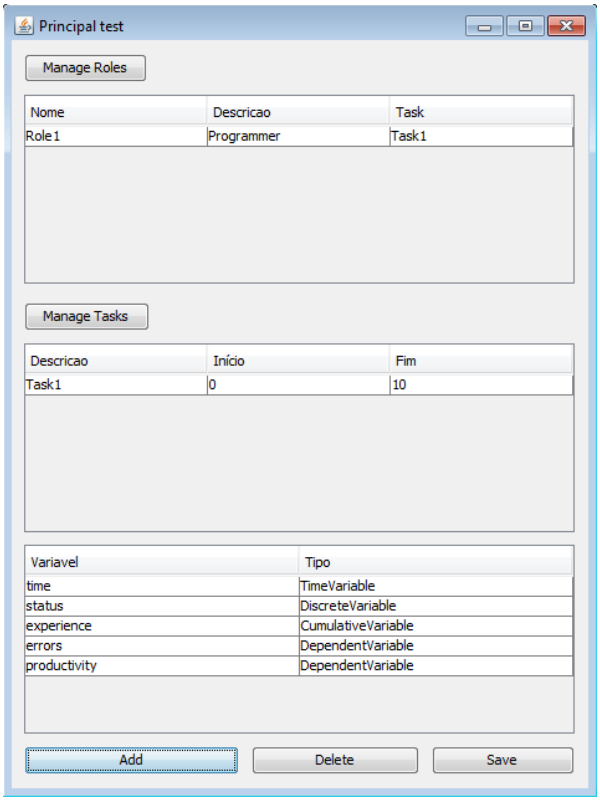

Figure 3 – Main screen. Shows the list of Role Users, Tasks and Variables created

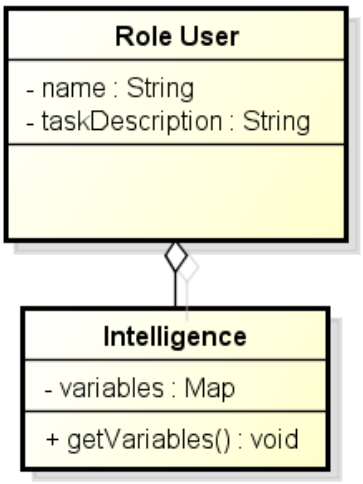

Figure 4 – Aggregation between Intelligence class and Role User class

By saving this configuration is automatically generated the Intelligence.java class with a instance of the variables created, this class is exemplified in an automatically generated code shown in Figure 5.

The code section describes the creation of the following objects: "time" (line 46), "status" (line 49) with the values of the attributes "state" (line 55, 62, 69 and 76) and "transition" (lines 56 -60, 63-67, 70-74 and 77-81); "experience" (line 83-87) with the function that defines it values.

5. Configuration and simulation of the behavioral model

To demonstrate the simulation process of MSCI-ep, a behavioral model was created obeying the following steps: (a) creation of variables to the software element Role User b) relationship between the variables using functions, (c) Simulation of the Behavioral Model and (d) verify the adherence of the MSCI-ep with the requirements defined in Section 3; below each item is explained in a subsection.

5.1. Creation of Variables

Five variables were created in the MSCI-ep for Role User.

a. Variable "time": controls the cycle time of the simulation.

b. Discrete variable "status": variable that has four different states, which represent the Role User emotional states, these are: Happy, Sad, Anxious and Relaxed.

c. Cumulative Variable "experience": That numerically represents the accumulation of experience on a task over time (cycles) the Role.

d. Dependent Variable "errors": The numerical representation of errors made by the Role User in a task.

5.2 Relationship between Variables

To represent the influence relationship that variables have with each other, a variable is placed in function of another. However the expressions and values used in the functions of this work were obtained through experimentation, so they are only for testing purposes and cannot be generalized as rules. See [OH 2006; BARROS 2001] for more research and accurate mathematical modeling of the software process element behavior.

```
TimeVariable time = new TimeVariable("time"); //Creation of a TimeVariable named "time"
variables.put("time", time);                   //TimeVariable added to the variables list

DiscreteVariable status = new DiscreteVariable("status", 4, 0); //Creation of a DiscreteVariable "status" with 4 different states, initial state set to 0
time.register(status);                          //Register "status" in TimeVariable, to recalculate its value in each cicle
variables.put("status", status);               //add "status" in variables list

State[] states = status.getEstados();      //gets the states list from discrete variable to sets its name and probabilities
double[] roulette = new double[states.length];//Creates a intermediary array to sets the roulette's probabilities for each state
states[0].setNome("Happy");                //Sets "Happy" to the name of state 0
roulette[0] = 0.40;                          //Sets the state 0 transition's probabilities for each other state
roulette[1] = 0.15;
roulette[2] = 0.20;
roulette[3] = 0.25;
states[0].setTransicoes(roulette);          //sets the composed roulette to state 0

states[1].setNome("Sad");       //sets the name "Sad" to the state 1
roulette[0] = 0.15;                 //set the state 1 transition's probabilities to each other state
roulette[1] = 0.40;
roulette[2] = 0.25;
roulette[3] = 0.20;
states[1].setTransicoes(roulette);  //Sets the composed roulette to state 1

states[2].setNome("Anxious");     //set the name "Anxious" to state 2
roulette[0] = 0.20;                  //Set the state 2 transition's probabilities for each other states
roulette[1] = 0.30;
roulette[2] = 0.35;
roulette[3] = 0.15;
states[2].setTransicoes(roulette); //set the composed roulette to state 2

states[3].setNome("Relaxed");  //Sets the name "Anxious" to state 3
roulette[0] = 0.35;               //set the state 3 transition's probabilities for each other state
roulette[1] = 0.15;
roulette[2] = 0.15;
roulette[3] = 0.35;
states[3].setTransicoes(roulette);//sets the composed roulette to state 3

CumulativeVariable experience = new CumulativeVariable("experience", 1, //Creates the CumulativeVariable with a sum function(0.001)
        new Soma().addParameter(new Constante(0.001)));
experience.getFuncao().addParameter(                                     //Adds to the function result the multiplication (experience * 0.25)
        new Multiplicacao().addParameter(experience).addParameter(new Constante(0.25)));
variables.put("experience", experience);        //puts the variable "experience" to the list of variables
time.register(experience);                       //register the variable "experience" in the time variable
```

Figure 5 – Auto-generated Code section

For discrete variables is used the roulette selection method [DE JONG 1965], which is commonly used in genetic algorithms. A roulette wheel is mounted where each slice is a state of the discrete variable, and the size of the slice represents the chance (probability) that this state has to be chosen according to a random number.

Status variable's auto-relation: For "status" variable, each turn of the MSCI-ep has two steps: a) checks the current state, and b) raffles the next state considering the weight of every other state has over the original state.

Figure 6 represents the wheel for each state, i.e. the probability that each state has to switch to another state.

Function of experience variable: For "experience" variable is used the following function:

$$experience = 0001 + experience + (experience * 0.25) \quad (1)$$

The "experience" variable is cumulative. In the course of simulation time (cycle) "experience" is always increased. This increase is due to the empirical principle that the Role User will always gain more experience in a task as long it spend more time allocated to it.

Function of "errors" variable: For the dependent variable "errors" the following function is used:

$$errors = 0.5 - (experience - errorLinesFromStatus) \quad (2)$$

Which *ErrorLinesFromStatus* is the numerical representation of errors for each state of the "status" variable; therefore the emotional state will affect the mistakes committed by the Role User. Table 1 shows the *errorLinesFromStatus* values for each state: happy, sad, anxious, and relaxed.

Function of "productivity" variable: For the dependent variable "Productivity" is defined the following function:

$$productivity = errors - statusProduction \quad (3)$$

Which StatusProduction is the numeric value that represents the production capacity of role in a given state. These values are fixed, and were configured as shown in table 2.

5.3. Simulation of behavioral mode

Figure 7 represents a small software process using the notation of SPIDER_ML [OLIVEIRA 2009]. This process is composed by: a task, an input artifact, and a role with the variables needed to simulate intelligent behavior through the MSCI-ep.

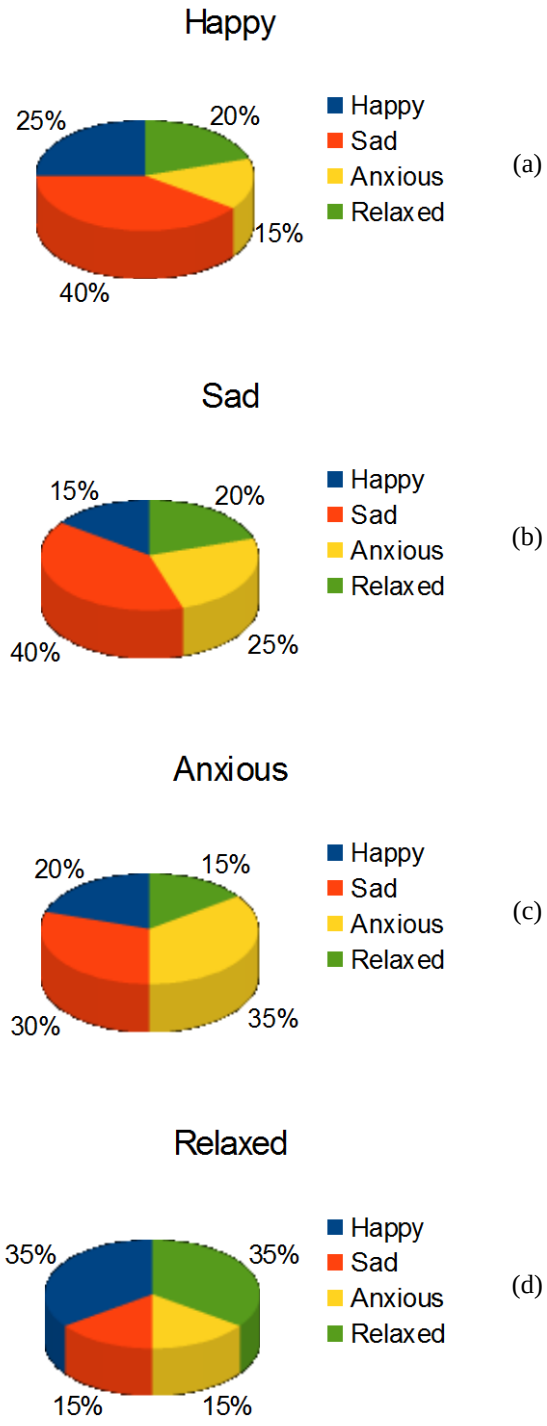

Figure 6 – Roulette for each state of "Status" variable

Table 1 – ErrorLinesFromStatus values according to the discrete variable state

status	errorLinesFromStatus
Happy	0.01
Sad	0.02
Anxious	0.02
Relaxed	0.01

Table 2 - StatusProduction values according to "Status" variable state

status	statusProduction
Happy	-0.3
Sad	0.3
Anxious	0.1
Relaxed	-0.2

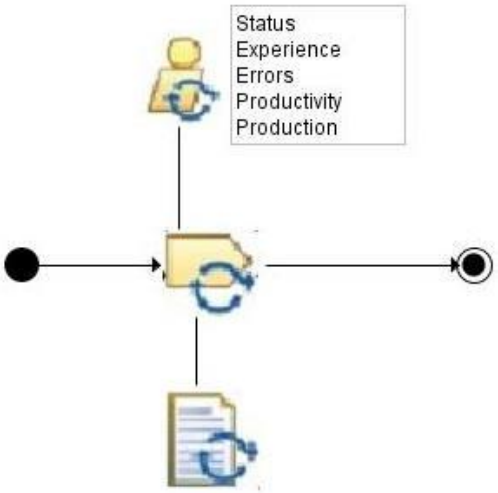

Figure 7 - Experiment overview

Two simulation rounds were performed to verify if happens a deterministic behavior, i.e. if the first results of the simulation would be the same or very similar to the results of the second simulation round. Labels: 0 – Happy, 1 – Sad, 2 – Anxious, 3 – Relaxed.

Figure 8 refers to the behavior of the "status" variable (y axis) relative to the "time" (x axis) in the first simulation round. The second round behavior is shown in figure 9. The difference between figures 8 and 9 confirms that the roulette method eliminates the deterministic effect.

From the figure 10, it can be seen the effect that the error variable decreases in the same proportion of the variable experience increases, as defined in the function (2). Observing figures 8 and 10 it can be seen another effect: the errors variable and status variable over the productivity according to the function (3).

For the simulation shown in figure 8, figure 10 shows the behavior of the role of continuous variables (errors, experience) and their influences on the variable productivity, according to the "Productivity" (3).

5.4. Evaluation of the adherence between intelligence module and the requirements

Table 3 shows each AI module requirements and how MSCI-ep adheres to each one.

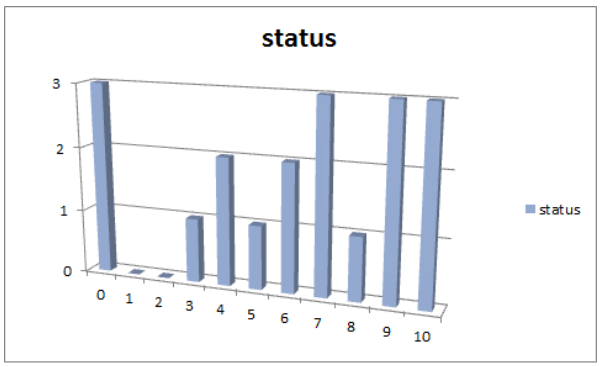

Figure 8 – First simulation round of the status variable behavior through time.

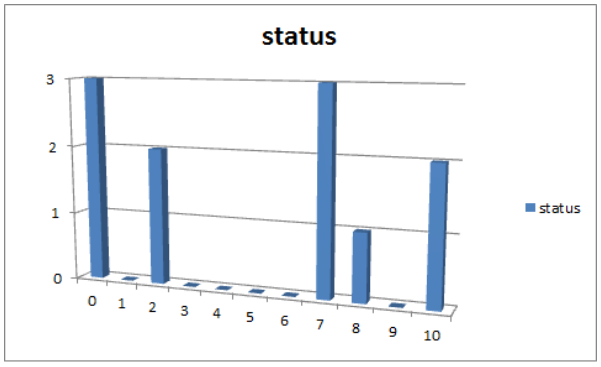

Figure 9 - Second simulation round of the status variable behavior through time.

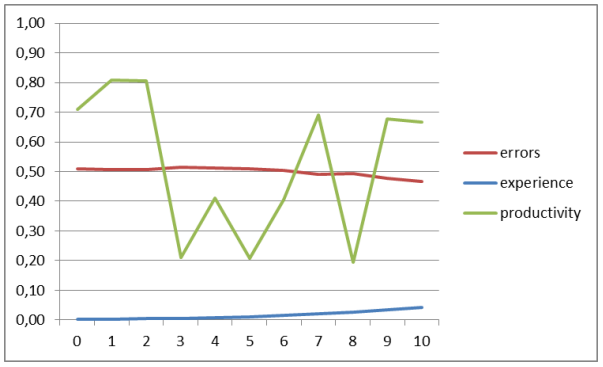

Figure 10 – Behavior of the continuous variable through time

Table 3 - Evaluation of the adherence between the intelligence module and the requirements

Requirement	Full Adherence	Description
Mirror the necessary programming for the behavior of the software process elements	Yes	This requirement is fully achieved. Since there is a graphical interface, see Figures 1, 2 and 3, which defines the attributes of the software process elements and the interactions between them. Thus, abstracting the complexity of programming required.
Be responsible for all the attributes of the behavioral model	Yes	This requirement is fully achieved. None of the attributes needed to define the behavior of software process element are external to the MSCI-ep. All attributes are variables created dynamically using the graphical user interface.
Stratify the behavioral models in difficulty levels	No	This requirement was not achieved. By the time of writing this paper don't exist a way to classify the behavioral models.
Reusable component	Yes	This requirement has been fully achieved. Therefore the MSCI-ep creates dynamically the Intelligence.java file that contains the behavioral model. This class enables the integration between the MSCI-ep and other components for the simulation games for ES development. The behavioral model can always be redefined to be more appropriate to a specific software process.
Eliminate the deterministic	Partially	This requirement is partially

effect		achieved. The use of roulette technique the discrete variables eliminates the deterministic effect of these and its dependent variables. e.g. "*errors*" function is influenced by errorLinesFrom Status, this uses the roulette technique, so "*errors*" is randomly influenced. In the case of "*experience*" function, it will still have a deterministic behavior, because it does not depend of a discrete variable that uses the roulette technique.
Behavioral models should be simulated and tested independently from other game components	Yes	This requirement has been fully achieved. As demonstrated in the tests, only the MSCI-ep was enough to build behavioral models and test them. There was no need of another game component.

6. CONCLUSIONS

Educational games represent a large market niche [BELLOTI F. et. al. 2009]. So, the development of this game type must use the methods and techniques of software engineering, emphasizing the reuse of components. This work was presented a set of requirements for the development of a reusable component to implement an intelligent behavior of software process elements, to the games created using this component be more realistic, simulating the actions of participants in real software process.

As a consequence to follow four of the six requirements defined in this work: the MSCI-ep becomes more transparent the necessary programming to describes the behavior of the software process element, this feature facilitates the development of behavioral models; MSCI-ep is responsible for all attributes of the behavioral model thus reduces the coupling with other components, and the dynamic generation of Intelligence.java class (containing the logic of behavioral model) increases the potential for reuse of this IA module; another positive consequence of the lower coupling between the MSCI-ep and the other components is the facility of testing behavioral models. However, the MSCI-ep is not able to classify behavioral models in difficulty levels.

A test was presented that demonstrated the MSCI-ep application as a AI component with the main features and functions to simulate the behavior of software process elements. Still is missing the scale tests. The MSCI-ep is promising, since the mainstream simulation games for teaching ES do not have specific components to deal with behavioral aspects of the software process simulation elements.

Acknowledgements

The authors would like to thank the Spider Project Team for their support to this paper.

References

WOODCOCK, S., 2001. Game AI: the state of the art industry 2000-2001. *Game Developer*, 8 (8), 36-44.

HOLLAND, M., 2004. *Citing references: the Harvard System* [online] Bournemouth University. Available from: www.bournemouth.ac.uk/library/using/harvard_system. html [Accessed 17 June 2006].

KARTCH, D., 2000. *Efficient rendering and compression for full-parallax computer-generated holographic stereograms*. PhD thesis, Cornell University.

PARKE, F.L. AND WATERS, K., 1996. Computer facial animation. Wellesley: AK Peters.

DUCHENEAUT, N., YEE, N., NICKELL, E. AND MOORE, J.R., 2006. "Alone together?": exploring the social dynamics of massively multiplayer online games. *In: Proceedings of the SIGCHI conference on Human Factors in computing systems, 22-27 April 2006 Montreal*. New York: ACM Press, 407-416.

BARROS, M DE O., 2001. *Gerenciamento de Projetos baseado em Cenários: Uma Abordagem de Modelagem Dinâmica e Simulação.* Thesis (PhD) held in the Postgraduate Program in Engineering Systems and Computing at UFRJ

BELLOTTI F ET AL.2009, *Enhancing the educational value of video game*s. Computers in Entertainment (CIE) -

SPECIAL ISSUE: Media Arts and Games (Part II). - New York, NY, USA .

BRENDAOU, R., ET. AL., 2007. *Definition of an Executable SPEM 2.0*. Procedings of the 14th Asia-Pacific Software Engineering Conference, IEEE CS Press, pp. 390–397.

CHAVES R. O., TAVARES E. M. DA C., OLIVEIRA, S. R. B., FAVERO E. L., 2010. *A Software Process Simulator Machine for Software Engineering Simulation Games*. In: SBGames

DANTAS, A. R., ET. AL.., 2004. *A Simulation-Based Game for Project Management Experiential Learning.* Proceedings of SEKE 2004, pp. 19-24.

DAREMA, F., 2004 - *Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulation and Measurements*. In: Computational Science - ICSS 2004. Springer Berlin / Heidelberg, 662-669.

DE JONG, K. A., 1975. *An analysis of the behavior of a class of genetic adaptive systems* (PhD thesis, University of Michigan)

DRAPPA, A. AND LUDEWIG, J., 2000. *Simulation in Software Engineering Training*. Proc. 22th Int'l Conf. Software Eng., ACM Press, pp. 199–208.

FILHO, V. V., 2005. REVOLUTION AI ENGINE – *Desenvolvimento de um motor de inteligência Artificial para a Criação de Jogos Eletrônicos*. TCC graduação em ciência da computação Universidade Federal de Pernambuco

FOLMER, E., 2007. *Component based game development: a solution to escalating costs and expanding deadlines?*. Proceedings of the 10th international conference on Component-based software engineering.

FORRESTER, J.W., 1961, *Industrial Dynamics, Cambridge*, MA: The MIT Press

NEVES, EVANDRO GREZELI DE BARROS ; BITTENCOURT, JOÃO RICARDO ; OSÓRIO, FERNANDO S. . *Proposta de um Motor de Inteligência Artificial para Jogos Digitais.*. In: SBGames / WJogos, 2005, São Paulo. Anais do SBGames 2005 / WJogos. São Paulo : USP - SBC, 2005. v. 1. p. 275-279.

OH, E. N., 2006. SIMSE: *A Software Engineering Simulation Environment for Software Process Education.* Thesis (PhD) held in the Postgraduate Program in Information and Computer Science at University of California, Irvine

OLIVEIRA, S. R. B., 2009. SPIDER - *Uma Proposta de Solução Sistêmica de um Suite de ferramentas de Software Livre de apoio à implementação do modelo MPS.BR.* Research Project. Instituto de Ciências Exatas e Naturais, Universidade Federal do Pará, Belém.

WANGENHEIM, C. G. V. AND SHULL, F. 2009. *To Game or Not to Game*. IEEE Software, vol. 26, no. 2, pp. 92-94.