Animating Organizational Patterns

Tomáš Frťala and Valentino Vranić
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava, Bratislava, Slovakia
E-mail: tomas.frtala@stuba.sk, vranic@stuba.sk

***Abstract*—Organizational patterns are the key to a stepwise adoption of agile and lean approaches and to a piecemeal growth of agile and lean organization of work. However, their text description is not easy to comprehend. In this paper, we introduce our initial efforts towards establishing an approach to animate organizational patterns as text adventure games. Players pass through a series of scenes described using Erickson's conversational hypnosis language patterns in order to better evoke their experience. The game scenario space is expressed using UML state machine diagrams. The approach is presented on adventure games we created for the Architect Also Implements organizational pattern.**

I. Introduction

Popularity of agile and lean approaches to software development in general—and Scrum in particular—is still on the rising curve for their flexibility and effectiveness while being close to the natural human attitude towards work [1]. However, despite vast efforts, there have been recurring difficulties in their adoption. They are even being rejected at different levels of organizational structure, including developers themselves. This is paradoxical, since agile and lean bring more freedom and involvement. Part of the reasons probably lies in not providing practitioners, who often have been conditioned (i.e., trained) to following heavyweight development processes, with an opportunity to *experience*—and not merely *understand*—what agile and lean are.

It is important to realize that agile and lean are actually not new nor did they appear suddenly. On the contrary, they are based on recurring patterns of organization that have been around long before they have been qualified as agile or lean. As Coplien and Harrison [2] showed, these *organizational patterns* can be identified in successful projects and presented in a concise, written form in a similar manner as design patterns in software development [3] or patterns in building architecture, where this idea has been first proposed by Alexander [4]. Community of Trust, Hallway Chatter, Architect Also Implements, and dozens of other organizational patterns constitute their catalogue [2].

Organizational patterns are the key to a stepwise adoption of agile and lean approaches and to a piecemeal growth of agile and lean organization of work. They can be applied to correct particular problems within an organization or to build a new organization. Scrum has been expressed in terms of Coplien and Harrison's organizational patterns [5] and there is even an ongoing work to define Scrum-specific patterns [6].

Having such pieces of experience in a condensed, apparently ready to consume form, may seem to be sufficient. However, this form is still an appeal to *understand* and not to *experience*, so the problem of acceptance we mentioned at the beginning remains. Agile games [7] attempt to overcome this problem by providing an non-expensive way to experience organizational problems without requiring any knowledge of software development, but consequently fail to relate to software development reality [8] as making paper airplanes or balloon blowing is quite distant from it.

In this paper, we introduce our initial efforts towards establishing a more involving and comprehensible form of presenting organizational patterns. The rest of the paper is organized as follows. Section II analyzes the possibilities of gaining experience through games as a way towards mastering organizational patterns. Section III introduces our approach to animating organizational patterns as text adventure games by the means of an example. Section IV discusses the approach. Section V points out work related to our approach. Section VI concludes the paper.

II. Gaining Experience Through Games

It is really hard to cut through all those forces, principles, and situations by which organizational patterns are typically described. It is even harder if you are to apply what you read in real life. Human nature requires experience. One way to gain it without risking failure in a real setting is to employ sociodrama in which participants can play the situation from the perspective of different roles [2]. However, people may be not willing to participate due to shyness or discomfort from conflicts that, though played, do arise and may evoke strong feelings among participants. Also, sociodrama is time consuming and moreover requires time coordination of many people.

A. Serious Games

One solution to the problems of employing sociodrama is offered by virtual games where person can act without feeling discomfort and as a bonus can be involved anytime in any place. Using the game format is not new and has been applied in learning software engineering under limited conditions [9], agile software development [7], or lean approach [10], [11], [12], [13]. Employing virtual reality in learning lean approach has also been reported [14]. This type of game is known as serious games. Apart from learning, serious games known as

human computation games are used to make people perform certain work while perceiving this as a fun activity [15], [16], [17].

B. Importance of Scenario

Although technical quality of the game is important, in serious games, the scenario is crucial to the perception of the situation by the user [18], [19]. If the scenario is involving, players easily identify with the main character, as we all know from popular games, despite simple graphics (recall Pacman) or no graphics at all, as in text adventure games.

When mimicking a sociodrama by a serious game, the goal is to make players feel the forces that drive the situation in the underlying organizational pattern as if they have been actually exposed to them. To achieve this, the scenario should sufficiently emphasize moments specific to that situation.

C. Power of Imagination

Erickson's conversational hypnosis [20] strives at utilizing subject's potential to induce a hypnotic trance and provide an effective problem treatment. For this, Erickson used specific language, in which Bandler and Grinder later identified sentence patterns [21] (e.g., Cause-Effect, Implied Causative, or Scope Ambiguity) that provided one of the cornerstones to their approach known as neuro-linguistic programming. Erickson's hypnotic mastery assumed careful yet quick tailoring of suggestions to each subject. However, there are attempts or even businesses based on generalized utilization relying on notions commonly perceived the same way by the majority of people (e.g., Uncommon Knowledge,[1] which offers recorded hypnotherapy sessions that should fit most of the people).

We are dealing with software development professionals or people that—though themselves not participating in programming on a daily basis—have a good idea of how software is developed and have been exposed to real critical situations that arise during software development. Using right words but avoiding binding to any specific programming languages or other notations, one can create a general setting capable of immersing a person being conditioned to software development process by simply being exposed to the respective environment into a desired fictional situation applicable to such environment.

The power of imagination is huge and can substitute other perceptions. For example, if someone says to you: "Your program suddenly stopped functioning after you implemented that new big algorithm," your mind employs the imagination and your memories and unfolds this condensed description into a specific situation that comprises also your feelings related to such situation. Thus, despite the description mentions no programming language, you instantly fill in your own. Accordingly, you gain a feeling of being in specific programming context, e.g., database development, mobile application development, etc.

It is important to note that not all details are worked out in your mind (e.g., what algorithm you are dealing with). That is not only unnecessary, but probably would just overwhelm your mental capacity. The description merely makes allusions on your experience and makes you recognize it. What is important is that the situation is *involving* and you are eager to solve it.

[1] http://www.uncommon-knowledge.co.uk/

III. Organizational Patterns as Text Adventure Games

To ease their mastering, we propose to represent organizational patterns as adventure games. An adventure game is a long-time known type of video game based on an involving story rather than action (as opposed to arcade games). The game consists of a number of scenes and the player as a protagonist is supposed to successfully solve the tasks in all or some of the scenes in order to finish the game.

The simplest form of realization of adventure games—known as text adventure games—involves only textual scene description and command line interface. This does not necessarily constitute a disadvantage as the stress is put on the game idea and puzzling tasks that often span over several scenes. Considering our audience to be highly educated, we assume that the textual description would be appropriate for organizational pattern adventure games. Moreover, as can be experienced from literary works, omitting direct graphical or video representation provides an opportunity to readers to develop their own representation they can more easily identify with while still maintaining conformance with the writer's intent in essential aspects.

A. Creating Scenarios

As we explained in Section II-B, scenarios are of utmost importance to the success of video games and in particular text adventure games. Applied to organizational patterns, this means attractive and involving scenarios have to be produced to successfully realize them as text adventure games.

To produce such a scenario for a given organizational pattern, one has to explore thoroughly it establishing its comprehension based upon the organizational pattern description and own professional experience. Subsequently, one has to come with one out of many possible situations fitting the pattern. Although such a situation embraces some particular issues (in order to appear as involving to players), these should be avoided as much as possible using Erickson's conversational hypnosis language patterns to give the opportunity to players to fill in their own details.

Erickson's language patterns lead towards using expressions that make players feel as if provided with a full situation description, yet making them fill in the details coming from their own experience with the organizational pattern instances they encountered in real life or, in the absence of such experience, at least makes them accept the description as their own. Erickson's language patterns can be constructed following their general form [21], but they also come intuitively from getting into the aforementioned attitude towards situation description.

Each role in the organizational pattern should be elaborated via a separate adventure game to allow players to experience the organizational pattern from different perspectives. Each

such adventure game scenario can be described as a set of scenes and transitions between them. UML state machine diagrams can be used to capture scenarios (not intended to be shown to players) with the scene description (to be shown to players) prepared separately. The following sections elaborate this further by the means of an example.

B. Architect Also Implements: Architect's Perspective

To demonstrate our approach, we present an adventure game for the Architect Also Implements organizational pattern [2]. This pattern puts the architect's role in a direct and active contact with program artifacts. This keeps architecture within the reality of implementation while providing an opportunity for "ordinary" developers to take part in it.

There are two key roles in this pattern: architect and developer. It should be possible to play the game from the perspective of each role separately. First, we will take the architect's perspective. Consider the UML state machine diagram in Figure 1. States represent scenes, each of wich is presented to the player in the form of text description, such as the one in Figure 2.

Transitions between scenes are determined by triggers and guards (introduced in square brackets as is common). Triggers are expressed by events that represent conscious decisions from the perspective of the player's role, e.g., *Adapt the architecture document*. After reading the scene description, the player chooses one of the events.

Guards represent conditions beyond the player's role perspective. These may come from the environment (*Problems have been found*) or other roles (*The architect joins you*). The game could be configured to allow the player to choose among the guards in the same manner as with events, to let the game make a random choice, or to always take the default choice (without displaying the guards at all). Default choices (i.e., transitions) are marked by the ≪default≫ stereotype.

In case no conscious decisions is available to the player, the generic *Continue* event is used. This means the player just continues to the next scene. Of course, if there are guards on a transition with the *Continue* event, the transition is determined by the guards.

Let us walk through the adventure game corresponding to the Architect Also Implements organizational pattern as depicted in Figure 1. At the beginning of the game, you are instructed to imagine extending the existing architecture (scene S1; see also Figure 2). As an architect, you capture your vision through the architecture document, but you encounter the identified problems have bigger impact on the entire system than you originally assumed.

Consider the first sentence of the scene description in Figure 2. This is effectively Implied Causative, one of the Erickson's language patterns [21]. This sentence makes the player infer the casual relationship between the meeting results and being given a task to extend the architecture. Avoiding any kind of explanation of this relationship and details in general (such as what kind of the system is being developed, the kind of architecture it is based upon, how is this architecture expressed, etc.), makes the player develop his or her own picture of the situation.

Getting on with our scenario, you and developers start to implement according to the architecture document (scene S2). However, you notice some problems (transition T2): you are unable to keep the implementation aligned with the architecture (scene S3; see also Figure 3). Initially, you decide to override the architecture document, i.e., to make changes to the implementation without reflecting them in the architecture document. However, new problems arise and you decide to adapt the architecture document (scene S4).

Let us stop at the first sentence of the scene description in Figure 3. This is Cause-Effect, another Erickson's language pattern [21]. The sentence makes the player connect designing and documenting architecture with realizing there are some problems as if there is a causal relationship between the two while—in reality—there is none. Or is there some after all? The elusiveness of the relationship is actually an additional allusion made upon what is the main idea of this organizational pattern: creating an architecture deprived of the connection to the real code brings problems. Avoiding any kind of explanation of this relationship and details in general (what kind of the system is being developed, the kind of architecture it is based upon, how is this architecture expressed, etc.), makes the player develop his or her own picture of the situation and consequently better identify with it.

Continuing with our scenario, a developer comes asking you about a particular problem in the implementation (scene S5; see also Figure 4). This also might have happened earlier, during your initial implementation efforts (transition T5).

If you decide to leave the implementation to developers (transition T7), they may do their best to solve the problems, but it may be counterproductive if developers create a solution that overrids the architecture document without reflecting this in the architecture document. By being indolent to technical details, you will hardly be able to solve similar situations in future (scene S9).

But if you decide to join the developer in implementation (transition T6), you will be able to better understand the problems in the context and see actual consequences of your architecture and decisions. The developer will also gain expertise from you (scene S6).

During the cooperation with the developer you may identify errors in the architecture document (scene S7). These may vary in importance. It is possible that the errors are a consequence of previous bad decisions and implementation.

If the identified errors are serious, you decide that the implementation cannot continue without adapting the architecture document (transition T10). However, now you know exact technical reasons for the failure of your architecture (scene S8).

You might also come to a solution not in accordance with the architecture document. Subsequently, you decide to apply it, i.e., to override the arcitecture document, and continue with the implementation together with the developer (transition T9). Nevertheless, during the implementation effort with the

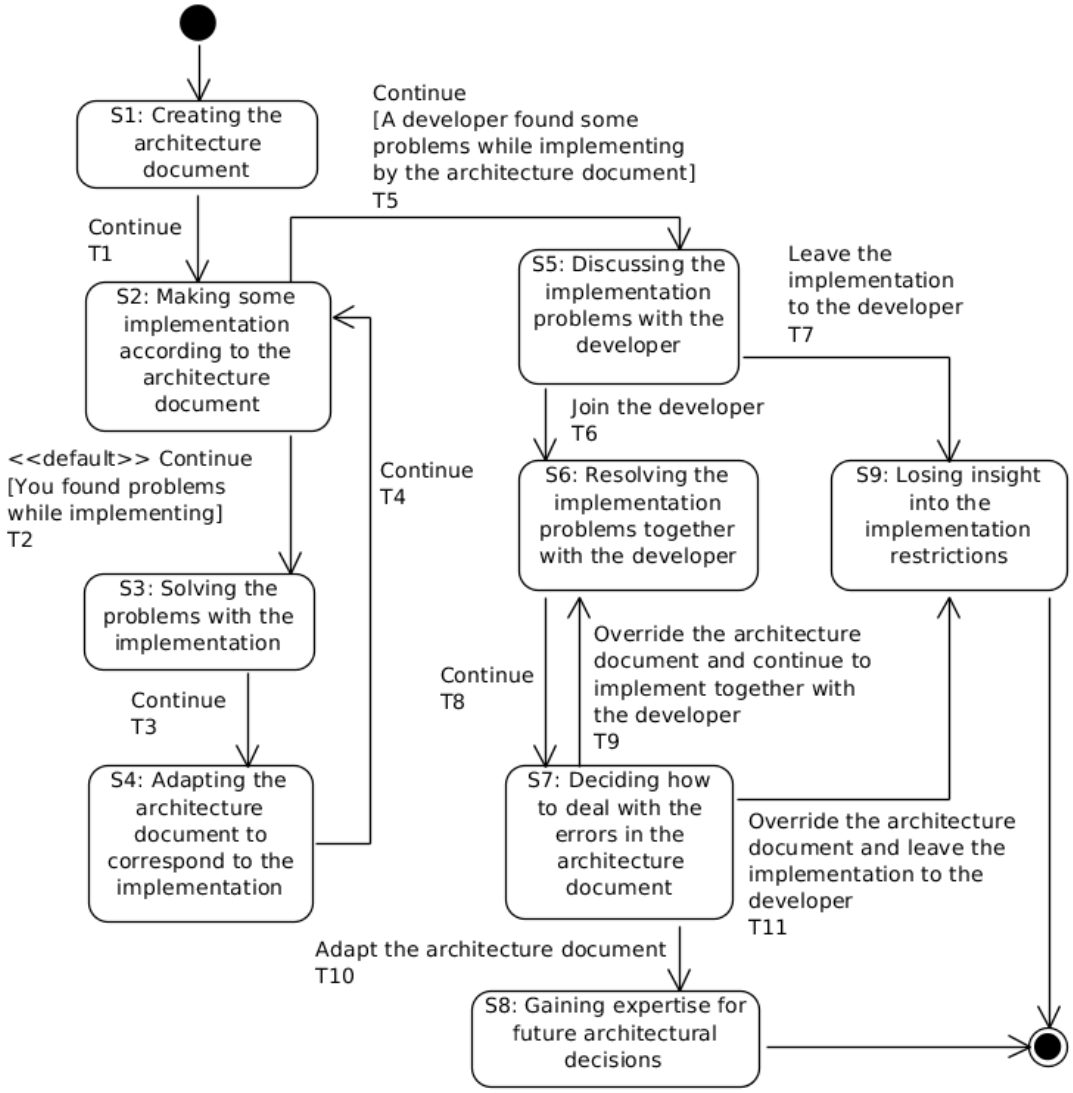

Figure 1. Architect Also Implements from the architect's perspective.

developer, yet another problem may arise (transition T8). Consequently, if this is a serious problem, the only right way of its solving, besides discussion and advising, is to adapt the architecture document.

You may be busy enough with solving other problems or doing some other work, or you even may lose interest. So when you propose a solution overriding the architecture document, you may just leave the implementation to the developer (transition T11).

C. Architect Also Implements: Developer's Perspective

Because the Architect Also Implements pattern is of concern to developers, too, a separate adventure game should be provided to them. This also provides an opportunity to architects to experience the developer's role.

The game we devised for the developer's role in the Architect Also Implements organizational pattern is depicted in Figure 5. In comparison to the architect, a developer has fewer possibilities to influence the development process, so the majority of transitions in the corresponding state machine diagram are determined by guards, which—as we explained—express the conditions beyond the player's role perspective.

IV. Discussion

In our approach to animating organizational patterns as adventure games, we used only textual rendering of scenes. Graphics, animated sequences, and video sequences can be used alternatively or complementarily to textual rendering. Another option is to employ virtual reality and thoroughly simulate reactions of people, but this would be too costly. Moreover, scenes would have to be fully elaborated leaving

After one of those meetings in which new functionality is being agreed upon, you find yourself carrying out the analysis of the prerequisites to extend the existing architecture. The architecture has to accommodate the new functionality.

To meet the requirements, you decide to study several versions of the frameworks used in implementation. At the same time, your mind starts shaping a vision of the architecture.

You document your findings and the vision in the architecture document double checking its consistency.

Having finished, you introduce the new architecture to developers.

→ Continue

Figure 2. Scene S1, Creating the architecture document.

As you design and document the architecture extension, you realize that the identified problems have bigger impact on the entire system than you originally assumed.

Since you are sure the architecture can't be incorrect—you've created it yourself and double checked it—you search for possible solutions of these problems without considering to adapt the architecture.

→ Continue

Figure 3. Scene S3, Solving the problems with the implementation.

almost nothing to player's imagination, which plays a significant role in our approach.

As we saw in the previous section, conditions determining transitions to scenes which are beyond the player's role perspective are manipulable by the player, decided randomly, or default transitions are triggered. These conditions may be sourced in the environment itself, such as a program failure, or in other roles, such as a boss deciding to engage another developer. The conditions sourced in other roles could be decided in a more sophisticated way using the agent-oriented approach, which would enable a more elaborated interaction between the player and these roles. This is also related to the possibility of having a multiplayer game instead of single-player games as it is currently in our approach. Other players could be humans instead being computer generated.

Different adventure games for the same organizational pattern are possible. These may be based on variant scenarios, but also may come from cultural and linguistic differences. Also, we consider separate adventure games for each organizational pattern. However, organizational patterns actually form pattern languages and a continuous adventure game reflecting the whole pattern language could be devised, too. Still, individual pattern games have an advantage of being readily accessible to target a problem at hand, requiring significantly less time than "playing" the whole pattern language.

A developer approaches you to discuss the implementation problems.

The developer was unable to follow your architecture. The solution seemed viable, but the technologies don't permit that kind of organization. Moreover, the developer is afraid of infringing compatibility with new framework versions.

You...

→ feel responsible for the whole system and moreover the technical solution to the architecture problem puzzles you. You decide make your time and join the developer in the implementation effort.

→ consider your architecture to be correct and the problems seem to you as an implementation detail. You explain your intent with the architecture and leave the implementation to the developer.

Figure 4. Scene S5, Discussing the implementation with the developer.

V. Related Work

Attempts have been made to evaluate and improve different pattern forms [22]. However, these forms remain passive in their essence as opposed to interactivity we strive for.

A notation similar to activity diagrams or BPMN has been used to express Scrum processes identified in real organizations [23]. Patterns are then identified or reinforced by rearranging corresponding activities. In this sense, our pattern state machine diagrams—albeit not directly in their current form—could be used to ameliorate this process or even to automate it.

Learning organizational patterns from a simulated experience as opposed to learning them from text description recalls establishing a correct perception of the matter being taught by preventing the divergence of the school model and intuitive model, which has been identified as a major problem in teaching by Mahajan in his course on teaching methodology [24] (see Mahajan's lecture 3 in particular).

VI. Conclusions and Further Work

We explained our initial efforts towards establishing an approach to animate organizational patterns as text adventure games. Players pass through a series of scenes described using Erickson's conversational hypnosis sentence patterns in order to better evoke their experience. Each role in the organizational pattern is elaborated via a separate adventure game. The game scenario space is expressed using UML state machine diagrams.

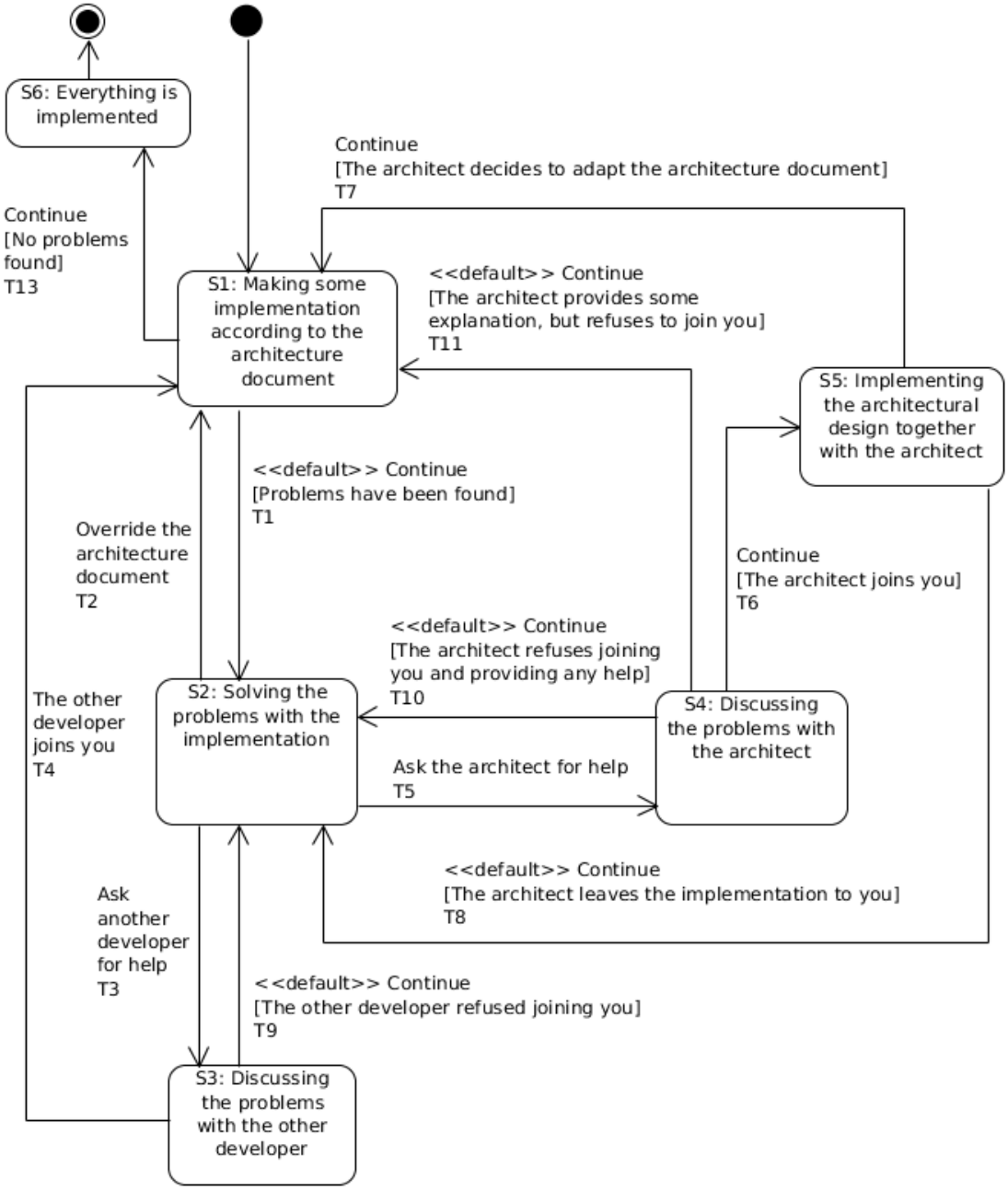

Figure 5. Architect Also Implements from the developer's perspective.

The approach is presented on adventure games we designed for the Architect Also Implements organizational pattern. Sample scenes (Figure 2, 3, and 4) in the form intended to be displayed to players are included to illustrate the kind of language we assume appropriate for this (Section III).

In Section IV we outlined several interesting directions for further work. Our next steps will be to express a greater number of organizational patterns as text adventure games and to evaluate efficiency of this representation with practitioners. These comprise our graduate students taking the course on team software development and our industrial partners involved in assisting their clients in switching to Scrum.

ACKNOWLEDGMENTS

The work reported here was supported by the Scientific Grant Agency of Slovak Republic (VEGA) under the grant No. VG 1/1221/12.

This contribution/publication is also a partial result of the Research & Development Operational Programme for the project Research of Methods for Acquisition, Analysis and

Personalized Conveying of Information and Knowledge, ITMS 26240220039, co-funded by the ERDF.

REFERENCES

[1] V. Vranić, "Promoting natural human attitude towards work: Scrum," in *Proceedings of Konferencija Mreža 2013 – Internet u edukacionom i poslovnom okruženju (Conference Mreža 2013 -– Internet in Educational and Business Environment)*, Valjevo, Serbia, 2013, pp. 8–12.

[2] J. O. Coplien and N. B. Harrison, *Organizational Patterns of Agile Software Development*. Prentice Hall, 2004.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[4] C. Alexander, *The Timeless Way of Building*. University of Oxford Press, 1979.

[5] Scrum Pattern Community, "Published patterns," https://sites.google.com/a/scrumplop.org/published-patterns/home, 2015.

[6] J. Sutherland, J. Coplien, J. Østergaard, G. Bjørnvig, and D. Friis, "Scrum as organizational patterns," https://sites.google.com/a/scrumorgpatterns.com/www/, 2011.

[7] B. Scharlau, "Games for teaching software development," *Proceedings of 18th ACM conference on Innovation and technology in computer science education*, 2013.

[8] M. Paasivaara, V. Heikkilä, C. Lassenius, and T. Toivola, "Teaching students scrum using LEGO blocks," *Companion Proceedings of 36th International Conference on Software Engineering – ICSE Companion 2014*, pp. 382–391, 2014.

[9] A. A. Deshpande and S. H. Huang, "Simulation games in engineering education: A state-of-the-art review," *Computer Applications in Engineering Education*, vol. 19, no. 3, pp. 399–410, Sep. 2011.

[10] H. McManus and E. Rebentisch, "Experiences in simulation-based education in engineering processes," *2008 38th Annual Frontiers in Education Conference*, pp. S1C–21–S1C–26, Oct. 2008.

[11] L. Zhou, Y. Xie, N. Wild, and C. Hunt, "Learning and practising supply chain management strategies from a business simulation game: A comprehensive supply chain simulation," *Simulation Conference*, 2008.

[12] I. D. Silva, A. R. Xambre, and R. B. Lopes, "A simulation game framework for teaching lean production," *International Journal of Industrial Engineering and Managemen*, vol. 4, no. 2, pp. 81–86, 2013.

[13] A. G. Ramos, M. P. Lopes, and P. S. Avila, "Development of a platform for lean manufacturing simulation games," *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, vol. 8, no. 4, pp. 184–190, Nov. 2013.

[14] A. Gamlin, P. Breedon, and B. Medjdoub, "Immersive virtual reality deployment in a lean manufacturing environment," *Interactive Technologies and Games*, pp. 51–58, 2014.

[15] J. Šimko and M. Bieliková, *Semantic Acquisition Games: Harnessing Manpower for Creating Semantics*. Springer, 2014.

[16] J. Šimko, M. Tvarožek, and M. Bieliková, "Human computation: Image metadata acquisition based on a single-player annotation game," *International Journal of Human-Computer Studies*, vol. 71, no. 10, pp. 933–945, 2013.

[17] ——, "Semantics discovery via human computation games," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 7, no. 3, pp. 23–45, 2011.

[18] J. P. Gee, "What video games have to teach us about learning and literacy," *Computers in Entertainment*, vol. 1, no. 1, p. 20, Oct. 2003.

[19] C. Hartog, "Scenario design for serious gaming – guiding principles for the design of scenarios and curricula in military job oriented training," Master's thesis, Delft University of Technology, 2009.

[20] M. H. Erickson and E. L. Rossi, *Hypnotherapy: An Exploratory Casebook*. Irvington Publishers, 1979.

[21] R. Bandler and J. Grinder, *Patterns of the Hypnotic Techniques of Milton H. Erickson, M.D.* Meta Publications, 1975, vol. I.

[22] M. Wieck, "Patterns: Lust for glory," in *Proceedings of 14th Annual Conference of the National Advisory Committee on Computing Qualifications, NACCQ 2001*, July 2001, pp. 145–151. [Online]. Available: http://www.citrenz.ac.nz/conferences/2001/145.pdf

[23] X. Meng, Y. Wang, L. Shi, and F. Wang, "A process pattern language for agile methods," in *Proceedings of 14th Asia-Pacific Software Engineering Conference, APSEC 2007*. Nagoya, Japan: IEEE Computer Society, Dec. 2007, pp. 374–381.

[24] S. Mahajan, "Teaching college-level science and engineering," Massachusetts Institute of Technology: MIT OpenCourseWare, http://ocw.mit.edu/5-95js09, 2009.