APPLYING GAMES-BASED LEARNING TO TEACH SOFTWARE ENGINEERING CONCEPTS

Thomas M. Connolly, Mark Stansfield and Tom Hainey
School of Computing, University of Paisley, High St, Paisley, PA1 2BE, Scotland

Keywords: Games-based learning, software engineering, pedagogical praxis.

Abstract: For some time now, computer games have played an important role in both children and adults' leisure activities. While there has been much written on the negative aspects of computer games, it has also been recognised that they have potential advantages and benefits. There is no doubt that computer games can be highly engaging and incorporate features that are extremely compelling. It is these highly engaging features of computer games that have attracted the interests of educationalists. The use of games-based learning has been growing for some years now, however, within software engineering there is still a dearth of empirical evidence to support this approach. In this paper, we examine the literature on the use of computer games to teach software engineering concepts and describe a computer game we have been developing to teach these concepts.

1 INTRODUCTION

It is generally accepted that computer games are an extremely motivating and engaging medium and represent a new form of popular culture. There is also a growing recognition of the potential benefits that can be gained in using computer games within teaching and learning, although there are still many critics of this approach. We have been attempting to use a games-based learning approach to support the teaching of requirements collection and analysis within a software engineering course in Higher Education for some years now (Connolly *et al.*, 2004). Recently we have teamed up with a Scottish games-based learning company to develop an expanded version of the game for both the academic and training communities. In this paper, we examine the literature on the use of games-based learning within software engineering and also examine some of the issues underlying the teaching of the abstract and complex domain of requirements collection and analysis and, more generally, software engineering. We then discuss the high-level requirements for our game and provide an overview of the game play and an outline subsystem design.

2 PREVIOUS RESEARCH

Software engineering has been described as a "wicked problem", characterized by incomplete, contradictory and changing requirements, and solutions that are often difficult to recognize as such because of complex interdependencies (DeGrace & Hulet Stahl, 1998). According to Armarego (2002), there is an educational dilemma in teaching such problems in software engineering because:

- complexity is added rather than reduced with increased understanding of the problem;
- metacognitive strategies are fundamental to the process;
- a rich background of knowledge and intuition are needed for effective problem-solving;
- a breadth of experience is necessary so that similarities and differences with past strategies are used to deal with new situations.

Oh and Van der Hoek (2001) identify a number of other issues that complicate the teaching of the software process:

- *Software development is non-linear*: activities, tasks and phases are repeated and multiple events happen at the same time. Managing two similar projects in the same way may not produce the same outcome due to the presence of several (possibly unexpected) factors (e.g., technical advances, client behaviours or expectations).

- *Software development involves several intermediate steps and continuous choices between multiple, viable alternatives*: even with careful planning, not all events that can occur can be anticipated at the start of a project. Difficult decisions must be made, tradeoffs considered and conflicts handled.
- *Software development may exhibit dramatic effects with non-obvious causes*: while software development has several cause-and-effect relationships (e.g., it is more cost-effective to identify flaws in the earlier phases of development than to identify them in the later phases), there are other situations that may arise in which the cause is not so apparent. For example, Brook's Law states that adding people to a project that is already late typically makes that project later.
- *Software engineering involves multiple stakeholders*: clients and non-development personnel in an organization all make decisions that impact development.
- *Software engineering often has multiple, conflicting goals*: software development includes tradeoffs between such things as quality versus cost, timeliness versus thoroughness, or reliability versus performance.

Two further issues arise with teaching software development that we are interested in taking into consideration in any learning environment we develop are:

- *Communication*: software engineers must be able to communicate, both verbally and in writing, with staff internal to the project (project manager, team leaders, analysts, designers, developers, testers, quality assurance) as well as with external stakeholders.
- *Pedagogical praxis*: Shaffer (2004a) proposes a theory of 'pedagogical praxis', which links learning and doing within an extended framework of communities of practice (Lave, 1991; Lave & Wenger, 1991). Pedagogical praxis is based on the concept that different professions (for example, lawyers, doctors, software engineers) have different epistemologies (epistemic frames) – different ways of knowing, of deciding what is worth knowing and of adding to the collective body of knowledge and understanding. For a particular community, the epistemic frames define "knowing *where* to begin looking and asking questions, knowing *what* constitutes appropriate evidence to consider or information to assess, knowing *how* to go about gathering that evidence, and knowing *when* to draw a conclusion and/or move on to a different issue" (Shaffer, 2004b, pp. 4). Implementation of pedagogical praxis requires a faithful recreation of the professional community, one that is "thickly authentic"; that is, one where (a) learning is personally meaningful for the learner, (b) learning relates to the real-world outside the classroom, (c) learning provides an opportunity to think in the modes of a particular profession and (d) learning where the means of assessment reflect the learning process (Shaffer and Resnick, 1999). Connolly and Begg (2006) have suggested that the term thickly authentic be extended to incorporate: (e) learning using the tools and practices of the modern-day professional.

According to Schön (1983, 1987) the following are some of the key problems in teaching an abstract subject of this nature:

- It is learnable but not didactically or discursively teachable: it can be learned only in and through practical operations.
- It is a holistic skill and parts cannot be learned in isolation but by experiencing it in action.
- It depends upon the ability to recognize desirable and undesirable qualities of the discovered world. However, this recognition is not something that can be described to learners, instead it must be learned by doing.
- It is a creative process in which a designer comes to see and do things in new ways. Therefore, no prior description of it can take the place of learning by doing.

Students often have considerable difficulty comprehending implementation-independent issues and analyzing problems where there is no single, simple, well-known, or correct solution (Connolly and Begg, 2006). They have difficulty handling ambiguity and vagueness and they can also display an inability to translate tutorial examples to other domains with analogous scenarios, betraying a lack of transferable analytical and problem-solving skills (Connolly & Stansfield, in press). Kriz (2003) highlights the point that the majority of students are not competent enough to put their knowledge into practice and they are unable to cope successfully with the everyday tasks associated with the practice of their chosen field. These problems can lead to confusion, a lack of self-confidence and a lack of motivation to continue.

Many of the above characteristics make teaching requirements collection and analysis and, more generally, the software development process, problematic using didactic approaches to teaching and learning and the practical experience provided falls far short of what a student can expect "in the real world". Instead, these issues suggest that students can only learn about software engineering by doing software engineering and rely less on overt lecturing and traditional teaching. This approach requires a shift in the roles of both students and teachers, with the student becoming an apprentice, exploring and learning about the problem in the presence of peers (who may know more or less about the topic at hand) and the teacher moving from being the "knowledgeable other" towards becoming a facilitator, who manages the context and setting, and assists students in developing an understanding of the material at hand (Koehler & Mishra, 2005).

We advocate an alternative teaching paradigm for software engineering based on *constructivism*. Cognitive constructivism views learning as an active process in which learners construct new ideas or concepts based upon their current/past knowledge. The learner selects and transforms information, constructs hypotheses and makes decisions, relying on a cognitive structure to do so (Piaget, 1968). Social constructivism, seen as a variant of cognitive constructivism, emphasizes that human intelligence originates in our culture. Individual cognitive gain occurs first in interaction with other people and in the next phase within the individual (Forman & McPhail, 1993). These two models are not mutually exclusive but merely focus upon different aspects of the learning process. In fact, Illeris (2003) believes that all learning includes three dimensions, namely, the cognitive dimension of knowledge and skills, the emotional dimension of feelings and motivation, and the social dimension of communication and cooperation – "all of which are embedded in a societally situated context".

Figure 1 provides a representation of the environment we will use as the basis for the development of the games-based learning application and the problems to be addressed.

3 THE SDSIM GAME

The development of the SDSim game is being underpinned by Participatory Design principles with users and other stakeholders playing a prominent role in all the stages relating to design, development and evaluation. The benefits of Participative Design are that it can provide better project control, better communication, more satisfied users and participants, lessens the need for costly corrective action post implementation and can provide more innovative and creative solutions than might have

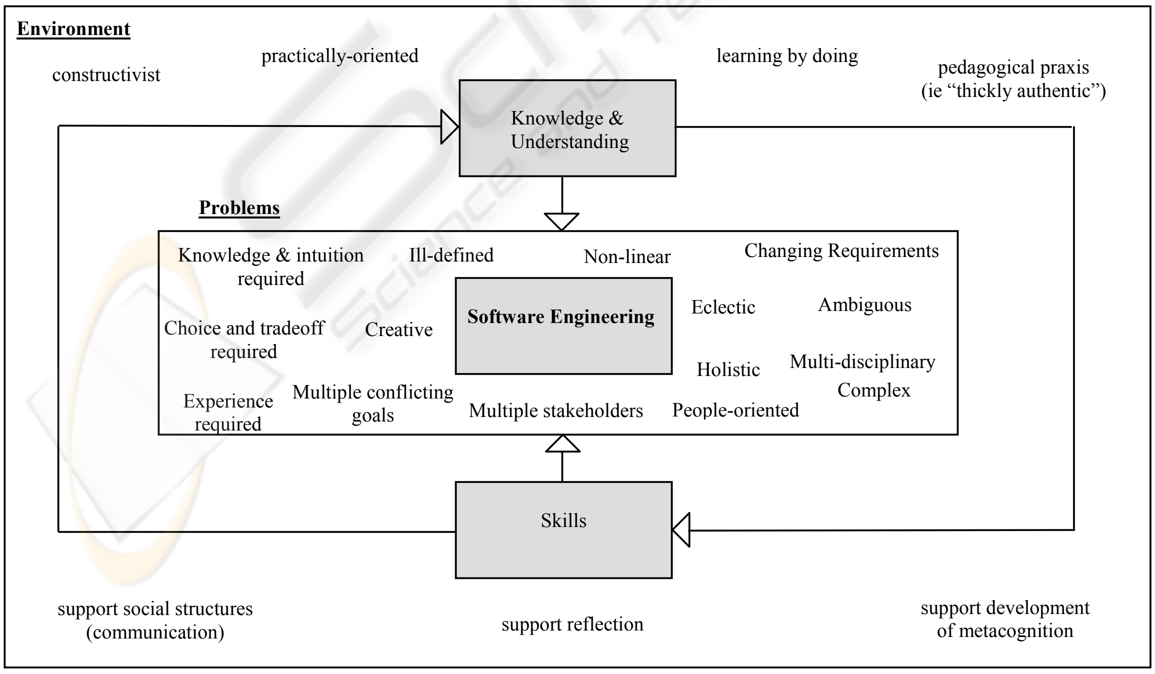

Figure 1: Environment for teaching software engineering (and problems that need to be addressed).

otherwise been possible (Kensing and Blomberg, 1998; Cherry and Macredie, 1999). To support the design, development and evaluation of the game, it was decided to establish a steering committee comprising senior representatives from industry and commerce, a number of academic representatives, the project managers and the developers of the game. By drawing upon the expertise and views of senior managers from industry and commerce it is hoped that the game will have a greater degree of relevance and significance to a wider audience other than students in higher education. In addition, it is hoped the SDSim game will utilise and develop a wider range of skills and knowledge that might be transferable across a wider section of industry and commerce. We now discuss the high-level objectives of the game, the game play and then provide an outline design of the game itself.

3.1 High-Level Objectives

The games-based learning environment should provide a rich learning experience through the creation of a range of project scenarios that will:

- Promote an engineering ethos that emphasizes fitness for purpose as the guiding principle in the design, development and assessment of information systems and their components.
- Enable the learner to take a disciplined approach to requirements collection and analysis, and to the high level specification, design and implementation of information systems and their components.
- Enable the learner to handle complexity, vagueness and ambiguity during the project.
- Enable the learner to develop a range of project management skills.
- Assist the learner to develop analytical and problem-solving skills and transferable skills.
- Assist the learner to develop the skills required for both autonomous practice and team-working.
- Assist the learner to develop reflection and metacognitive strategies.

In discussion with the advisory group, the following requirements were identified:

- The game will be targeted at both university students in a computing-related subject and also the professional training market.
- The game must support a number of players carrying out different roles (for example, analyst, developer, project manager) as well as a facilitator. Communication between players should be supported.
- The facilitator will be able to see what the players are doing, will be able to intervene in the game (for example, to modify the frequency of new projects, to modify the number of people assigned to a project) and will be able to call team meetings to discuss issues that have arisen in the team's play.
- Ideally, in a team-based activity when a player is not available the game (AI) should play that role.
- The game must be scenario-based to allow the players access to a range of project scenarios to provide practical experience.
- The game must have a reasonably authentic underlying business model to model clients, projects, staff, suppliers and competitors. The model should take cognisance of a range of project variables such as project budget, time, staff, staff specialisations, staff costs, resource costs. These variables would be scenario-specific.
- The game should run in an online environment.
- Game play should be recorded wherever possible to support debriefing, post-game analysis and evaluation.

3.2 Game Play

The basic idea of the game is for the team (comprising one or more players) to manage and deliver a number of software development projects. Each player has a specific role, such as project manager, systems analyst, systems designer or team leader. A bank of scenarios have been created based on case studies the authors have been using for many years in teaching and learning; for example, the *DreamHome Estate Agency* (Connolly & Begg, 2005), the *StayHome Online DVD Rentals* company and the *Perfect Pets Veterinary Clinic* (Connolly & Begg, 2002), the *Blackwood Library* and the *Fair Winds Marina* (Connolly *et. al.*, 2004). Each scenario has an underlying business model; for example, there will be a budget for the delivery of the project, a set timescale for the delivery of the project and a set of resources (for example, staff with specified technical specilisations) that can be used on the project. Additional resources can be brought in for a project although this will have a cost and timescale (delay) associated with it. The project manager has overall responsibility for the delivery of each project on budget and on time and is given a short brief for each project. Communication is one of the key aspects of the game and the project

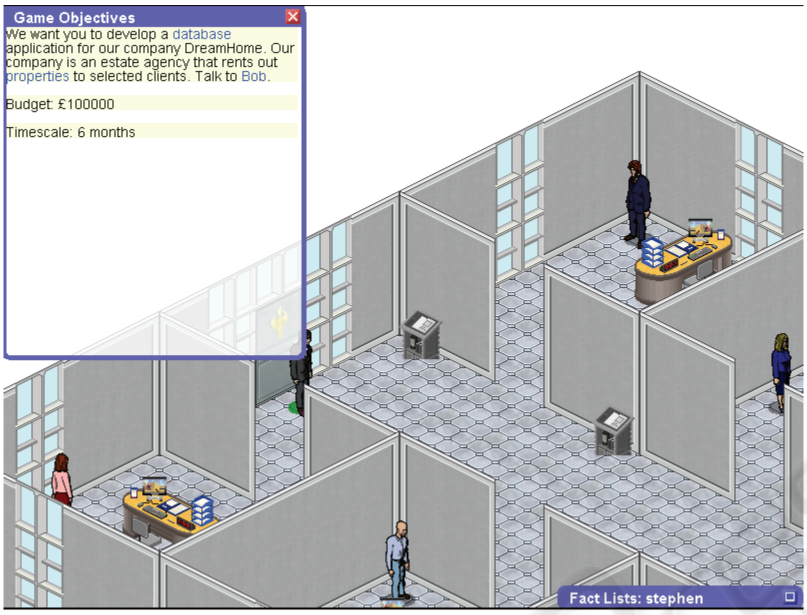
Game Objectives
We want you to develop a database application for our company DreamHome. Our company is an estate agency that rents out properties to selected clients. Talk to Bob.
Budget: £100000
Timescale: 6 months
Fact Lists: stephen

Figure 2: Screen during requirements collection.

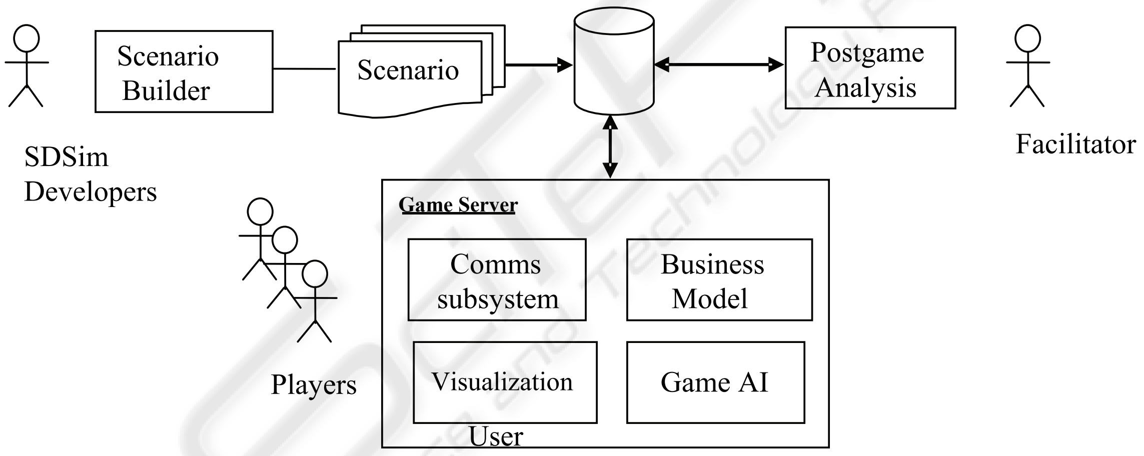

Figure 3: Internal subsystems of the SDSim game.

manager must communicate relevant details of the project to the other players. This will be done using a message metaphor – any player can communicate with any other player(s) by sending a message. Players have a message board that indicates whether there are any unread messages.

The player(s) assigned to the system analyst role has to identify the requirements for the project. To do this, the player must move through the game and 'talk' to the non-player characters (NPCs) in the game, as illustrated in Figure 2. In addition, there are objects in the game that can also convey relevant information when found (for example, a filing cabinet may convey requirements). For the prototype game we are using written transcripts in place of NPC speech. We hope shortly to use lip synching within the game to have the NPCs 'talk' to the system analyst. Each NPC's 'speech' will contain some general background details and a number of requirements (the analyst has to distinguish the requirements from the general details). Visiting the same NPC may generate the same speech or a new speech. Each speech will generate a transcript that the analyst can visit at any point in the game. The transcript is presented as a numbered list of requirements. During the play, the analyst can use the transcripts to produce an initial 'wishlist' of requirements, which can be refined until such time as the analyst believes all requirements have been identified, at which point the analyst can send the completed requirements to the project manager. The project manager now has two choices: send the requirements to the designer to produce an outline high-level design or consider the requirements to be incorrect and ask the analyst to rework the requirements (asking for rework will have a 'cost' associated with it).

During this period, the designer will be provided with some background information relevant to the design phase (for example, high-level components that the company might have developed previously, technical experience of the staff, technical resources the designer has access to and software and hardware that can be bought externally). Upon receiving the requirements, the designer must produce a high-level design that addresses the clients' requirements and must identify what will be developed 'in-house' and what software/hardware will be bought in. In addition, the designer must provide some estimate of cost and timescale to implement the system. Again, the design will go back to the project manager to accept or reject (in which case the design must be reworked by the designer at 'cost').

The implementation phase is handled by the team leader who is given a brief by the project manager (high level design, available budget, available staff). The team leader is responsible for the delivery of the implementation phase. However, during this period the team leader may have to handle a number of planned events (such as staff holidays) and unexpected events (such as staff becoming ill, leave, and some activities taking longer than planned). Some events the team leader may be able to handle autonomously within the remit provided; however, with others the team leader may need to consult the project manager to seek a solution.

The facilitator will have access to the game play and will be able to intervene during the play. One intervention is to call a (physical or virtual) team meeting because of problems identified with the running of the project. There are a number of other interventions, such as changing the requirements during the design or implementation phase, reducing the number of staff available for the project, making staff go off sick, making staff leave the company.

3.3 Game Design

The game is based on the traditional multi-client/single-server architecture. The subsystem design is shown in Figure 3:

- The Scenario Builder is an offline utility to allow us to create and update the game scenarios. These scenarios are stored in the server-side database.
- The Postgame Analysis is a second offline utility to allow us to provide data to the facilitator on how the team has performed. The utility will also provide us with data to evaluate the impact of the game and to eventually produce longitudinal analyses.
- The Game Server consists of four main subsystems:
  - The Comms subsystem, which allows players to communicate with each other.
  - The Visualisation/User Interface, which handles what the players see on the screen and what they can do.
  - The Business Model, which implements both the general business rules and the business rules specific to each scenario. This will be loosely based on the SESAM model (Mandl-Striegnitz, 2001).
  - The Game AI (Artificial Intelligence), which implements 'missing players'. At the time of writing, this subsystem has not been fully designed.

The input to date from the advisory group has been extremely useful and has helped shape the design of the game. We are currently implementing a prototype of the game and an early version of the fully system is due for completion in early 2007.

4 CONCLUSIONS

In this paper we have examined previous approaches to the application of games-based learning to software engineering and have found a significant dearth of empirical research to support this approach. Software engineering has been described as a "wicked problem", characterized by incomplete, contradictory and changing requirements and solutions that are often difficult to recognize as such because of complex interdependencies. Other issues that complicate the teaching of software engineering are that software development is non-linear, it involves several intermediate steps and choices between multiple, viable alternatives, it may exhibit dramatic effects with non-obvious causes and it involves multiple stakeholders. Finally, we have described the design of a new games-based learning application aimed at the teaching of requirements collection and analysis, design and project management aimed at both the academic and training markets. We consider evaluation to be key to the entire development process and have adopted a Participatory Design approach from the outset. In the design of the game we have included a Postgame

Analysis utility to support the collection of empirical evidence on the use of this game.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the Scottish Executive for their support of this work and TPLD for their collaboration.

REFERENCES

Armarego, J. 2002. Advanced Software Design: A Case in Problem-Based Learning. *Proceedings of the 15th Conference on Software Engineering Education and Training*, 25–27 February 2002, Covington, Kentucky, USA, pp. 44–54.

Cherry, C. and Macredie, R.D. 1999. The importance of context within information system design: an assessment of participative design, *Requirements Engineering*, 4: 103-114.

Connolly, T.M. & Begg, C.E 2006. A constructivist-based approach to teaching database analysis and design. *Journal of Information Systems Education*, 17(1): 43-54.

Connolly, T.M. & Begg, C.E 2005. *Database Systems: A practical approach to design, implementation, and management*, 4th edition. Addison Wesley Longman: England.

Connolly, T.M. & Begg, C.E 2002. *Database Solutions: A step-by-step approach to building databases*, 2nd edition. Addison Wesley Longman: England.

Connolly, T.M., McLellan, E., Stansfield, M.H., Ramsay J. & Sutherland J. 2004. Applying Computer Games Concepts to Teaching Database Analysis and Design. *International Conference on Computer Games, AI, Design and Education*, Reading, UK, November 2004.

Connolly, T.M. & Stansfield, M.H. in press. From eLearning to games-based eLearning: using interactive technologies in teaching an IS course. *International Journal of Information Technology and Management*.

DeGrace, P. & Hulet Stahl, L. 1998. *Wicked Problems, Righteous Solutions: A Catalog of Modern Engineering Paradigms*, Prentice Hall.

Forman, E. & McPhail, J. 1993. Vygotskian perspectives on children's collaborative problem-solving activities. (*In* Forman, E.A., Minick, N. & C. Addison Stone (eds), *Contexts for learning. Sociocultural dynamics in children's development*, Oxford University Press, Oxford).

Illeris, K. 2003. Towards a contemporary and comprehensive theory of learning. *International Journal of Lifelong Learning*, 22(4): 396-406.

Kensing, F. & Blomberg, J. 1998. Participatory design: issues and concerns. *Computer Supported Cooperative Work*, 7: 167-185.

Koehler, M.J. & Mishra, P. 2005. Teachers Learning Technology by Design. *Journal of Computing in Teacher Education*, 21(3), Spring 2005.

Kriz, W.C. 2003. Creating effective learning environments and learning organizations through gaming simulation design. *Simulation and Gaming*, 34(4): 495–511.

Lave, J. 1991. *Situating learning in communities of practice*. Washington, DC: American Psychological Association.

Lave, J. & Wenger, E. 1991. *Situated learning: Legitimate peripheral participation*, Cambridge University Press, Cambridge, England.

Oh, E. & Van der Hoek, A. 2001. Adapting Game Technology to Support Individual and Organizational Learning. *Proceedings of the 13th International Conference on Software Engineering and Knowledge Engineering,* Buenos Aires, Argentina, June 2001.

Piaget, J. 1968. *Six Psychological Studies*. Vintage Books, New York.

Schön, D.A. 1983. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York.

Schön, D.A. 1987. *Educating the Reflective Practitioner: Towards a New Design for Teaching in the Professions*. Jossey-Bass Inc., San Fransisco.

Shaffer, D.W. 2004a. Pedagogical Praxis: The Professions as Models for Postindustrial Education. *The Teachers College Record*, 106(7): 1401–1421.

Shaffer, D.W. 2004b. Epistemic frames and islands of expertise: Learning from infusion experiences. In *Proceedings International Conference of the Learning Sciences*, Santa Monica, CA. Retrieved 28 July 2005, from http://www.education.wisc.edu/edpsych/facstaff/dws/papers/epistemicframesicls04.pdf.

Shaffer, D.W. & Resnick, M. 1999. Thick authenticity: New media and authentic learning. *Journal of Interactive Learning Research*, 10(2): 195–215.