

# Evaluating Computer-Supported Learning Initiatives

**John B. Nash**

*Stanford University, USA*

**Christoph Richter**

*University of Hannover, UK*

**Heidrun Allert**

*University of Hannover, UK*

## INTRODUCTION

The call for the integration of program evaluation into the development of computer-supported learning environments is ever increasing. Pushed not only by demands from policy makers and grant givers for more accountability within lean times, this trend is due also to the fact that outcomes of computer-supported learning environment projects often fall short of the expectations held by the project teams. The discrepancy between the targets set by the project staff and the outcomes achieved suggests there is a need for formative evaluation approaches (versus summative approaches) that facilitate the elicitation of information that can be used to improve a program while it is in its development stage (c.p., Worthen, Sanders & Fitzpatrick, 1997). While the call for formative evaluation as an integral part of projects that aim to develop complex socio-technical systems is widely accepted, we note a lack of theoretical frameworks that reflect the particularities of these kind of systems and the ways they evolve (c.p., Keil-Slawik, 1999). This is of crucial importance, as formative evaluation will only be an accepted and effective part of a project if it provides information useful for the project staff. Below we outline the obstacles evaluation faces with regard to projects that design computer-supported learning environments, and discuss two promising approaches that can be used in complimentary fashion.

## BACKGROUND

According to Worthen et al. (1997), evaluation is “the identification, clarification, and application of defensible criteria to determine an evaluation object’s value (worth or merit), quality, utility, effectiveness, or significance in relation to those criteria.” In this regard evaluation can

serve different purposes. Patton (1997) distinguishes between judgment-, knowledge- and improvement-oriented evaluations. We focus on improvement-oriented evaluation approaches. We stress that evaluation can facilitate decision making and reveal information that can be used to improve not only the project itself, but also outcomes within the project’s target population. The conceptualization of evaluation as an improvement-oriented and formative activity reveals its proximity to design activities. In fact this kind of evaluative activity is an integral part of any design process, whether it is explicitly mentioned or not. Accordingly it is not the question if one should evaluate, but which evaluation methods generate the most useful information in order to improve the program. This question can only be answered by facing the characteristics and obstacles of designing computer-supported learning environments.

Keil-Slawik (1999) points out that one of the main challenges in evaluating computer-supported learning environments is that some goals and opportunities can spontaneously arise in the course of the development process and are thus not specified in advance. We believe that this is due to the fact that design, in this context, addresses ill-structured and situated problems. The design and implementation of computer-supported learning environments, which can be viewed as a response to a perceived problem, also generates new problems as it is designed. Furthermore every computer-supported learning experience takes place in a unique social context that contributes to the success of an intervention or prevents it. Therefore evaluation requires that designers pay attention to evolutionary and cyclic processes and situational factors. As Weiss notes, “Much evaluation is done by investigating outcomes without much attention to the paths by which they were produced” (1998, p. 55).

For developers designing projects at the intersection of information and communication technology (ICT) and

the learning sciences, evaluation is difficult. Evaluation efforts are often subverted by a myriad of confounding variables, leading to a “garbage in, garbage out” effect; the evaluation cannot be better than the parameters that were built in the project from the start (Nash, Plugge & Eurlings, 2001). Leaving key parameters of evaluative thinking out of computer-supported learning projects is exacerbated by the fact that many investigators lack the tools and expertise necessary to cope with the complexity they face in addressing the field of learning.

We strongly advocate leveraging the innate ability of members of the computer science and engineering communities to engage in “design thinking” and turn this ability into a set of practices that naturally becomes program evaluation, thereby making an assessment of the usefulness of ICT tools for learning a natural occurrence (and a manifest activity) in any computer-supported learning project.

### **Design-Oriented Evaluation for Computer-Supported Learning Environments**

There are two approaches that inherently relate themselves to design as well as to evaluation. Therefore they are useful tools for designers of computer-supported learning initiatives. These two perspectives, discussed below, are scenario-based design and program theory evaluation. Both approaches assume that the ultimate goal of a project should be at the center of the design and evaluation discussion, ensuring a project is not about only developing a usable tool or system, but is about developing a useful tool or system that improves outcomes for the user. Beyond this common ground, these approaches are rather complementary to each other and it is reasonable to use them in conjunction with one another.

### **Scenario-Based Approaches**

Scenario-based approaches are widely used in the fields of software engineering, requirements engineering, human computer interaction, and information systems (Rolland et al., 1996). Scenarios are a method to model the universe of discourse of an application, that is, the environment in which a system, technical or non-technical, will be deployed. A scenario is a concrete story about use of an innovative tool and/or social interactions (Carroll, 2000). Scenarios include protagonists with individual goals or objectives and reflect exemplary sequences of actions and events. They refer to observable behavior as well as mental processes, and also cover situational details assumed to affect the course of actions (Rosson & Carroll, 2002). Additionally it might explicitly refer to the

underlying culture, norms, and values (see Bødker & Christiansen, 1997). That said, scenarios usually focus on specific situations, only enlighten some important aspects, and generally do not include every eventuality (e.g., Benner, Feather, Johnson & Zorman, 1993).

Beside their use in the design process, scenarios can also be used for purposes of formative evaluation. First of all, as a means of communication, they are a valuable resource for identifying underlying assumptions regarding the program under development. Stakeholder assumptions might include those related to instructional theories, the learner, the environmental context, and its impact on learning or technical requirements. Underlying assumptions such as these are typically hidden from view of others, but easily developed and strongly held within individuals developing computer-supported learning environments. Scenarios help to reveal the thinking of designers so that others can participate in the design process and questionable assumptions can come under scrutiny. The use of scenarios also allows identification of pros and cons of a certain decision within the design process. In this vein Carroll (2000) suggests employing “claim analysis.” Claims are the positive or negative, desirable and undesirable consequences related to a certain characteristic of a scenario. Assuming that every feature of a proposed solution usually will entail both positive and negative effects helps to reflect on the current solution and might provoke alternative proposals. The analysis of claims is thereby not limited to an intuitive ad hoc evaluation, but also can bring forth an explicit hypothesis to be addressed in a subsequent survey.

### **Program Theory Evaluation**

Program theory evaluation, also known as theory-based evaluation, assumes that underlying any initiative or project is an explicit or latent “theory” (or “theories”) about how the initiative or project is meant to change outcomes. An evaluator should surface those theories and lay them out in as fine detail as possible, identifying all the assumptions and sub-assumptions built into the program (Weiss, 1995). This approach has been promoted as useful in evaluating computer-supported learning projects (Strömdahl & Langerth-Zetterman, 2000; Nash, Plugge & Eurlings, 2001) where investigators across disciplines find it appealing. For instance, for designers (in mechanical engineering or computer science), program theory evaluation reminds them of their own use of the “design rationale.” And among economists, program theory evaluation reminds them of total quality management (TQM). In the program theory approach (Weiss, 1995, 1998; Chen, 1989; Chen & Rossi, 1987), one constructs a project’s “theory of change” or “program logic”

by asking the various stakeholders, “What is the project designed to accomplish, and how are its components intended to get it there?” The process helps the project stakeholders and the evaluation team to identify and come to consensus on the project’s theory of change. By identifying and describing the activities, outcomes, and goals of the program, along with their interrelationships, the stakeholders are then in position to identify quantifiable measures to portray the veracity of the model.

Theory-based evaluation identifies and tests the relationships among a project’s inputs or activities and its outcomes via intermediate outcomes. The key advantages to using theory-based evaluation are (Connel & Kubisch, 1995; Weiss, 1995):

- It asks project practitioners to make their assumptions explicit and to reach consensus with their colleagues about what they are trying to do and why.
- It articulates a theory of change at the outset and gains agreement on it, by all stakeholders reducing problems associated with causal attribution of impact.
- It concentrates evaluation attention and resources on key aspects of the project.
- It facilitates aggregation of evaluation results into a broader context based on theoretical program knowledge.
- The theory of change model identified will facilitate the research design, measurement, data collection, and analysis elements of the evaluation.

Both scenario-based design and program theory stress the importance of the social context while planning computer-supported environments. They also represent means to facilitate the communication among the stakeholders and urge the project team to reflect their underlying assumptions in order to discuss and test them. Furthermore, both approaches are particularly suitable for multidisciplinary project teams. Scenarios and program logic maps are not static artifacts; they are a starting point for discussion and have to be changed when necessary. With these similarities there are also differences in both approaches. The major difference between them is that program theory offers a goal-oriented way to structure a project, while scenario-based design proffers an explorative approach that opens the mind to the complexity of the problem, alternatives, and the diversity of theories that try to explain social and socio-technical process. That is, scenario-based design highlights the divergent aspects of project planning, and evaluation program theory stresses the convergent aspects. Program theory evaluation helps to integrate each scenario, decision, and

predefinition into the whole process. Scenarios force users not just to use terms, but to give meaningful descriptions. They force users to state how they actually want to instantiate an abstract theory of learning and teaching. This helps to implement the project within real situations of use, which are complex and ill structured. Program theory helps to focus on core aspects of design and prevent getting ‘lost in scenarios’. Scenarios and program theory evaluation can be used in an alternating way. Thereby it is possible to use both approaches and improve the overall development process.

The program theory of an initiative can be a starting point for writing scenarios. Especially the interrelations between the goals and interrelation between ultimate goal and inputs can be described with a scenario. The scenario can help to understand how this interrelation is meant to work and how it will look in a concrete situation. Scenarios on the other hand can be used to create program theory by pointing out main elements of the intended program. They can also be used to complete already existing program theory by presenting alternative situations of use. For developers of computer-supported learning environments, scenario-based design and program theory represent complementary approaches, which when used together or separately, can add strength to the implementation and success of such projects.

## FUTURE TRENDS

It is clear that formative evaluation will become more important in the future, and it will be especially crucial to think about how to integrate evaluation into the design process. Essentially, designers will need to answer the question “Why does the program work?” and not just “Did it work?” It becomes obvious that the design of a computer-supported learning environment, like the development of any other complex socio-technical system, is a difficult process. In fact the necessity for changes in the original plan is practically preordained due to the ill-structured and situated nature of the domain. The mere act of engaging in a design process suggests that designers will engage in planned as well as evolutionary, unplanned activities. Therefore it is important that the project designers use methods that support divergent thinking and methods that support convergent processes. While scenario-based design and program-theory evaluation represent complementary views on the design and evaluation of computer-supported learning environments that can facilitate these processes, there is still room for improvement.

## CONCLUSION

Formative evaluation is an important means to ensure the quality of an initiative's outcomes. Formative evaluation directed towards improvement of an initiative can be understood as a natural part of any design activity. While this is widely recognized, there is still a lack of program evaluation frameworks that reflect the uniqueness of the design process, the most crucial of which is the inherent ambiguity of design. In spite of great inspiration portrayed by project teams, usually manifested by visions of a certain and sure outcome, no project can be pre-planned completely, and midcourse corrections are a certainty. Scenario-based design and program theory evaluation provide a theoretical foothold for projects in need of collecting and analyzing data for program improvement and judging program success.

In sum, scenario-based design and program theory hold many similarities. The major difference between them is that program theory offers a goal-oriented way to structure a project, while scenario-based design provides an explorative approach that opens the mind to the complexity of the problem, alternatives, and the diversity of theories that try to explain social and socio-technical process.

Scenario-based design highlights the divergent aspects of project planning, and evaluation program theory stresses the convergent aspects. For developers of computer-supported learning experiences, scenario-based design and program theory represent complementary approaches, which when used together or separately can add strength to the implementation and success of ICT learning projects.

## REFERENCES

Benner, K.M., Feather, M.S., Johnson, W.L. & Zorman, L.A. (1993). Utilizing scenarios in the software development process. In N. Prakash, C. Rolland & B. Pernici (Eds.), *Information system development process* (pp. 117-134). Elsevier Science Publishers.

Bødker, S. & Christiansen, E. (1997). Scenarios as springboards in design. In G. Bowker, L. Gaser, S.L. Star & W. Turner (Eds.), *Social science research, technical systems and cooperative work* (pp. 217-234). Lawrence Erlbaum.

Carroll, J.M. (2000). *Making use: Scenario-based design of human-computer interactions*. Cambridge: MIT Press.

Chen, H.T. (1989). Issues in the theory-driven perspective. *Evaluation and Program Planning*, 12, 299-306.

Chen, H.T. & Rossi, P. (1987). The theory-driven approach to validity. *Evaluation Review*, 7, 95-103.

Connell, J.P. & Kubisch, A. (1995). Applying a theory of change approach to the evaluation of comprehensive community initiatives: Progress, prospects, and problems. In K. Fulbright-Anderson et al. (Eds.), *New approaches to evaluating community initiatives. Volume 2: Theory, measurement, and analysis*. Washington, DC: Aspen Institute.

Keil-Slawik, R. (1999). Evaluation als evolutionäre systemgestaltung. aufbau und weiterentwicklung der paderborner DISCO (Digitale Infrastruktur für computerunterstütztes kooperatives Lernen). In M. Kindt (Ed.), *Projektelevaluation in der lehre—multimedia an hochschulen zeigt profil(e)* (pp. 11-36). Münster, Germany: Waxmann.

Nash, J.B., Plugge, L. & Eurlings, A. (2001). Defining and evaluating CSCL evaluations. In A. Eurlings & P. Dillenbourg (Eds.), *Proceedings of the European Conference on Computer-Supported Collaborative Learning* (pp. 120-128). Maastricht, The Netherlands: Universiteit Maastricht.

Patton, M.Q. (1997). *Utilization-focused evaluation* (3rd Edition). Thousand Oaks, CA: Sage Publications.

Rolland, C., Achour, C.B., Cauvet, C., Ralyté, J., Sutcliffe, A., Maiden, N.A.M., Jarke, M., Haumer, P., Pohl, K., Dubois, E. & Heymans, P. (1996). *A proposal for a scenario classification framework*. CREWS Report 96-01.

Rosson, M.B. & Carroll, J.M. (2002). *Usability engineering: Scenario-based development of human-computer interaction*. San Francisco: Morgan Kaufmann.

Strömdahl, H. & Langerth-Zetterman, M. (2000). *On theory-anchored evaluation research of educational settings, especially those supported by information and communication technologies (ICTs)*. Uppsala, Sweden: Swedish Learning Lab.

Weiss, C. (1995). Nothing as practical as good theory: Exploring theory-based evaluation for comprehensive community initiatives for children and families. In J. Connell et al. (Eds.), *New approaches to evaluating community initiatives: Concepts, methods, and contexts*. Washington, DC: Aspen Institute.

Weiss, C. (1998). *Evaluation research: Methods for studying programs and policies*. Englewood Cliffs, NJ: Prentice-Hall.

Worthen, B.R., Sanders, J.R. & Fitzpatrick, J.L. (1997). *Program evaluation—alternative approaches and practical guidelines* (2nd Edition). New York: Addison Wesley Longman.

## KEY TERMS

**Computer-Supported Learning:** Learning processes that take place in an environment that includes computer-based tools and/or electronically stored resources. CSCL is one part of this type of learning.

**Evaluation:** The systematic determination of the merit or worth of an object.

**Formative Evaluation:** The elicitation of information that can be used to improve a program while it is in the development stage.

**Program:** A social endeavor to reach some predefined goals and objectives. A program draws on personal, social, and material resources to alter or preserve the context in which it takes place.

**Program Theory:** A set of assumptions underlying a program that explains why the planned activities should lead to the predefined goals and objectives. The program theory includes activities directly implemented by the program, as well as the activities that are generated as a response to the program by the context in which it takes place.

**Scenarios:** A narrative description of a sequence of (inter-)actions performed by one or more persons in a particular context. Scenarios include information about goals, plans, interpretations, values, and contextual conditions and events.

**Summative Evaluation:** The elicitation of information that can be used to determine if a program should be continued or terminated.