Analysis of Design from a Community of Practice Dialogue: Negotiating the Meaning of Auditing Information System Development

A. Faye Borthick
Georgia State University

ABSTRACT: A persistent question associated with theories of situated learning is how to put them into action. The concept of situated learning, in which the learner and the task are placed in the context of the overall social practice, stems from Lave (1988) and Lave et al.’s (1988) recognition that a traditional view of learning as the end result of a process of transmitting knowledge is inadequate. Instead, they propose looking at learning from the learner’s perspective and viewing learning as a continuous process in which what is learned depends on what has been learned before and the context in which the learner is immersed—a process known as “legitimate peripheral participation” (Lave and Wenger 1991). This paper explains how a situated-learning experience was operationalized in a context in which the learners’ task was to develop a plan for auditing the development of a dysfunctional information system. It identifies and analyzes the situated learning from the viewpoint of dimensions of educational design associated with communities of practice (Wenger 1998). Here, the community of practice consisted of master’s-level students assuming the auditor’s role. In a community of practice, members of the group learn from each other by working together as they develop a common sense of purpose, including a common way of thinking about how work gets done and what is necessary to accomplish a task. The analysis is based on the text of synchronous class discussion from a web class session. Also discussed are implications of communities of practice implemented through synchronous discussion for educational design and professional practice.

Keywords: auditing information systems development; collective intelligence; community of practice; course design; information systems assurance; legitimate peripheral participation; situated learning.

I. DIMENSIONS OF EDUCATIONAL DESIGN IN A COMMUNITY OF PRACTICE

Arguments for learning organizations and characterizations of them have accumulated anecdotally and systematically over the last decade. The concept of learning organization embodies the idea that to be competitive requires continuous innovation that can be applied pervasively and quickly throughout the organization. Achieving continuous innovation, however, requires the engagement

---


The author is indebted to Susan Galbreath, Bill Hillison, discussants and participants at the 1999 University of Waterloo Research Symposium on Information Systems Assurance, and anonymous reviewers for helpful comments.
of everyone (not just of managers) to realize the organization’s previously untapped potential. This implies learning on the part of everyone, embodied in the term “learning organization,” whose objective is to develop a collective intelligence (Lévy 1997). In learning organizations, the leaders’ task is not making decisions but designing learning processes.

Because of the growing emphasis on learning in organizations, it is important to understand how people learn in organizations. The purpose of this article is to analyze a discussion in a group of learners to see whether the intended learning occurred and how it emerged. Insights gained from such analyses could be used to improve the design of learning events.

The analysis is conducted from the perspective of the dimensions of educational design proposed for promoting learning groups as communities of practice (Brown and Duguid 1991; Wenger 1998). In a community of practice, members of the group learn from each other by working together as they develop a common sense of purpose, including a common way of thinking about how work gets done and what is necessary to accomplish a task. The theory of learning as a community of practice is based on the assumption that “engagement in social practice is the fundamental process by which we learn and so become who we are” (Wenger, iii).

The unit of analysis is the group. The motivation for a community of practice as a theory of learning is the emerging view, consistent with the management literature, that organizations benefit most when their members participate “inventively in practices that can never be fully captured by institutionalized processes” (Wenger 1998, 10).

If the contributions to organizational goals cannot be specified in advance, it is unlikely that educational courses designed to transfer knowledge as discrete pieces packaged in tight units delivered to recipients in classrooms as succinctly as possible are likely to prepare people for participating in practices that have not yet evolved. If ensuring organizational performance were simply a matter of rolling out successive refinements of organizational processes, then learning events characterized by teachers standing in front of classes are efficient and direct. If, however, ensuring organizational performance comprises making sure that “participants have access to the resources necessary to learn what they need to learn in order to take actions and make decisions that fully engage their own knowledgeable” (Wenger 1998, 10), then learning events ought to be about encouraging participants’ participation in the communities of practice they are seeking to enter. This kind of learning requires that learners “situate the decomposed task in the context of the overall social practice” (Brown and Duguid 1993, 12). For the educational designer, the challenge is to redesign learning events “so that newcomers can legitimately and peripherally participate in authentic social practice in rich and productive ways—to, in short, make it possible for learners to ‘steal’ the knowledge they need” (Brown and Duguid 1993, 11). Such participation is known as “legitimate peripheral participation” (Lave and Wenger 1991).

Wenger (1998) proposed four design dimensions for a community of learning: participation and reification, designed teaching and emergent learning, local and global practice, and identification and negotiability. Reification is “the process of giving form to our experience by producing objects that congeal this experience into ‘thingness’” (Wenger 1998, 58), e.g., words that explain understanding of a concept, a written procedure that delineates how something should be done, or a flowchart documenting a process.2 Wenger’s (1998) dimensions are the basis for the analysis of the dialogue in this research. This approach was selected because of its focus on a group of learners situated in context interacting with each other. This analysis does not speak to whether other kinds of analysis based on other theoretical underpinnings would yield useful information about learning or even whether this approach is the “best” approach of all possible candidate approaches.

There is no formal statement of hypotheses because the purpose of the analysis is to determine whether Wenger’s (1998) dimensions can be used to interpret participants’ interaction. Because the learners are situated in a university course setting rather than actual practice, some of the accoutrements of a community of practice are not as pervasive as they would be in an actual community of practice.

---

2 Reification covers “a wide range of processes that include making, designing, representing, naming, encoding, and describing, as well as perceiving, interpreting, using, reusing, decoding, and recasting….2[I]n all these cases, aspects of human experience and practice are congealed into fixed forms and given the status of object” (Wenger 1998, 59).
example, practitioners were not present in all the meetings. The setting, however, does not hinder the analysis because the focus is on the participants’ learning how to learn in a group organized as a community of practice rather than on the authenticity of the group as a community of practice. It also does not matter that the actual object of the learning, a concept associated with system development audits, would likely be familiar to auditors with computer programming backgrounds. The point is that no matter how much a community of practice (or a group learning to behave like a community of practice) already knows, it will always be confronted with the new, with which it must come to terms. The newest of the new will not be available in the form of a reification, which means that community members will have to negotiate among themselves if they are to make sense of it. It is the process of “coming to terms with” or “negotiating the meaning of” that is the focus of this article.

Participation and Reification

In a community of practice, there are two principal ways to acquire meaning: through participation or through reification (Wenger 1998). Through participation, one negotiates new meanings by participating in learning with the group or community, and the more meaningful the participation the more one learns. Reification of knowledge is the result of codifying it into an institutional form intended to ease the learner’s grasp of it, e.g., a textbook. Unfortunately, although absorbing reified learning may be efficient, its acquisition leads to “a brittle kind of understanding with very narrow applicability” (Wenger 1998, 265). This makes the design choice a matter of deciding how much to reify learning and how much to rely on participation in a community of practice.

Designed Teaching and Emergent Learning

Teaching and learning are not the same. What is taught may not correspond with what is learned, and what is learned may not be what is taught. This does not, however, mean that learning events should not be designed. On the contrary, it is important to think through “the interaction of the planned and the emergent” (Wenger 1998, 267) to maximize learning, to let learning and teaching inform each other, and to maximize the processes of participants’ negotiating meaning.

Local and Global Practice

To avoid school learning being “just learning school” (Wenger 1998, 267), one’s learning has to apply beyond the classroom, i.e., the learning needs to help one move within a practice and from one practice to another. The issue for educational design is to ensure depth of coverage with broad scope and to create links to other practices so the learner can move easily among them.

Identification and Negotiability

In a community of practice, learning is tantamount to assuming new identities and negotiating new meanings and practices. Educational design issues include deciding what to identify with and how to foster it, facilitating the negotiation of new identities, and defining success and failure in the new identities. It is a “process of colonizing learning, of claiming a territory, of deciding what matters, and of defining success and failure” (Wenger 1998, 269).

II. CONTEXT FOR THE COMMUNITY OF PRACTICE

Preparation of a Plan for Auditing System Development

The learning situation that is analyzed here is a segment of a class discussion from a master’s-level course in information-systems assurance. The course learning objective pertinent to this research is to learn to develop assurance plans for highly automated information systems. Developing plans for auditing

---

3 The syllabus for the course is available at: http://www.gsu.edu/~accafb/ac863.htm. The course has prerequisites of undergraduate courses in auditing and accounting information systems or a master’s-level course in application development. Most of the students had an accounting background rather than an information systems background. The rationale for, design of, and results from the course are explained in Borthick and Jones (2000).
system development is germane to this objective in that giving assurance for highly automated information systems may entail giving assurance for system processes, some of which are examined in audits of system development.

The specific assignment was to prepare an audit plan for the development of the magazine-fulfillment system presented in Rorer (1997). In this system, magazine fulfillment encompasses the process of determining how many magazines to print, updating master subscription files, and creating mailing labels in mail code order. The directions to students were for them to:

Prepare an audit plan that, if executed timely, would have detected the system development and execution flaws soon enough for the publisher to have avoided them. To minimize the likelihood that you fail to identify all the unsuccessful system development practices, start by making a list of them. Then develop an audit plan that addresses each one.

An illustrative audit plan that addresses the major system development failures appears in Table 1.

<table>
<thead>
<tr>
<th>Unsuccessful System Development Practices</th>
<th>Audit Step to Detect the Unsuccessful Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Inattention to documentation, which is voluminous but unusable, out of date, and lacks inline code comments</td>
<td>Examine documentation for completeness, currency, and usefulness of code comments</td>
</tr>
<tr>
<td>2. Lack of change control over program code</td>
<td>Examine change control logs and authorizations</td>
</tr>
<tr>
<td>3. Program too big to understand or modify</td>
<td>Verify that program code exists in manageable units with obvious entry and exit points</td>
</tr>
<tr>
<td>4. Return codes from I/O operations not checked</td>
<td>Examine program code, e.g., with a code analyzer, to verify return code checking. This step would be invoked only after obtaining evidence of the need for it.</td>
</tr>
<tr>
<td>5. Run-time anomalies (i.e., gibberish for mail bag routing codes, excess copies printed, proliferating messages from fulfillment program indicating bad DASD tract) not reported or ignored if reported</td>
<td>Examine change control records for evidence of user reports initiating bug fixes; interview users about system stability</td>
</tr>
<tr>
<td>6. Inadequate program testing due to inadequate test facilities</td>
<td>Inspect the development and testing environment (computing); interview development staff</td>
</tr>
<tr>
<td>7. Fall-through program logic that lets unsorted addresses go to labels without being sorted into mailing code order</td>
<td>Examine program code, e.g., with a code analyzer, to verify that all code values are tested for explicitly. This step would be invoked only after obtaining evidence of the need for it.</td>
</tr>
<tr>
<td>8. Subscriber update program permitting invalid codes to be written to master file</td>
<td>Verify that the application rejects invalid codes. This step would be invoked only after obtaining evidence of the need for it.</td>
</tr>
<tr>
<td>9. Console messages indicating sorting on bad DASD tracks ignored</td>
<td>Verify that computer operations logs and follows up on unusual conditions</td>
</tr>
<tr>
<td>10. Volume testing not performed</td>
<td>Examine test results for volume testing</td>
</tr>
<tr>
<td>11. Uncontrolled program code installation</td>
<td>Verify that production libraries are subject to change control</td>
</tr>
<tr>
<td>12. Management ineptitude in overseeing system development</td>
<td>Interview management and development staff</td>
</tr>
</tbody>
</table>
Although some of the failures in Rorer (1997) concern the operation rather than the development of the system, all of them are included in the audit plan because of their implications for system development. Collectively, the development failures led to the system not being maintainable and its failing to satisfy the requirements established for it, which constituted a catastrophic system failure.

Assignments and Resource Materials

Students obtained all course materials from the course web site. Each day’s assignment incorporated resource materials (the course had no textbook) with study questions. These materials provided background and explanations of concepts, techniques, and applications in organizational situations that would be helpful in preparing the day’s assignment. After preparing assignments, students converted them to *.html files and loaded them on the presentation server, which was accessible to all class participants.

Resource materials and study questions for this assignment, the first one in the course concerning system development, are listed in Table 2. The resources were either prescriptive in explaining how to audit systems development, descriptive of system-development failures (where, in hindsight, remedies become apparent), or descriptive of system-development practices. On the premise that practices “can never be fully captured by institutionalized processes,” representative, but not exhaustive, prescriptive material about how to audit systems development was provided to avoid discouraging “the very inventiveness that makes practices effective” (Wenger 1998, 10). Instead, the larger share of the materials created a diversity of contexts in system development through sagas of failed development projects and illustrations of current development practices.

Data Collection: Synchronous Discussion

The course was posed to students as discovery learning through web-enabled collaboration. In discovery learning, learners actively seek knowledge or principles for solving problems that they have identified (Anzai and Simon 1979; Kulkarni and Simon 1988; Dunbar 1993; Norman and Spohrer 1996; Brandt 1997; Sabelli 1998; Schank 1998). In discovery learning, learners recognize pertinent knowledge not because the teacher identifies it beforehand but because it enables learners to make progress solving the problem. Learning through web-enabled collaboration means that students and teacher share a common workspace in which they work together to solve problems (Okada and Simon 1997).

The learning space in the course was implemented with online discussion coupled with a web-based presentation frame for student-completed assignments. Using synchronous space for class sessions fosters discovery learning by (1) promoting students’ consideration of more information and more alternatives for solving problems and (2) creating a community of practice (Wenger 1998) in the class that promotes student participation rather than passive acquisition of abstract knowledge. When they had questions they could not answer, students sought insight from other students, the facilitator, and guest professionals.

The data for this research were taken from the log of the class session in which the subject was the audit of Rorer’s (1997) magazine-fulfillment system. The class session was neither immediately before nor after a session in which an examination was administered (online). It was the tenth of 20 class sessions in the course. This means that students were as accustomed to the class’s synchronous discussion as they were likely to be and that the feeling of being overwhelmed at the end of the term that some students experience had not yet occurred. Students’ directions for participating in class discussions were:

- Participate in class discussions by (1) entering your comments and questions during class discussions and (2) pushing your *.html files for assignments at the appropriate times. To push a web page during discussion, type its URL (starting with “www”) in the comment line. Seek out resources (Web and non-Web) that pertain to class topics and share them as appropriate during class discussions.
- Participate in class discussions by commenting on others’ work for the purpose of helping to improve it. This kind of participation may be uncomfortable at first because non-online...
TABLE 2
Resource Materials for Auditing System Development

Prescriptive in Explaining How to Audit System Development
1. What project management practices are associated with failed projects?
2. What should be the objectives for audits of project management?
3. How would an auditor achieve each objective?

1. Why is change control so important?
2. What limits the kind of change control that can be implemented?

Descriptive of System Development Failures
1. Why is it so hard to obtain an honest assessment of a project’s status?
2. What is the danger in keeping bad news from customers?
3. What limits the kind of change control that can be implemented?
4. What can be done when developers seem to be too busy to follow their own procedures, e.g., for change control or testing?
5. Why might developers be so proud of circumventing established procedures?
6. How should development schedules be set? What happens when they are unrealistic?
7. What finally focuses attention on projects in which development is not on schedule?
8. How is it that developers can delude themselves into thinking that subverting change control is productive?

1. In spite of the best-intentioned designs, some computer-made decisions are apt to be misguided or just wrong due to inadequate data being considered. When such instances are called to system users’ attention, how should they behave?

Descriptive of System Development Practices
1. How should development audits of applications developed iteratively, e.g., synch-and-stabilize, differ from those developed with sequential (waterfall) methodology?
2. How does the existence of frequent integrations change what an auditor would do in an application-development audit?
3. Which is a riskier approach to systems development-sequential or iterative? Why? Explain the risks that matter.
4. What aspects of parallel development with frequent stabilization reduce the risk of projects falling behind?
5. Compared to a sync-and-stabilize approach, what are the biggest risks of sequential development?
6. Why doesn’t following a sync-and-stabilize process guarantee on-time and bug-free products?
7. What aspects of sequential development increase the risk of developer turnover? What aspects of any development project tend to promote developer turnover?

1. What is the likely outcome of ignoring configuration management?
2. Why is testing able to show the presence of errors but not their absence?
3. In a system development audit, what evidence would give an auditor assurance that each of unit, integration, usability, function, system, acceptance, and regression testing were completed adequately?
classes typically do not feature much of this kind of participation. Imagine yourself in a work setting in which your organization’s survival (and hence your continued employment there) depends on your work group improving its products or services substantially. No improvement will be possible if everyone is bashful and refrains from commenting on the work. A human tendency is to refrain from pointing out flaws (which is important because a flaw is unlikely to be remedied if it is never identified) on the grounds that doing so might hurt someone’s feelings. But we need not be personal in our comments about real or perceived flaws. For example, we can pose our comments as follows: “I believe this audit procedure would give better evidence for the audit objective if…because….” Focusing on the work or the ideas in the work will let us avoid hurting someone’s feelings unintentionally. Therefore, you are just as responsible for discussing others’ ideas in class as you are for presenting your own. This is a tall order for people not accustomed to behaving this way, but learning to do so is likely to enable you to be much more successful in your career.

III. NEGOTIATING THE MEANING OF FALL-THROUGH LOGIC

One of the system-development lapses in the magazine-fulfillment system was the use of fall-through logic in the program code. Fall-through logic occurs when a program fails to check all data code values explicitly, with the result that the program branches as if it had received one value when the data field had some other value, i.e., the program makes an erroneous assumption about a data value, which leads to unintended results. The undesirability of fall-through logic and the coding to avoid it are usually understood by programmers with a few years’ training and experience. Its incidence usually arises from (1) the assumption that all possible values are known and that they will not change over the life of the program and (2) programmers’ desire to finish program development or maintenance quickly (fall-through logic contains fewer coding statements than explicitly coded logic.) The paragraph in Rorer (1997, 13) explaining that fall-through logic was associated with the unintended results of mailing labels being printed in the wrong sequence is:

Occasionally, the system would output long runs of mailing labels in the wrong sequence. (They were arranged alphabetically by subscriber name, instead of by delivery address zip code). This condition was found to be the result of fall-through logic. The subscriber master file contained records that included invalid type codes. If the subscriber list program did not branch on a matching type code, it fell through to logic that output these records directly to the mailing labels. Some may feel that allowing this condition to occur was the magazine-publishing organization’s fault. However, not having the processing routine issue a warning when an unexpected condition was detected reflected bad program design.

Class Dialogue: Fall-Through Logic

This passage about the fulfillment system (Rorer 1997) prompted participants to name it as one of the development failures meriting audit attention. The dialogue below shows how the group came to understand (1) the meaning of fall-through logic, (2) how fall-through logic caused the erroneous printing sequence, (3) how auditors might detect fall-through logic, and (4) how auditors might acquire the programming competence for auditing systems development. The dialogue is taken directly from the recorded log of the session (misspellings retained, beginning about 1 hour 20 minutes into the 2 hour 15 minute session. The class had 14 students (corresponding to participant numbers 207–220), eight of whom visibly participated in the dialogue on fall-through logic. Five other students were logged on during this sequence, although one student was experiencing connectivity lapses, and one student was absent.

The dialogue contains several occasions of multiple conversations occurring at the same time, which is typical of synchronous discussion due to the time delay engendered by typing rather than speaking responses. For example, line 36 starts a new thread, but the previous thread persists through line 37.
Similarly, line 56 is a response to line 54. Although some participants in synchronous discussion are initially distracted by multi-threaded conversation, they soon appear to become accustomed to it, even if they did not grow up in the “twitch generation” (Lancaster 1998), for whom attending to multiple tasks seems normal.

The analysis below is based on the following dialogue:

<table>
<thead>
<tr>
<th>Line</th>
<th>Source</th>
<th>Dialogue on Fall-Through Logic (17 minutes, 28 seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Facilitator</td>
<td>sounds like it’s time to work on the next problem. who’s got it?</td>
</tr>
<tr>
<td>1</td>
<td>Participant 215</td>
<td>the printing problem?</td>
</tr>
<tr>
<td>2</td>
<td>Participant 214</td>
<td>mailing codes</td>
</tr>
<tr>
<td>3</td>
<td>Facilitator</td>
<td>what caused the mailing codes to end up gibberish?</td>
</tr>
<tr>
<td>4</td>
<td>Participant 215</td>
<td>the fall-through logic</td>
</tr>
<tr>
<td>5</td>
<td>Facilitator</td>
<td>how is fall-through logic the culprit</td>
</tr>
<tr>
<td>6</td>
<td>Participant 208</td>
<td>two-points in the code?</td>
</tr>
<tr>
<td>7</td>
<td>Participant 210</td>
<td>the wrong record was being moved to the output file area</td>
</tr>
<tr>
<td>8</td>
<td>Participant 216</td>
<td>there were invalid type codes</td>
</tr>
<tr>
<td>9</td>
<td>Participant 215</td>
<td>invalid type codes would fall through to logic and output directly to the mailing labels</td>
</tr>
<tr>
<td>10</td>
<td>Facilitator</td>
<td>what does “fall through the logic” mean?</td>
</tr>
<tr>
<td>11</td>
<td>Facilitator</td>
<td>a code “falls through the logic” when ...?</td>
</tr>
<tr>
<td>12</td>
<td>Participant 213</td>
<td>because of the bad result of return code?</td>
</tr>
<tr>
<td>13</td>
<td>Facilitator</td>
<td>no, not return codes here</td>
</tr>
<tr>
<td>14</td>
<td>Participant 217</td>
<td>sort operator is bypassed</td>
</tr>
<tr>
<td>15</td>
<td>Participant 215</td>
<td>I guess the invalid codes would pass without any warnings</td>
</tr>
<tr>
<td>16</td>
<td>Participant 210</td>
<td>it was not following the correct path?</td>
</tr>
<tr>
<td>17</td>
<td>Facilitator</td>
<td>why did the invalid codes get passed through as good ones?</td>
</tr>
<tr>
<td>18</td>
<td>Facilitator</td>
<td>what about the way the program was written let that happen?</td>
</tr>
<tr>
<td>19</td>
<td>Facilitator</td>
<td>have you ever written a program for which there were 3 possible codes, say, a, b and c</td>
</tr>
<tr>
<td>20</td>
<td>Participant 218</td>
<td>the value neither within valid range nor in invalid range</td>
</tr>
<tr>
<td>21</td>
<td>Participant 214</td>
<td>have never written a program</td>
</tr>
<tr>
<td>22</td>
<td>Facilitator</td>
<td>participant 218’s getting there, but it’s not a range check, but an explicit check for each of the possible values.</td>
</tr>
<tr>
<td>23</td>
<td>Participant 214</td>
<td>I am confused</td>
</tr>
<tr>
<td>24</td>
<td>Facilitator</td>
<td>what a program is supposed to do is check for each possible value, and any value that doesn’t correspond to the ones in the permissable set is deemed garbage and rejected.</td>
</tr>
<tr>
<td>25</td>
<td>Participant 217</td>
<td>so there is no wrong logic choice, anything goes or falls through.</td>
</tr>
<tr>
<td>26</td>
<td>Participant 218</td>
<td>check a and b, if not true, assume it is c</td>
</tr>
<tr>
<td>27</td>
<td>Facilitator</td>
<td>in fall through logic, one or more legitimate values are not explicitly tested for, with the result that some illegal value can flow through and be treated as the untested for legal value.</td>
</tr>
<tr>
<td>28</td>
<td>Participant 214</td>
<td>what?</td>
</tr>
<tr>
<td>29</td>
<td>Participant 215</td>
<td>so why would use such a logic?</td>
</tr>
<tr>
<td>30</td>
<td>Facilitator</td>
<td>so, instead of the program rejecting values that are clearly erroneous, it treats them as if they are good ones.</td>
</tr>
<tr>
<td>31</td>
<td>Facilitator</td>
<td>it’s easy to program because you don’t have to put in the last test.</td>
</tr>
<tr>
<td>32</td>
<td>Participant 214</td>
<td>so how does this print garbage</td>
</tr>
<tr>
<td>33</td>
<td>Facilitator</td>
<td>in this case, the garbage got printed because any code that didn’t match the ones explicitly tested for got sent immediately to the print queue—this is really poor programming!</td>
</tr>
</tbody>
</table>
Borthick—Analysis of Design from a Community of Practice Dialogue

Dialogue Analysis: Meanings Negotiated

Even from the beginning of the dialogue, several participants understood that fall-through logic was associated with mailing labels being printed out of sequence (lines 4, 7–9), but they had varying degrees of understanding of fall-through logic and how it led to the erroneous sequence. The facilitator, aware of the wide variance in participants’ programming backgrounds (ranging from accomplished to never attempted), used the occasion to encourage participants to gain a shared understanding of fall-through logic.
logic and the implications of its incidence for auditing the development of information systems. Until line 12, participants made comments consistent with fall-through logic, but this line introduces an irrelevant concept (return codes from I/O operations, which were a factor in another development lapse) that the facilitator closed off from further discussion (line 13). Line 14 returns to a comment relevant to the effect of fall-through logic in the system. At line 19, the facilitator posed an example of fall-through logic, which prompted the idea of a domain check (line 20). To keep the discussion focused on fall-through logic rather than a domain or range check, the facilitator noted the explicitness of value checking (line 22). Participant 214 (an acknowledged nonprogrammer, line 21) admitted being confused (line 23). Participant 217 was on the right track (line 25), and participant 218, who had programming experience, gave a succinct definition of the fall-through logic phenomenon (line 26). Participant 214, however, is still confused (line 28). Now the discussion turns to why code would be written in such an obviously flawed way (lines 29–31), and expediency was accepted as a plausible answer (lines 35, 37). Participant 214 is catching on to the extent of wanting to know how fall-through logic results in printing errors (line 32) and appears to understand (line 34) after another explanation (line 33).

Next, the discussion turns to how auditors would be expected to be aware of such programming arcana (lines 36, 38), and a participant supplied an answer—have some programming background (lines 39, 45). However, regardless of one’s programming background, auditors have a job to do. There being nothing in the resource materials that covered what auditors did in such situations, the facilitator supplied the information—use a code checker/mapper to identify such instances. Using a code checker, once one understood how to use it, would be easier than running specially prepared test data (lines 41, 44).

The facilitator confirmed participant 208’s suggestion (line 39) that some programming background would be helpful to an auditor (line 54) and attempted to move the discussion to another system development lapse (line 55). This attempt was unsuccessful because participants wanted to understand what acquiring a programming background might entail (lines 57, 59–64). The facilitator tried again to direct attention to another development lapse (line 65) and was eventually successful (although the sequence is not shown in the dialogue here.) The concluding exchange in the sequence was participant 214’s comment about being in the last quarter of the degree program (line 66) and receiving congratulations on that achievement (line 68).

**Dialogue Analysis: Community of Practice**

The preceding section showed how the dialogue achieved the facilitator’s specific objectives for the session (participants developing an understanding of (1) the meaning of fall-through logic, (2) how fall-through logic caused the erroneous printing sequence, and (3) how auditors could detect fall-through logic) and a general course objective (participants gaining an understanding of the need for systems auditors to have competence in information systems). This section analyzes the dialogue in terms of an educational design for promoting a community of practice (Wenger 1998). This evaluation is organized by the following dimensions: participation and reification, designed teaching and emergent learning, local and global practice, and identification and negotiability.

**Participation and Reification**

In the dialogue, participants gained their understanding of fall-through logic through participation rather than through reification, i.e., they negotiated its meaning rather than absorbed it through written materials (reifications) that explained it. The concept of fall-through logic was likely the most technically difficult of the four (from the list of 12 in Table 1) unsuccessful development practices whose understanding required some familiarity with programming. It merited attention because without understanding fall-through logic, an auditor would neither grasp its significance nor know how to detect its incidence, which would increase the difficulty of the auditor assessing system development.

The educational design dilemma is the choice of how to promote learning associated with the most technically difficult concepts: by having participants negotiate their meaning, by having participants study reifications of their meaning, or through a combination of the two approaches. Having participants negotiate meaning has the advantage of engaging participants in the group’s learning and increasing the likelihood of participants gaining ownership because they participated in creating the shared understanding.
A disadvantage is that the process may be more time consuming than individual study of reifications of the concept.

Having participants study reifications has the advantage of permitting participants that complete assigned tasks before class to maximize their acquisition of context-specific knowledge. For example, the concept of fall-through logic could be reified by creating a web page that explains it, with an example in pseudo-code that would apply to many programming languages, and linking from the words *fall-through logic* in the web page rendition of Rorer (1997) so that participants could study it when they first encounter the term. Such a link would likely be helpful to nonprogramming participants but would not interfere with programming-savvy participants’ creation of their audit plans. Such links could also be created for the other technically difficult unsuccessful practices (in Table 1: 4, return codes from I/O operations; 8, invalid codes written to master files; 9, console messages for DASD I/Os). This approach, however, has the disadvantage of decreasing the likelihood that participants will, as a group, learn to negotiate meanings of technical concepts when no reifications of them are readily available, which would promote even farther participants’ identity of themselves as solo learners. Another disadvantage of a strictly reified approach is that even though the course design might provide reification for the four most technically difficult practices, the designer cannot anticipate every learner’s need for clarification or explanation. When these learners join workplace communities of practice, there is unlikely to be a course designer attempting to anticipate their learning needs.

A hybrid approach of negotiating meanings for some concepts and studying reifications for others has all the advantages and disadvantages of both approaches. If, however, the designer ensured that, although some concepts were accessible through reifications, some crucial ones were only attainable through negotiations of meaning, the group might make the most progress in creating a community of practice. During discussion, the facilitator could verify that the reifications were effective. The results of the group’s negotiation of the meaning of the other concepts might be reified in the form of web pages of the group’s creation that are then linked from the concepts (in Rorer 1997). Now, the group would be creating its own reifications rather than merely absorbing those of others, thereby lessening the likelihood of the learning being merely extractive rather than integrative.

**Designed Teaching and Emergent Learning**

The teaching in the dialogue was designed by the facilitator to encourage the group to identify as many of the unsuccessful development practices as they could and, for each one, develop a shared meaning of it including the implications for the auditor of system development. Before the class session, the facilitator did not know which of the practices in Rorer (1997) would elicit participants’ interest. That participants would have the most difficulty with the most technically difficult concepts was predictable. That negotiating the meaning of fall-through logic would prompt participants to pursue knowledge of how to acquire programming competency was surprising.

Although the facilitator’s plan ensured there was an agenda to organize the discussion, the plan did not control how the learning would emerge—that was under the control of the participants. At intervals, when discussion about a development practice seemed complete, the facilitator invited participants to name another unsuccessful development practice. When participants had not reached closure on a practice, they declined to respond to the facilitator’s attempts to initiate discussion of another practice. Thus, participants controlled the order in which they discussed the practices and, in many respects, the scope and richness of the discussion. From previous class sessions, participants had most likely learned that any subject could be discussed, but that if participants waited well into the session to ask about it that there might be insufficient time for it.

**Local and Global Practices**

The learning represented by the dialogue is local to audits of system development with respect to fall-through logic. Furthermore, the discussion took up its incidence in only one context (Rorer 1997). The idea of using a code analyzer/mapper to identify lapses in programming statements is generalizable to another practice in Rorer (1997), i.e., in Table 1: 4, identifying return codes and lack of appropriate testing of their values. The technique of using a code analyzer/mapper also arose in other class sessions.
There are two global ideas embodied in the dialogue. First, some lapses in program code lead to situations likely to be associated with unintended system behavior, the cause of which may not be immediately apparent. Second, once an auditor has some reason to suspect these kinds of lapses in a program, there are aids, such as code analyzers/mappers, that will help identify such instances. The defect in this characterization of the global application is that the behavior ensuing from such programming lapses is usually idiosyncratic and thus not easily traced back to its cause.

**Identification and Negotiability**

At the beginning of the class session, most of the participants, by virtue of previous training and experience (including in the course) identified with information system (IS) auditors as auditors of application systems. Not having had training or experience in auditing systems development, they did not appear to identify with IS auditors as auditors of IS development at the beginning of the class session. This starting point sets up the session as an opportunity for participants to negotiate a new identity for themselves. A surprise in the dialogue is that, as participants became sufficiently intrigued with the skills that an IS auditor would need to audit systems development, they identified enough with IS developers to want to know what courses would enable them to achieve the requisite skill. This can be taken as evidence of their negotiating a new identity for themselves—IS auditor that is competent to audit systems development.

**IV. IMPLICATIONS**

This article has analyzed a dialogue from the synchronous discussion of a community of practice composed of learners engaged in learning to audit information systems development. The dialogue achieved the facilitator’s specific objectives for the session (participants developing an understanding of (1) the meaning of fall-through logic, (2) how fall-through logic caused the erroneous printing sequence, and (3) how auditors could detect fall-through logic) and a general course objective (participants gaining an understanding of the need for systems auditors to have competence in information systems). As discussed below, the dialogue has implications for educational design and professional practice.

**In Formal Courses of Study**

The dimensions of educational design that were considered in the analysis were those associated with learning in a community of practice: participation and reification, designed teaching and emergent learning, local and global practice, and identification and negotiability. Of these, the dialogue most informs design with respect to participation and reification.

On participation and reification, the conclusion is that a hybrid approach might work best if it were designed such that even though reifications of many concepts might be available, there would always be instances for which meaning was only obtainable through negotiation. This approach would ensure that participants learned to negotiate meanings for themselves so that they could do so when they moved to the edge of practice, where reifications do not yet exist, and ensure that learners avoid reinforcing their identities as exclusively solo learners. This approach might have even more promise if it incorporated into its repertoire the community’s creation of reifications. This process would ensure that participants really did understand the meaning that they had negotiated and make it more accessible to newcomers, those in the next class or those who might read it in published form.

From a research-method standpoint, using this data source (synchronous discussion) lets the facilitator’s strategy and participant learning (or lack of learning) be apparent because all comments are recorded. The existence of the recording makes them accessible in a way that evanescent speaking is not, even if it were recorded on tape.

From an educational design standpoint, course designers need to be aware that no two participant groups will be identically situated, which means that the learning will differ across groups. For this course in particular, successive classes of students are apt to be more familiar with programming practices and system development. For courses in general, future classes of students are apt to be more computer savvy than their predecessors and more adept at learning in a community of practice.
Furthermore, the problems to be solved will change over time. Taken together, these changes lead to concluding that designs for learning events will need to evolve with the learners’ situations.

In Professional Practice

In an era in which knowledge is becoming the infrastructure, “[s]kill flow conditions cash flow. Once the process of renewal slows down, the company or organization is in danger of petrifaction and extinction” (Lévy 1997, 2). In capitalism’s current rendition, organizations prosper to the extent they are able to import, produce, and introduce new abilities, not just once but continuously. Synchronous discussion of the kind analyzed here has the potential to help organizations cultivate the continuous renewal of skills required for them to prosper. For example, synchronous discussion might be one of the ways for an organization to:

- Create training experiences
- Deliver training experiences
- Plan engagements
- Conduct short time-horizon engagements
- Develop policies or procedures
- Review audit or project files
- Brainstorm on innovations
- Develop virtual team capabilities
- Facilitate meetings

To make these opportunities more concrete, suppose several clients of an auditing firm are preparing to implement new enterprise resource-management software and that the auditing firm has either no auditors with the requisite skills or inadequate numbers of such auditors. In an earlier time, one or more auditors at the firm’s national office, likely in conjunction with an auditor in a practice office, would be charged with creating an audit guide and training materials and conducting training sessions. This approach would still work if it could commence a year before the skills were needed. Now, however, the planning horizon is likely to be shorter. In this case, synchronous discussion might be used as follows: identify a small number of firm professionals whose expertise collectively best covers the requisite skills (which might include some individuals from other units of the firm), publish to a secure web site the audit plans and plan drafts that are closest to the needed plans, and charge these professionals with facilitating synchronous discussions whose purpose is to develop appropriate audit plans. Subsequent synchronous discussions could be focused on helping the audit teams’ members learn to conduct the audits and evaluate the evidence they gather. Because this approach avoids the need for the existing skilled professionals to meet face to face, the time and expense of travel are avoided.

Some auditors might say that an audit firm should be able to plan far enough ahead to be able to prepare for such engagements through traditional means. That may be, although undertaking a long planning horizon means accepting the risk that a client will abandon some or all of its installation of the new software.

With respect to the length of the planning horizon, lengthy preparation is rarely possible in due-diligence engagements because of the short lead times for accepting and completing the engagements. The professionals who are needed for any specific due-diligence engagement are mostly likely scattered around the globe, already working at capacity. With some shifting of their existing assignments, they might be able to plan the engagement in synchronous discussion sessions, which would let them be better prepared when they go on-site. To the extent the target of the due-diligence engagement has machine-accessible records of its operations, the team might be able to decrease or eliminate some phases of on-site work, which would let it accelerate the completion of the work.

At this point, auditors might say that members of _ad hoc_ teams still need to learn to trust each other and that they believe this process requires face-to-face interaction. In due-diligence engagements, there may not be time for this acculturation. Instead, a firm might rely on the individuals’ sense of professionalism, their previous working relationships, and the culture instilled through periodic training in face-to-face settings. Training through synchronous discussion could, however, be part of the acculturation
experience. For specific learning objectives, text and case materials could be made available online and a facilitator could guide learners in preparing a case solution. For example, if the learning objective was to develop competence with a firm’s audit software, learners could be writing and executing procedures in one frame as they report their results and ask questions in a synchronous discussion frame. In addition to task learning, participants would also become aware of other participants’ skills through the questions they ask and the comments they make. This kind of awareness of others’ talents helps participants develop a sense of whom they could rely on later for different kinds of expertise.

A synchronous discussion approach to training could help staff learn to be at home in electronic environments that mediate their interaction with their fellow professionals, which would prepare them for functioning in virtual teams that form and disband as engagements dictate. This approach has the additional advantage of making learning experiences more accessible whenever staff need the training without disrupting work schedules for a multiday training session. Learning in synchronous discussion could be conducted in short segments, just as staff need new skills.

REFERENCES


