

TOWARD HABITABLE LEARNING ENVIRONMENTS

Michael Feldstein
Assistant Director, SUNY Learning Network
State University of New York
Michael.Feldstein@suny.edu

Abstract

Today's online learning environments, as defined principally by the Learning Management System, are imagined as virtual analogs to the physical classroom—as neutral containers that are cleanly separable from their contents and the activities that occur with in them. This paper will argue that the separation between environment, interactions, and content in an online environment is false and is retarding the progress of developing educationally rich virtual learning spaces. It will also argue that successfully changing this state of affairs will entail enabling teachers and students to become system programmers.

The profile of today's Learning Management System (LMS) is fairly homogenous. According to the Edutools Web site, 26 of the LMS's they track have discussion forums, calendars, test engines, group work spaces, and grade books. Of these 26, 12 also have chat capabilities and instructional templates. As my colleague Patrick Masson is fond of pointing out, LMS's tend to differentiate themselves not with substantial additions to basic capabilities but with fine-grained enhancements to the standard feature set. Are chats archiveable? Are the archives searchable? Can I download the discussion posts for offline reading? Occasionally, we see a new application such as a blog or a wiki being added to the LMS's bag of tricks. By "occasionally," I mean once every few years or so. Blogs and wikis have been around for quite some time now and yet the major LMS's—both Open Source and proprietary—are just now getting around to adding them to their toolsets.

Why is the pace of evolution so slow? Nobody that I know of has argued that the LMS product category is mature and meets the needs of most users. Yet the rate of progress, at least in the sense of re-imagining the affordances that defines an LMS, has been much slower than one normally expects in an evolving software category. I will suggest that we have mistakenly imagined that our LMS is like a physical classroom, that our learning objects are like physical textbooks, and that online class interactions are like physical class interactions. In this view, our LMS deserves about as much attention as our physical plant. We make sure there are enough chairs for the students, we try to avoid certain structural defects such as a column in the middle of the room that blocks our students' view, and once in a while we need to check for specialized features such as a projector screen or lab table. But for the most part, our virtual classrooms are imagined to be like their physical analogs in the sense that we mostly ignore them except when there is something blatantly wrong or missing about them. Contrary to this view, I will argue that virtual class spaces are not simply neutral containers and that their shape has an enormous impact on how students interact with content, with the teacher, and with each other.

Code of Behaviors

Consider the most basic of classroom activities: The conversation. In a physical classroom, a teacher may employ different sets of rules for conversational behavior based on the activity of the moment. During a lecture, students may raise their hands and ask questions related to the content at hand. Sometimes these comments may play out into longer all-group discussions, but that is tightly controlled by the teacher's decision to call on others or not. On the other hand, the teacher may also pose a question and have students respond to both the question and each other. During this class discussion, the teacher may comment on each student's remarks or simply call on another student. She may tightly control who speaks in what order by calling on particular students, loosely control the conversation

by calling on students mostly in the order that they raise their hands, or not control it at all, allowing students to sort out conversation order themselves through verbal and non-verbal cues to each other. She may also call several students to the board, have them simultaneously write out answers to different problems (or the same one), and lead a class inquiry of the students at the board. This is a form of conversation too. Importantly, a teacher is free to employ all of these strategies, and many more, in any combination or order, and switch them on-the-fly as the class dynamic demands.

In an asynchronous teaching environment, many of these tools are missing. The teacher has very limited capacity to call on students and dictate an order of responses. And even if she had this capability, there is no body language from the students for her to read and make decisions about how to structure the conversation. Communicative bandwidth is very narrow in the asynchronous class. To compensate for the loss in ability to *communicate* the rules of educational conversations, we end up *encoding* them into our user interfaces. If I want my students to post solutions to the group and encourage their classmates to scan them for particular information, I might choose a discussion board with a threaded interface. Threading affords a great deal of informational density in terms of showing what content is available to review at the expense of atomicizing the comments themselves and reducing the view of the detailed conversational context. Conversely, in cases where I want the students to focus on one conversation and attend to the details of each comment, I may choose a flat discussion board interface. Non-threaded discussions afford browsing through the entire conversation in order and having previous comments available on the page when responding to the conversation, at the expense of narrowing the focus to one stream of conversation. If I am mainly interested in eliciting comprehension questions regarding content, I may forego a discussion board entirely, choosing instead to attach a single thread to the bottom of a content page, or even encouraging students to email me with their questions. Every interface design choice effectively enforces a particular conversational style. If your discussion board is threaded and you have to click around to see every post, you're not likely to quote or even refer to posts other than the immediate parent with any degree of regularity. It's just too much work. Discussion threads enforce conversational norms in an environment where the teacher is not free to eyeball the room and shape the conversation accordingly. Norms are created and communicated by software affordances. Far from being an empty room in which conversations happen to take place, the affordances of the learning environment software strongly influence the shape of the educational conversations that they support.

Note that the combinations of affordances could, in theory, be as varied and granular as the styles of discussion in a physical class. I can take a non-threaded discussion board and turn it into a blog in three steps. First, I make the body of the first post in each thread appear along with the subject line in the topmost view of the discussion board. Next, I set the rule so that the posts always display in reverse chronological order of the first post. What I have at this point is a group blog. Anyone can post and the bodies of the post are displayed on one page in reverse chronological order. If I want to turn it into a traditional single-author blog, I need only take the last set of setting the permissions so that only I can start a new thread. With three changes, I have transformed a discussion board into a blog with commenting capabilities. The interesting aspect of this transformation is that highlights the choices we are ignoring by thinking about discussion boards and blogs as fundamentally different beasts. For example, are there situations in which I would want to take only the first step, making the body of the first post appear in the thread list but leaving all other features untouched?

The logical conclusion of this thought process is that our learning conversation environment should be designed not as a set of disparate applications—a discussion board, a blog, a wiki—but as an erector set of affordances that teachers and students can combine and recombine at will in order to suit the needs of the educational conversation at hand. Under today's LMS architectures, this would require teachers and students to be software developers. We need to erode the distinction between user and developer, creating a gentle slope for users who want to gain progressively more control over their environments. Think of Microsoft Excel. Users start by learning how to put information into the cells. They move on to learn formulas. People who write Excel formulas don't generally consider themselves to be "programming" Excel. And yet that is precisely what they are doing. In the days before spreadsheets, those formulas would have to be coded by expert programmers. Finally, "power users" can graduate to become "programmers" when they begin to extend Excel with Visual Basic. One can imagine a similar process with the communication tools in a learning environment. One starts with prepackaged applications—discussion boards, blogs, wikis, and the like. One becomes a "power user" by granularly changing the capabilities that comprise these packages. And one graduates to "programmer" when one uses the application APIs to actually code new capabilities. With this capability, online teachers and students may regain the rich variety and flexibility of options

that they have for conversation in the physical classroom. But that won't happen until we reject the false dichotomies between environment and action as well as between user and developer.

Content or Discontent?

A third false dichotomy we must confront is the one between the learning environment and the learning content. In a physical classroom, the vast majority of learning content is presented as static text and static images. There are some notable exceptions to this, of course; a science class will have lab materials, a cinema class will have movies, and so on. But the vast majority course materials are still written words and still images. We can (and often do) take the same approach with online content, of course. But doing so ignores the power of the computer. And the minute we go beyond static words and images, particularly the minute we add any sort of interactivity, we have blurred the line between the content and the environment.

Consider a financial spreadsheet as a “learning object.” What is the locus of educational value in the spreadsheet? Is it the data itself? Is it the set of formulas that describe how some of the data is converted into useful information? Or is it the capability to plug in new numbers and run different financial scenarios? The answer is that it can be any of these or a combination thereof, depending what you're trying to teach. As long as we are focused on the data, on the nouns, then the learning object is completely separable from the environment. But once we start getting into the verbs, into the *manipulation* of the data, then we are talking about software affordances and therefore about the learning environment itself. If an image of a painting is considered the locus of educational value, then your “learning object” is an image that is neutral with respect to the environment. But what if you want your students to mark up the image and comment on it? What if you want them to share those comments? Is the image by itself the learning object? Is it the commentary on the image by other students? Or is it the ability to add new comments? As with the spreadsheet, the answer could be any or all of these, depending on the educational purpose.

Interestingly, we tend to think about spreadsheets—complete with formulas and the ability to alter the inputs—as content objects and sharing and annotation web services (such as Flickr) as environments. But this distinction is mostly an historical accident. If the canonical image annotation tool were a desktop application that shared discreet files and the canonical spreadsheet application were a web-based service that had no user-manipulable representation of a file as an object in the operating system, then we might have entirely different instincts about these tools. Regardless of the details of the tool, any time content comes with built-in tools for manipulating it, we blur the line between content and the environment.

Yet the design of our learning environments insists on hard distinctions. Learning objects are wrapped in metadata for re-use and are generally treated as bundles of content. While some learning objects may have applets or other interactive devices, they are still categorized as objects—as nouns, rather than verbs. As such, our learning environments tend to have fairly impoverished tool sets for building content-centric interactivity. Where are the data manipulation tools? Where are the online spreadsheets? Where are the image manipulation tools like Flickr? Where are the map manipulation tools like Frappr? What we mainly have in today's online learning environment is file sharing with a little metadata thrown in for seasoning. This state of affairs in no way allows us to compensate for the vast amounts of lost communication bandwidth in a physical classroom. In a physical class, you can point to a spot on an image. You can draw a map or calculate a result while simultaneously talking about it. The physical classroom has affordances just as the virtual one does. We are so attuned to them by millions of years of evolution that we don't even notice them or have names for them. And this is a problem. Because we are so unaware of them that we have no way to think about what we lose in today's online analog. To think about a physical classroom as a neutral container for what happens inside is to ignore every capability that physics, physiology, and psychology provide us for communicating and learning inside that container.

Situated Software

Framed this way, producing an adequate virtual learning environment starts looking less like a mundane or even trivial task and more like an impossible one. How can we hope to design an environment that equals the infinite richness of affordances found in a real-world classroom. In my opinion, the answer is that we can't. We can't design

such an environment. But we may be able to evolve one or, more accurately, we may be able to co-evolve with it, just as we have co-evolved with our physical environment. To do so, we need two things: A language for describing what we want and economically practical tools for building what we describe. In the former case, we need to evolve pattern languages such as the one that architect Christopher Alexander and his colleagues evolved to describe living spaces (Alexander et al 1977). We need to articulate patterns of educational activities so that we know what we want our software affordances to support. For example, Clay Shirky is advocating for the creation of a moderation pattern language to describe nuances of communication environments such as the one I discussed earlier (Shirky et al 2006). What are the possible goals of conversations? What are the software affordances that support or impede those goals? What are the problems with conversations that different structures can mitigate? We need to develop languages that describe what it is that we do when we interact with each other and content in the experience we call a “class.”

Second, we need learning environments that are evolvable by the users. We need to be able to change the rules that the software imposes on us at will to suit our needs. We need to be able to experiment. There are all sorts of practical barriers to making this happen, of course. To begin with, we’re talking about making ordinary users into something like programmers. There have been many failed experiments and few successes in this sort of endeavor (spreadsheets being one of the rare successes). And since this software must run on machines that support many users, it needs to provide this kind of flexibility without sacrificing certain kinds of scalability. How can you design the system so that no amount of tinkering by the users will bring the system down? How can you encourage the introduction of new capabilities and applications by non-programmers while still supporting authentication, common user interfaces, and all the other capabilities that one expects from enterprise software? And how do you support software like this? How do you document it? How do you provide help desk support? These are all deep questions that are beyond my area of expertise. But this is what we need. This is the challenge.

I close with a quote by a software architect who also saw the need for software to support this kind of flexibility. Reflecting on Christopher Alexander’s work, Richard Gabriel wrote,

There is, I think, a better goal [for software development], to which I want to draw your attention. It’s a characteristic that you’ve perhaps not thought of and which perhaps should have some influence over the design of programming languages and certainly software methodology. It is **habitability**.

Habitability is the characteristic of source code that enables programmers, coders, bug-fixers, and people coming to the code later in life to understand its construction and intentions and to change it comfortably and confidently....

Habitability makes a place livable, like home. And this is what we want in software--that developers feel at home, can place their hands on any item without having to think deeply about where it is....

Buildings like the Superdome lack habitability. In this instance people inhabit the building, but only for very short periods of time, and for very special occasions--and such buildings are not easily altered. The Superdome is a static building, and therefore it can stand as a monument, being little else.

A modern skyscraper, to take another example, has a fixed inflexible interior, which is secondary to the designed beauty of the interior. Little attention is paid to the natural light, and often the interiors are constructed as “flexible office space,” which means cubicles. The flexibility is for management to set up offices for the company, not for the inhabitants--the employees--to tailor their own space. When you run out of space in the skyscraper, you build another; you don’t modify the existing one or add to it.

Contrast this with the New England farmhouse. It starts as a small home with a barn out back. As the family grows and the needs of the farm grow, a back room is added to the house, then a canning room, then a room for grandma; stables are added to the barn, then a wing for milking more cows. Finally the house and barn are connected because it is too

difficult to get from the house to the barn in a blizzard. The result is rambling, but each part is well-suited to its needs, each part fits well with the others, and the result is beautiful because it is a living structure with living people inside. The inhabitants are able to modify their environment because each part is built according to familiar patterns of design, use, and construction and because those patterns contain the seeds for piecemeal growth.

I think this should be the goal for computer science practice. Most programming languages are excellent for building the program that is a monument to design ingenuity--pleasingly efficient, precise, and clear--but people don't build programs like that. Programs live and grow, and their inhabitants--the programmers--need to work with that program the way the farmer works with the homestead (Gabriel 1996).

This is what we need. We need our learning environments to be habitable. Until they are, the thing we call a Learning Management System today will continue to be inhabitable for only short periods of time, for special occasions, and with significant discomfort.

References

Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A Pattern Language: Towns, Buildings, Construction*. New York, NY: Oxford University Press, USA.

Gabriel, R. (1996). *Patterns of Software: Tales From the Software Community*. New York, NY: Oxford University Press, USA.