Roumani's Teaching Philosophy

My philosophy stems from a deep conviction that university education is about providing **insights** and imparting **skills** that transcend the details of subject matter. In fact, I view these details as mere contexts in which insights and skills can be developed. When I teach computer programming, for example, it is not the details of a particular programming language that I am after. I use these details as a mean to an end, with the end being such big ideas as "programming by delegation" and such critical skills as "how to approach programming problems". Details are often forgotten, and can even become obsolete in time, but a deep insight into knowledge, and the skill to confront challenge, will last a lifetime. These two things are the focus of *everything* I do and they shape my view of my role as a teacher. I want my students to attain a level of understanding that is unshakable by evolutionary changes in the field and to have the confidence to approach problems not seen before. Transforming these goals into pedagogy has been my pursuit for some thirty years and has culminated in two directives:

D1. Don't mix abstraction levels; stay on message.

What is the relation between driving a car and the combustion engine? Do I need to known one in order to learn the other? And more to the point, why did you keep referring to spark plugs when you were showing me how to accelerate?

D2. Don't teach me the "how"; help me discover it.

Try as you may but I don't think your lectures will ever teach me how to skate. You have explained the techniques really well, even made me an expert in the physics of skating, but I still fall flat on my face every time I set foot in the rink!

The first tells me how to orchestrate the topics within a course to enable deep understanding. The second helps me choreograph the discovery of concepts so that students can pick up skills while remaining engaged. I explain below what each directive means; how it can be implemented; and the way it played out in my work as an educator.

The First Directive
Don't mix abstraction levels; stay on message.

Scenario:

What is the relation between driving a car and the combustion engine? Do I need to known one in order to learn the other? And more to the point, why did you keep referring to spark plugs when you were showing me how to accelerate?

Context:

We use abstraction all the time to confront complexity. Rather than dealing with many details at the same time we focus only on the ones that are essential to the subject matter and replace the rest with simple artefacts that encapsulate them;

i.e. capture their effects while hiding their causes. This allows us to operate in one abstraction level and reason about things within it without needing to expose details from the level underneath. In the driving abstraction, the gas pedal, the steering wheel, and the other driving controls are the artefacts that encapsulate everything there is to know about the engine. Using abstraction as pedagogy allows us to teach someone to how drive without ever involving anything from under the hood.

Problem:

The topics we teach often span several abstraction levels. And since we are familiar with the contents of all these levels, it is natural for us to tend to mix them up as we teach. We do so, for example, when we describe properties in one level using terminology that belongs to another; or, when we reason about behaviour in one level by attributing it to causes in a lower one. We do this in the name of "completeness" to enable students to see the richness of the subject and appreciate its various cause-effect facades. I see two serious problems with such inter-level "crosstalk":

- Mixing abstraction levels makes things overly complex for first-time learners because it inflates the scope, deepens inter-topic dependency, and entangles concepts. Only a few students, ones with particularly long attention spans, will be able to learn in such a setting. Those students will have to sit through seemingly unrelated lectures until all the pieces are in place and at that point everything will suddenly make sense. I call this phenomenon an *induced* threshold concept[1]—one that emanates *not* from the subject itself but from abstraction muddling.

- Level interference conditions students to always think "bottom up"; i.e. they can reason about something only if they understand all its dependencies first. This inhibits "system thinking" and does not prepare students to face most real-life situations in which information about the lower levels is either un-known or incomplete. Whether you are a scientist who needs to study a new phenomenon in the absence of an underlying theory; an engineer who needs to build a system out of black-box components manufactured by others; or an analyst who needs to create technical specifications for software yet to be developed, you will have to be comfortable working with unknowns by building abstractions around them.

Discussion:

In the opening scenario, when the teacher invoked "spark plugs" to explain "acceleration", the intention was to provide a complete picture. To the learner, however, this was tantamount to noise! There is nothing wrong with teaching students about spark plugs provided you do so *after* they learn how to drive. In fact, you will likely find them more receptive to learning about how something works once they are comfortable using it. From the perspective of this pedagogy,

[1] See http://personal.strath.ac.uk/ray.land/thresholds/home.htm

progress through the curriculum can be viewed as chartering a path across abstraction levels. I find this pedagogy quite profound and advocate sharing it with the students (perhaps at the end of the course).

Implementation:
Generally speaking, early courses in the program, or early lectures within a course, would concentrate on the high abstraction levels and refrain from subordinating them to lower levels. This not only keeps the material accessible but also **relevant** and engaging because students are often familiar with, or can at least relate to, the domain of the higher levels. Furthermore, learning to deal with encapsulations nurtures **communication skills** early on because when you work with black boxes, everything must be articulated carefully and nothing can be taken for granted. Later, more advanced, lectures would expose the lower abstraction levels and show how they can "explain" what was observed in the higher levels. Exploring this cause-effect relationship develops **analytical skills** and breeds curiosity. And since the upper and lower levels are treated as independent entities, rather than parts of a continuum, they can be analyzed and assessed separately; e.g. a mechanism or theory in a lower-level may get rejected if it failed to account for a requirement or observation in an upper level. Permitting students to look at theory, not as a given, but as a hypothesis to be verified, is truly liberating because it allows them to explore "what can be" rather than merely recite "what is"—**critical thinking** in action.

Progress:
I have toyed with various aspects of this directive for a long time but it was not until 2003 that all the pieces came into sharp focus. It was then that I initiated a number of projects that included conducting research, writing a textbook, and revising the curriculum. The work culminated in a number of achievements including the publication of the paper *"Practise What You Preach: Full Separation of Concerns"* in the proceedings of the ACM technical symposium on Computer Science Education [1], and the publication of the book "*Java By Abstraction*" (now in its second edition) with Pearson Addison Wesley [2].

The Second Directive
Don't teach me the "how"; help me discover it.

Scenario:
Try as you may but I don't think your lectures will ever teach me how to skate. You have explained the techniques really well, even made me an expert in the physics of skating, but I still fall flat on my face every time I set foot in the rink!

Context:
This directive is orthogonal to the first: having re-organized the material so that there is no level interference, you now need to teach the material of a given level. Such material, especially within Science and Engineering, typically involves concepts whose learning requires both application and synthesis. How do you

teach such material? There are a number of requirements here: you know you have to explain the abstract concepts (deep understanding) but you also want to give students a sense of discovery; a chance to explore, observe, and evaluate (problem solving skills). But while you do want students to explore and try things, you don't want them to go off on tangents. We thus seek a pedagogy that meets all these, seemingly-conflicting requirements.

Problem:

To establish a baseline, let us imagine adopting the traditional, one-way pedagogy, and then assess the learning outcomes vis-à-vis providing insight and imparting skills. In this pedagogy, you start by explaining the underlying concepts, then pose a problem that depends on them, and then present its solution. If you did that then most students would likely not engage; they would find the concepts too abstract to sit through. If you grew up in an environment in which everything was interactive, and even "live TV" could be paused and rewound, then you too would probably not engage. In addition, little problem-solving skills can be learned from watching someone solve a problem. In short, this approach treats all students as reflective and ignores the larger subset of active learners [3].

Discussion:

The opening scenario, teaching how to skate, suggests a diametrically opposite approach, one that says "don't teach me—let me figure it out"! Back in 500 BC, Confucius hinted to something similar when he wrote:

I hear and I forget; I see and I remember; I do and I understand.

But how does one turn this rudimentary idea of learning-by-doing into pedagogy? It is clear that we must first set things up so that students can indeed see what we like them to see, comprehend what they are seeing, and then combine what they learned with their existing knowledge. Specifically, we must choreograph the encounter of new ideas, whether in lecture or in guided explorations, such that:

1. **One idea at a time:**
   This may seem obvious but we often ignore it by presenting examples that expose two ideas at once. We find such examples “rich” and “efficient” but they are in fact counterproductive because students will likely either miss both ideas or internalize a cognitive model that falsely links them.
2. **No cyclic dependencies:**
   Many skills are cyclic in their dependency graph: in order to learn A, you must first learn B, but in order to learn B, you must first learn A! Suppose you want to teach someone how to write a simple program that adds two numbers. For that, you first need to teach them how to read numbers from the user. But this requires[2] a knowledge of “exception handling”, which is an advanced concept that can only be understood after one learns how to program! We must break such circular dependencies and construct a linearized path in concept space.

[2] At least in Java versions prior to 5.0.

3. **Irregularities are hidden or delayed:**
   If you were to teach someone about the shortest distance between two addresses in a city with grid-like streets, you could give two Manhattan addresses as example, but you would make sure neither is near Broadway[3]! We all know that simplicity favours regularity but it is a fact that most real-life systems are not regular—their designs involve tradeoffs and compromises, and this implies exceptions and special cases. If we start with the rule and delay the exception, students can see the patterns and form mental models and this will help them when they later encounter the exceptions.

4. **Subtle differences are highlighted:**
   We often encounter concepts that are similar on the surface but with a subtle difference. In that case, we continue to observe the one-idea-at-a-time guide-line above but then follow with an exploration that involves both concepts and, hence, exposes the difference and puts it under the spotlight. When we say "the book *has* pages", for example, or the "wallet *has* coins", the meaning of "has" is similar in both but there is a subtle difference: the pages are part of the book but the coins do not come with the wallet! In object-oriented pro-gramming lingo, the book-page relationship is known as composition whereas that of wallet-coin is aggregation, and first-year computing students need to distinguish the two. After encountering each concept separately, students are presented with a situation that makes them experience the otherwise-dormant difference.

These guidelines help us set the flow of encounters so as to optimize the gained experience. But as the student progresses within this framework, it is possible to reach a dead-end, an observation the student cannot explain. It is here that the final link in the chain is needed: **just-in-time information**. Information that pro-vides background knowledge that seems missing and gives a "nudge" in the right direction. Such on-demand information is highly effective and illuminating, and its impact cannot be paralleled by fixed, pre-planned information delivery.

Implementation:

Creating an environment that incorporates the guidelines discussed above and immersing the students in it can be done in various settings. The instructor can prepare the encounters and then go through them pausing after each to prompt for ideas and initiate discussions as needed. This can also be done by playing a sequence-based game such as Jeopardy. A third possible setting is a self-paced lab with guided explorations to control the encounter flow (or sequencing). These settings provide varying degrees of engagement and participation. Regardless of the setting, the student should be encouraged to explore, and this implies **no penalties when mistakes are made**. In general, I find it counterproductive to overlap assessment with teaching.

[3] Broadway is the only major street that defies the grid.

I mention in closing that this directive shares many principles with the design of strategy-based computer games. The designers of these games discovered how to get players to learn long and complex games without reading a manual first and to keep them engaged hours on end. A study [4] of the pedagogy that underlies these games reveals features that are strikingly similar to the guidelines we derived above: the game starts in a **fish-tank** which is a stripped-down version of the game. In it, the game's features are revealed gradually (**one idea at a time**) to enable the player to master the basic skills before moving on to more complex situations (**no cyclic dependencies**). The game in the fish-tank is scaled down so only its main storyline is played (**irregularities are hidden or delayed**) with an emphasis on key relationships (**subtle differences are highlighted**). Besides fish-tanks, successful games come with a **sandbox**. A sandbox plays much like the real (not stripped-down) game except things cannot go too wrong too quickly, which means players can take risks and try out hypotheses (**no penalties when mistakes are made**). Finally, the study revealed that good games provide information on demand (**just-in-time**) because players don't read manuals but do use the manual as a reference after they have played for a while.

It has been a dream of mine to take the implementation of this directive beyond lectures and beyond exploratory labs to the realm of games; i.e. have the student play a game in order to learn something. This will *not* dumb down the content, or lower the expected learning outcome, of the course because, as we saw, games incorporate the very principles that we seek to implant in our teaching. I have not achieved this dream yet.

Progress:

I have recognized the need for learning environments for our students since the mid 90's and have developed one called **TYPE** (*T*he *Y*ork *P*rogramming *E*nvironment) to act as a fish-tank. And I have been advocating learning through guided explorations since 2002 when I published my lab design paper [5]. And to enable explorations, I created **eCheck** to act as a sandbox.

---

[1] H. Roumani. Practice What You Preach: Full Separation of Concerns. In *Proceedings of the 37th ACM SIGCSE technical symposium on Computer Science Education*, pages 491-494, Houston, TX, USA, March 2006. ACM.

[2] H. Roumani. *Java by Abstraction: A Client-View Approach*. Second Edition. Pearson Addison Wesley, 2008.

[3] R. M. Felder. Matters of Style. *ASEE Prism*, 6(4):18-23, December 1996.

[4] J. P. Gee. Learning by Design: good video games as learning machines. *E-Learning Journal,* 2(1), 2005.

[5] H. Roumani. Design Guidelines for the Lab Component of Objects-First CS1. In *Proceeding of the 33rd ACM SIGCSE technical symposium on Computer Science Education*, pages 222-226, Cincinnati, OH, USA, March 2002. ACM.