CodeRefinery teaching philosophies

Objectives

- Get to know the teaching philosophies of CodeRefinery instructors

Instructor note

- Teaching: 10 min
- Discussion: 20 min

Introduction

During this episode we split into breakoutrooms and discuss own teaching philosophies. Collect your teaching philosophies in the collaborative document. We will be going through these and the end of the lesson.

✍️ Ice-breaker in groups (20 minutes)

- Share your approach to teaching and your teaching philosophy with your group.
- Please share your tricks and solutions in the live document for others.

Additional ice-breaker questions:

- What is your motivation for taking this training?
- How structured or informal are your own teaching needs?
- What difference do you notice between the teaching what we (also Carpentries) do and traditional academic teaching?
- What other skills need to be taught, but academic teaching isn't the right setting?

Instructor views

Here CodeRefinery instructors share their training philosophy to show that we all have different teaching styles and how these differences are beneficial to CodeRefinery workshops.

It is important to explain how much we value individuals and that there is not one way to teach but as many ways as individuals. We want to help each other to find the way that is best for each of us.

⚙ Video recordings

We have recorded some of the below as videos: https://www.youtube.com/playlist?list=PLpLblYHCzJAAHF89P-GCjEXWC8CF-7nhX

✍ Bjørn Lindi

My teaching style has changed a bit since I started with CodeRefinery. In the beginning I had this “BLOB” (Binary Large OBject) of knowledge and experience that I wanted to to convey to the participants. With experience and some help from the Carpentries Instructor training, I have realized I need to serialize the “BLOB”, to be able to share it with others.

In a similar fashion as you would do with a binary large object which you intend to send over the wire, you will need stop signals, check-sums and re-transmissions, when you give a lecture. I have come to appreciate the natural periods/breaks the lessons offers, the questions raised, the errors that appear during type-along and the re-transmission. Co-instructors are good to use for re-transmission or broadening a specific topic.

When I started with CodeRefinery my inclination was to give a lecture. Today I am trying to be a guide during a learning experience, a learning experience which includes me as well. That may sound a bit self-centric, but is in fact the opposite, as I have to be more sensitive to what is going on in the room. The more conscious I am of being a guide, the better lesson.

Tools that I find useful in preparing a lesson is concept maps and Learner Personas (though I have developed too few of them):

- Concept Maps
- Learner Personas

✍ Radovan Bast

My teaching changed by 180 degrees after taking the Carpentries instructor training. Before that I used slides, 45 minute lecture blocks, and separate exercise sessions. After the Carpentries instructor training I embraced the interaction, exercises, demos, and typos.

My goal for a lesson is to spark curiosity to try things after the lesson, both for the novices (“This looks like a useful tool, I want to try using it after the workshop.”) and the more experienced participants (“Aha - I did not know you could do this. I wonder whether I can make it work with X.”). I like to start lessons with a question because this makes participants look up from their browsers.

Keeping both the novices and the experts engaged during a lesson can be difficult and

offering additional exercises seems to be a good compromise.

For me it is a good sign if there are many questions. I like to encourage questions by asking questions to the participants. But I also try not to go into a rabbit hole when I get a question where only experts will appreciate the answer.

I try to avoid jargon and “war stories” from the professional developers’ perspective or the business world. Most researchers may not relate to them. For examples I always try to use the research context. Avoid “customer”, “production”, also a lot of Agile jargon is hard to relate to.

Less and clear is better than more and unclear. Simple examples are better than complicated examples. Almost never I have felt or got the feedback that something was too simple. I am repeating in my head to not use the words “simply”, “just”, “easy”. If participants take home one or two points from a lesson, that’s for me success.

I prepare for the lesson by reading the instructor guide and all issues and open pull requests. I might not be able to solve issues, but I don’t want to be surprised by known issues. I learn the material to a point where I know precisely what comes next and am never surprised by the next episode or slide. This allows me to skip and distill the essence and not read bullet point by bullet point.

I try to never deviate from the script and if I do, be very explicit about it.

A great exercise I can recommend is to watch a tutorial on a new programming language/ tool you have never used. It can feel very overwhelming and fast to get all these new concepts and tools thrown at self. This can prepare me for how a participant might feel listening to me.

I very much like the co-teaching approach where the other person helps detecting when I go too fast or become too confusing. I like when two instructors complement each other during a lesson.

✍️ Sabry Razick

My approach is to show it is fun to demystify concepts. Once a concept is not a mystery anymore, the learners will understand is what it means, where it is coming from, why it is in place and what it could it offer for their future. I try to relate concepts to real life with a twist of humour whenever possible if the outcome is certain not be offensive to any one. I use diagrams whenever possible, I have spent weeks creating diagrams that is sometime three or four sentences. That effort I consider worthwhile as my intention is not to teach, but to demystify. Once that is achieved, learners will learn the nitty gritty on their own easily and with confidence, when they have the use-case.

✍️ Richard Darst

Like many people, I've often been teaching, but rarely a teacher. I tend to teach like I am doing an informal mentorship. I've realized long ago that my most important lessons weren't learned in classes, but by a combination of seeing things done by friends and independent study after that. I've realized that teaching (the things I teach) is trying to correct these differences in backgrounds.

My main job is supporting computing infrastructure, so my teaching is very grounded in real-world problems. I'm often start at the very basics, because this is what I see missing most often.

When teaching, I like lots of audience questions and don't mind going off-script a bit (even though I know it should be minimized). I find that sufficient audience interest allows any lesson to be a success - you don't have to know everything perfectly, just show how you'd approach a problem.

✍️ Stepas Toliautas

I aim for my learners to understand things (concepts, techniques...), instead of just memorizing them. The phrase I use from time to time when teaching information technology topics (be it hardware or software) is "there is no magic": everything can be explained if necessary. While CodeRefinery also emphasizes usefulness/ immediate applicability of the content, often having the correct notions actually helps to start using them sooner and with less mistakes.

I try to guide my audience through the presented material with occasional questions related to previous topics and common knowledge – to help them link the concepts already during the lesson. I'm also fully aware that waiting for someone to answer is quite uncomfortable in-person and doubly so in an online environment, especially if no one uses their cameras :)

And if I get the question I don't have the answer for (or material to show it) at the moment, in university classes I prepare and come back to it next time. While with one-off workshops there might not be a "next time", open-source material used by CodeRefinery allows to have the same outcome: the latest and greatest version stays available for self-study afterwards.

Summary

❗ Keypoints

- People have different viewpoints on teaching.