Lesson design and development

Objectives

- We share our design processes for teaching material and presentations.
- Learn how to design lessons “backwards”, starting from learning objectives and learner personas.
- Learn good practices for improving existing material based on feedback.

Instructor note

- Discussion: 20 min
- Exercises: 35 min

Exercise: How do you design your teaching material?

✍️ **We collect notes using a shared document (5 min)**

- When you start preparing a new lesson or training material, where do you start?
- What tricks help you with “writer’s block” or the empty page problem?
- Maybe you haven’t designed training material yet. But how do you start when creating a new presentation?
- If your design process has changed over time, please describe what you used to do and what you do now instead.
- What do you know now about preparing lessons/training/presentations that you wish you knew earlier?

Creating new teaching material

Typical problems

- Someone creates a lesson, but they think about what is interesting to them, not what is important for the learners.
- “I want to show a number of things which I think are cool about tool X - how do I press these into 90 minutes?”
- Write down material you want to cover and then sprinkle in some exercises.
- Thinking about how I work, not how the learners work.
- Trying to bring learners to their level/setup, not trying to meet the learners where they are.

- Not really knowing the learning objectives or the learner personas.

Better approach

Good questions to ask and discuss with a group of colleagues **from diverse backgrounds**:

- What is the expected educational level of my audience?
- Have they been already exposed to the technologies I am planning to teach?
- What tools do they already use?
- What are the main issues they are currently experiencing?
- What do they need to remember/understand/apply/analyze/evaluate/create (Bloom's taxonomy)?
- Define learner personas.
- It may be an advantage to share an imperfect lesson with others early to collect feedback and suggestions before the lesson "solidifies" too much. Draft it and collect feedback. The result will probably be better than working in isolation towards a "perfect" lesson.

The process of designing a lesson "backwards"

As described in "A lesson design process" in the book Teaching Tech Together:

1. Understand your learners.
2. Brainstorm rough ideas.
3. Create an summative assessment to know your overall goal.

    [Think of the things your learners will be able to do at the end of the lesson]

4. Create formative assessments to go from the starting point to this.

    [Think of some engaging and active exercises]

5. Order the formative assessments (exercises) into a reasonable order.
6. Write just enough material to get from one assessment (exercise) to another.
7. Describe the course so the learners know if it is relevant to them.

Improving existing lessons

All CodeRefinery lessons are on GitHub

- Overview: https://coderefinery.org/lessons/
- All are shared under CC-BY license and we encourage reuse and modification.
- Sources are all on GitHub: https://github.com/coderefinery
- Web pages are generated from Markdown using Sphinx (more about that in the episode Lessons with version control).
- We track ideas and problems in GitHub issues.

Collect feedback during the workshop:

- Collect feedback from learners and instructors (Example from a past workshop).
- Convert feedback about lessons and suggestions for improvements into issues so that these don't get lost and stay close to the lesson material.

Collect feedback before you start a big rewrite:

- First open an issue and describe your idea and collect feedback before you start with an extensive rewrite.
- For things still under construction, open a draft pull/merge request to collect feedback and to signal to others what you are working on.

Small picture changes vs. big picture changes:

- Lesson changes should be accompanied with instructor guide changes (it's like a documentation for the lesson material).
- Instructor guide is essential for new instructors.
- Before making larger changes, talk with somebody and discuss these changes.

Use case: our lessons

As an example to demonstrate the process of designing and improving lessons, we will have a look at one of our own lessons: Introduction to version control with Git.

- Initial 2014-2016 version
  - https://github.com/scisoft/toolbox-talks and https://toolbox.readthedocs.io/
  - Amazingly they are still findable!
  - Format: Slides and live coding.
  - Exercises were separate, during afternoon sessions.
- Some time in 2014-2015 attended Carpentries instructor training.
- 2016: CodeRefinery started.
- 2017: Started a new repository based on the Carpentries lesson template (at the time using Jekyll).
  - Exercises become part of the lesson.
  - We start in the **command line** and only later move to GitHub.
- 2019: A lot more thought about learning objectives and personas.
  - Also license change to CC-BY.
- 2022: Convert lesson from Jekyll to Sphinx.
  - Using the tools that we teach/advocate.
  - We can have tabs and better code highlighting/emphasis.
  - Easier local preview (Python environment instead of Ruby environment which we were not used to in our daily work).
- 2024: Big redesign. We move the lesson closer to where learners are.

  - Start from GitHub instead of on the command line.
  - Start from an existing repository instead of with an empty one.
  - Offer several tracks to participate in the lesson (GitHub, VS Code, and command line) and learners can choose which one they want to follow.
  - Blog post: We have completely changed our Git lessons. Hopefully to the better.
- Next steps?
  - Making the lesson citable following our blog post.
  - Improvements based on what we learn from this workshop.

The overarching trend was to make the lesson simpler and more accessible - to meet the learners where they are instead of pulling them to the tool choices of the instructors. Looking back, we learned a lot and the learning process is not over yet.

Exercise: Discussion about learning objectives and exercise design

✍️ **We work in groups but use the shared document as result (20 min)**

1. As a group **pick a lesson topic**. It can be one of the topics listed here but you can also choose something else that your group is interested in, or a topic that you have taught before or would like to teach. Some suggestions:
   - Git: Creating a repository and porting your project to Git and GitHub
   - Git: Basic commands
   - Git: Branching and merging
   - Git: Recovering from typical mistakes
   - Code documentation
   - Jupyter Notebooks
   - Collaboration using Git and GitHub/GitLab
   - Using GitHub without the command line
   - Project organization
   - Automated testing
   - Data transfer
   - Data management and versioning
   - Code quality and good practices
   - Modular code development
   - How to release and publish your code
   - How to document and track code dependencies
   - Recording environments in containers
   - Profiling memory and CPU usage
   - Strategies for parallelization
   - Conda environments
   - Data processing using workflow managers
   - Regular expressions
   - Making papers in LaTeX

- Making figures in your favorite programming language
- Linux shell basics
- Something non-technical, such as painting a room
- Introduction to high-performance computing
- A lesson you always wanted to teach
- …

2. Try to define 2-3 learning objectives for the lesson and write them down. You can think of these as "three simple enough messages that someone will remember the next day" - **they need to be pretty simple**.
3. Can you come up with one or two engaging exercises that could be used to demonstrate one of those objectives? **They should be simple** enough people can actually do them. Creating simple exercises is not easy. Some standard exercise types:
   - Multiple choice (easy to get feedback via a classroom tool - try to design each wrong answer so that it identifies a specific misconception).
   - Code yourself (traditional programming)
   - Code yourself + multiple choice to see what the answer is (allows you to get feedback)
   - Inverted coding (given code, have to debug)
   - Parsons problems (working solution but lines in random order, learner must only put in proper order)
   - Fill in the blank
   - Discussions, self directed learning exercises

Great resources

- Teaching Tech Together
- Our summary of Teaching Tech Together
- Ten quick tips for creating an effective lesson
- Carpentries Curriculum Development Handbook
- Our manual on lesson design