Software Carpentry Evaluation
Updated 2012-04-10
Jorge Aranda (mostly) & Greg Wilson (a bit)

We are halfway through the SWC funding period, and it would be good to set down in writing our current thinking on assessment.

To Date

- A preliminary evaluation proposal, to guide assessment efforts until our data analysis shows a better way
- A pre- and post-intervention survey. We applied the pre-intervention survey at four sites so far, with 82 responses:
    - Indiana, 17 responses
    - Monterey Bay, 28 responses (2 partial)
    - Berkeley Lab, 27 responses
    - Utah State, 10 responses (*in progress*)
- A pilot interview visit with UBC SWC participants. We conducted three hour-long face-to-face interviews, in which we tested the survey and talked about their computing and software development and about SWC.
- A site visit during the Berkeley Lab workshop. Though the intention was to interview participants during workshop downtimes, there were few of those, and they were not appropriate for interviewing. All chats were short, informal, and with respondents generally aiming to please (interviewer was also a helper throughout the bootcamp and was perceived to be "part of the team"). Emphasis was therefore on observations during the workshop, and on the links between the workshop and the principles and recommendations that are the goal of SWC.
- An analysis of ten videos prepared by STSCI participants as part of their "homework", where they all individually code a solution to the same problem, and of experienced developers tackling the same problem.

Thoughts

Our final assessment protocol needs to be:

1. lean (not time-intensive for subjects);
2. scalable to at least an order of magnitude more participants than SWC currently has; and
3. applicable (and interpretable) by people without particularly sophisticated research skills.

What we are doing now is not any of these things—it is our attempt at figuring out a proper assessment protocol by producing several tentative procedures.

Interviews and video analyses are time-consuming, so if they are to be used for evaluation, we will only be able to apply them to a subset of participants. This should be fine because we are not conducting assessment to provide a grade to participants, but to evaluate our own efforts.

Surveys scale much better, but present two other problems: the things we really want to assess (e.g., principles like "paranoia makes us more productive") do not lend themselves well to self-reporting, and they are not easy to ask about in short, closed questions.

One interesting finding from the MET Gathering Feedback project (http://metproject.org/) is that a variety of evaluation techniques yields better results than any one evaluation technique on its own, at least in the context of assessing school teachers. We should therefore use several techniques concurrently to assess SWC.

I am currently leaning toward the following mix: pre- and post-surveys for all participants of all upcoming bootcamps, video analysis of a random sample of them (corresponding to about one day of work per bootcamp site), and remote pre- and post-intervention interviews (via Skype) with a handful of attendees to each bootcamp (corresponding to about half a day of work per bootcamp site, or to about two days of work including set-up and analysis time—the interviews could be conducted by non-experts with some interview training and monitoring.) Including time for analyzing the survey and preparing a summary report, this means that an evaluation of a single intervention could take about one week, and that therefore a person fully dedicated to assessment could conduct a few dozens in a year. It also means that assessment time for a bootcamp would be roughly in the same ballpark as instruction time for it, or, alternatively, that it would imply double the effort for a bootcamp. This is clearly out of scope of the existing funding, but on par with some other efforts: for example, most of the US$3M that the Open University spent on developing their new "Introduction to Computing" curriculum was spent on proofreaders, testers, and editors (http://computinged.wordpress.com/2012/03/26/teaching-4500-students-to-program-ubiquitous-computing-mike-richards-and-my-digital-life/).

Further Thoughts on Video Analysis

Perhaps the most interesting proposal we have (certainly the most innovative) is to conduct video analysis of participants at work, through screen recording and think-aloud methods. The technique shows promise, but it's still early to tell if it will be appropriate. A few problems became clear after analyzing the videos we have so far:

- Video durations for what is supposed to be a fairly simple task are long. 30-60 minutes for the "helmet" task was standard. This does not include the time that some participants clearly spent making sense of the task (reading the spec, planning a strategy), but that they did not record.
- Several of the submitted solutions would not have satisfied the spec in full, though participants seemed to think they did.
- The task did not include anything other than coding and testing. That is: no interaction with a version control system, no (or rather, very little) interaction with the shell, no SQL. It should be possible to design tasks that require these tools, but that will increase analysis time. *(Note: one person did create a version control repository for the task on his own.)*
- Several participants had trouble recording their screens. If we are going to ramp this up, we'll need to find what works and what doesn't work on each platform.

There were also several benefits or otherwise interesting things that became clear with the video analysis:

- I found it almost always easy to tell, after only a short time into the video, whether it was going to be a smooth task for the participant or not. I don't think this would have come out as clearly with any other of the techniques we are considering. Several hints were immediately available:
    - whether they had an idea of the overall structure of what they were going to do before doing it
    - whether their decomposition was conceptually clear
    - whether it seemed that their tools felt familiar for them
    - whether it seemed that the language felt familiar for them (including, when things went wrong, whether they could make sense of the error messages efficiently)
    - whether they were using overly complicated structures and idioms
    - whether they had grasped the problem spec when they narrated it
- The think-aloud process was also useful: it often helped explain why participants went down weird routes, and was also useful for figuring out whether they knew they were doing something sloppily or not ("I know you're not supposed to copy this like that, but for now this should be enough...").

- Watching the videos had an educational value for me, too: I learned about some features of Python I didn't know of, and it helped me clarify, for myself, what software development expertise looks like. I return to this point below.

We should therefore:

- Build the need to interact with tools other than just a programming language into the task description.
- Experiment with debugging and maintenance tasks, as opposed with only new code tasks.
- Get participants to submit their code as well, so it can be tested and graded.
- Experiment with peer commenting of performance, as an educational tool. I've learned from watching these videos; it's reasonable to expect that participants could also learn from them. They learn from watching others tackle the same task in different ways, perhaps with different tools, and making the same or different mistakes. They also learn from getting other people's feedback. In some domains (public speaking training, for instance) this is done routinely. This is not an assessment strategy, but it still seems valuable.
- Produce a rubric for assessing videos to make analysis of them more systematic.

One complicating factor is the potential conflict between using videos to assess the training, and using them as a graduation exercise (i.e., requiring students to submit a video of themselves doing a task "well enough" to earn a Software Carpentry badge).

Next Steps

The next three steps are:

1. Begin conducting the post-intervention interviews with STSCI participants (the workshop is now, perhaps, far enough into the past for them that things have returned to normal, except for whatever benefits they extracted from SWC). Action: JA.
2. Design a task for video assessment that is short (so that learners will complete it) but comprehensive (so that it exercises all four of the major skill categories covered in the boot camp). We recognize the tension between these requirements... Action: GW
3. Prepare a rubric for assessing these videos. Action: JA