# Chapter V

## Construction versus sandbox

One process problem arising from the use of the Django framework is the issue of how to track time spent in the sandbox versus time tracked as construction. Typically, any time spent writing test code or production code would be counted as part of the construction phase. However, a significant amount of time in this project was spent trying to perform simple tasks in production code using Django idioms. Since many of these idioms were unknown to me at the start of the project, the typical “red-light, write production code, green-light” pattern of TDD was skewed. One of those cycles would normally take a small amount of time (measured in minutes). However, I had to learn how to idiomatically accomplish those unit-level tasks in many cases after writing the failing test in order just to produce the simple green-light production code. In some cases this meant that the length of a single unit test cycle could span an hour or more instead of two minutes.

Since I was *technically* writing production and test code during that time, I made the choice during iteration 1 to document this time as construction time. Within the scope of this project, that decision worked for estimating effort in subsequent iterations. A problem would arise if I were to use data from this project to estimate effort in a second Django project. Because my data tracking was not granular enough to distinguish “learning-Django-construction” from “normal-TDD-construction”, using this data again would produce skewed effort estimation because I would not be learning Django in a second project. I believe this would hold true for any project heavily based on a framework that was being used for the first time (or even a project in a new language). Once you know the framework and have added it to a conceptual toolbox, coding is smooth and the learning curve is low because you are merely applying things you already know in different combinations.

For this reason, if I were starting the project again today, I would track all time spent learning Django (even time when I was technically writing production code or tests) as sandbox time. That would still enable me to accurately estimate effort required for subsequent iterations by including sandbox time in those calculations. In a subsequent project, however (when I ideally would know Django and not require sandbox time) I could estimate effort based only on actual construction time.

## Replicating complex system behavior in the sandbox

The “sandbox” in PCSE is so named because you can climb in, build what you want, and destroy it when you’re through, all without worrying about affecting production code. For many tasks, this is adequate both as a metaphor and a technique. An example of a situation when the sandbox shines is familiarizing yourself with some aspect of how a language works – opening up a sandbox file, filling up a linked list and plucking the elements you think you want out of it, then going back to production code and rewriting that code to take advantage of your confidence.

When building a large system, however, there are limitations with the sandbox. Bugs were prone with Django (as in many systems) at the interfaces between multiple components. Furthermore, some bugs only manifested when components tried to access or iterate over entries in the database. Replicating these problems in the sandbox is difficult because the unit-level components or tests may appear to be working fine without the complexity of the entire system. During iteration 2, I reached a point at which the system was large and complex enough that I could no longer replicate problems in production code using from-scratch sandbox code.

There are at least a few ways to handle this. At first I attempted to make small changes to the production code in an attempt to tease out the nature of problems I was getting. However, even small changes to interfaces would break half or sometimes more of my test cases. This was problematic from an immediate standpoint (making it a pain to scroll through all of my expected failing test cases to find the one I actually cared about) and also from a process standpoint (I was losing confidence that failing test cases actually represented a problem or even that passing test cases meant something worked). I briefly tried replicating the system in the sandbox, but quickly discarded this idea as it mean prohibitive amounts of stubbing and recoding just to get simple test cases working again.

I eventually decided on a third method, made possible by my decision to track my code in a Git repository. At each stable point in the project, I made a commit to the repository, making sure I had test cases to match production code, etc. When beginning implementation of the next feature, I simply cloned the repository as a sandbox, keeping all the functionality of the system since the last stable commit but without the danger of breaking functional code. If I encountered a problem that needed to be solved in the sandbox, I did so. When I had solved it, I rewrote the code from scratch in the production codebase to prevent side effects, wiped out the sandbox, and committed the new stable version.