Maxitize

A Project Introduction

by Mike Kramlich

2016 October 5

This project is purely a portfolio showpiece. The goal is to have an excuse to play with certain ideas, tech and toolkits related to data science, marketing, business analytics, math, machine learning and some recently popular cloud services. To apply and integrate them in a useful and coherent way. And to demonstrate in public my understanding or expertise in these areas.

The topic domain chosen for an application is that of a mock online store and advertiser (like Amazon but simpler and much smaller) and the ostensible end goal is to maximize profits for this imaginary business. I plan to publish and revise code and write-ups periodically, as pieces become mature enough. And to include more details such as the implementation tech mix & architecture, and any test data or performance metrics.

First, about the name. Chosen quickly. It may not be unique and there may already be something else out there using this name. Initially we don't care. But its good to have a name, especially one that is relevant and exciting. We can change it later if it ever becomes important. For now I liked it because the name is evocative of three key features or themes of the project: *maximizing* the *monetization* of products and/or the return on *advertising*. Therefore my own made-up term for the pursuit of this goal is... *to maxitize*.

I will write the code in Python, at least initially. This helps with rapid development, both because its a concise and low ceremony language, and because its already my preferred language for getting things done. Python also has a lot of mature libraries and good ecosystem support for dealing with data and math.

I will use sqlite3 as the initial data store. This gives us SQL for CRUD operations and a proven ACID database. We could migrate to PostgreSQL in the future if that became necessary. We might augment with more exotic data stores like Redis, Neo4J or Cassandra, to give a few examples, in cases where they might be a better fit.

There will be no automated tests, at least initially. This helps us to move faster. Ultimately we may switch to complete test coverage and even strict TDD if the code were to acquire a significant user base, or customers. Because in those situations the benefits would outweigh the costs. At the very beginning, however, we want to be fast and agile. Perhaps the project will be abandoned anyway. Tests are a kind of premature optimization, or speculative makework, at this stage. Though tests are so helpful in

production code, especially with a large number of customers, users and developers, that they should be considered such best practice that their absence in those situations would be inexcusable.

I want to be very very clear about a certain point to anybody reading this who is rabidly pro-test: I love tests, I love automated tests and TDD, I totally appreciate them. But they do impose a burden and there are cases and time periods where it is wiser to not have them. For this project, at start, it's wiser to not have them. It's different from other common best practices (such as backups, version control, RDD, CLI bias, thinking carefully, incremental development, Always Working Code, KISS, etc.) in that most of those other best practices impose very little additional marginal cost on the development workflow, in order to reap their benefits. Especially early on. Therefore, it is always wise to follow those practices, no matter what. And regardless of the company, project or its stage in the lifecycle it is always wise to have a smart programmer who thinks carefully, prioritizes, decides, types fast, solves and ships. There's a cost to this, too, but it's always worth paying.

I'll use git for version control and publish code to GitHub in a public repo or gists.

I'll develop on a Mac laptop, and only support Mac, initially. Ultimately we'll add Linux and that would likely become our ideal platform to target.

I'll practice RDD (README-driven development). Or something similar to it.

I'll practice CLIFMO. This is a term I've coined. Its an acronym for "CLIs First, Maybe Only." The basic idea is not original with me, lots of programmers practice something like this. But I wanted to make a term for it. Plus I've expanded on the core element, the bias to CLIs and Terminal based workflows, to add some additional rules and architectural patterns which I think are aligned with them, and achieve a kind of bonus synergy. For those who are interested, I'd like to go into more detail in a separate piece about my vision for CLIFMO. It is such good practice, such a strong strategy and set of ideas, that it should be taught as best practice. The absence of CLIFMO in a modern software project should raise critical eyebrows in the same way that the absence of version control, or the absence of backups, RDD or TDD should raise eyebrows.

Anyway, that's it for now. I'll go into more detail in the next post in the series, I think probably including more about the vision and architecture of the Maxitize engine.

Until then, remember: relax, stay sharp and keep your lasers handy. The Computer is your friend!

(The cool kids will get that reference.)

---

author: groglogic@gmail.com
http://synisma.neocities.org/resume_Mike_Kramlich__Software_Engineer.pdf