Agile/Scrum Basics plus tips for Capstone Success

Benjamin S. Skrainka

July 22, 2016

Objectives

Today's objectives:

- List tips for successfully completing a good capstone project
- Describe capstone phase of DSI
- Define basic Agile concepts
- Explain how to scrum works under Agile

Le Grand Dessin

The big picture:

- Complete *Minimum Viable Product* (MVP) in 1 week
- . . . aka "shitty first model"
- Polish in second week
- Should have completed draft of presentation on Friday of second week
- Week 11 of the course is hiring day prep and hiring day

Tips to finish your project (1/2)

Some pro tips:

- "It is better to have a bad plan than no plan at all" – Alex Dunne
- Divide & Conquer is usually a winning strategy for complex/large projects
  - Keep it simple
  - Pursue one idea at a time
  - Break tasks down into ~4hr/task
  - $\Rightarrow$ Can complete two tasks per day
- Create deadlines – deadlines make you get stuff done
- Ask yourself, "Will this get my project done?" If not, then stop and work on correct task

Tips to finish your project (2/2)

Write down (pipeline) requirements:

- Stages
- Input
- Output
- Contract
- Unix filter & pipes model
- How will you test?

Methodology

Many methodologies have been used through the years:

- Waterfall:
  - Old school
  - Big, monolithic
  - Tries to think through and pre-plan the entire project up front
  - Requirements $\rightarrow$ Analysis $\rightarrow$ Design $\rightarrow$ Code $\rightarrow$ Software product
  - . . . But hard to to solve a problem until you learn more about it by working on it
- Agile:
  - Reaction to totalitarian methodologies, wasted resources, missed deadlines, building stuff customer doesn't want/need
  - Goal: be nimble & responsive to "customer"
  - Only build what you need
  - Light, incremental, small
  - Team rules, not manager
  - Success measured in software

Agile process

Agile process breaks work into manageable *sprints*:

- Break work into sprints
- Sprints last 2-4 weeks, typically
- Break sprints into tasks
- Tasks should take no more than two days to complete
- Start each sprint with *sprint planning*, where you agree on the goal of the sprint and the tasks you will (attempt to) complete

Control

Use a tool to track your team:

- Provides insight into state of project
- Track with PivotalTracker, JIRA, or equivalent
- Provides accountability by tracking progress, e.g., *Burn Down* charts
- Daily check-in in *Scrum*

Scrum

Team holds scrum every day in the morning:

- Run by *scrum master*
- Ensures everyone is working on the right task
- Identifies problems early
- Time boxed: should be short ($\leq$ 15 minutes)
- Can only discuss:
  - What you did yesterday
  - What you will do today
  - Any blockers
- Can say "Pass, no blockers"
- Take everything else off line

Scrum is at **10 am** at Galvanize

MVP by end of first week

Your mission is to build an MVP in next week:

- *MVP* means "minimum viable product"
- Start simple
- Only build what you need
- . . . but plan for reasonable possible changes/extensions
- Break down by tasks:
  - Task should take 4-8 hours
  - Break anything bigger down to this grain
- List requirements to identify tasks
- Identify *Use Cases*

Use Cases

Uses cases describe a possible use:

As a *[type of user]* I want to *[perform some task]* so that I can *[achieve a goal, benefit, outcome]*

Homework

Your first tasks:

1. Create your project report
2. Structure your files (preliminary)
3. Complete the most detailed CRISP-DM plan – more detail and pre-planning will save you time

$\Rightarrow$ don't forget to keep a notebook of all the random thoughts you have about your project

I like to start writing my final report from the beginning of the project, so that I can document data clean, methodology, and other decisions while they are fresh in my mind

Pro tips

Some pro tips:

- *Always be committing* (ABC)
- Structure your project sensibly
- Use CRISP-DM
- Prototype and test on a small subset of the data so that you can iterate quickly
- Use Test-driven development
- Debugging tips
  - Use the debugger (PDB)
  - Something you think is true is not, so reexamine your assumptions
  - Explain your code to your rubber duck . . . or a friend

Basic design

Some software design advice:

- IPython notebook is great for EDA and exploring ideas, but you should write code/pipeline using an editor + TDD
- An interface is a contract
- Loose coupling between modules . . . & avoid cycling dependencies
- Fail early

Ask for help

Make sure you maintain your momentum:

- Ask for help when you are blocked more than a reasonable amount of time
- Use deadlines to stay on track