Basic Software Engineering

SURP 2022 Python Bootcamp

Ohio State Astronomy

Slides by: James W. Johnson

Software Engineering

The application of the principles of engineering to software development.

All of the usual principles apply

- Making efficient use of resources
- Extensions of existing features
- Testing of individual components
- Maintenance

Why care if you're an astronomer? If you build anything substantially large, these principles can save you not just headaches but migraines.

DRY: Don’t Repeat Yourself

You should *never* write the same code twice within the same application.

Always assume that every software component you write (no matter how well it performs) is wrong or bugged in some subtle, nuanced way

- When you find a bug, you want to be able to fix it *once* and have that change propagate through the *entire code base*

Caveat: Some choose to break this rule for software written in a *modular* manner – that is, if each component of your software is implemented independently of the other components. In this case DRY applies within the individual *modules*.

No Ifs, Ands, or Buts

When describing the function fulfilled by a method, object, file, etc. you should be able to state it in a simple, declarative manner with no ifs, ands, or buts.

If you can’t state it without an if, and, or but, then you need to split up that function/object/file/whatever it is into more than one component.

This is a mnemonic for “every component should do exactly one thing.”

Version Control

Cataloging of previous copies of a software’s source code

- Popular tools: GitHub, Bitbucket
- Command line: *git*
- Keeps track of all previous changes to your code base for you

The standard for public codes is to use the 3-digit system (e.g. 1.2.1, 2.3.0)

- First is for back-compatibility, second is for new features, third is for bug-fixes

You don’t need to use GitHub/Bitbucket for version control – you can also store your code, plots, etc. there to manage copies between multiple systems. I also use it to share my research notes and plots with collaborators.

GitHub

First and foremost: file-sharing system

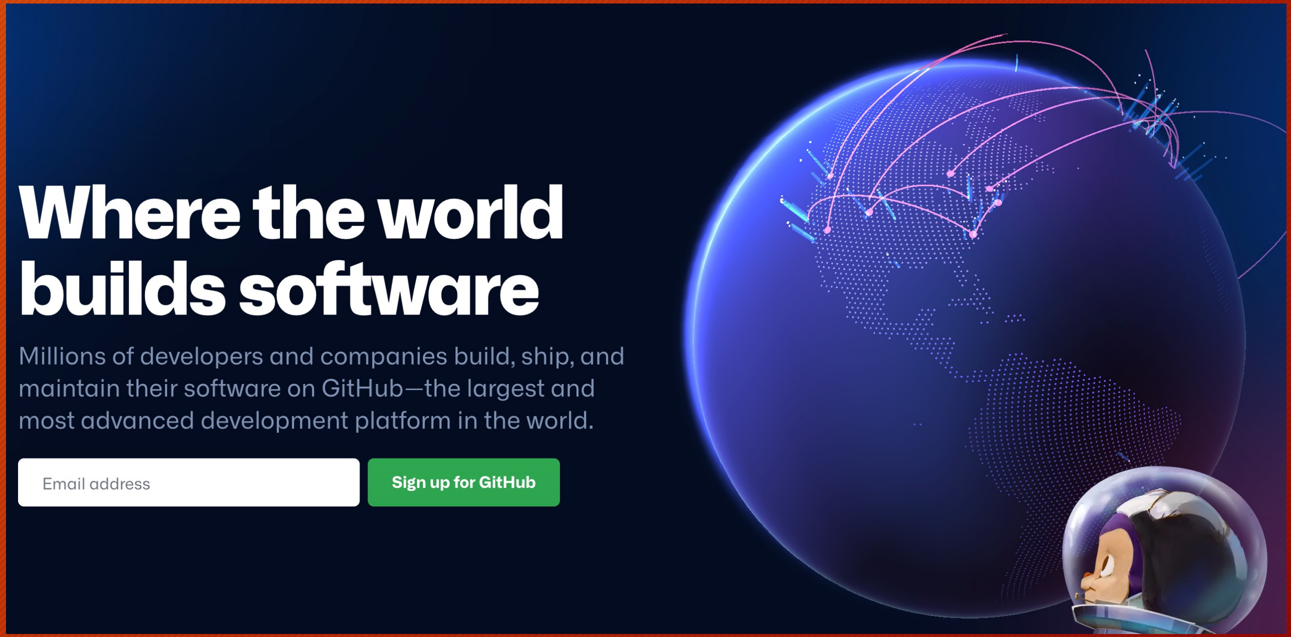
- Keeps a catalog of the changes to your code over time, making it a great tool for version control for developers

For astronomers:

- Share with collaborators
- Manage files between multiple computers

Terminal: *git [add, commit, pull, push, …]*

- Their website will show you how to set up a repository



Where the world builds software

Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.

Minimization of Dependencies

If the code base for your software uses another, external software (e.g. NumPy), that is called a *dependency* (your code *depends on* NumPy).

It's quite common to have a ~few dependencies, but too many is *problematic*.

As of now my own python package (VICE) is ~90k lines of code. If one of its dependencies releases a new version with changes that affect my code, I have to go look at every file to ensure this doesn't break anything.

Advice: Don't be afraid to write the small but challenging stuff yourself.

Testing

Well-written software implements *unit tests*

- A means of assigning *Success/Failure/Skipped* messages to the smallest possible components of a code base

When you make a future change to your code base, this gives you an automated means of finding out if your change broke anything you weren't expecting to break.

This is as simple as *actually saving* the code you write to test a new component of your software, and the same for when you resolve an issue.

Think Before You *Write*

Many scientists just start writing functions when posed with a problem

- This has its place and its usefulness within research
- If you’re going to use some function >few times, I advise more thought

If you’re going to be spending even as much as five minutes writing a given file, stop and ask yourself: *What objects might be useful? What functions might be useful?*

There are many ways to write the same program, so don’t always settle for the first idea that comes into your head. *You can usually improve upon the first thing that comes to mind.*

Think Before You *Write*

This is how professional developers and software engineers view scientist-written code

Astronomers
Write Code
Make Tests
Pseudocode
Whiteboard
Idea or Problem
imgflip.com

Rule Number One

Code is read much more often than it is written.

- Advice: Assume this applies to all of the code you will ever write

Emphasize *readability* above all else. It’s okay to break any of these rules if the result is a much more readable solution.

Bottom line: *Well-written code is clear, verbose, and does not cut corners.*

- Viewing code as a mere means to an end solves zero problems while causing many. That line is usually crossed *long* before a public release, and the solution is knowing how to engineer your code.