### Code Archeology

Rohan Padhye and Michael Hilton



### Learning goals

- Understand and scope the task of taking on and understanding a new and complex piece of existing software.
- Appreciate the importance of configuring an effective IDE.
- Contrast different types of code execution environments including local, remote, application, and libraries.
- Enumerate both static and dynamic strategies for understanding and modifying a new codebase.

Context: big ole pile of code.



...do something to it.

# You cannot understand the entire system.

## Goal: develop and test a working model or set of working hypotheses about how (some part of) a system works.

- Working model: an understanding of the pieces of the system (components), and the way they interact (connections).
- It is common in practice to consult documentation, experts.
- Prior knowledge/experience is also useful (see: frameworks, architectural patterns, design patterns).
- Today, we focus on individual information gathering via observation, probes, and hypothesis testing.

# TWO PROPERTIES OF SOFTWARE THAT ARE USUALLY ANNOYING THAT WE CAN TAKE ADVANTAGE OF.

### Software constantly changes → Software is easy to change!



Guess so!



institute for SOFTWARE RESEARCH

Is this wall load-bearing?

### Software is a big redundant mess → there's always something to copy as a starting point!





Key insight in grokking unfamiliar code/apps

### **CODE MUST RUN TO DO STUFF!!**

### 1. If code must run, it must have a beginning





#### 2. If code must run, it must exist

```
DWUKU PTR [ebp+0x8],0x1
         ---- <+16>;
   0x08048416 <+18>:
                       jg
                              0x804843c <main+56>
  0x08048419 <+21>:
                       MOV
                             eax, DWORD PTR [ebp+0xc]
  0x0804841b <+23>:
                             ecx, DWORD PTR [eax]
                       Mov
 0x08048420 <+28>:
                      mov
                            edx,0x8048520
 0x08048425 <+33>:
                      MOV
                            eax,ds:0x8049648
0x08048429 <+37>:
                           DWORD PTR [esp+0x8],ecx
                     Mov
0x0804842d <+41>:
                          DWORD PTR [esp+0x4], edx
                     MOV
0x08048430 <+44>:
                          DWORD PTR [esp],eax
                    mov
7x08048435 <+49>;
                    call
                         0x8048338 <fprintf@plt>
x0804843a <+54>;
                   MOV
x0804843c <+56>;
                         eax, 0x1
                   jmp
                         0x8048459 <main+85>
0804843f <+59>;
                  MOV
                        eax, DWORD PTR [ebp+0xc]
98048442 <+62>:
                  add
8048444 <+64>;
                        eax, 0x4
                 mov
                       eax, DWORD PTR [eax]
3048448 <+68>;
                 mov
                       DWORD PTR [esp+0x4], eax
04844c <+72>:
                lea
                      eax,[esp+0x10]
94844f <+75>;
                mov
                      DWORD PTD .
48454 =100
                Cal1
```



### The Beginning: Entry Points

Some trigger that causes code to run.

- Locally installed programs: run cmd, OS launch, I/O events, etc.
- Local applications in dev: build + run, test, deploy (e.g. docker)
- Web apps server-side: Browser sends HTTP request (GET/POST)
- Web apps client-side: Browser runs JavaScript



### Code must exist. But where?

Helps to identify what's knowable and what's changeable

- Locally installed programs: run cmd, OS launch, I/O events, etc.
  - Binaries (machine code) on your computer
- Local applications in dev: build + run, test, deploy (e.g. docker)
  - Source code in repository (+ dependencies)
- Web apps server-side: Browser sends HTTP request (GET/POST)
  - Code runs remotely (you can only observe outputs)
- Web apps client-side: Browser runs JavaScript
  - Source code is downloaded and run locally (see: browser dev tools!)

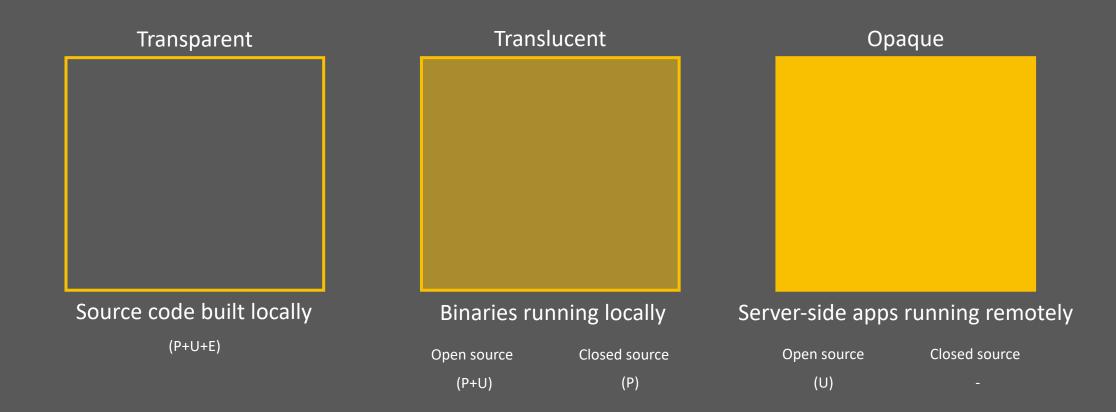


### Side note on build systems

- Basically the same across languages / platforms
  - o Make, maven, gradle, grunt, bazel, etc.
- Goal: Source code + dependencies + config -> runnables
- Common themes:
  - Dependency management (repositories, versions, etc)
  - Config management (platform-specific features, file/dir names, IP addresses, port numbers, etc)
  - Runnables (start, stop?, test)
  - Almost always have 'debug' mode and help ('-h' or similar)
  - Almost always have one or more "build" directories (= not part of source repo)



### Can running code be Probed/Understood/Edited?







Source code built locally

## CREATING A WORKING MODEL OF UNFAMILIAR CODE

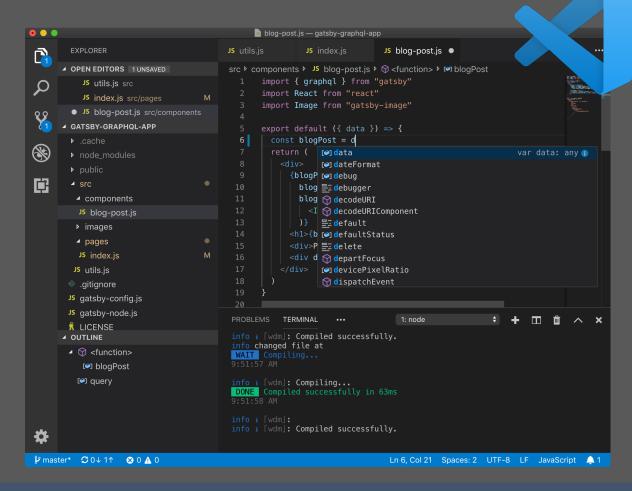
### Static (+dynamic) information gathering

- Basic needs:
  - Code/file search and navigation
  - Code editing (probes)
  - Execution of code, tests
  - Observation of output (observation)
- Many choices here on tools! Depends on circumstance.
  - o grep/find/etc. Having a command on Unix tools is invaluable
  - A decent IDE
  - Debugger
  - Test frameworks + coverage reports
  - Google (or your favorite web search engine)

At the command line: **grep** and **find**! (Do a web search for tutorials)

### Static Information Gathering

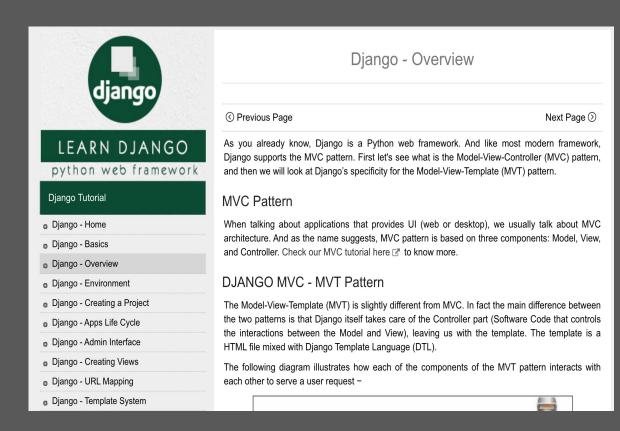
- Please configure and use a *legitimate* IDE.
  - No favorites? We recommend VSCode and IntelliJ IDEA.
- Why?
  - "search all files"
  - "jump to definition"
  - "download dependency source"
- Remember: real software is too complicated to keep in your head.





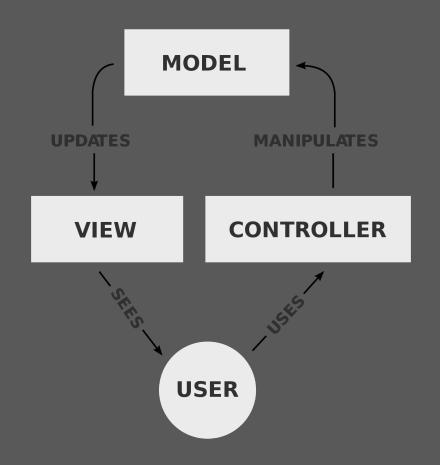
### Consider documentation/tutorials judiciously

- Great for discovering entry points!
- Can teach you about general structure, architecture.
  - Forward-reference to architectural patterns!
- As you gain experience, you will recognize more of these, and you will immediately know something about how the program works.
- For example, next time you work on a webapp...





### Consider documentation/tutorials judiciously





#### Django - Overview

O Previous Page

As you already know, Django is a Python web framework. And like most modern framework, Django supports the MVC pattern. First let's see what is the Model-View-Controller (MVC) pattern, and then we will look at Django's specificity for the Model-View-Template (MVT) pattern.

#### MVC Pattern

When talking about applications that provides UI (web or desktop), we usually talk about MVC architecture. And as the name suggests, MVC pattern is based on three components: Model, View, and Controller. Check our MVC tutorial here 🗗 to know more.

#### DJANGO MVC - MVT Pattern

The Model-View-Template (MVT) is slightly different from MVC. In fact the main difference between the two patterns is that Django itself takes care of the Controller part (Software Code that controls the interactions between the Model and View), leaving us with the template. The template is a HTML file mixed with Django Template Language (DTL).

The following diagram illustrates how each of the components of the MVT pattern interacts with each other to serve a user request –



### Dynamic Information Gathering

- Key principle 1: change is a useful primitive to inform mental models about a software system.
- Key principle 2: systems almost always provide some kind of starting point.
- Put simply:
  - 1. Build it.
  - 2. Run it.
  - 3. Change it.
  - 4. Run it again.
- Can provide information both bottom up or top down, depending on the situation.



## Probes - Observe, control or "lightly" manipulate execution

- Printf("here")
- Turning on automatic debug info logging
- Breakpoints
- Sophisiticated debugging tools
  - Breakpoint, eval, step through / step over
  - (Some tools even support remote debugging)
- Delete debugging (equivalent of `kill -9`)

### Step 0: sanity check basic model + hypotheses.

- Confirm that you can build and run the code.
  - o Ideally *both* using the tests provided, *and* by hand.
- Confirm that the code you are running is the code you built.
- Confirm that you can make an externally visible change.
- How? Where? Starting points:
  - Run an existing test, change it.
  - Write a new test.
  - Change the code, write or rerun a test that should notice the change.
- Make sure the changes persist if you want them to.
  - Distinguish between source repository and build/deploy directories.



### Demonstration: Live Coding

