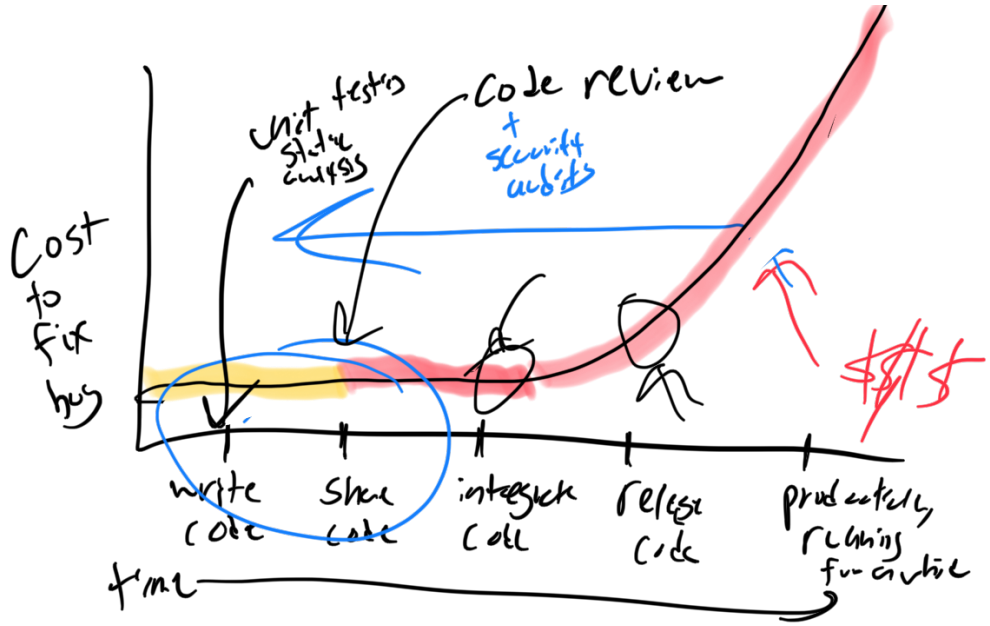
3-22: Continuous Development

Agenda:

- HW4 poll and discussion - https://pollev.com/jbell
  - Q: what about 1 week each for HW1, HW2, 3 weeks for HW3, HW4 2 weeks, but now finishes 1 week sooner?
    - A: Overwhelming "No" - we like 2 week deadlines
      - 1 week for HW1, 2 weeks for rest - finish HW4 1 week sooner
    - Socket part of HW3 needed more guidance
    - More examples with mocks and spies
      - Activities in class with mocks + spies?
- Titus Winters talk recap + discussion
  - Was there a recording? - Prof Bell will check
  - Q: Knowledge sharing - should it be the case that "everyone can resolve an issue" - how does this reconcile with "frontend people and backend people"
    - Balancing redundancy vs specialization
- Discussion - motivations for continuous development
  -
- Project notes + discussion
- Team meetings + activity - setting up CI/CD pipelines



Q: Why is it more expensive to find a bug later in development process?

PR cost

Human cost of context switching to figure out and understand the code

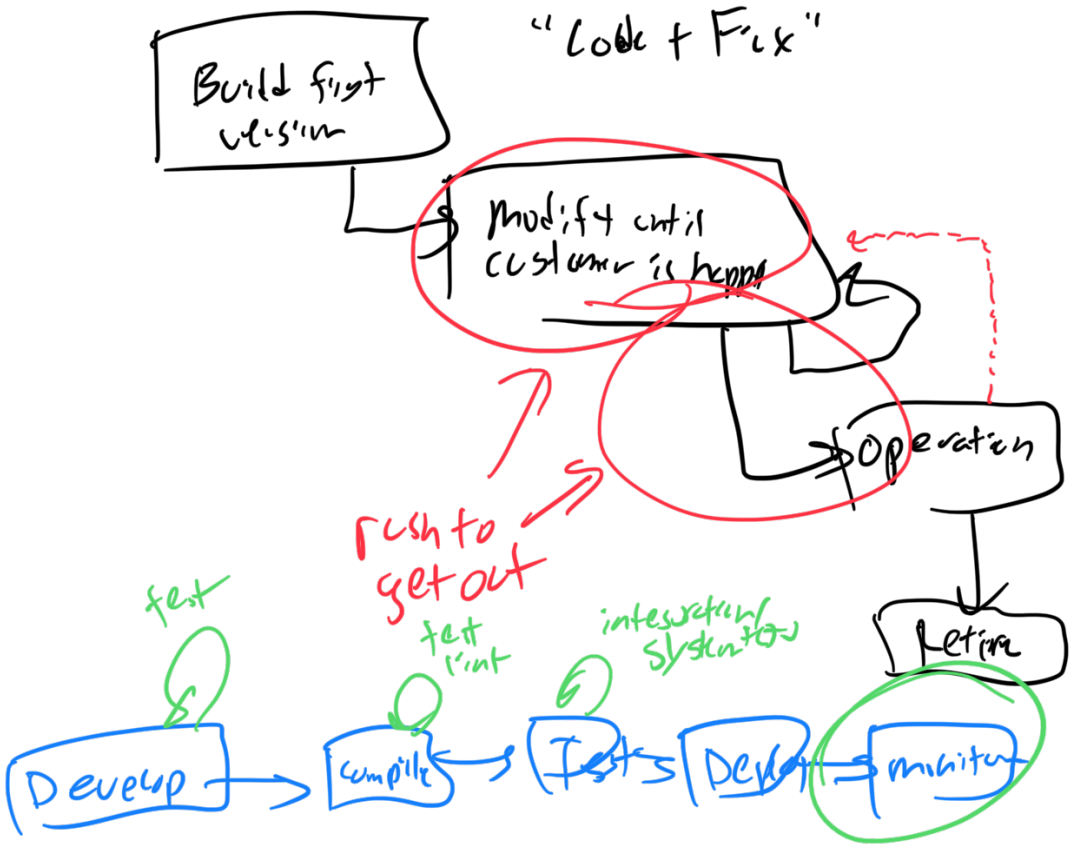
Restructuring of other code if that depends on it

3/24: Continuous Development II
Agenda:

- Project Discussion
  - Final deliverables information posted: https://neu-se.github.io/CS4530-CS5500-Spring-2021/assignments/project-deliverable
  - Demo due on 16th, not 15th
  - No class on Apr 12, Apr 15 will be work on projects + demos if someone wants to show
- Continuous development discussion
  - Q: How have you deployed applications before?
    - Personal website -> GitHub Pages (note - class website is like this!) - “works magically”
    - AWS Code Pipeline -> Automatically deploy to cloud, works magically
    - Vercel -> Like GitHub pages, integrates multiple repositories + runs react build script to create a deployment, integrates pull-requests with GitHub
    - Copy files to a server?
      - Mostly crickets, but some “yes”
    - Jenkins pipelines
  - Q: What are these different deployment infrastructures based on?
    - Use a “bare metal” machine
      - Good: Control - I can do whatever I want, including of resources
      - Bad (Not automated): Not efficient to manually copy files to a server each time, maybe have a manual process to trigger a deployment. Systems administrators need to handle this
      - Bad (Resource utilization) - Difficulty of having multi-tenancy. Can’t enforce resource limitations.

        - Bad (Latency to set up a new machine) - Probably on the order of weeks, certainly hours
    - Use a virtual machine running on bare metal (Early 2000's) - VMWare/VirtualBox/EC2/GCP
        - Good: Can limit resources per-application (limit CPU + RAM)
        - Bad: Hogs resources/compute power - there is some overhead (running an OS)! Run Windows VM on Linux machine, run Ubuntu VM on a Gentoo machine
    - Use containers (Getting closer to 2010) - Docker
        - Good: Can limit resource per-application (CPU + RAM)
        - Good: Don't run an OS in each container
        - Good/bad: Same OS as bare-metal machine
    - Orchestrate these deployments using a process like Kubernetes
        - Declaratively specify what we need: "3 docker containers running TownsService, with a load balancer in front of them, and spin up more TownsService if load is high"
        - DevOps Engineers/Site Reliability Engineer
- Q: How has development of these computing infrastructures enabled new ways or generally changed how we build software?
    - Continuous deployment has become the norm
    - "Democratizes" building software - don't need as much specialized knowledge just to get something up and running
        - Platform as a service (Heroku/Netlify) - enables access to these resources without training
        - Provide this benefit not only to the kinds of developers who would have built large-scale software already - but to everyone!
    - Easier to scale - build on common available abstractions
    - Increases performance (latency) - Speed of light can dominate the time to server a request
        - Easy way to say "Replicate across the world" - take advantage of Content Delivery Network (CDNs) that locate our services close to users
    - Not all glory: When services go down, out of luck
        - Not just one administrator -> this is good AND bad. Classic examples: Amazon Northern Virginia outage(s)
    - Not all glory: Difficult to monitor!
    - Not all glory: Doesn't necessarily eliminate having a gate keeper to determine when/what to deploy
    - Can find bugs sooner - deploy faster, feedback faster
- Q: Does continuous delivery mean that features are released fast?

- Things are “in the pipeline” faster - releases happen often, but still probably takes at least a week from code being written it is deployed
-



Q: What kinds of things can we monitor from deployment?
A: KPI’s (key performance indicators) - business metrics, like: if you put out a new feature, how many users actually interact with that? If the feature is to make a process better/faster, does it now take less time for a user to complete that process?
A: Technical metrics (Response time, resource utilization, CPU load, error rate)