

# INTRODUCTION TO SOFTWARE DEVELOPMENT

## **Week 4 Day 3**

Led by: Emily Crose  
for  
Oakland University

# PREVIOUS SESSION REVIEW



**QUESTIONS FROM PREVIOUS  
SESSION?**







Disney Photo Snapper







[Yesterland.com](http://Yesterland.com)









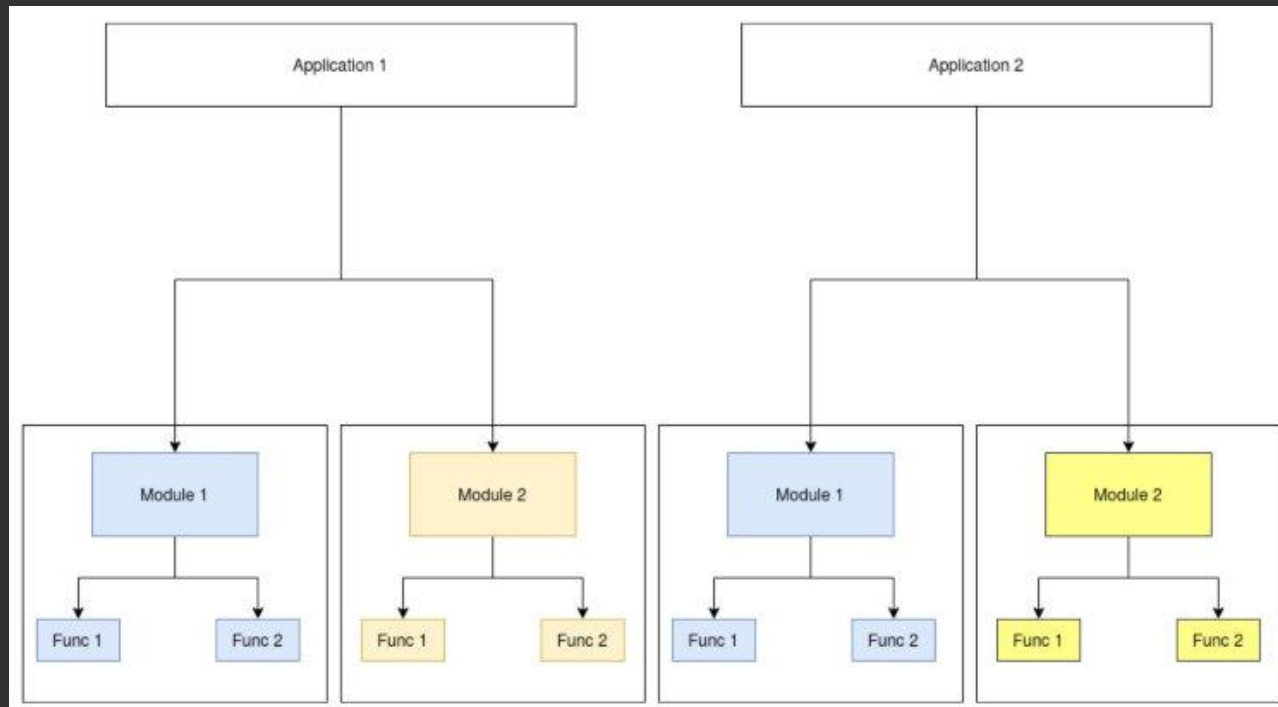
# MODULAR CODING

# MODULARITY DEFINITION

- the degree to which a system's components may be separated and recombined, often with the benefit of flexibility and variety in use.



# HOW TO PROGRAM MODULARLY

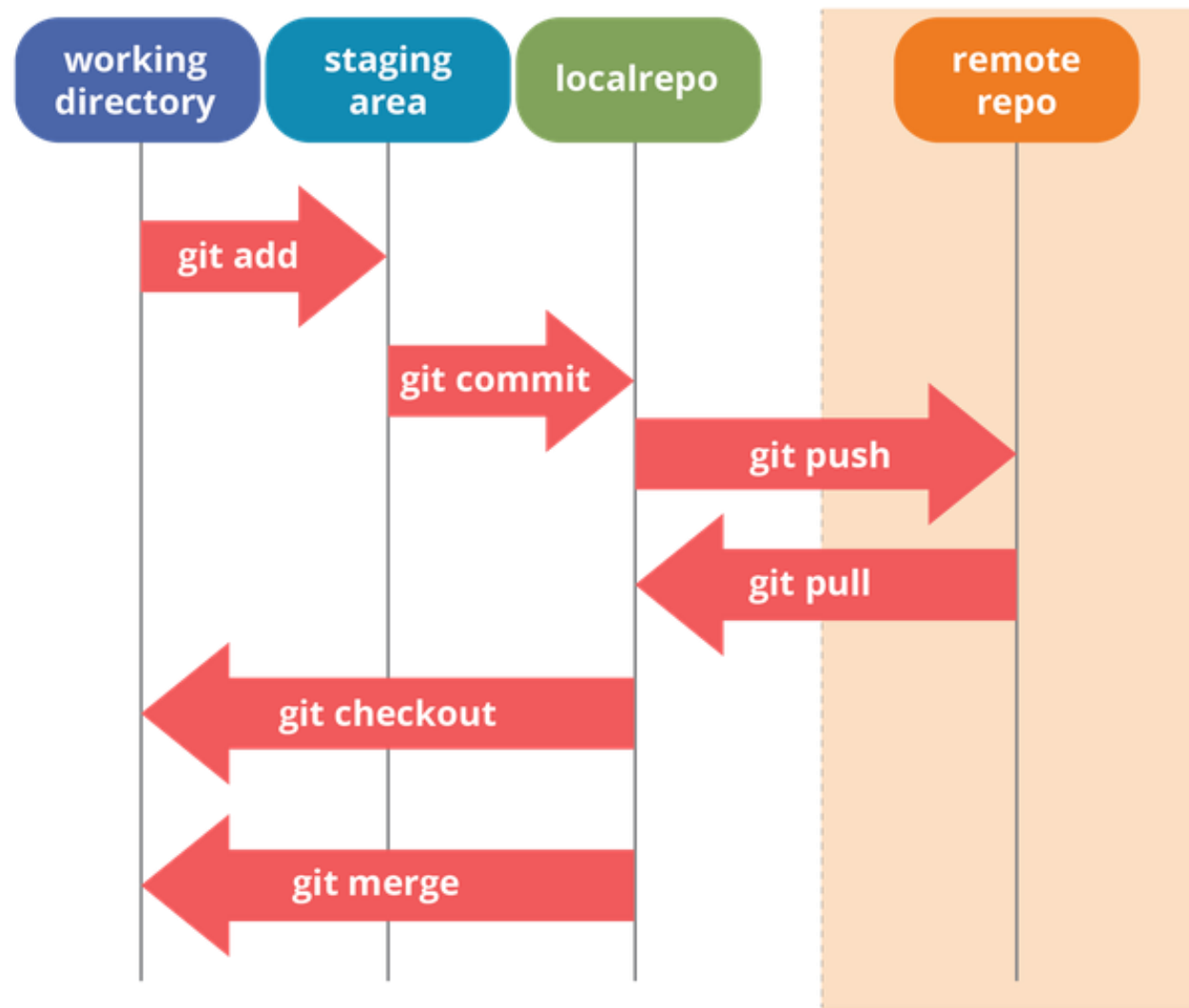


- Equivalent Coding Standards
- Standard APIs
- Reliable frameworks
- Designing for testability

# WHY PROGRAM MODULARLY?

- Plug & Play code
- Cheaper
- Easier to build





# WORKFLOW REVISITED

The background features a dark gray field with a repeating pattern of light gray, stylized mountain peaks or triangles. A horizontal white band runs across the middle of the image. In the center of this band is a faint, light gray circle. The title text is centered within this white band.

# BRANCHING WITH TEAMS



# BRANCH NAMING CONVENTIONS

- What type of branch naming convention works for you?
- [Username] - <feature>
- [Jira ticket #] - <feature>
- [github issue number] – {username} - <feature>



PULL



# FORKING



# EDITING A FORKED REPOSITORY

PULL REQUEST

# CODE REVIEWS



# CODE REVIEW

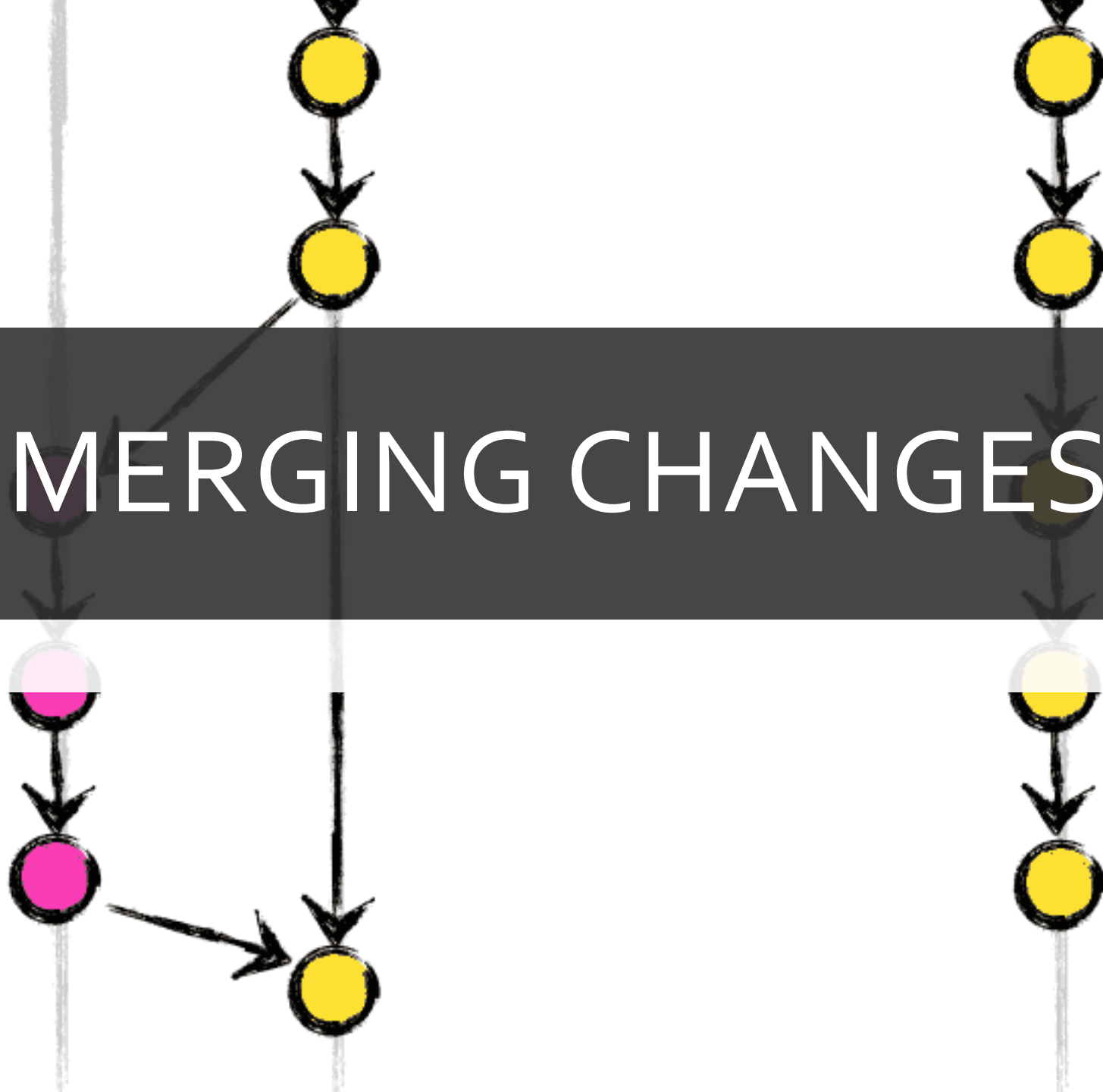
- Allows changes to be approved
- Limits bad code merges
- Adds a layer of safety

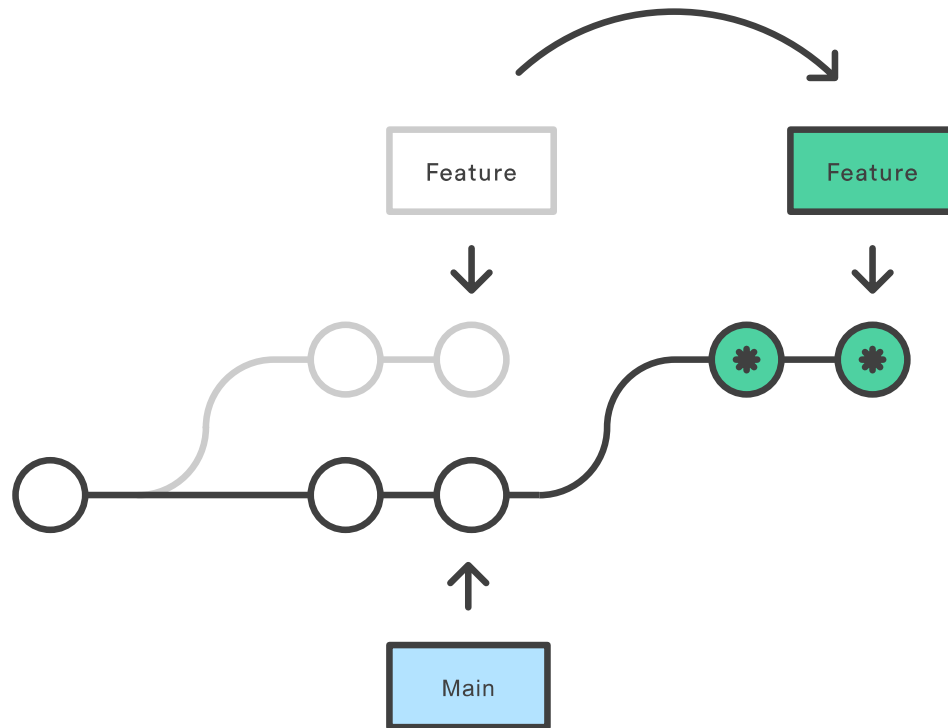
```
or object to mirror
mirror_mod.mirror_object = ob
operation == "MIRROR_X":
    True
    False
    False
operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True
selection at the end -add
modifier.select=1
context.scene.objects.active
("Selected" + str(modifier
mirror_ob.select = 0
bpy.context.selected_obj
data.objects[one.name].select
print("please select exactly
-- OPERATOR CLASSES -----
types.Operator):
    on X mirror to the selected
    object.mirror_mirror_x"
    mirror X"
```

# MERGING CHANGES

feature

feature





\* Brand New Commits

# REBASING



The background of the image is a dense crowd of people, represented by stylized human figures. The figures are mostly in shades of blue and purple, with some in the foreground appearing more distinct. A central figure is highlighted in white, standing out from the crowd. The overall image has a blurred, bokeh-like quality, suggesting a large gathering of people.

# RELEASE CANDIDATES



# BUILDING BINARIES



# RELEASE NOTES

# PRACTICING WITH GIT

- <https://learngitbranching.js.org/>
- Alternatives
  - [https://gitimmersion.com/lab\\_o1.html](https://gitimmersion.com/lab_o1.html)





10 MINUTE BREAK



# LICENSING

# LICENSING



- Software can be copyrighted
- How do we protect our “Intellectual Property”?

# BROAD CATEGORIES OF SOFTWARE LICENSING

- Weak Copyleft
- Copyleft
- Commercial or Proprietary
- Dual
- Public Domain



# PERMISSIVE

- Permissive
  - Minimally restrictive on modifications or redistribution
  - Typically, only require acknowledgements







# WEAK COPYLEFT

- Allows linking to open-source libraries
- Minimal requirements
- Modifying the library is more complicated than permissive
- Examples:
  - Gnu Lesser General Public License (GLPL)
  - Mozilla Public License (MPL)
  - CDDL
  - Eclipse



# COPYLEFT

- A.K.A. “Reciprocal” licenses or “Restrictive” licenses
- Not as commercially friendly
- Requires publication of source code for derivative works
  - Not good for commercial products!
- Examples:
  - Gnu Public License (GPL)





# MIT

- Allows for unlimited copy, modify, merge, publish, distribution, sublicense and sale
- Designed to encourage software innovation
- Does NOT provide any warranty, and excludes all liability on the author
- This is my favorite type of license.

# COMMERCIAL/PROPRIETARY

- Most restrictive
- Typically used by corporations
- Typically closed source



# DUAL

- Differs for different types of users
- Examples:
  - Server-Side Public License (SSPL)





# PUBLIC DOMAIN

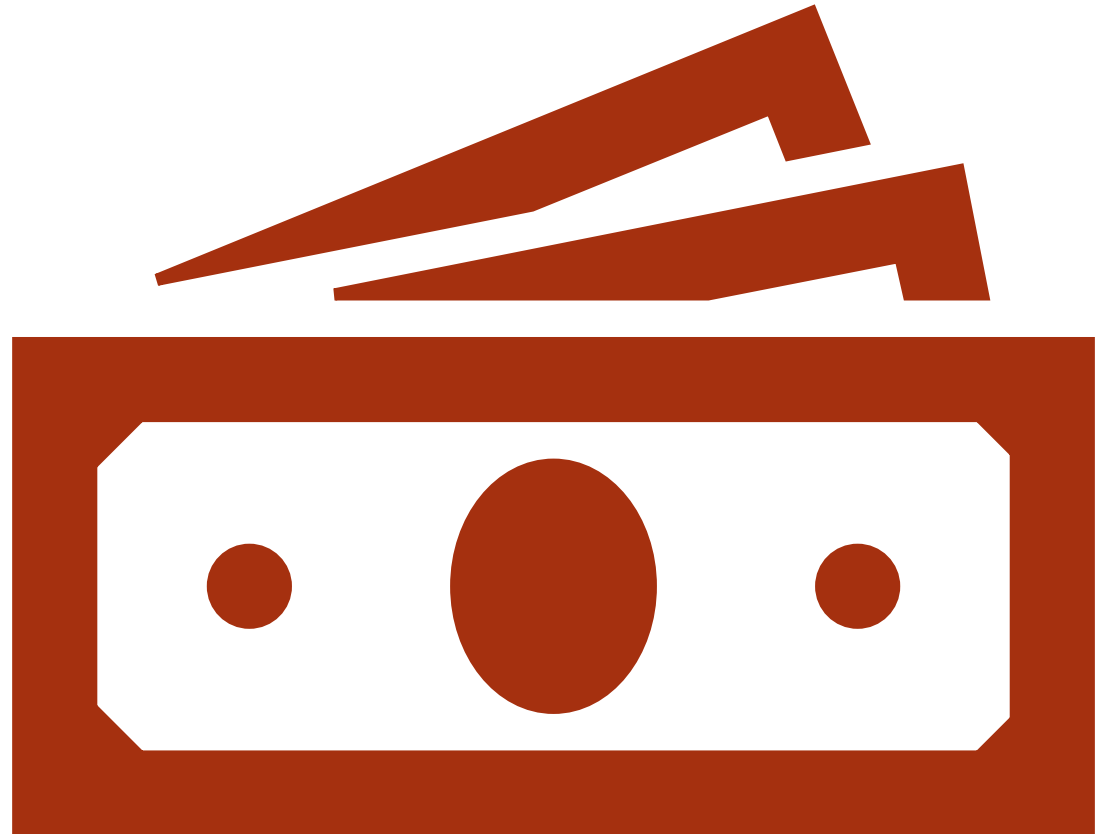
- Copyright protections do not apply
- Hard to find software in this category
- Most permissive



# UNLICENSED

Can be complicated

Can be legally risk





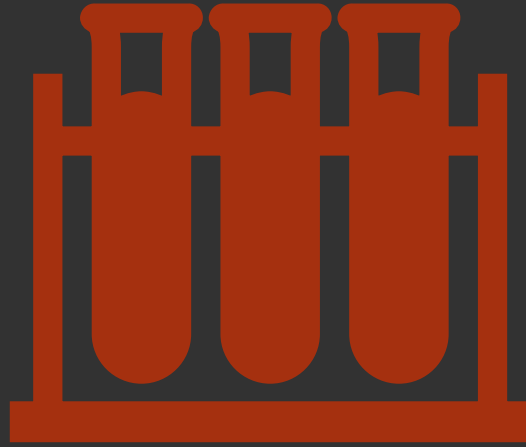
# LICENSING CONCERNS



10 MINUTE BREAK

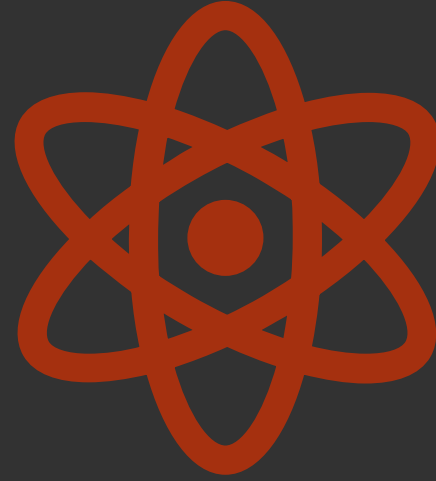


# ARCHITECTURE DESIGN



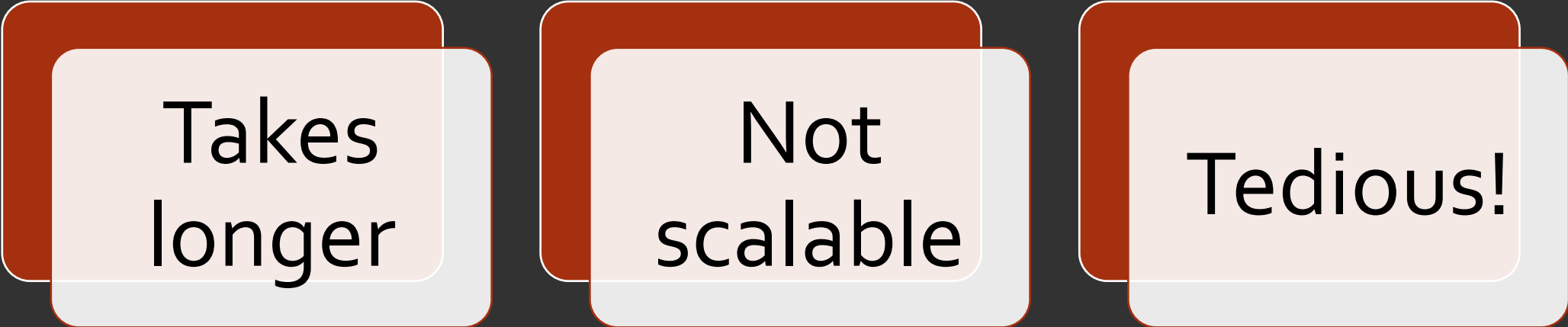
# FUNCTIONAL TESTING





WHAT IS THE PURPOSE OF  
TESTING?

# MANUAL TESTING?



Takes  
longer

Not  
scalable

Tedious!

# AUTOMATED TESTING

**What does our testing pipeline look like from an architecture perspective?**

**What types of tests do we need to perform?**

- Regression testing
- QA testing
- User Acceptance Testing
- Load testing
- Performance Testing
- Security Testing

# TESTING IN DEPTH



LOAD TESTING



STRESS  
TESTING



PERFORMANCE  
TESTING



SECURITY  
TESTING

# UNIT TESTING

- We added a feature!
  - Does it do what it's supposed to do?
- This should be done for every new feature!

# LOAD TESTING



Average number of concurrent users in the tool at a time



Beneath “peak” use



This is the REGULAR amount of traffic you expect to get



Does our app scale with more users? We can test for this!



# STRESS TESTING

- ABOVE average use
- Uncommon amount of traffic
- Can we load balance?

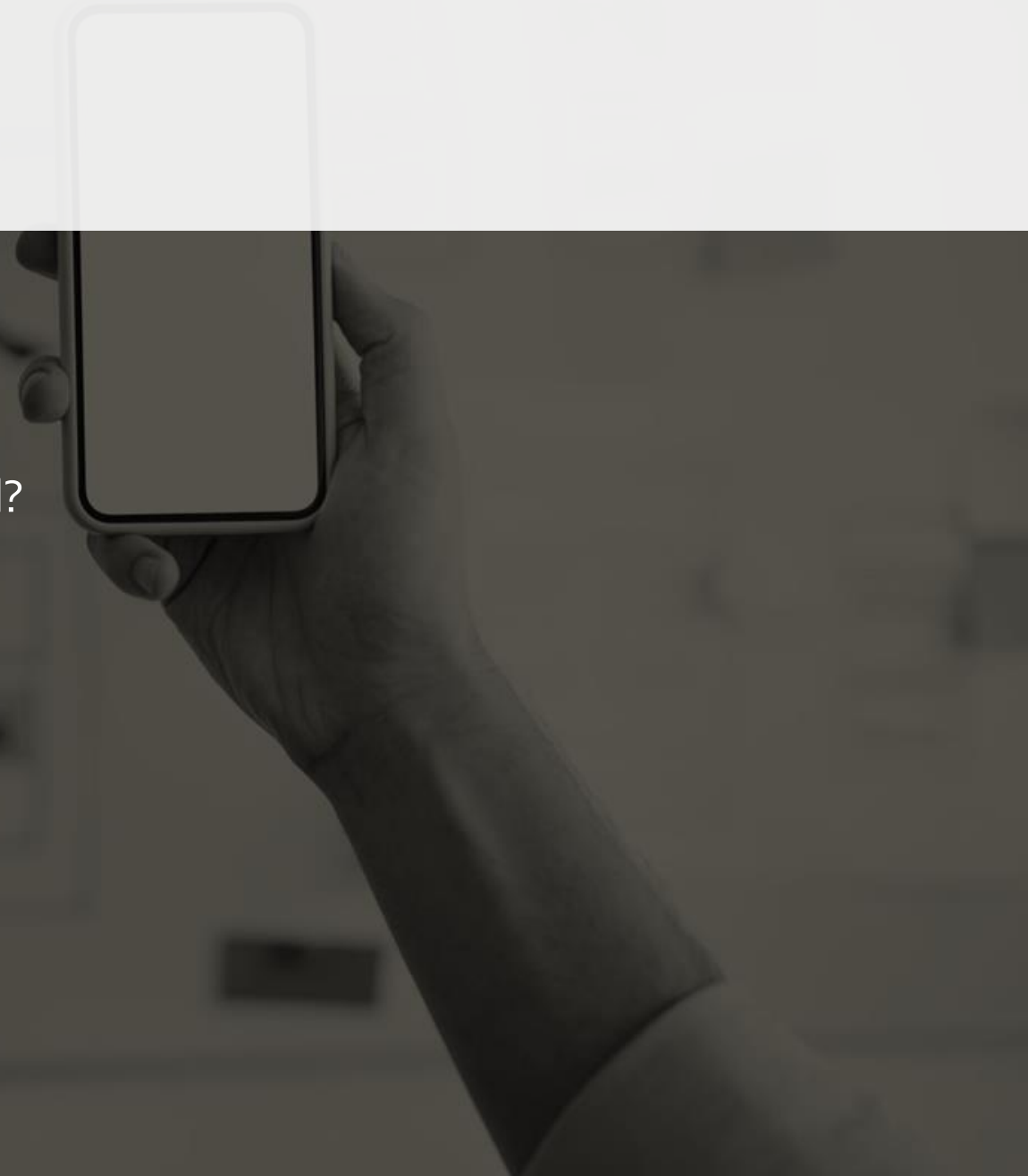


# PERFORMANCE TESTING

- How quickly can we load information?
- Do we see any lags in transfer under regular use?
  - Lags in higher-than-average use?

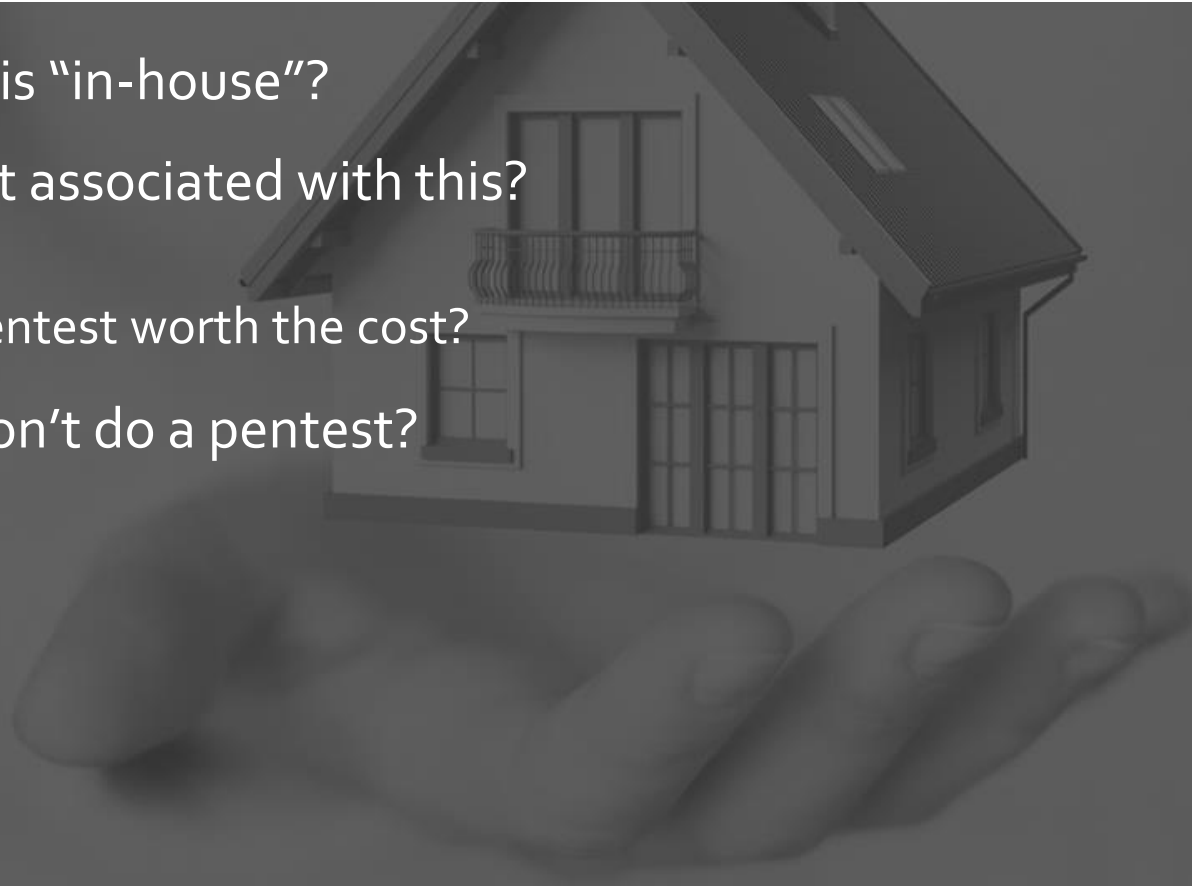
# SECURITY TESTING

- How can we break this app?
  - Can we get access we shouldn't have?
  - Can we get information we shouldn't have?
  - Can we make the app do something unexpected?
- Do we need to fully pentest this app?



# PENTESTING?

- Can we do this “in-house”?
- Is there a cost associated with this?
  - Probably!
  - Is doing a pentest worth the cost?
- What if we don’t do a pentest?



# PRODUCT PENTESTING OPTIONS

- In-house penetration test (pentest)
  - Do we have the expertise to do this in-house?
- Third-party testing
  - Is there a legal or regulatory reason we should do a third-party pentest?

# ARCHITECTURE PENTESTING

- Should we pentest our back-end architecture?
  - Is there a cost associated with this?
    - Probably!

# OUTCOMES FROM TESTING

- Bug tickets
- Feature Requests
- Reporting
  - We can show this to regulators!



# SESSION REVIEW

QUESTION OR  
CLARIFICATIONS?



SEE YOU NEXT  
TIME!