Software Design & Architecture

Continuous Integration & Release Engineering

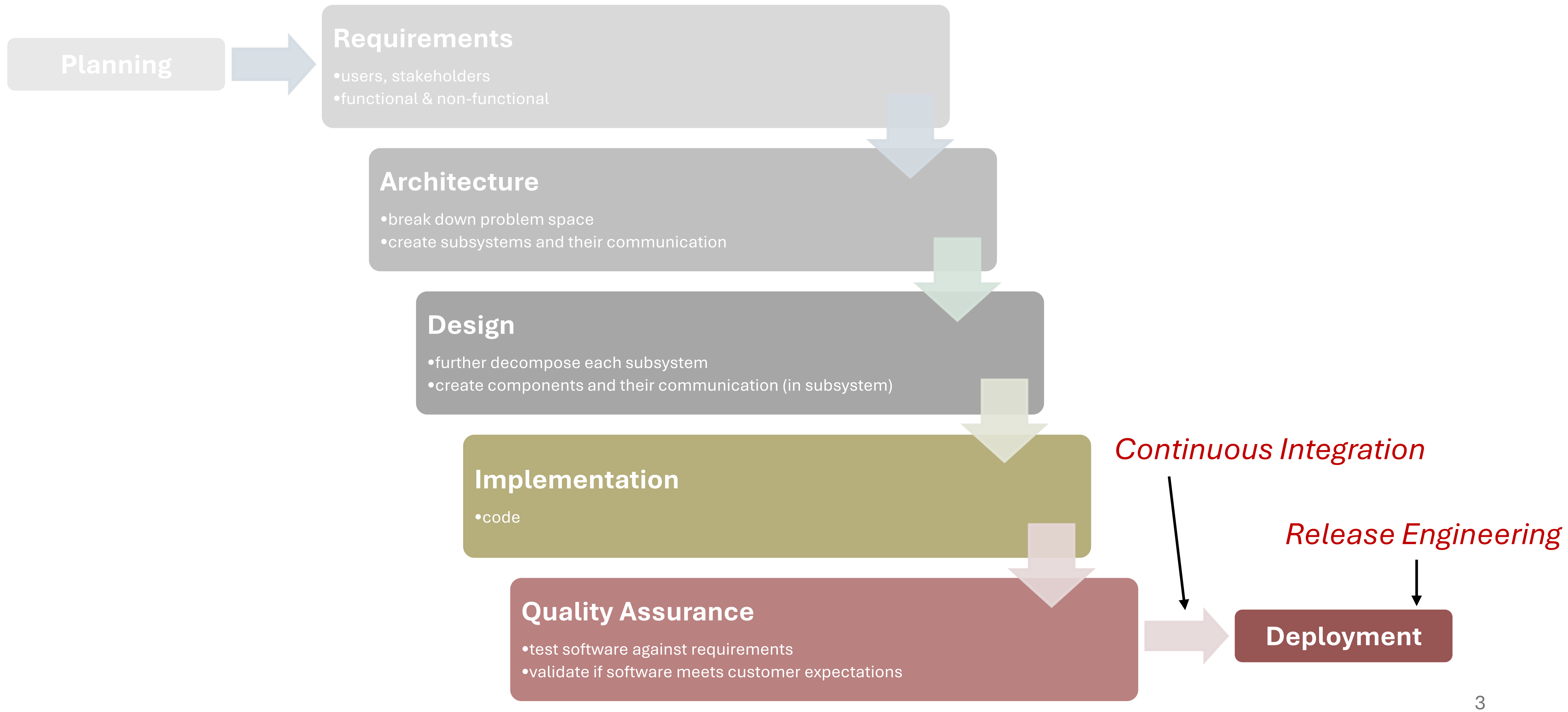
Pengyu Nie

Acknowledgements: slides adapted from previous versions by Mei Nagappan and Shane McIntosh, which are adapted from previous versions by Zhen Ming Jiang, Ahmed E. Hassan, Reid Holmes.

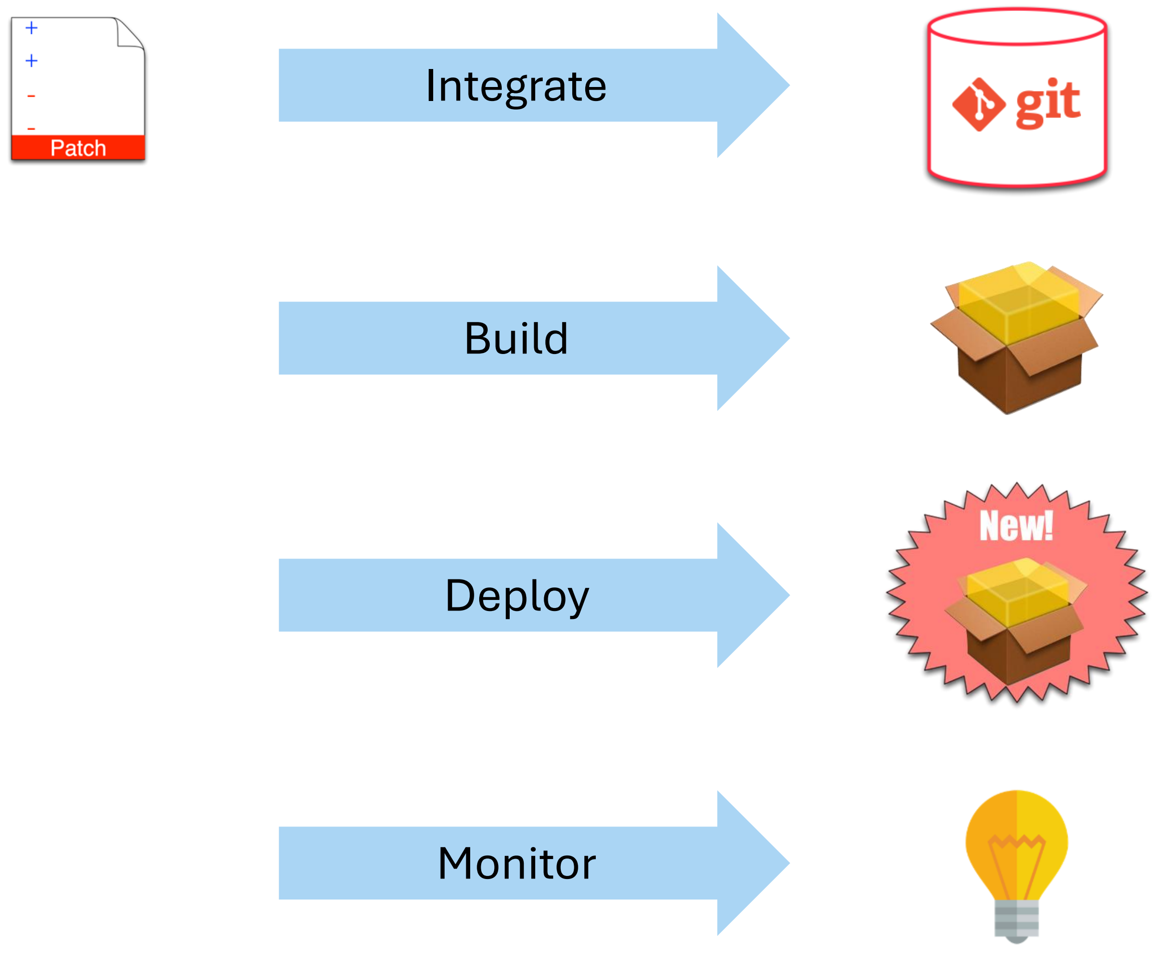
Agenda

- Release pipeline
- Continuous integration
- Release engineering
  - Green-blue deployment
  - Canary releases

Revisiting Software Development Lifecycle

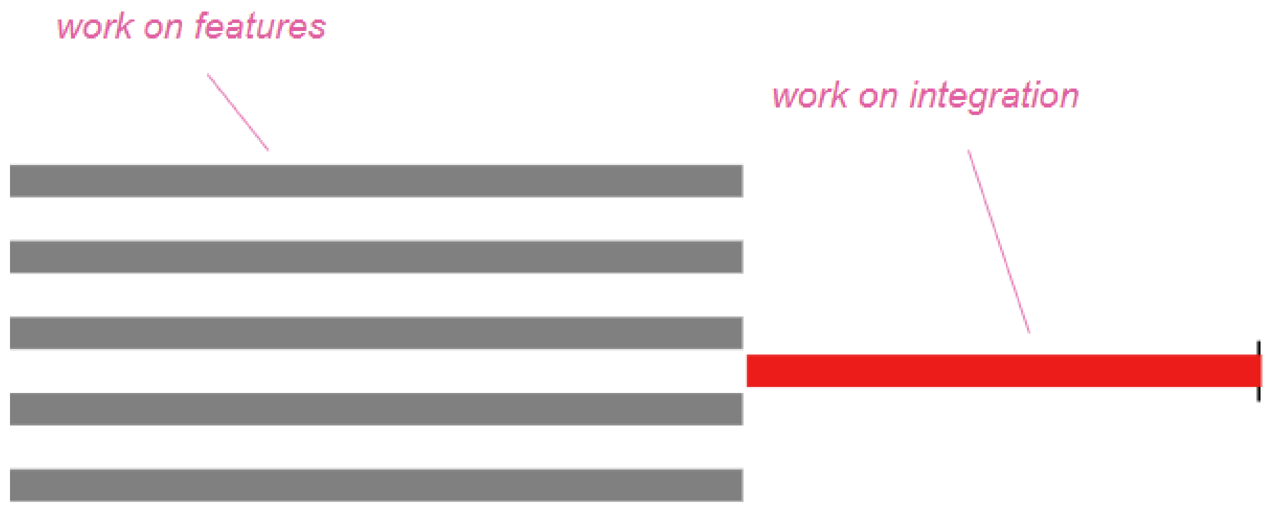

Planning
Requirements
•users, stakeholders
•functional & non-functional
Architecture
•break down problem space
•create subsystems and their communication
Design
•further decompose each subsystem
•create components and their communication (in subsystem)
Implementation
•code
Quality Assurance
•test software against requirements
•validate if software meets customer expectations
Deployment
Continuous Integration
Release Engineering

Release Pipieline



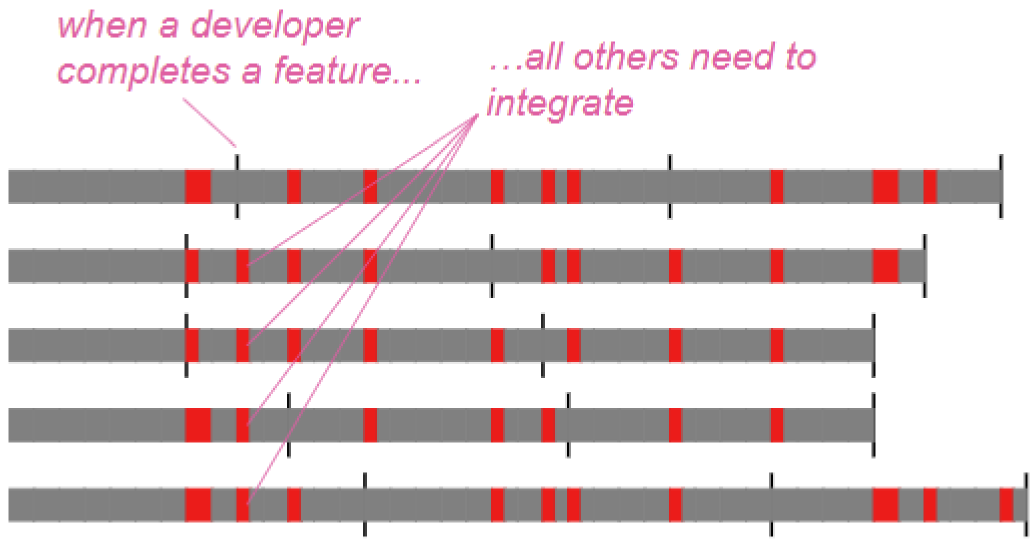
Styles of Integration (1)

- Pre-release integration
  - components are implemented and tested individually (unit tests)
  - integration happens once after all features are done (integration tests)
- The integration “phase” can become chaotic and take long time



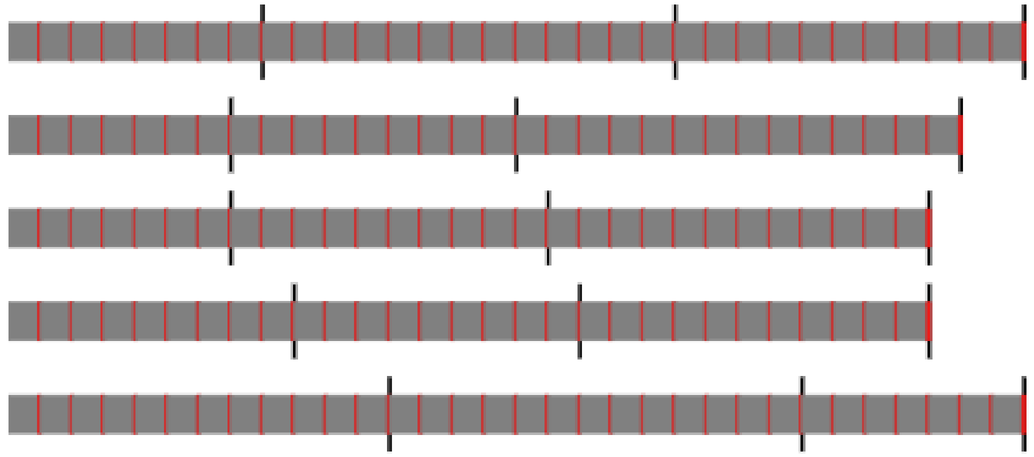
Styles of Integration (2)

- Feature branches
  - each developer pulls from mainline, implements a feature, then pushes the changes to mainline
  - integration happens more frequently (during pull and push)


when a developer completes a feature...
...all others need to integrate

Styles of Integration (3)

- Continuous integration
  - pull and push changes continuous (e.g., every day!)
  - integration happens more frequently, but each becomes easier



Continuous Integration

- Put everything in a version controlled mainline
  - Everyone pushes commits to the mainline every day
  - Everyone can see what's happening
- Automate the build
  - Include tests
  - Keep the build fast
- Every push to mainline should trigger a build
  - Fix broken builds immediately
- Automate deployment

https://martinfowler.com/articles/continuousIntegration.html

Continuous Integration – Pros and Cons

- Put everything in a version controlled mainline
  - Everyone pushes commits to the mainline every day
  - Everyone can see what's happening
- Automate the build
  - Include tests
  - Keep the build fast
- Every push to mainline should trigger a build
  - Fix broken builds immediately
- Automate deployment

+ Reduce time and effort wasted in integration

+ Less bugs

+ Refactoring becomes easier

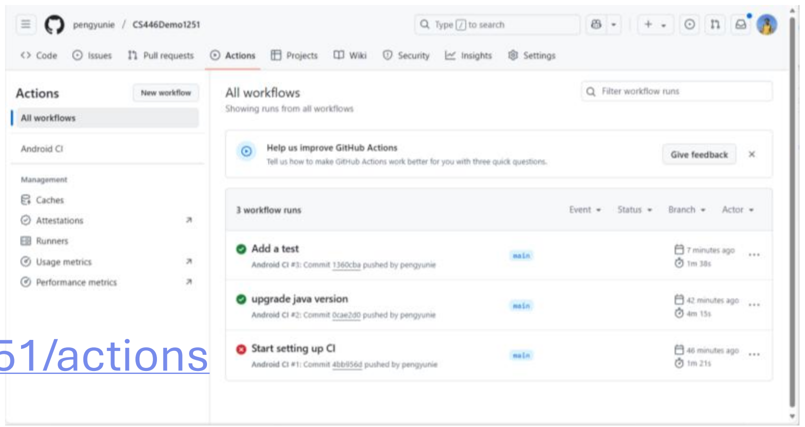
+ Release becomes easier

- You need to be committed to the project (more suitable for industry projects, less suitable for open-source projects)

- Automation is the key

Setup Continuous Integration

- Put everything in a version controlled mainline
  - Everyone pushes commits to the mainline every day
  - Everyone can see what's happening
- Automate the build
  - Include tests
  - Keep the build fast
- Every push to mainline should trigger a build
  - Fix broken builds immediately
- Automate deployment



All workflows
Showing runs from all workflows

Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.

3 workflow runs

Add a test
Android CI #3: Commit 1360cba pushed by pengyunie — main — 7 minutes ago — 1m 38s

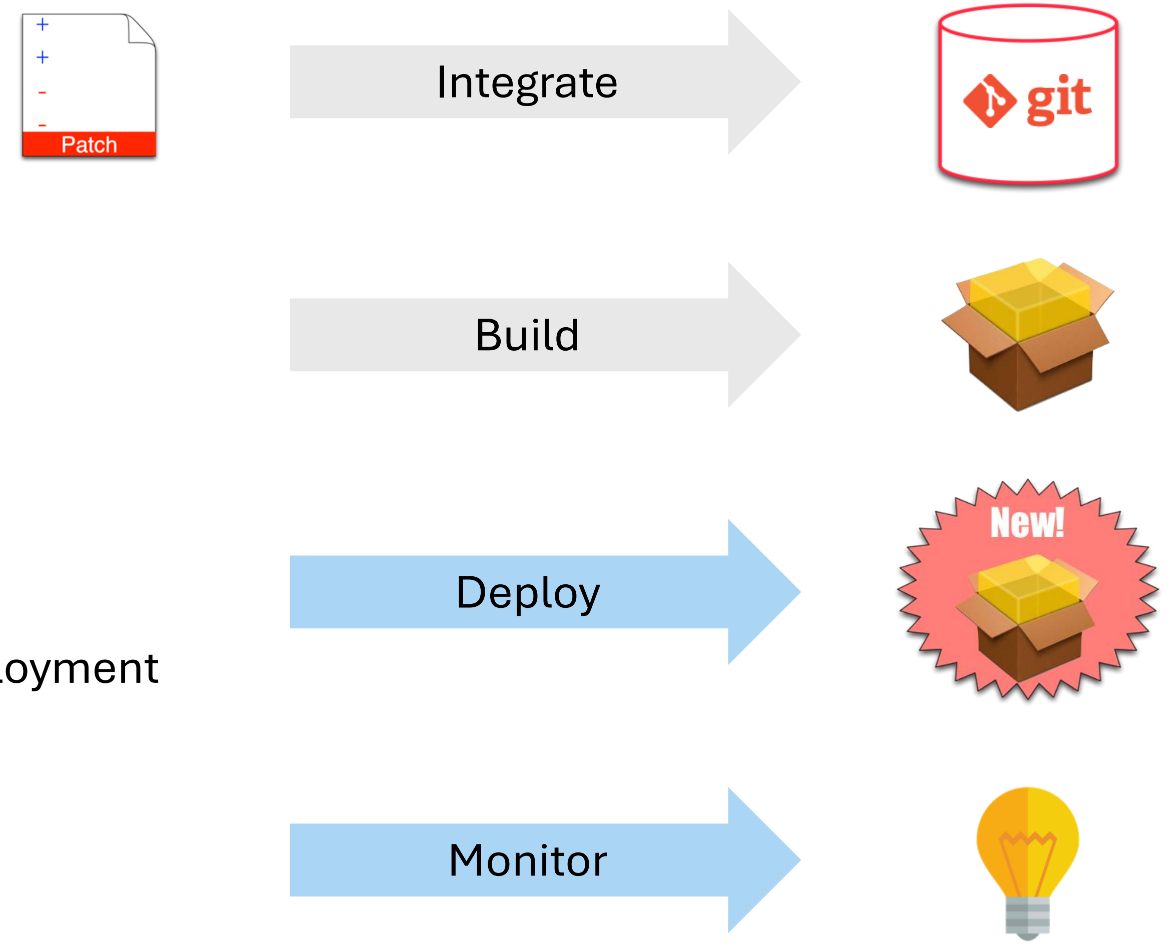
upgrade java version
Android CI #2: Commit 0cae2d0 pushed by pengyunie — main — 42 minutes ago — 4m 15s

Start setting up CI
Android CI #1: Commit 4bb956d pushed by pengyunie — main — 46 minutes ago — 1m 21s

(simple) demo: https://github.com/pengyunie/CS446Demo1251/actions
more examples:

- https://github.com/amirisback/automated-build-android-app-with-github-action (including automated deployment)
- https://github.com/topics/android-ci

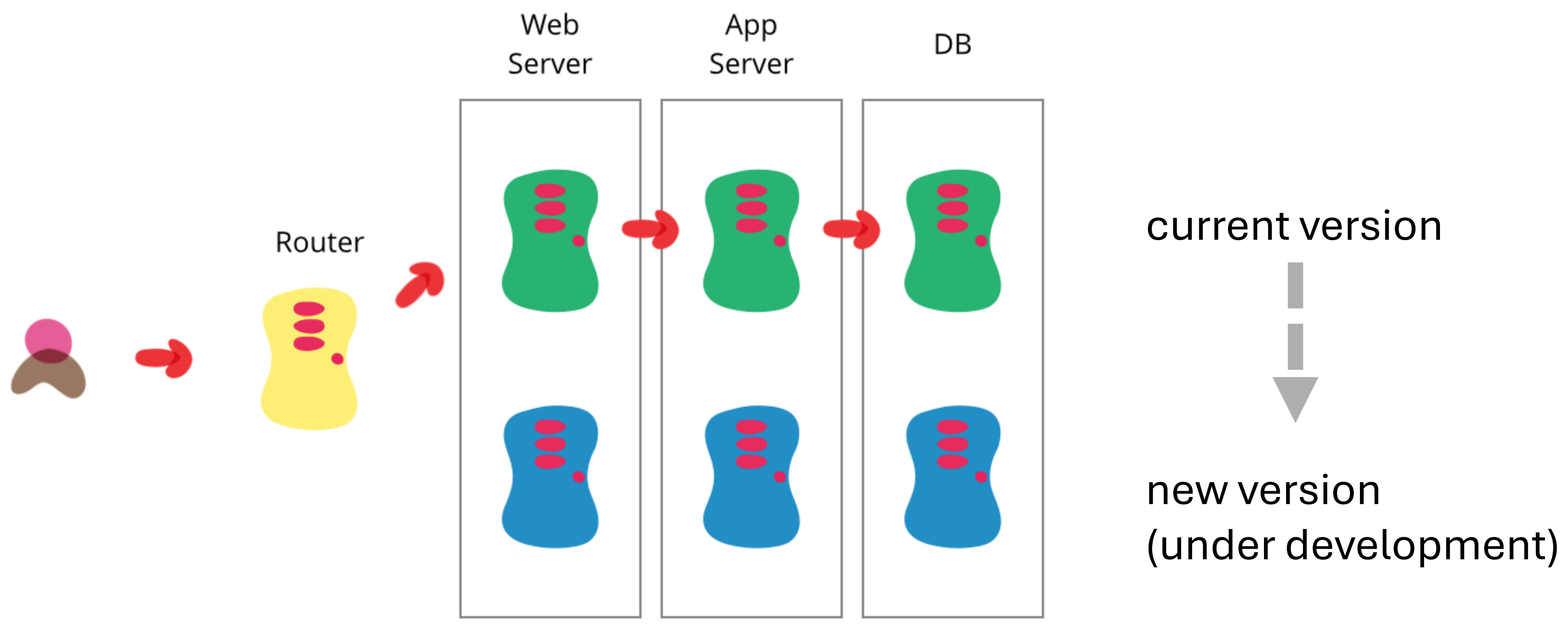
Release Pipieline (Part 2)



- Green-blue deployment
- Canary release

Blue-Green Deployment

- Challenge: downtime during "cut over"
  - When a release candidate is promoted from testing to production environments
  - Bring servers down and update them? Too costly!

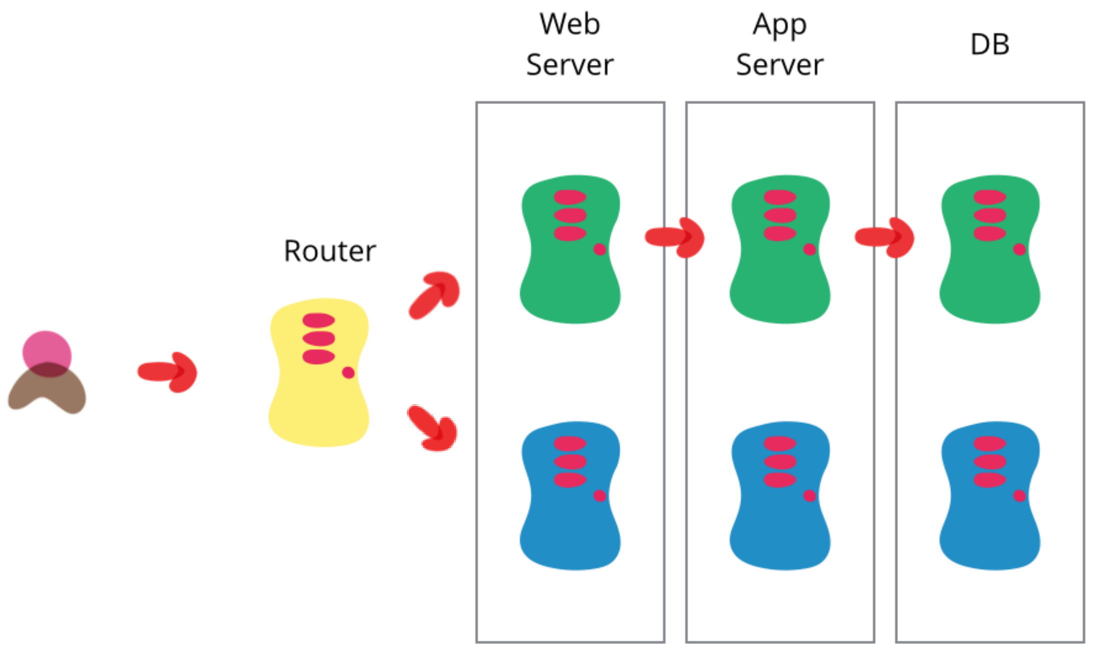


Disaster in Deployment

- Disasters: catastrophic failure of hardware/software components needed to deliver a service
- Two schools of thought about how to deal with disasters
  - Disaster prevention: design and deploy systems in a way that disasters cannot happen
  - Disaster readiness: design and deploy systems in a way that should a disaster occur, the system can quickly (and automatically) recover

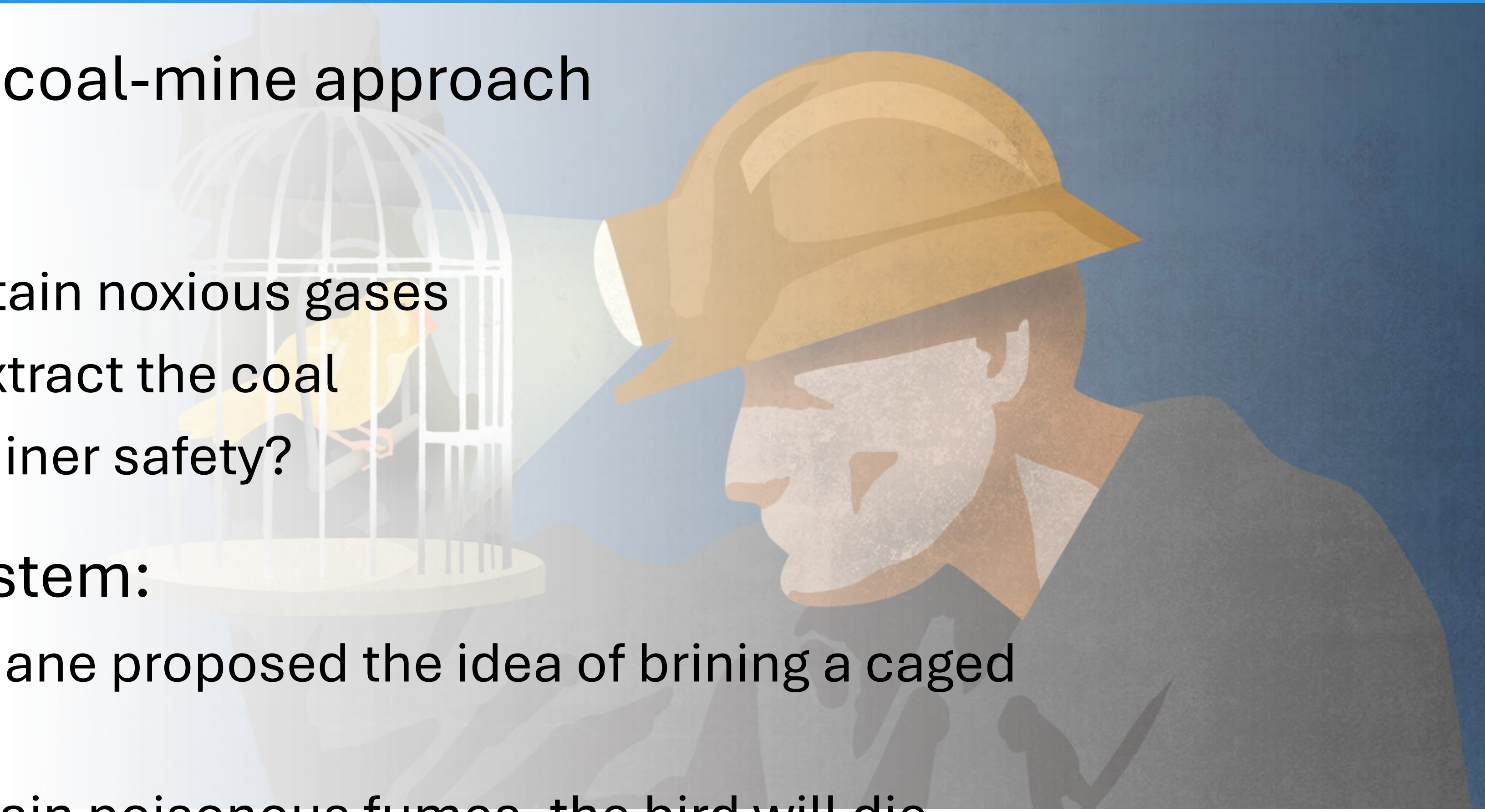
disaster recovery

Blue-Green Deployment as Disaster Recovery Plan



- Send requests to both blue and green deployments during the cut over
- If the new environment fails, the previously running environment can resume the operation seemlessly

Canary Release



- Origin: canary-in-the-coal-mine approach
- The problem:
  - coal mines often contain noxious gases
  - miners still need to extract the coal
  - how can we ensure miner safety?
- An “early warning” system:
  - Physiologist J. S. Haldane proposed the idea of brining a caged bird into the mine
  - Should the mine contain poisonous fumes, the bird will die, giving the miners some time to escape

Canary Release

- Applying the approach to software deployment...
- The problem:
  - each software release introduces some risk
  - how can we minimize the risk of deploying broken releases to a large userbase?
- The solution:
  - canary releases!
  - if the canary dies, flee the scene!

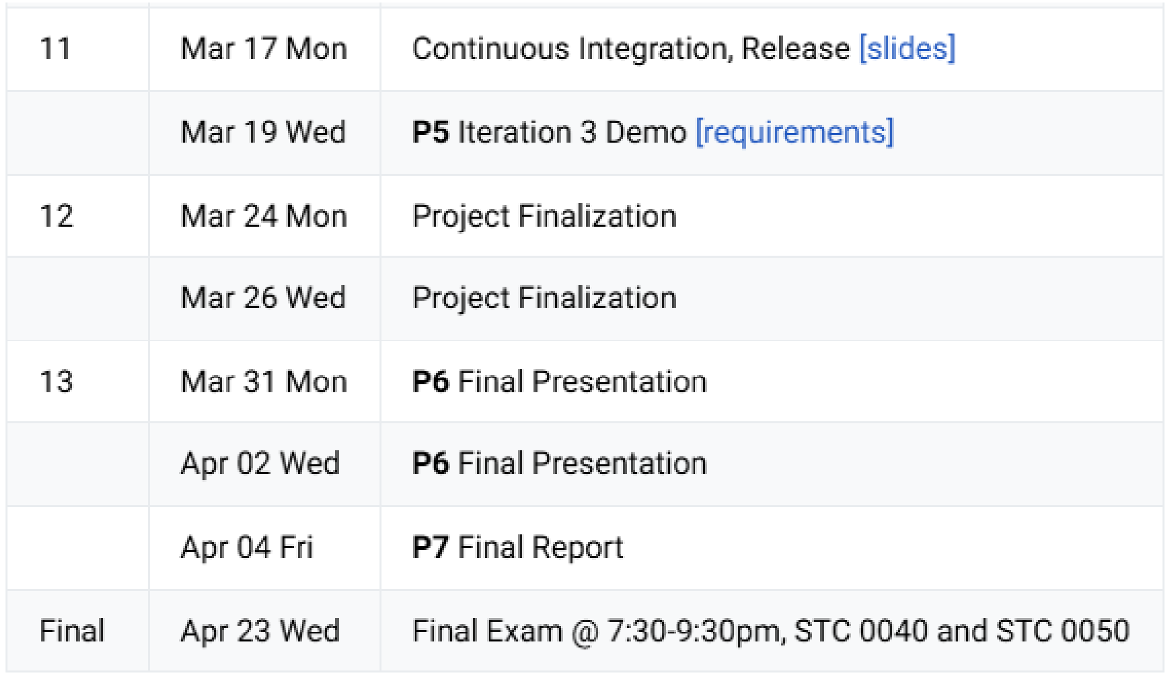
Canary Release

- Partial, time-limited deployment of a change in a service
- Followed by an evaluation of the safety of the changed service
- Production may then:
  - roll forward (to a bigger population)
  - roll backwards (undo the change)
  - alert an operation (e.g., email)
- Called “staged rollouts” on Google Play

Agenda (recap)

- Release pipeline
- Continuous integration
- Release engineering
  - Green-blue deployment
  - Canary releases

Plan for the next few weeks (end of term!)



11	Mar 17 Mon	Continuous Integration, Release [slides]
	Mar 19 Wed	**P5** Iteration 3 Demo [requirements]
12	Mar 24 Mon	Project Finalization
	Mar 26 Wed	Project Finalization
13	Mar 31 Mon	**P6** Final Presentation
	Apr 02 Wed	**P6** Final Presentation
	Apr 04 Fri	**P7** Final Report
Final	Apr 23 Wed	Final Exam @ 7:30-9:30pm, STC 0040 and STC 0050

review of final presentation/report requirements

review of final exam practice questions

~Apr 11: release of project grades

- one page cheat sheet allowed
- assigned seats

~Apr 27: release of exam grades

Apr 29: last day to rebuttal any grading issue