Lecture 24: Delivery

CS 5150, Spring 2022

Test results

- See Gradescope for scores (out of 75 points); solutions in rubric
  - Not sure about something? Ask on Ed or in OHs
  - If you see a scoring *error* (invalid rubric item applied), submit a regrade request on Gradescope by Thursday (11pm)
- Statistics:
  - Median: 79%
  - Standard deviation: 12%
- Most commonly missed problems:
  - Amdahl's Law: parallelization
  - Latency vs. throughput
  - Test brittleness

Project reminders

- Keep meeting with your client
  - Reports are not an efficient way to get feedback
  - Client gets final say on design (user testing informs your *recommendation*)
- CI infrastructure is behind schedule
- Final presentation & delivery
  - Details on Thursday
- Peer feedback
  - Still trouble following directions
  - If you give a "1", must include a comment

Report #4 feedback

- Manual tests require a detailed test plan, including step-by-step procedures and conditions for passing
- Make sure screenshots are legible at print sizes
- Involve client in design decisions
- Document alternative flows
- Double-check report requirements (e.g. current state of testing)

Remaining topics

- Deployment
- Security
  - Input validation & taint tracking, injection, bugs, cryptography
- Safety & reliability
- Legal considerations
  - Licenses, copyright, contracts, patents, privacy, exports
- Ethics

- Assignment: analyze dependencies and vulnerabilities

Poll: PollEv.com/cs5150

Ubuntu 22.04 LTS (long-term support) was released last week. Should you upgrade your Linux (virtual) machines before the end of the semester?

Vulnerabilities

- CVE: Common Vulnerabilities and Exposures
  - Common identifier for specific vulnerabilities (not vulnerable systems)
  - May be crosslinked with other databases (e.g. severity, product, weakness category)
    - NIST's National Vulnerability Database (NVD) includes common links and history
    - Common Vulnerability Scoring System (CVSS) standardizes measures of severity
    - Common Platform Enumeration (CPE) standardizes identifiers for vulnerable components

Releasing software

When is software ready to release?

- When it is feature-complete?
- When it is bug-free?
- When it has soaked for long enough"?
- On the release date?
- Continuously?

- "The biggest risk to any software effort is that you end up building something that isn't useful."
  - Martin Fowler

Traditional release process

Example: GCC

1. Merge window (time-boxed)
   - Branches that are "ready" are merged to trunk
2. Bug fixes (time-boxed)
   - Cut release branch
   - No new (coupled) features
3. Regression & doc fixes
4. Release
   - When all high-priority bugs are fixed

- Challenges
  - Need to coordinate process
  - Features that miss merge window must wait until next cycle
  - Problematic features are difficult to remove
  - Branch divergence

Time-based vs. feature-based releases

Feature-based

- Product manager decides which major features define the next release
- Developers argue for their features to be included
- Which features should hold up release? (tendency for inflation)

Time-based

- All features that are completed (tested) by release deadline are included in release
- Features that are not ready must wait until next release
- Shorter release cadence reduces cost of missing deadline

Risks of long release processes

- Delay in providing value to client
  - May fall behind competition
- Slow feedback on feature utility
- Drain on morale
  - Churn among release managers prevents building expertise
- Difficult to diagnose issues
- Pain leads to over-conservatism, which leads to irrelevance

- Example: YouTube
  - Monolithic Python application
  - Manual regression testing (50 hours)
  - Requires release volunteers (lack of automation)
    - Burnout leads to loss of expertise
- CD recommendation: don't slow down; speed up!

Principles of Continuous Delivery (CD)

- Agility
  - Release small updates frequently
- Automation
- Modularity
  - Isolate changes
  - Enable delegation
- Data
  - Monitor health metrics
  - Evaluate feature effectiveness
- Rollout control
  - Phased rollouts
  - Rollbacks

- Component-based design and microservice architectures provide modularity

- Most benefit comes from being *able* to release frequently, not necessarily from actually doing it

Feature flags

- Tying new features to binary builds is risky
  - Binary rollout and rollback takes time
    - Risk of version skew
    - Can't synchronize feature availability with announcement
  - Fixing a regression requires rolling back *all* new features
- Runtime flags allow more granular control
  - Faster to propagate changes
  - Can enable for arbitrary subset of users
  - Can toggle independently of other features
- Build-time flags can be used to avoid leaks