lecture 17
software development best practices

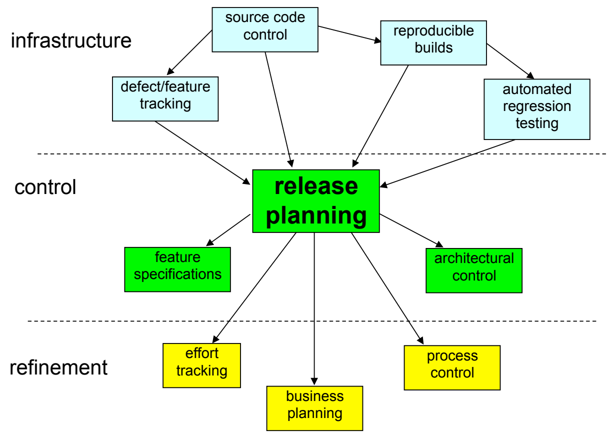
csc302h
winter 2014

recap from last time

- structure of a "typical" software company
- responsibilities of the various roles
  - shareholders, b.o.d., CEO, mgmt. team.
- responsibilities of the different functional units
  - marketing, sales, admin, services, finance…
  - & most importantly, R&D

software development best practices
the top 10

best practices – top 10



1. source code control

- usually a central repo (yes, even with git)
- backed up very regularly
- complete change history
- maintenance streams via branching & tagging
- reproduce old builds/revisions
- required for teams
- commits tied to issue tracking system
- these days, often hosted by a 3rd party
  - sourceforge, github, many others

2. issue tracking

- keeps track of all issues (defects, features, chores, etc.)
- follows one or more workflows
- often incorporates time tracking
- reports for management
- prioritization
- can suffer from "junk-drawer" phenomenon
  - different schools on this: hoarding or purging
- these days, often hosted by a 3rd party
  - ex. pivotal tracker…many, many others!

3. build automation

- single command (scripted) to checkout & build
- consistent environment, no hacked developer environments
  - email: "just use the attached dll…"
- developer builds & production builds
- spits out "the installer," whatever that means depending on context
- (part of) continuous integration through automation
  - "Matt broke the build again!"

4. automated regression tests

- scripted to run after each build – run all unit tests, integration tests, etc.
  - continuous integration (other part)
- prevents previous errors from creeping back in
- enhances developer confidence when making critical changes
- find problems earlier
  - last check-in is smoking-gun
- critical to improving quality over time

5. release planning

- ahem, i mean, agile horizon planning
- with 1 – 4 in place as support, this is arguably the most important practice!
- Release planning determines & tracks:
  - what we are building?
  - by when will it be ready?
  - how many people it will take?
- track for entire duration of release, adjusting along the way
- enables management visibility & decision making
- enables quality by enforcing proper testing

6. design specifications

- complicated features require specs
- helps to make better estimates
- helps eliminate integration problems later
- first candidate for review
- yes, agile requires specifications too!

7. architectural control

- maintain a clean architecture, esp. with many developers and lots of commits
- document the architecture – uml diagrams can help here (class, package)
- review the design before implementing
- enforced by the chief architect

8. effort tracking

- essential for improving #5 – release planning
- how much was estimated, and then spent on:
  - each feature?
  - fixing defects?
  - everything else?
  - any anomalies?
- helps improve estimation accuracy
  - but we don't actually care too much about that… seriously!
- in turn, improves estimates of staff time available for next cycle

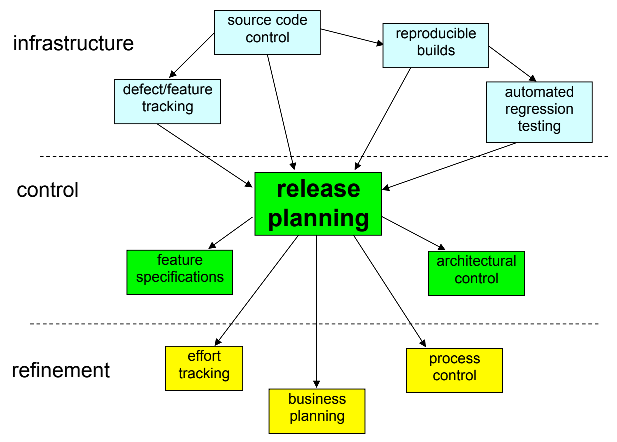
9. process control

- written process for the release cycle
  - protocol to follow, who does what & when
- helps training new staff
- enables collection of metrics
  - how many features are in what state?

10. business planning

- development occurs within a business context
  - this is the the only one that (potentially) differs for open-source projects
- get it wrong, and it can be a bigger problem than technical problems
- involves writing effective proposals
- integrates with budgeting
  - tradeoffs with other departments
- staff, equipment, alternatives, etc.

R&D Best Practices – Top 10


infrastructure
source code control
reproducible builds
defect/feature tracking
automated regression testing
control
release planning
feature specifications
architectural control
refinement
effort tracking
business planning
process control