Tuesday 12/2/14 4156

review - Head First Software Development
ch. 6-12

ch. 6 version control
book uses subversion, this class uses git
What is difference?

- git is distributed, svn is centralized

| | |
|---|---|
| commit changesets locally | commit changeseb to server |
| push/pull to others | need network connectivity |
| may optionally be authoritative central repo | requires less local space |

- git stores content ast meta data (.git)
while svn stores just files (.svn)
includes all tags, branches, version histories, etc.

- git has better support for merging branches
svn branches are just a separate folder in repository
git allows quick switching between branches w/in same working directory

Tuesday 12/2/14 4156

- sun has global revision numbers, git doesn't (other than SHA-1 hash)

- git has better content integrity due to using SHA-1 hash (can check for corruption)

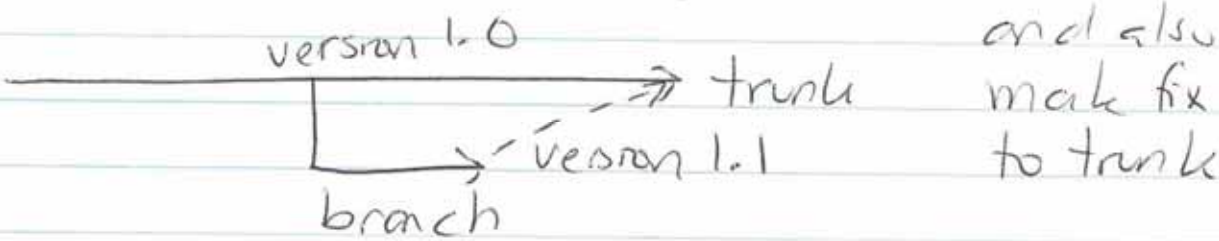
fixing bugs in previous versions

could check logs to figure out which revisions of each file were part of that (released) version

better to associate same tag with every file in that version
- tag is snapshot of repository

tags also used for milestones, not just releases

branch at that tag to make fixes




branch only when need to, not for mundane changes, ~~es~~ sandboxes or separate platforms

?

Tuesday 12/2/14 4:56

ch. 6.5 build — ant
you do not need to know ant
but must know some build tool &
how build tools work in general
→ should have been using one!!

ch 7 testing + continuous integration

black vs. grey vs. white box testing

blackbox functionality from
user perspective
user input validation
output results including
error conditions
state transitions —
use map of states
boundary cases +
off by one errors
(little typos from
user, not internal)

grey box verify logging + auditing
data destined for other systems
- check output format

Tuesday 12/2/14 4/156

mor grey box — system-added info
hashes, checksums,
timestamps, etc.
scraps left lying around
security +/or resource
risk leak
clean uninstall

white box — test all branches
error handling, clean
up file handles,
mutexes, allocated
memory
working as documented
thread safe, null arguments,
security role
resource constraints
what happens when
cannot get
needed resources?

state, error messages } not just

need testing framework that will run
entire suite of tests on one command
continuous integration =
"regression testing" run the entire suite for text describes
every commit cruise control
(or every nightly build)

Tuesday 12/2/14 4156

the longer a test suite takes to run, the less often it will be run
- Jon Bell's VmVm is one approach to speeding up testing
- also test suite minimization
  test selection
  test prioritization

(deals with test dependencies)

test coverage - a huge test suite may still have poor coverage

hard to get 100%, shoot for 85-90%

ratio $\frac{\text{bugs found during testing}}{\text{bugs found after release}}$

if too small need to shoot for higher coverage before deployment

ch. 8 TDD -
write tests first
should initially fail
- not compile
- not run correctly

red

then write code to get test to pass
"simplest code possible"

green

→ then Refactor... (evolutionary design)

Tuesday 12/2/14 4156

TDD - each test should verify
only one thing
avoid duplicate test code
use setup, teardown,
mock objects to
call common code
mirror directory w/ source code
don't mix w/ source code

need to find way to separate
dependencies - loose coupling

use design patterns to help create
mock objects/ stubs (standins) for testing
→ need to test w/ real subsystems
later - integration & system
testing
careful with dependencies - clean
up after each test to
known, restorable state

Ch. 9 ending an iteration

System testing - 2 iteration cycles (ideally)
in parallel w/ separate
testing team - but timeboxes may
not match scope

Tuesday 12/2/14 4156

development team will get bug reports during iteration
- treat like any other story & prioritize although harder to estimate time

seventh vs. priority

or, if same team, alternate development & test/fix iterations

bug reports - not always clear fail
maybe ambiguity in documentation, missing feature, break from UI style guide

need steps to recreate bug & how bug was detected (~~e.g. error messages~~ ~~vs. what expected~~)

what happened vs. what expected

new test case

only original reporter should mark bug as closed, not developer who "fixed"

Use bug tracking tool - record everything

generate metrics — new-bug submission rate, bugs per code region, how many left to fix, zero-bug-bounce before release

Tuesday 12/2/14 4156

iteration review – process improvement

- is everyone happy w/ quality of work? doc? testing?
- pace of iteration?
- developers comfortable w/ area of system
- tools helping or hurting productivity? new tools?
- process changes to consider
- code identified for refactoring or rewriting
- performance problems?
- discuss open bugs
- testing effective? sufficient test coverage for confidence?
- deployment under control & repeatable?

ch. 10 the next iteration

additional user stories / use cases
revise estimates & priorities

(reductions bug fixes)

calculate project velocity

new big board w/ new burn down

Tuesday 12/2/14 4156

ch. 11 integrating w/ 3rd party code

DO NOT assume anything works

cannot use TDD, code already exists

but need to test everything yourself & bring into your process everything your requirements rely on
→ take ownership

spike debugging

if lots of tests fail, pick random selection & fix w/in limited time window — maybe one week

then get estimate of how long to fix bugs for full set of tests

→ maybe could only fix 80% — or 100% use for time estimate (& confidence)

ch 12 summary
pointers to other books & websites