Construction

CSE 403, Spring 2003

Software Engineering

http://www.cs.washington.edu/education/courses/403/03sp/

Readings and References

- Chapter 18, Daily Build and Smoke Test, *Rapid Development*, McConnell
- The Joel Test: 12 Steps to Better Code, Joel Spolsky
  http://www.joelonsoftware.com/printerFriendly/articles/fog0000000043.html

Some construction fundamentals

- Agreed-on coding standards
  - » naming, layout, documentation
- Data-related concepts
  - » scope, persistence, binding times
- Control-related
  - » complexity, control structures, exceptions
- Errors and exceptions
  - » assertions, defining and handling exceptions

More construction fundamentals

- Integration strategies
  - » Unit-testing and debugging
  - » Build and packaging practices
- Code tuning and performance measurement
- Programming tools
  - » editors, IDE, interoperability
  - » group work support tools (email, change visibility)
  - » source code revision management
  - » bug tracking

The Joel Test

- Do you use source control?
- Can you make a build in one step?
- Do you make daily builds?
- Do you have a bug database?
- Do you fix bugs before writing new code?
- Do you have an up-to-date schedule?
- Do you have a spec?
- Do programmers have quiet working conditions?
- Do you use the best tools money can buy?
- Do you have testers?
- Do new candidates write code during their interview?
- Do you do hallway usability testing?

Disclaimer (Spolsky)

- Of course, these are not the only factors that determine success or failure:
  - » in particular, if you have a great software team working on a product that nobody wants, well, people aren't going to want it.
  - » And it's possible to imagine a team of "gunslingers" that doesn't do any of this stuff that still manages to produce incredible software that changes the world.
- But, all else being equal, if you get these 12 things right, you'll have a disciplined team that can consistently deliver.

Software Configuration Management (SCM)

- SCM is the practice of managing project artifacts so the the project stays in a consistent state over time
  - » processes for evaluating proposed changes
  - » tracking changes and enabling roll-back
  - » handling multiple versions
- Most often applied to source code, but also beneficial for requirements, design, test cases, user documentation, scripts, etc, etc

Source Control

- The team product is a *complete working program*
  - » correctly built from synchronized and correct source code and resources and tested appropriately
- Multiple people working on one collection of sources can be a nightmare unless managed well
  - » Overlapping changes, old and inconsistent versions
  - » Disks crash, houses burn, computers are stolen
  - » There are good tools to help you manage integration!
    - use CVS, not caffeine

Make a build in one step

- On good teams, there's a single script you can run that
  - » does a full checkout from scratch
  - » rebuilds every line of code
  - » makes the binary executable files in all versions, languages and #ifdef combinations
  - » creates the installation package
  - » creates the final media - CDROM, web site, …
- All steps are automatic and exercised regularly

Daily Build and Smoke Test

- Build the entire product every day and run a good test suite against the new version
  - » automatic and frequent
  - » canary in the mine - find out early that you've got problems and fix them before disaster strikes
- Benefits
  - » Minimizes integration risk
  - » Reduces risk of low quality
  - » Supports easier defect diagnosis
  - » Improves morale - developers, managers, customers

Using Daily Build and Smoke Test

- Build daily
  - » Developers check in working modules
  - » The build is the heartbeat or sync pulse of project
- Check for broken builds and fix problems
  - » Define appropriate quality level
  - » At minimum, build should be useful for testing
    - complete compile, link, package, and pass smoke test
- Smoke test daily
  - » exercise entire system from end to end
  - » grow the tests with the system

Use a bug data base

- You need to know
  - » how to reproduce the bug
  - » expected behavior, actual behavior
  - » current owner of the bug
  - » status (open, fixed)
- You can't keep the bug list in your head!
- There are numerous tools available
  - » Don't use something that is so fussy that it is a big pain to enter, comment on, and close bugs
  - » free trial version of FogBUGZ is available
  - » an Excel spreadsheet can do the job

Fix bugs before writing new code

- Don't build the termites into the structure
  - » Bugs are always easier to find soon after creation rather than after time has gone by
- Sometimes bugs reveal fundamental problems
  - » you may have a basic concurrency problem!
- You can't accurately schedule the repair and release of a system made from defective parts held together with duct tape and prayer

Up to date schedule

- "It will be done when it's done!"
  - » When will my computer be repaired?
  - » When will you finish your degree?
  - » When will you have a releasable product?
- Confidence in the schedule enables all sorts of decision making and planning to go on
  - » lower stress, higher morale all around
- A good schedule helps you resist feature creep
  - » Don't let the doodads build up and delay delivery

Have a Good Specification

- Know what you are building
  - » Write it early
  - » Keep it up to date
- The spec is the tool that can help you see where you are going to have problems
  - » Are the scenarios realistic?
  - » How you are going to accomplish the promises?
  - » It's a lot easier on everybody to change the promise now than to break the promise later

Have quiet working conditions

- Minimize uncontrollable distractions
  - » turn off your email
- Be focussed when you are alone and working
  - » get in the zone and blast away
- Be focussed when you are meeting and discussing with others
  - » communication is important, so make good use of the time you are together

Use the best tools money can buy

- This doesn't mean the most expensive tools!
- Spend the time to understand
  - » which tools you need
  - » which tools you already have
  - » what you need to be more productive
- If you need an investment, think about how to request it then *stand up and request it*
  - » There is a lot of money available, why should it be spent on you?

Use testers as basic part of the team

- Testing is a different mindset from developing
- It can be interesting to do and very revealing in its results
- Your customers are going to test all the nooks and crannies of your system anyway
  - » testers are your friends, not your enemy!
  - » find out the problems now, not after shipping

Write code during interviews

- We are not hiring, but still ...
  - » You are writing code while learning the processes
  - » You are using a variety of tools and processes
- Think about your projects at an abstract level
  - » Could you describe the successes and problems in the project life cycle?
  - » Could you lay out a project plan for a hypothetical system product that uses a reasoned selection of tools and techniques?

Hallway Usability Testing

- Does this project and its design make sense to somebody who is not married to the project?
- Let somebody new use the product
  - » Do they understand what it is?
  - » Do they like it?
  - » Do they make assumptions that you never thought of?
  - » It only takes a few people doing this to understand if you are on track.

Some support tools

- Ant - build, package, test integrator
- JUnit - testing framework
- JavaNCSS - simple code metrics
- JDepend - design quality metrics
- Checkstyle - coding standard checker
- FogBUGZ - bug tracking
- CVS - source code revision management