Software Engineering Principles

With examples from Google and Cray

Asaph Zemach
Background

• About Cray
  – Builds large supercomputers for technical computing.
  – Serves the government, national lab and similar agencies abroad.
  – About 1000 employees, ~200 SWE, ~200HWE
  – Competitors: IBM, SGI, commodity clusters

• About Google
  – Internet search, Web ads
  – Serves all web users, serves advertisers
  – About 6000 employees, “a few thousand” SWE
  – Google.com, AdSense, Talk, Earth, desktop, Picasa, Pack, Blogger, Toolbar, Maps, Scholar, Gmail, Calendar, …
  – Competitors: Yahoo, MSN, Baidu

• About me
Cray and Google

• Small customer base, every deal is important and large.
  – Service is significant portion of revenue.
• Strong ties to existing customers.
  – “Customer X has this code…”
  – Drives and inhibits innovation
• Limited product line: X1, XT3, XD1
  – Products take years and millions of dollars to develop.
  – Tight HW/SW Integration
• Cray projects driven by engineering improvements to create “the next generation.”

• Huge user base
  – Service the “long tail”.
  – Many advertisers, some big ones (AOL, Amazon).
• No personal relationship with users or (most) advertisers.
  – Must think about large numbers of varied users.
  – “What do people want?”
• Varied product line
  – Products can take months to get to beta.
    • Let’s get something out there.
  – Loosely integrated over Google infrastructure.
  – Google give engineers 20% time to pursue their own ideas
    • Source for a lot of innovative ideas.
Flow of Ideas
What to Look for in a Project

Interesting (to you) gets you started
Doable (in your opinion) gets you to first prototype
Meaningful (to you) gets you to a product
What to Look for in a Project

Other Considerations:
- Profile
- Impact
- People
- Funding
- Company Buy in
- Challenge
- Coolness
- Fun

**Interesting** gets you start
**Doable** gets you to first prototype
**Meaningful** gets you to a product
Design Phase

• How to get good input on your ideas
  – Cray - Send description by email, plenty of meetings, send meeting summary by email, eventually write document. Repeat.

• Tightly integrated vs loosely integrated projects.
  – Not every project needs an equal amount of vetting.

• Make sure you cover: Background, Motivation, Goals vs Non-Goals.
  – Too easy to jump right into the technical details.
Development Phase - The Work

• Choose the right language. Consider language properties and support for 3rd party SW needed for your project.

• Have a style guide
  – Cray C++ style: per project, primarily covered look
  – Google C++ style: company wide, look & language features.
    • Example: `page_ = kStartPage;`

• Use a source control system. Don't be afraid to layer your own scripts on top
  – Cray had CVS + review/commit + MySQL
  – Google has 3rd party source control + scripts (policy, branching, features).

• Source control should enforce project/company policies
  – Example: Has the code been reviewed? By who?
Comments and Change Descriptions

• Very easy to just explain what you did.
  – Obj = new Foo(x); // Create new Foo. **Gee thanks…**
  – Here’s an example to explain why a certain parameter is passed to a constructor.
    /* Why couldn't Foo create a Bar for you and delete it along with itself? Because Foo ... */ (details omitted).
    – “I made this mistake and I knew I’d forget and make it again if I didn’t leave a note.”

• Another example:
  – This commit adds feature Foo to class Bar. I also considered these two other approaches […], but this one was the least amount of work and should be fast enough for the size we are looking at.

• Explaining the “What” will save the reader few minutes of head scratching.

• The “Whys” will potentially prevent days of unnecessary work.
Integration Testing

• It’s never too early to integrate.
  – Always try to have a working system, even if you need to simulate missing parts.
    • Best way to test interaction between parts.
    • Great for demos.
  – Beware of over reliance on simulated parts, the real modules might not work the same way.

• It’s easy to make fast progress when you work in isolation, and it always leads to underestimation of integration costs.
  – If it’s not checked in, it can’t be integrated with it doesn’t exist.

• Constant background integration, at least at the build level.
  – Automated Nightly build and regression testing.
Can anyone really follow 200+ tasks?

We are tracking weeklong tasks the same way as 19 week tasks!

Weekly reporting in this format foisted on Cray by the Government.

After all if the Project™ is fine then the project is fine. Right?
Project Management:
How I Do It Now

• Well defined End of Quarter goal
  – “Support 100% load by EOQ”
  – Everybody on the project has agreed to this goal.
    • Do what is needed to keep this goal foremost in the teams mind.
• We defined 5 Priority areas for this Quarter e.g.
  – “1. Improved monitoring”
  – “2. Complete integration with Project X”, etc.
  – Easy to decide if what someone is doing is in one of these five areas.
    • If so, fine. Trust them to do the right thing and work towards the goals.
    • If not, find out why and see if the scope can be reduced.
• Keep milestones on the web.
  – Anyone can update.
  – Everyone in Engineering can see.
• Less burden, while still keeping the team focused.
• Less information overall
  – No consistent way to account for interdependencies between projects.
  – Up to managers/directors to make sure different groups are talking.
• OK for small, independent projects.
Project Management: Getting the Best of Both

- Projects clearly define dependencies on each other.
  - Reduce the things that need to be tracked in detail.
- LW mgmt for each project.
  - Within each project assume people know who depends on what.
- We tried this at Cray.
  - It was an improvement over the great-wall-of-Project™.
  - Problem of figuring out when “Feature X” will be late and by how much still not solved. Just reduced (somewhat).
Deploying Code

• Release branch vs development branch
  – Flying fish development
    • Works well when little code dependencies
    • Hard when you depend on a lot of external code

• "Early and Often" vs "No wine before its time"
  – Capture market & mind share
  – Don’t alienate customers with subpar products

• Support issues
  – Assume that once a project launches, the team will spend 50% of it’s time in support.
  – Google needs a large scale model (millions of users…)
    • Tracks issues in general
    • Not all bugs are obvious e.g. query search results.
  – Cray has multilevel support:
    • On-site support – knew customer specific setup
    • Phone support
    • Support organization – trained in solving many problems, able to modify configurations, write scripts, etc.
    • Development – often knew bugs by site-name
Development Ecosystem

• How to be productive
  – Co-location vs Remote teams
    • Consider cultural issues.
  – Projects need periodic reviews
  – Communication is key: mailing list vs a Wiki
    • Common types of mailing list: project, project-users, project-announce
    • Wiki good for “How to”, user guide, latest performance.
  – Report your productivity: weekly status reports.
    • Helps you explain what you did come review time.
    • Support opt-in for people interested in your project.

• Hardware
  – Developer Hr ~ $75, Another monitor ~ $500. Get the monitor.
  – Upgrade a workstation to 8GB ~ $3K. For the whole team ~ $15K. Does everybody need 8GB or can some people work with 4GB (for now).
  – Don’t forget ergonomics.
    • Many large employers offer support in this area.
Work Life Balance

• Take advantage of benefits
  – Ask around for what’s available.
  – Don’t be afraid to suggest additional benefits.

• Take vacations!
  – 100 accrued vacation days is not a badge of honor.
  – There’s never a good time. Consider deadlines, but don’t be a slave to milestones.
  – If you bring your laptop – it’s not a vacation.

• Remember:
  – Nobody on his deathbed ever said, "I wish I had spent more time at the office."
Thank You!

Questions?
Backup
Source Control and Build Systems

- Need to go beyond `make`.
  - In large project make files quickly become unmanageable.
  - Need to talk about whole modules.
- Can you `make clean` correctly?
  - How to deal with dependence on some remote module?
- Can you build the tools? The compiler?
- Build management and source control is a full time job.