Wrapup of Refactoring example

What did we do?

- Made date calculator easier to read and understand using simple *refactorings*
- Found a bug
- Observation: if we had developed method using TDD, might have gone easier!
- Improved our **`flog`** & **`reek`** scores

Other Smells & Remedies

Smell	Refactoring that may resolve it
Large class	Extract class, subclass or module
Long method	Decompose conditional Replace loop with collection method Extract method Extract enclosing method with `yield()` Replace temp variable with query Replace method with object
Long parameter list/data clump	Replace parameter with method call Extract class
Shotgun surgery; Inappropriate intimacy	Move method/move field to collect related items into one DRY place
Too many comments	Extract method introduce assertion replace with internal documentation
Inconsistent level of abstraction	Extract methods & classes

Which is NOT a goal of method-level refactoring?

- ☐ Reduce code complexity
- ☐ Eliminate code smells
- ☐ Eliminate bugs
- ☐ Improve testability

Legacy Code & Refactoring: Reflections, Fallacies, Pitfalls, etc.

(ESaaS §9.8-9.10)

First Drafts

When in the Course of human events, it becomes necessary for a people to advance from that subordination in which they have hitherto remained, & to assume among the powers of the earth the equal & independent station to which the Laws of Nature & of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the change.

We hold these truths to be sacred & undeniable…

First Drafts

When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, & to assume among the powers of the earth, the separate & equal station to which the Laws of Nature & of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.

We hold these truths to be self-evident…

Fallacies & Pitfalls

Most of your design, coding, and testing time will be spent refactoring.

- ⚠ "We should just throw this out and start over"
- ⚠ Mixing refactoring with enhancement
- ⚠ Abuse of metrics
- ⚠ Waiting too long to do a "big refactor" (vs. continuous refactoring)

Which is TRUE regarding refactoring?

- ☐ Refactoring usually results in fewer total lines of code
- ☐ Refactoring should not cause existing tests to fail
- ☐ Refactoring addresses explicit (vs. implicit) customer requirements
- ☐ Refactoring often results in changes to the test suite

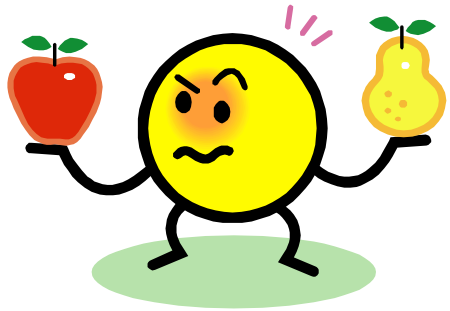
Plan-And-Document Perspective on Software Maintenance:

(Engineering Software as a Service §9.7)

David Patterson

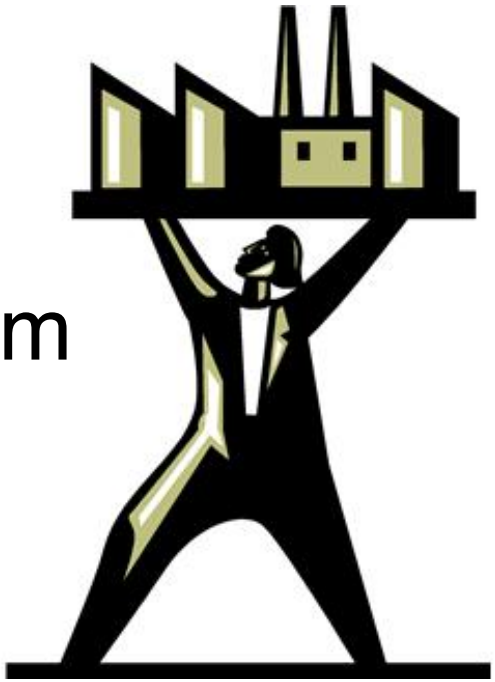
P&D Maintenance?

- How much spent on P&D development vs. P&D maintenance?
  - How does this compare to Agile?
- Agile developers maintain code
  - Does P&D use same or different people for maintenance?
- What does the P&D Maintenance Documentation look like?



P&D Maintenance

- P&D spends 1/3 on development, 2/3 on maintenance
  - Customers pay 10%/year SW maintenance fee
- Development ≠ Maintenance Team
  - Maintenance Managers
  - Maintenance SW Engineers
  - Typically less prestigious



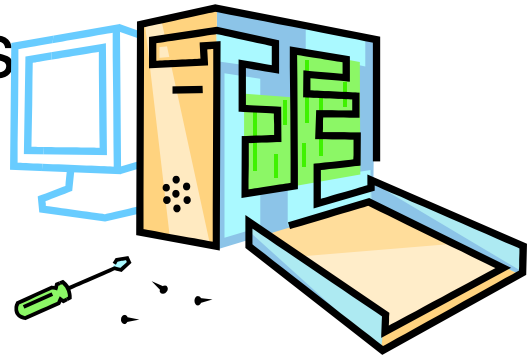
Maintenance Manager

- Like Development Manager
  - Estimate costs, maintain schedule, evaluate risks & overcomes them
  - Recruits maintenance team
  - Evaluate software engineers performance, which sets salary
  - Document project maintenance plan (maintain documents & code)
    - IEEE standard to follow
  - Blamed if upgrade takes too long or too expensive

P&D Maintenance Process

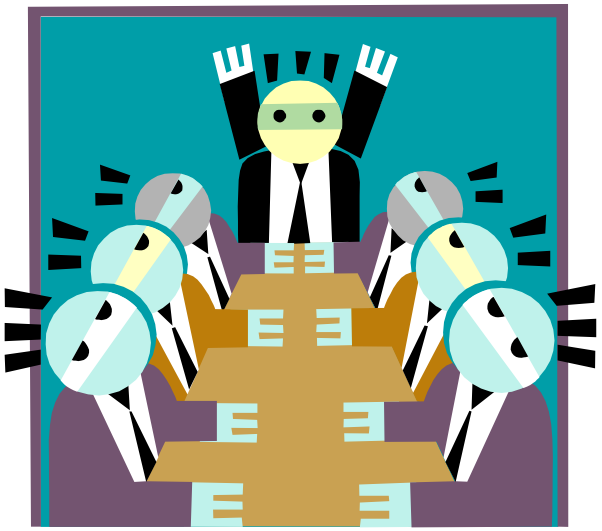
- Differences vs. Development Process:

1. Working SW in field
   - New releases can't break features
2. Customer collaboration
   - Work with customer to improve in next release vs. meet contract spec
3. Responding to change
   - Customers send *change requests*, which SW engineers must prioritize
   - *Change request forms* have ticket tracking



Change Control Board

- Board (not Manager) decides
- Manager estimates cost/time per change request
- QA team gives cost of testing for change request, including regression testing + new tests
- Documentation teams gives cost of updating docs
- Customer support group decides if urgent or workaround



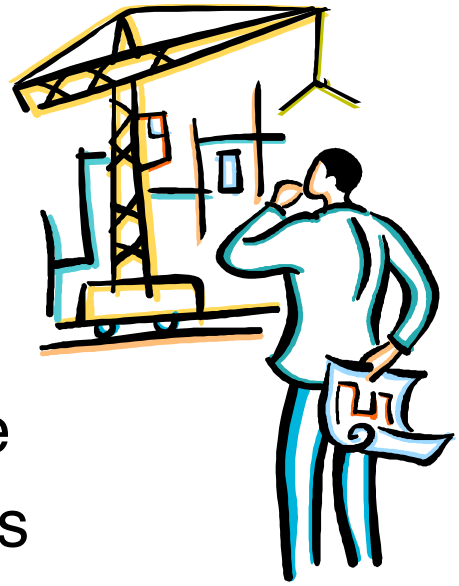
Urgent Change Request

- No time to update docs, plans & code
  - Software product crashes
  - New laws affect product
  - Security hole => data vulnerable
  - New releases of underlying OS/library
  - Must match competitor's new feature

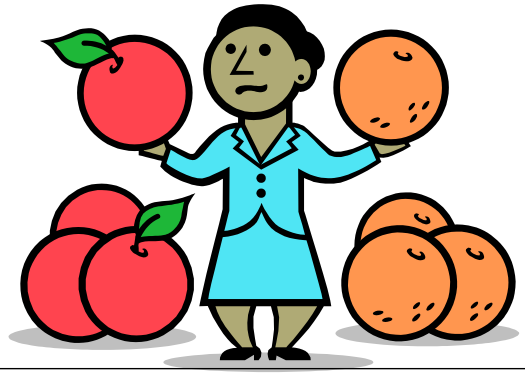
- Synch after emergency?
  - Emergencies too frequent to catch up?
- Time to refactor to improve maintainability
  - Too expensive for Change Control Board?

Time to Re-Engineer?

- Time to refactor to improve maintainability?
  - Refactor continuous during dev & main
- Re-Engineer to Improve vs. Replace?
  - Use automated tools to upgrade as SW ages and maintenance hard
    - Change database schema
    - Improve documentation by reverse engineering
    - Code analysis tools to point to bad code
    - Programming Language translation tools



Maintenance: P&D vs. Agile



Tasks	*In Plan and Document*	*In Agile*
Customer change request	Change request forms	User story on 3x5 cards in Connextra format
Change request cost/ time estimate	By Maintenance Manager	Points by Development Team
Triage of change requests	Change Control Board	Development team with customer participation
Roles		
	Maintenance Manager	n.a.
	Maintenance SW Engineers	Development team
	QA team	
	Documentation teams	
	Customer support group	

Which statement regarding P&D maintenance is FALSE?

1. The cost of maintenance usually exceeds the cost of development in P&D
2. The Agile equivalent to P&D change requests is user stories; equivalent of change request cost estimates is points; P&D releases are iterations
3. The Agile lifecycle is similar to the P&D maintenance lifecycle: enhancing working software product, collaborating with customer vs. negotiating by contract, continuously responding to change
4. All the above are true

Provocative Question

- If 2/3 cost of product are in the maintenance phase, why not use same maintenance-compatible software development process for whole lifecycle (Agile) vs. separate processes (and teams) for development and maintenance?

The Affordable Care Act software fiasco (a CS169 special report)

Armando Fox & David Patterson

Which number best describes the total number of lines of code (LOC) in the Affordable Health Care Web service?

- ☐ 0.5 Million LOC
- ☐ 5 Million LOC
- ☐ 50 Million LOC
- ☐ 500 Million LOC

Space Shuttle: ~0.4 MLOC

Linux 3.10: ~13 MLOC

Windows Vista: ~50 MLOC

Banking: ~100 MLOC

“One specialist said that as many as _____ lines of software code may need to be rewritten before the site runs properly.” *Slate,* “The One Disheartening Number That Suggests Healthcare.gov Will Not Be Fixed Anytime Soon”

- ☐ 0.5 Million LOC
- ☐ 5 Million LOC
- ☐ 50 Million LOC
- ☐ 500 Million LOC

Space Shuttle: ~0.4 MLOC

Linux 3.10: ~13 MLOC

Windows Vista: ~50 MLOC

Banking: ~100 MLOC

TL;DR

- **Should** they have used Agile?
- **Could** they have used Agile?
- Regardless, is it **good code?**

What's in the code?

- Logic for insurance exchanges for 36 states
- Integration: 14 states' existing insurance exchanges
- Integration: IRS & Social Security data systems (verify identity, income, residence)
- Federated identity management (all systems have different concept of identity)
- "Front end": registration, account creation, steering applications to correct exchanges

What's in the code?

- Disclaimer: only small part of code released (Combination of client-side JS & Node.js)
- JS code doesn't pass JSHint (code smells)
- No tests directory
- No caching of static assets (JavaScript); over 2MB JS served per page view
- No CDN serving of static assets
- No compression (minification) of JavaScript
  - reduced size & load time by ~75% when done

Project size as predictor of success?

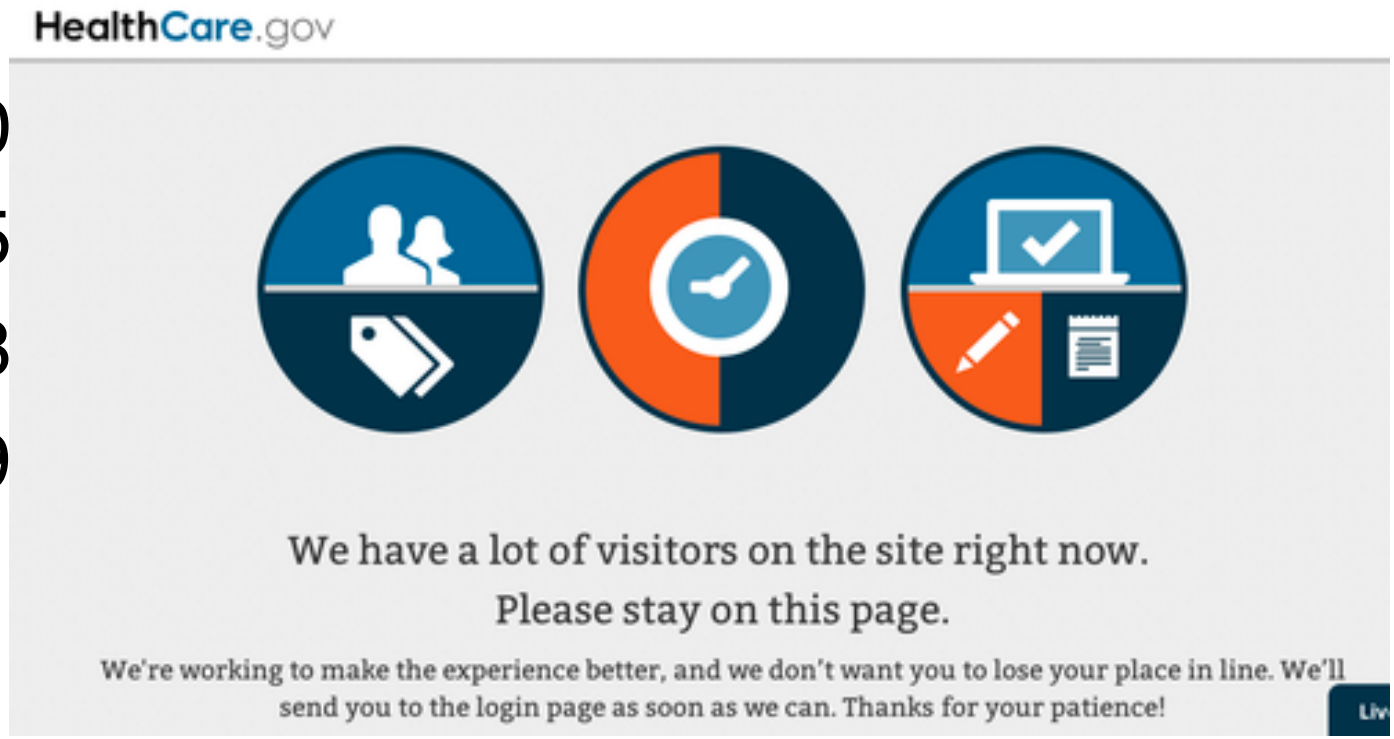
- 2013 CHAOS report (Standish Group): for large (>$10M) software projects,
  - 10% on time and on budget
  - 52% challenged (late, over budget, incomplete)
  - 38% failed (cancelled, or delivered but not used)
- *"The real key to success is doing less for less. The key to doing less for less is splitting large projects into a sequence of small ones..."*

Contract background

- 2007: 16 companies including CGI Federal "qualified" $4B for "Indefinite delivery, indefinite quantity" Enterprise Systems Development for Centers for Medicaid & Medicare Services (CMS)
- 55 contractors, $394M total
- Largest CGI Federal $88M
  - CGI also has ~$8B in Federal contracts over next few years

Woes

- United Sample poll of 832 users trying to sign up:
  - 50
  - 25
  - 38
  - 19



The runup

- "It's fast, built in static HTML, completely scalable and secure," said Bryan Sivak, chief technology officer of HHS, in an interview. [Referring to front-end code built by Development Seed, Inc]

These are not new problems

- "More users than expected" (>20M)
  - So did FarmVille! 3➔150 cloud servers in weeks
- "Interfaces with legacy systems" (IRS, SS)
  - Airlines, banks, stock trading solved pretty well
  - Did anyone stub the APIs to test the system?
- "Database could be bottleneck" (Talk to 14 states' insurance sites, serve 36 others)
  - Queue tasks and deliver results later
- "Not enough time to test"
  - More of a risk if testing is left til last (Waterfall) and requirements changing during development

Brooks's Law

"Adding manpower to a late software project makes it later."

Fred Brooks Jr., *The Mythical Man-Month*

- Response: "Tech surge" of "best & brightest" called in to fix
  - Verizon engineers
  - "contractors & experts from insurance industry"
  - "veterans of top Silicon Valley companies"
  - "This new infusion…will bring a powerful array of subject matter expertise and skills, including extensive experience scaling major IT systems."

Code Reviews?

- "With enough eyeballs, all bugs are shallow"
  Eric S. Raymond, *The Cathedral & The Bazaar*
- Healthcare.gov: withdraw code initially posted on GitHub
  - Happily someone forked repo (STRML/Healthcare.gov-Marketplace) and has started to inspect/fix

Division of Labor?

- Pitfall: dividing labor based on front-end/ back-end, vs. **end-to-end** story owner
- Healthcare.gov: front- & back-end contracts
  - couldn't talk to each other
  - front end delivered early, good code & UI
  - no ownership of end-to-end UI or stress test
  - back-end integrations difficult but apparently insufficiently tested end-to-end

Customer changes mind?

- Pitfall: failing to build right thing, even if built the thing right
- Healthcare.gov: "In the last 10 months alone, government documents show, officials modified hardware and software requirements for the exchange seven times." *Insurance site seen needing weeks to fix,* NY Times, 21-Oct-2013

Agile Cost Estimation

- Real world needs to estimate cost before customer can agree to project
- Agile: quality, schedule, scope => pick 2
- Healthcare.gov: Set rollout date, demand upfront plan, commit money for contractor

Incremental rollout?

- "Feature flags" for incremental rollout
  - A few features at a time
  - A few users at a time
  - Every major successful SaaS site uses this, and grows site "organically" over time
- Healthcare.gov: whole thing goes live for all states, all users, on a single "flag day"

Yes: Plan-and-Document
No: Agile (Sommerville, 2010)

1. Is specification required?
2. Are customers unavailable?
3. Is the system to be built large?
4. Is the system to be built complex (e.g., real time)?
5. Will it have a long product lifetime?
6. Are you using poor software tools?
7. Is the project team geographically distributed?
8. Is team part of a documentation-oriented culture?
9. Does the team have poor programming skills?
10. Is the system to be built subject to regulation?

Federal procurement broken?

- 58 “preferred” providers tend to dominate Federal IT contracts
  - 6500 pages of regulations
  - Process inherited from DoD that requires full upfront specification
  - Contractor fear of being sued for breach of regulations and losing contract to competitor
  - Govt contract officer fear of being “fingered” by companies not selected for contract for not following rules
  - (These companies spend millions of $ lobbying)

Federal contracting rules work against Agile

- "If I were to bid on the whole project, I would need more lawyers and more proposal writers than actual engineers to build the project … If people don't see the need for procurement reform after this, we're in trouble."
  - Eric Gundersen, Development Seed, Inc.
  - Development Seed was technically a subcontractor to Aquilent
- Can't change course in midstream without breaking contracting rules
- Waterfall used because process inherited from HW projects, and "easy to procure for"

Summary

- Technical challenges not trivial, but not new
- Splitting into smaller projects could have helped
- Agile could have helped (and would work with small teams), but Federal procurement rules basically make Agile illegal
- Quality of code revealed so far is embarrassing