Class 10 CS540

Gregg Vesonder
Stevens Institute of Technology
(© 2005, Gregg Vesonder)

Roadmap- Class 10

- Clarifications from last class
- Log Book volunteer
- Brooks
- Heavy versus Light
- Extreme Programming
- Crystal Methodology
- Reading: Brooks Chapter 13
- Reading next class: Brooks 14 & 15, BY 9

Calendar -Key Dates

- November 7th - second test
- November 21st - log books due
- December 12th - final exam

Clarifications

- Testing, refactoring and OO - next page
- Overloading is one type of Polymorphism
  - Polymorphism - process objects differently depending on their data type or class
  - Overloading (number and type of arguments) --many methods same class vs overriding a superclass method -- many methods different classes.
- Abstract data - internal form hidden by access functions - OO classes should be abstract data type

Testing in OO

- Junit - http://www.junit.org
- TestCase is an object runTest() is an overridden method.
- Philosophy (from Refactoring book)-- classes should contain their own tests:
  - Make sure all test are fully automatic and checks their own results
  - Do testing frequently -- at least daily
  - When you get a bug report start by writing a unit test exposing the bug
  - It is better to write and run incomplete tests than not to run complete tests (some is better than none)
  - Think of boundary conditions under which things go wrong and concentrate there
  - Don't forget to test exceptions to make sure they are handling things that go wrong

Logbook

- Your Entry
- Stone Soup
- Elephant and Monkey - Boehm and Turner

Brooks Chapter 13

- Building a system to work
- Conceptual integrity of a design not only makes it easy to use but easy to build
  - Careful function definition
  - Careful specification
  - Eliminate frills and exploration of techniques
  - Outside group tests the specification
  - Vic Vyssotsky's quote: "They won't tell you they don't understand it; they will happily invent their way through the gaps and obscurities."

Top-down Design

- Design as a sequence of refinement steps
  - First sketch rough task definition and rough solution, refine
    - Each refinement in task definition results in a refinement in the algorithm
- Avoid bugs by:
  - Clarity of structure and representation makes precise statements about requirements and functioning of the module easier
  - Partitioning and independence
  - Suppression of detail
  - Design can be tested at each of the refinement steps
- On testing - budget for scaffolding - programs and data built for debugging and testing but never intended as a final product - results in 50% more code!

Light vs. Heavy

- Heavy, as in heavy in process, what we have been studying
- Light well, here's Jim Highsmith's take from the Cutter Consortium: "Thin, lean, adaptive, or light- these emerging approaches are thin on process, thick on skills, and focus on people: collaboration, communication and excitement."
- A light methodology provides guidance and boundaries whereas a heavy methodology is prescriptive
- A reactionary movement and representative of what was really happening - an attempt to gain some control or at least document what was really happening -- RAD was not the answer (although some consider RAD agile)

ASD vs. RSM

- Agile Software Development vs. Rigorous Software Methodologies
- **Key concept in agile methodologies is to embrace change**
  - Agility is a way of life in a constantly emerging and changing response to business turbulence
  - Improvise
  - Trusting in one's ability to respond rather than trusting in one's ability to plan
  - Focus on individuals and self adapting their own processes
  - Chaordic perspective -- chaos to order (product goals not predictable, process not repeatable)
  - Collaborative values and principles - human dynamics, may be the "soft" sciences but they are the hardest!
  - Barely sufficient methodology - **programming usually adds value, process management usually adds overhead**

Agile Manifesto

Manifesto for Agile Software Development

http://agilemanifesto.org

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

© 2001, the above authors
this declaration may be freely copied in any form,
but only in its entirety through this notice.

Focus on Light Methodology

- Customer value
- Creating a culture of innovation, creation and rapid delivery
- Appealing to skilled, talented staff
- Facilitating collaboration, knowledge sharing and decision making
- Reducing, time, cost and defect levels significantly.

Some Light Methodologies

- Extreme Programming - Kent Beck
- Crystal Methods - Alistair Cockburn
- Lean Development - Bob Charette
- SCRUM - K. Schwaber, J. Sutherland
- Adaptive Software Development - Jim Highsmith
- …

XP

- XP's basic premise - coding is THE KEY activity
- Geared for small to medium sized teams
- Calls for implementing highest priority features first
- Customer is integral part of the team
- Define smallest code release possible
- Programmers accept responsibility for estimating and completing work - feedback
- Encourages human contact, incorporates method for staff turnover

Underpinnings of XP

- If code review is good, do it all the time, pair programming
- If testing is good, everyone will test (unit testing), customers (functional testing)
- If design is good, it is every day for everyone (refactoring)
- If simplicity is good, we'll leave system with simplest design that supports the functionality - the simplest thing that can possibly work
- If architecture is important everyone does/defines/refines architecture all the time
- If integration testing is important then we will integrate and test several times a day
- If iterations are important we'll make iterations really short, minutes and hours, not weeks and months!

Differs from other methods

- Short cycles provide early, concrete and continuing feedback
- Incremental planning that quickly generates an overall plan that evolves
- Responds to changing needs by flexibly scheduling implementation of functionality
- Reliance on automated tests to catch defects early, monitor progress and allow system to evolve
- Reliance on oral communication, tests and source code to communicate system structure and intent
- Reliance on evolutionary design process throughout development
- Reliance on close collaboration of programmers with ordinary skills
- Reliance on practices that mesh with short term instincts of programmers and long term interests of project

A Day in the Life of an XPer

- Stand up meeting begins day
- Stack of task cards provides tasks
- Invite programmer to be your partner
- Develop together - both at the screen, mouse and keyboard concerned with implementation, other more globally
- First build tests, run tests
- Develop program, assess design, collaborate
- Test
- Programming pairs evolve design of the system - they can change everything!

Four Values of XP

- Communication: unit testing, pair programming, and task estimation cause programmers, customers and managers to communicate
- Simplicity - better to do a simple thing today and change later, than a more complicated thing that will not be used
- Feedback - unit tests, customers write stories (feature descriptions), programmers estimate -- when all the tests are run you are done
- Courage - within the context of the first three values - "go like hell!" However, courage by itself is "just plain, bad hacking!"
- Other comments:
  - XP resembles hill climbing local optimas require large change
  - Need a real team that respect each other and have passion for what they do

Fundamental Principles of XP

- Rapid feedback
- Assume simplicity
- Incremental change - smallest change that makes a difference
- Embracing change
- Quality work - excellent or insanely excellent

Less Central Principles

- Teach learning
- Small initial investment
- Play to win, rather than playing not to lose
- Concrete experiments - especially regarding requirements
- Open honest communication
- Work with people's instincts, not against them - "XP matches observations of programmers in the wild"
- Accepted responsibility
- Local adaptation
- Travel light
- Honest measurement

Key Aspects of XP

- **Whole team** - development + customer
- **Metaphor** - everyone use common analogy in discussing the system, e.g., desktop metaphor -Scandinavian School
- **The Planning Game** - specify the next step of development and, as project progresses, provides better and better picture of what will be delivered. **User stories** lead to development cost estimates, leads to client assigning priorities, leads to evaluating estimates for the next round
- **Simple design** - design should only incorporate at best next iteration - if it becomes complex, refactor
- **Small releases** - every development cycle (~ 2 weeks) client gets new software.

Key Aspects of XP - 2

- **Customer tests** - customer develops acceptance tests based on user stories, automated and used frequently by development team
- **Pair programming**
- **Test-driven development** - test first, add to suite
- **Design improvement** - refactoring and small improvements in design, simple design
- **Collective code ownership** - source code control, "refrigerator in frat house" phenomenon (anything you put in, you should not expect to see next time)
- **Continuous integration**
- **Sustainable pace**
- **Coding standards**

Comments on XP

- Some of the old is there - incremental, prototyping, variant of use cases
- Takes code reviews, inspections to extreme
- Focuses on people
- Very experimental
- Fun, enthusiasm with discipline!

Crystal Programming

- Alistair Cockburn - consultant for IBM
- Crystal Light permits developers maximum individual preference
- XP assumes everyone is following tight, disciplined practices
- According to Cockburn:
  - XP is more productive through increased discipline but is harder for the team to follow
  - Crystal Clear permits greater individuality within the team and more relaxed work habits, for some loss in productivity
  - Crystal Clear is easier to adopt but XP provides better results if the team can handle it
  - A team can start with Crystal and move to XP, or start with XP and backup

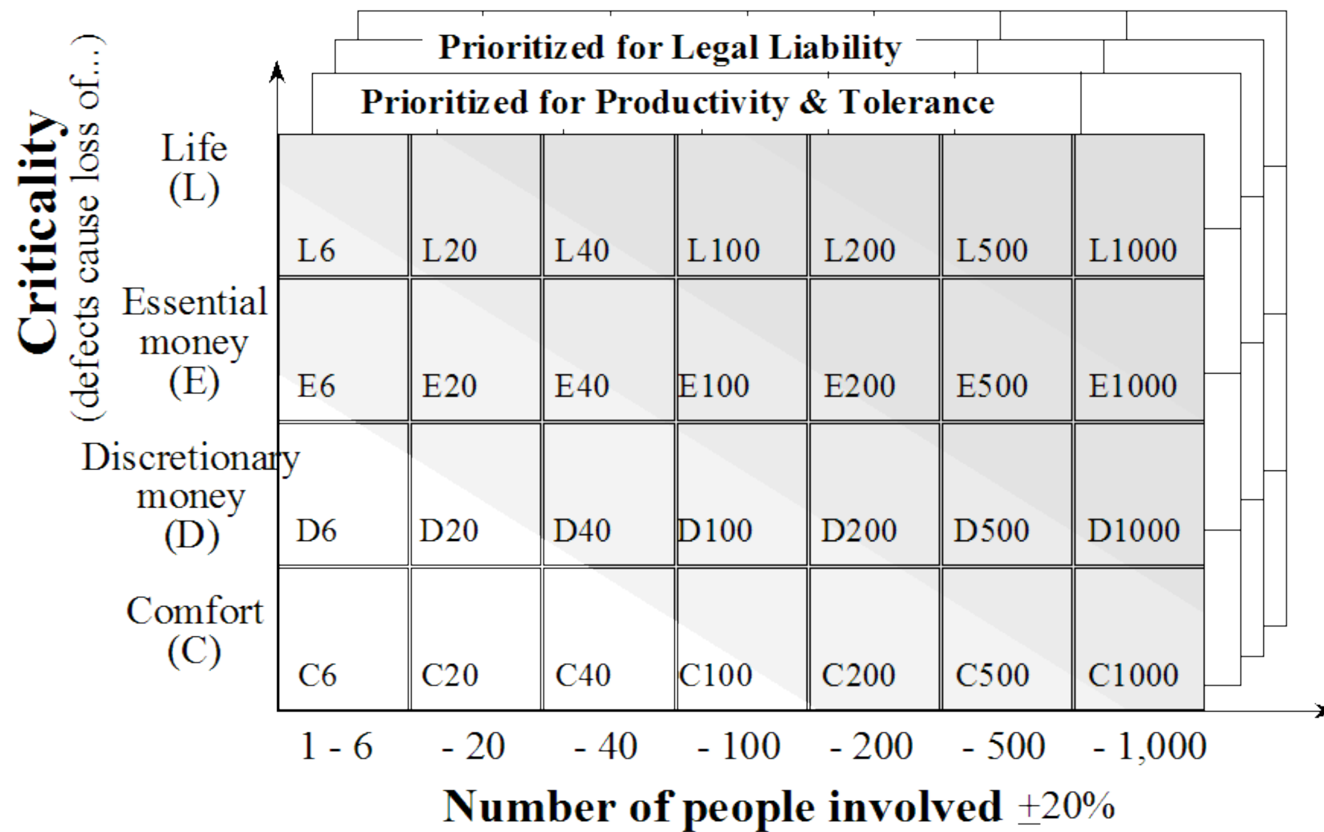
Cockburn's Principles

- Replace written documents with face to face interaction, reduce reliance on written products
- Deliver frequent, running, tested slices of the system rather than rely on promissory notes
- Different projects have different needs
  - As staff grows so does need to coordinate communication
  - As risk increases, increases the need for public scrutiny and decreases tolerance for personal stylistic variations
  - Some projects depend on time to market, others aim for traceability or legal liability protection --- differ in 9s

More on Crystal

- Selecting the variant of Crystal for your project:
  - 3 factors: communication load (staff size), system criticality and project priorities
  - Different planes represent different project priorities from time to market to legal liability
  - X-axis is staff size
  - Y-axis is damage effect
  - From this select Crystal light family of methodologies:
    - 2-6 person projects clear, 6-20 yellow, 20-40 orange
    - Vertical axis hardens each of the methodologies (but not much)
- "Software development is a cooperative game." "The endpoint of the game is an operating system. …"

Cockburn's Table



Street Programming - work in progress

- Just enough process, meeting adverse, improve on the 50%
- Match talent to task
- **Reality rules - no project is the same, every person is unique (find the joy)**
- Experience + apprenticeship (reinforcing)
- Long duration teams
- Customers in/ Management out
- Manage overdemanding customers & management - Kobayahsi Maru
- Process is there
- Don't scrimp on hardware, tools, environment, but don't decorate
- Continuous discovery
- Automate
- Test!!!
- Painless metrics that are used
- Think globally, code locally
- Quality begins with staff and management - inhomogeneity of technical understanding
- **Enlightened self interest**

Thought Problems (next time)

- Are lightweight methodologies, agile software development techniques short on process?
- How would you begin to experiment with agile methodologies, what are some of the project/staff characteristics?

So Far

- Software Process Models, Software Project Planning (woosh!), Requirements, Estimation, Risk Analysis, Multics case study, Architecture Reviews, Questionnaire Design
- Software Quality Assurance, Configuration Management and Testing, Architecture and Design, Software Engineering skills: Problem Solving, meeting, stat, … (and finished Arch and Design) and OO
- This Time: Lightweight Methodologies, XP
- Next Time: CHI and Human Factors (next two classes)

References

- Brooks
- Kent Beck, eXtreme Programming explained, Addison-Wesley, 1999, isbn = 0-201-61641-6
- http://alistair.cockburn.us/crystal
- http://agilemanifesto.org
- Steinberg and Palmer, Extreme Software Engineering: A Hands on Approach, Pearson Prentice Hall, 2004, isbn = 0-13-047381-2
- Boehm and Turner, Balancing Agility and Discipline, Addison-Wesley, 2004, isbn = 0-321-18612-5
- Martin Fowler, Refactoring: Improving the design of existing code, Addison-Wesley, 2000, isbn =0-201-48567-2