

# Class 1 CIS 573

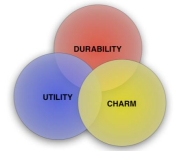
Gregg Vesonder  
University of Pennsylvania  
Penn Engineering - Computer & Information Science  
© 2009 Gregg Vesonder

# Roadmap- Class 1

- Class Mechanics
- My History
- Software Engineering Overview
- Software Process Models
- CMM and the SEI
- Project Management
- Software Engineering micro and macro aspects
- Software Engineering Certification
- Reading: S - chapters 1-4; Brooks - Preface, chapters 1, 7, 10 & 14
- Reading for next class: S - chapters 6-8 & 26; Brooks - chapters 2 & 8

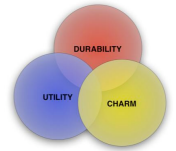
# Class Mechanics

- Review Syllabus - note dates
- Three texts plus supplementary reading
- 10 lectures
- mid-term 25%
- Final exam 25% of grade
- Log book 25% of grade
- Project 25% of grade
- email will be used, vesonder@cis.penn.edu
- Web site will have additional resources, especially lecture slides
  - <http://homepage.mac.com/vesonder>
  - <http://www.seas.upenn.edu/~vesonde/>
- Blog
- Project wiki - more on that
  - <http://cis573.wikispaces.com/>



# Log Book

- Preferably electronic
- Contains thoughts and insights about software engineering and how you practice it
- Should have at least a paragraph/week - at least 5 entries
- Review at least two each week in class or on blog- with text copy
- NOT CLASS NOTES!
- Examples on the blog
- Due at Session 10
- Method to this - should be part of your professional life, time to start!



# Class Project

- Teams: next page
- Presentation each class
  - Green, yellow, red -simplified model + gaps
  - Current pressing issues
  - What was done since last class
  - What will be done before next class
  - Gaps
- Project Presentation - **August 11<sup>th</sup>**

# Teams

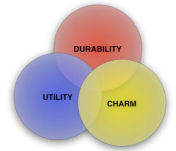
- Team 1 - Klein, Beck, Buchman, Richardson, Nunez
- Team 2- Wilmarth, Caputo, Xiang, Francis
- Team 3- Noronha, Fang, Treatman, Han
- Team 4-Whitehead, Liu, Chang, Ratnakar

# Projects

- Mashups
- Other Web Applications - Andersson, et.al.
- Expert System
- HCI system
- Examples of previous classes -
  - <http://spam.seas.upenn.edu>

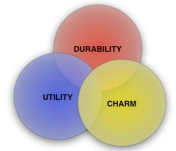
# Crucial

- Define a project - next Tuesday latest
- Roles and Responsibilities
- Maintain a project log (wiki)
- Schedule
- Project Plan/report - example on site



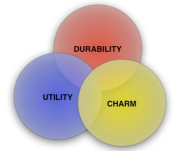
# Truth in Lending

- Brooks comments
- Slide comments -- see blog entry Slideways
- More Discussion
- "It's not that easy bein' green", --Kermit the Frog and Frank Sinatra



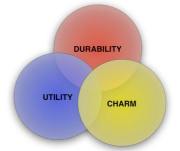
# Vesonder's Relevant Bio

- Software for 30+ years
- PhD in Cognitive Psychology - Computer modeling of learning and memory
- [Bell|AT&T] labs for 30 years
- Dozens of projects
- Architecture Reviewer and served software engineering corporate stint at Bell Labs
- Executive Director, Communication Science and AI Research, AT&T Labs - Research
- Adjunct Professor at University of Pennsylvania and Stevens Institute of Technology
- Now yours and characterize your experience in software engineering (none, some, a lot)



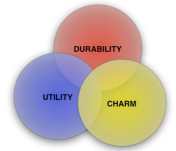
# My Log Book Entry

- Kobayashi Maru



# Views of Software Engineering

- Your view - we will review in the last class



# Birth of Software Engineering

- Born in 1968 at the NATO Software Engineering Conference - in response to the software crisis
- “The phrase ‘software engineering’ was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering” -p13 (8)

# Preface of NATO Conference

- Although much of the discussions were of a detailed technical nature, the report also contains sections reporting on discussions which will be of interest to a much wider audience. This holds for subjects like:
  - the problems of achieving **sufficient reliability** in the data systems which are becoming increasingly integrated into the central activities of modern society
  - the difficulties of **meeting schedules and specifications** on large software projects
  - the **education** of software (or data systems) engineers
  - the highly controversial question of whether software should be priced separately from hardware

# An Example

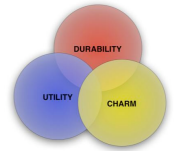
- 1983: FAA AAS (Advanced Automation System) - overhaul of computers, radar units, software and communications network
- 1988: contract awarded to IBM with budget of \$4.8B completed by 1994
- December 1990: project was 19 months behind schedule
- Late 1992: project was 33 months behind schedule, cost now \$5.1B, project scaled back
- December 1993: cost of smaller project \$5.9B
- April 1994: independent commission project has "high risk of failure"
- June 1994 project shifted to Lockheed Martin, further slimmed, now \$6B and estimated for completion in 2000
- September 1996 Lockheed announced a new project STARS (Standard Terminal Automation Replacement System) - estimates of AAS costs ranged from \$7.6-23B.

# Software Crisis Persists

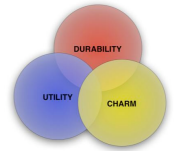
[http://www.standishgroup.com/sample\\_research/chaos\\_1994\\_1.php](http://www.standishgroup.com/sample_research/chaos_1994_1.php)

- (in mid '90s) \$250B over 175,000 projects
  - Average cost of projects: large company \$2.3M, medium \$1.3M small \$4M
- Almost a third will be cancelled before they are completed
- Over half will cost 189% of their current estimates
- Only 16.2% will be finished on time and on budget
- IBM FAA Air Traffic Control project, Mars probe, Denver airport baggage handler ...
- Add your own
- From paint to building -- long life span --**increasing demand for COBOL**
- the University of Oxford has this remark in their Engineering Science description, "The qualities of a good engineer include not only a high degree of technical competence but also imagination, strength of purpose, commonsense - and a social conscience."
- Sommerville places a finer point on issue with discussions of sociotechnical systems and critical systems





What are the major issues  
with modern software?



# Sommerville - Key Challenges

- Heterogeneity - integrate new software with legacy software
- Delivery - time is the devil!
- Trust - it is everywhere

# Sommerville

- Fundamental notions of process and system organization (and human factor) are the essence of software engineering
- There is no ideal approach
- Engineering discipline
- All aspects of software production
  - Software = programs + documentation + configuration data + ...
- *Component based software engineering - parts already exist*

# Views of Software Engineering

- SEI:
  - Engineering is the systematic application of scientific knowledge in creating and building cost-effective solutions to practical problems in the service of mankind.
  - Software engineering is that form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems.

# IEEE and Software Engineering

1. The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is the application of engineering to software
2. The study of approaches as in (1)

The study of conceiving, building and maintaining **successful** software products And systems.

# Software Engineering Knowledge

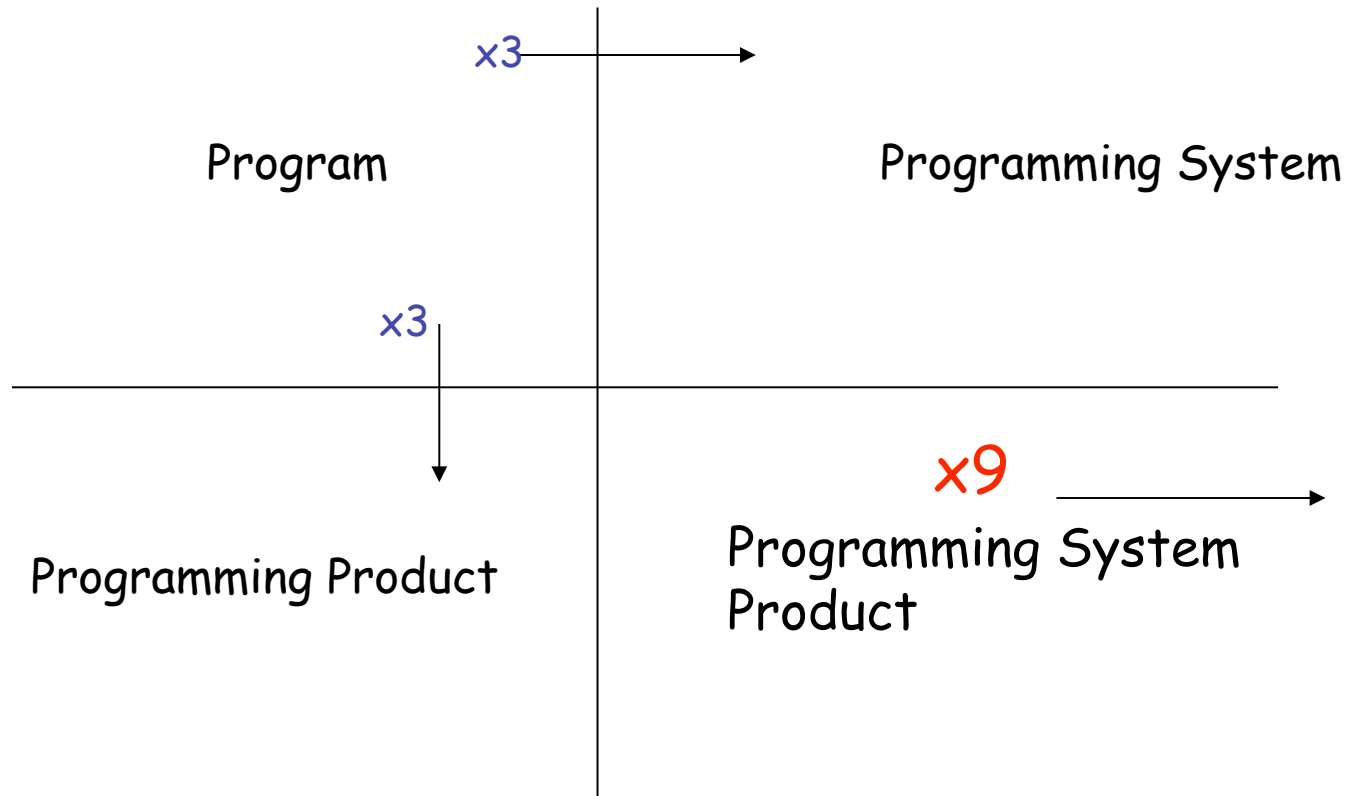
- SWEBOK, SoftWare Engineering Body Of Knowledge:
  - Software requirements analysis
  - Software design
  - Software construction
  - Software testing
  - Software maintenance
  - Software configuration management
  - Software quality analysis
  - Software engineering management
  - Software engineering infrastructure
  - Software engineering process

# Reality Check

- There is theory
- There is engineering
- There is state of the art
- There is state of the practice
- There is reality - whatever it takes to conceive, design, build and maintain a successful software system product
- **Your biases?**

**"In theory there is no difference between theory and practice. In practice there is."  
--Yogi Berra**

# Brooks: System Production Differs from Coding



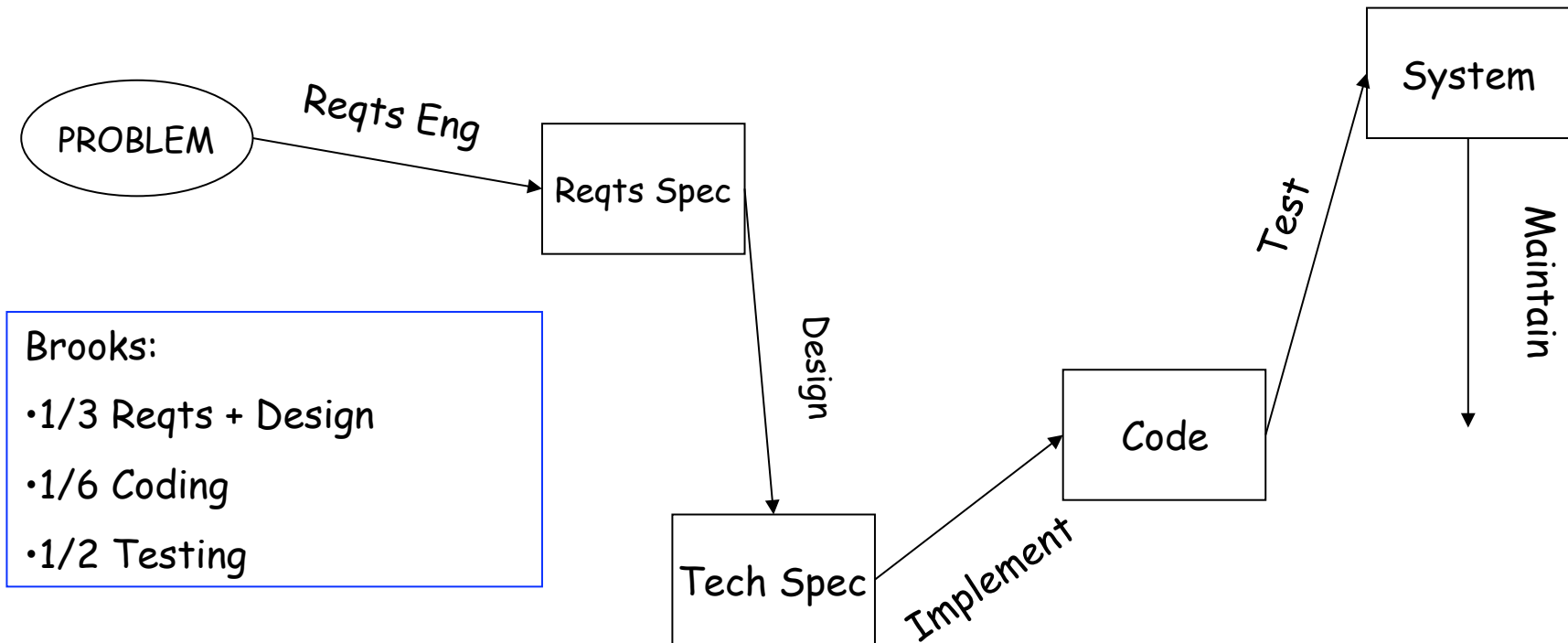
# Software Process Models

- The cost of constructing most software occurs during development (broadly defined, development is not equivalent to coding!) and not during production
- Process is a series of predictable steps, a roadmap
- We will cover:
  - Simplified -> waterfall (single step, once through)
  - Prototyping
  - Incremental
  - RAD
  - Spiral (evolutionary -- avoids defining all requirements)
  - Component based
  - (Agile Models)

## But First

- Code and Fix, Do Until Done Models
- No planning, general idea of product, informal "design" mostly through code
- Code, debug, test until ready for release
- No way to tell you are done or if requirements met

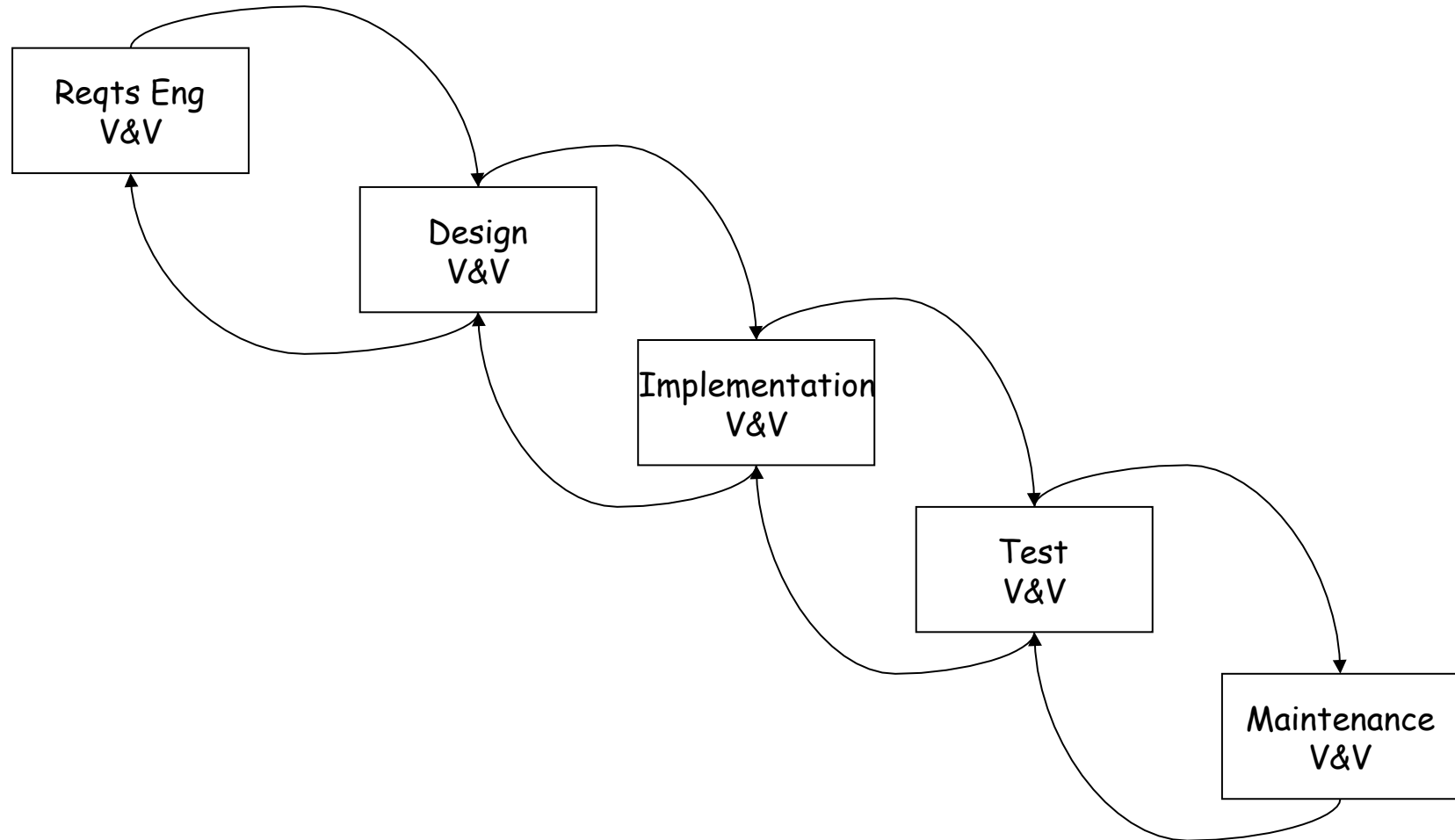
# Simplified Model



# Main Milestones

- Requirements engineering -> baselined Requirements Specification
- Design -> baselined Technical Specification
- Implementation -> baselined Code
- Test -> test report

# Waterfall Model (Royce 1970)



# Development Activities by Lifecycle Phase (p.51)

	Design Phase	Coding Phase	Integration Test Phase	Acceptance Test Phase
<b>Integration Test Activity</b>	4.7	43.4	26.1	25.8
<b>Coding Activity</b>	6.9	70.3	15.9	6.9
<b>Design Activity</b>	<b>49.2</b>	34.1	10.3	6.4

e.g., only 50% of Design occurs in Design Phase!

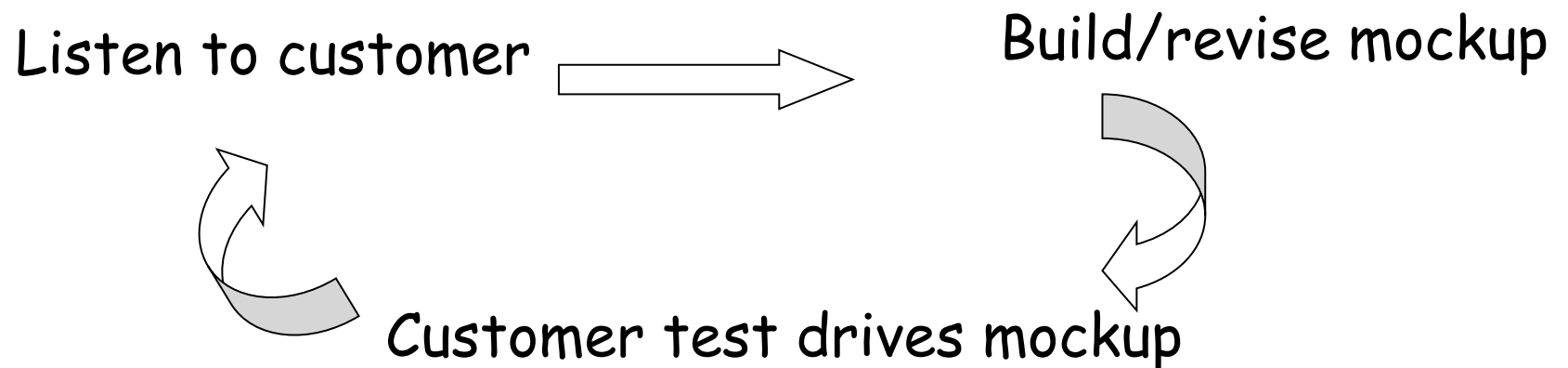
# Review of Waterfall

- Not change tolerant - premature freezing of requirements
- Difficult for Customer to state all requirements upfront - no customer preview until late
- Document driven - excessive and expensive
- System not available until late in the process - false comfort in X% done
- Strong Development - Maintenance Distinction
- Came from an era when coding was difficult, expensive
- *Energy before system is built*
- Still being used

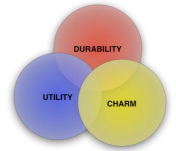
# Why Waterfall is Still Used

- Familiar to customers, steps make intuitive sense - easy to understand
- Structure for new staff or teams - tight control by project management
- Requirements are stable
- It is documented

# Prototyping-1



When finished: **Design, Implement, Test, Maintain**



# On Prototyping

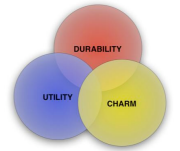
- Evolutionary versus throwaway prototypes
- Prototyping takes advantage of high level languages, sacrifices efficiency for speed
- Great when few initial requirements
- People (dev and users) like prototype
- Danger of feature creep
- Documentation, performance of final system may suffer - perceived lack of discipline
- Customer and management may think it is done
- Quality can go either way - process not visible and may suffer from premature structure
- Requires experienced developers

# Advantages of Prototyping

- Evolving requirements are visible in the system
- Minimizes miscommunication, language gap barrier
- Spec is prototype
- Progress can be seen - non trivial
- Early user involvement may increase quality

# Disadvantages

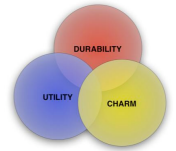
- Has a bad rap with some managers
- Performance, documentation, quality issues
- Prototype environment may not equal target deployment environment
- Prototype does not equal finished system, often tough to convince users
- Potential for much coding, little analysis



# Prototyping: Love it or Hate it? Why?

# Incremental

- Functionality of system is produced and delivered in small increments
- “prototyping + waterfall” - but focuses on delivery of operational product
- Focuses on **assigning priorities to features for each release** - Rolling Stones ... don't always get what you want ... you get what you need
- Especially useful with small staff, to manage technical risks and to build to current capability (e.g., hardware)
- Not good when delivery/installation is expensive
- Many systems require a base or infrastructure layer that cannot be done incrementally



# RAD- Rapid Application Development

- Incremental development where time is driver
- Introduced by IBM in the 80's - James Martin's book
- JRPs (Joint Requirements Planning) - requirements triaged, structured discussion of requirements
- JADs (Joint Application Design)-developers and users work together through prototyping to a finalized design
- Product developers are SWAT (Skilled with Advanced Tools) team - highly dependent on productivity tools (generators)
- Cutover- final testing of system takes place, users trained, system installed
- Best used in information systems where technical risks are not high
- Typically 60-90 days

# RAD Advantages

- Tools reduce cycle time - when describing RAD it is usually paired with some software tool (Oracle, Visual Basic, ...)
- Project team usually knows problem domain, key
  - Developers are willing to dive deeply into domain - key success factor in any model
- Time-box, usually 60 days, bounds development - maintain interval, slip functionality (what is essential)
- Customer involvement
- Installation and user training are an explicit part of the process

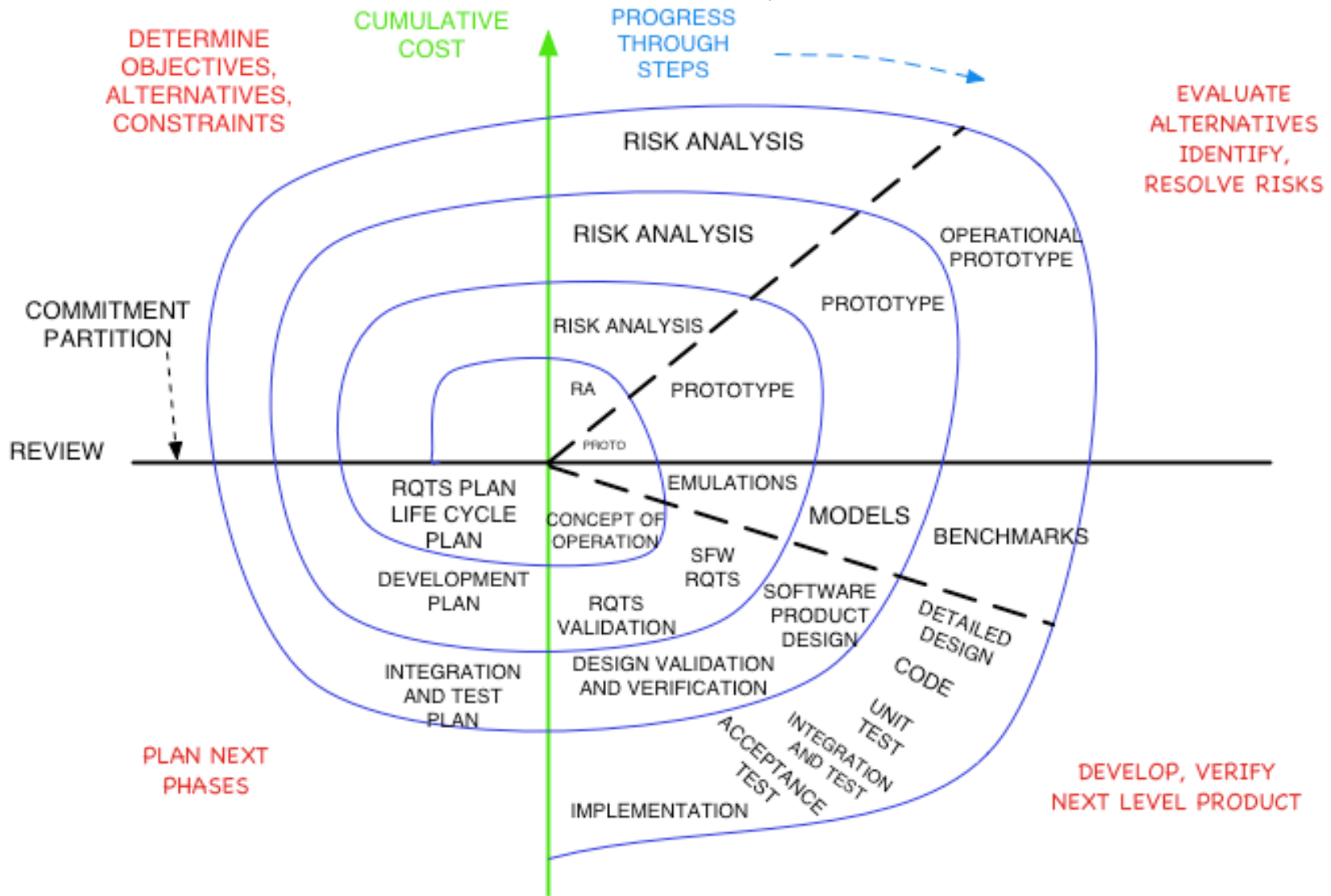
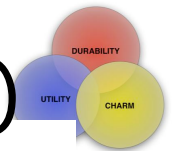
# RAD Disadvantages

- Users have to be involved
- Technical risks should be low
- Developers have to be very good and experienced with RAD - good developers are a success factor in any model
- System can be modularized in 2 month chunks
  - Users have to be willing to deal with constant involvement and change
- Difficult to attach to legacy systems that did not use RAD

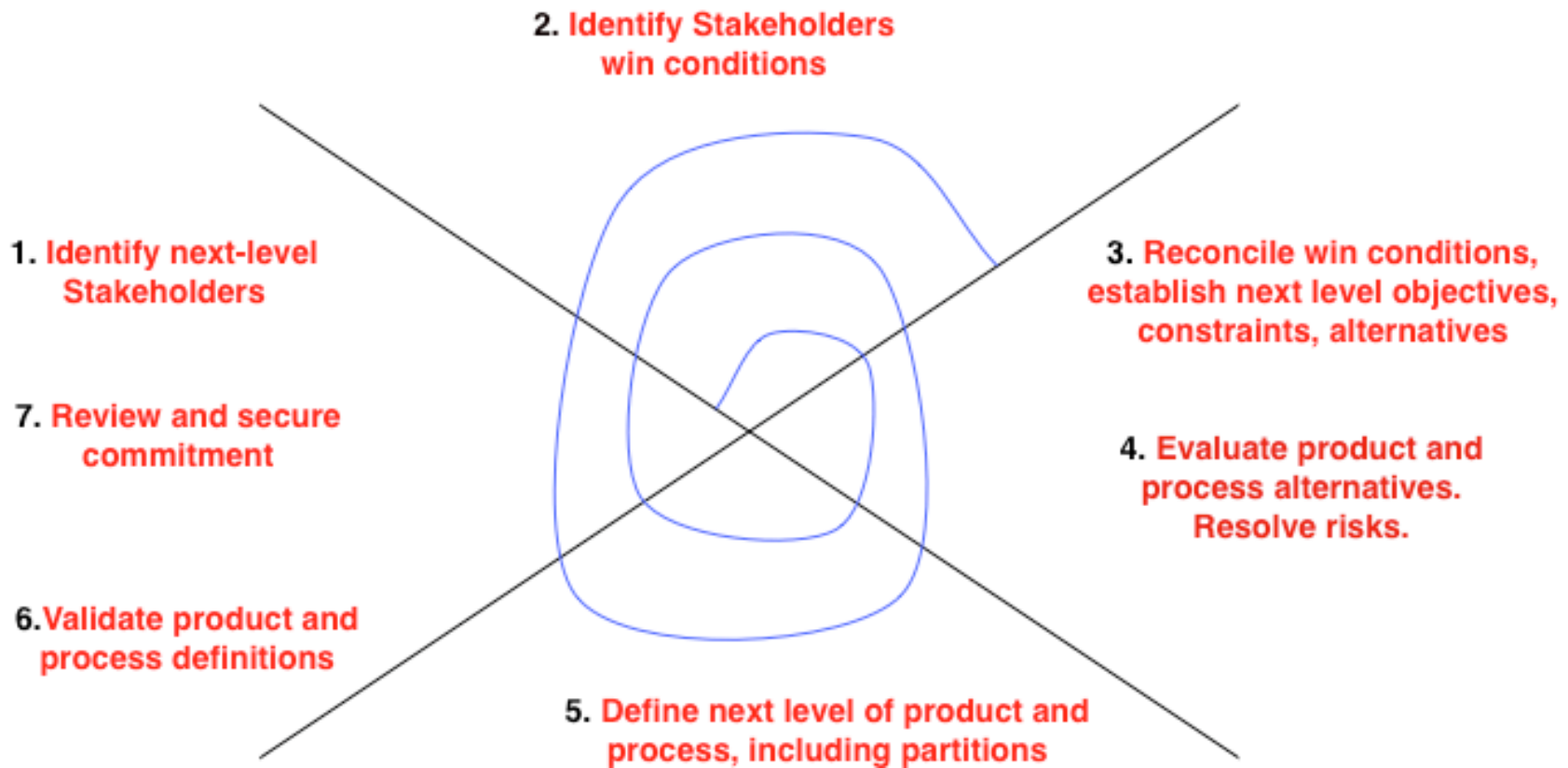
# Spiral Development

- Recognizes that at each iteration you go through most phases
- **At each iteration you pinpoint sub-problem with highest risk and solve** (highest risk versus highest priority feature - could converge if you are selling software)
- Other models are subsumed

# Spiral Model (Boehm)



# WinWin Spiral Model Variant

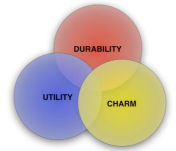


# Spiral Advantages

- Risk analysis may uncover show stoppers early
- Chunks development so that it is affordable
- Waterfall like characteristics add some discipline, management control
- Lots of feedback from all stakeholders

# Spiral Disadvantages

- Expensive for small projects - more mechanism than proto
- Complex and requires risk assessment expertise
- Development is on again/off again so the other stages can be accomplished - in proto development is continuous.
- Not really used as much as folks claim



# Model Based Software Development

- What is old is new again, collection of simulations
  - Great if you can do it (and it is done)
- A dream in some ways
- The prototype can be the model
- Domain Driven Design - Eric Evans, approach for agile processes

# Component Based Software Development

- Process
  - Component analysis -search for components that match spec
  - Requirements modification -reflect selected components
  - System design and *reuse*
  - Development and integration
- Related to integrated web services

# Component Based Software Development

0 25 50 75 100

## WATERFALL



Sommerville  
Fig. 1.2

## ITERATIVE

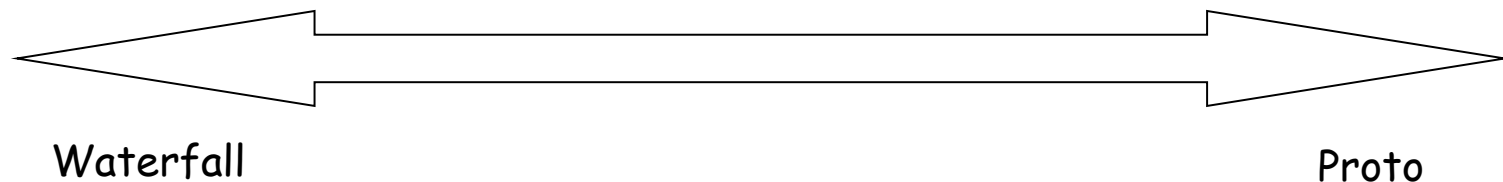


## COMPONENT BASED



# Project Space - van Vliet, p186

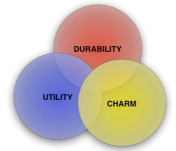
	Realization	Allocation	Design	Exploration
Product Certainty	HIGH	HIGH	HIGH	LOW
Process Certainty	HIGH	HIGH	LOW	LOW
Resource Certainty	HIGH	LOW	LOW	LOW



# Requirements Issues

(adapted from Futrell, et.al.(2002) van Vliet p 147)

Requirements	Water	Proto	Spiral	RAD	Inc
Well known	+	-	-	+	-
Defined early	+	-	-	+	+
Change often	-	+	+	-	-
Proof of concept	-	+	+	+	-
Complex system	-	+	+	-	+
Early Functionality	-	+	+	+	+



# Agile Manifesto

<http://agilemanifesto.org>

Manifesto for Agile Software Development

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

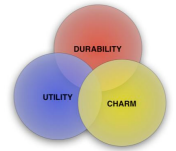
- Customer value
- Creating a culture of innovation, creation and rapid delivery
- **Appealing to skilled, talented staff**
- Facilitating collaboration, knowledge sharing and decision making
- Reducing, time, cost and defect levels significantly.

# 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

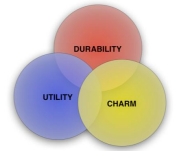
- If code review is good, do it all the time, pair programming
- If testing is good, everyone will test (unit testing), customers (functional testing)
- If design is good, it is every day for everyone (refactoring)
- If simplicity is good, we'll leave system with simplest design that supports the functionality - 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

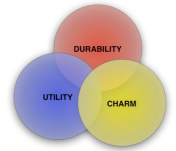
# A Day in the Life of an XPer

- 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

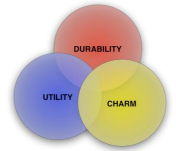


# Fundamental Principles of XP

- 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**



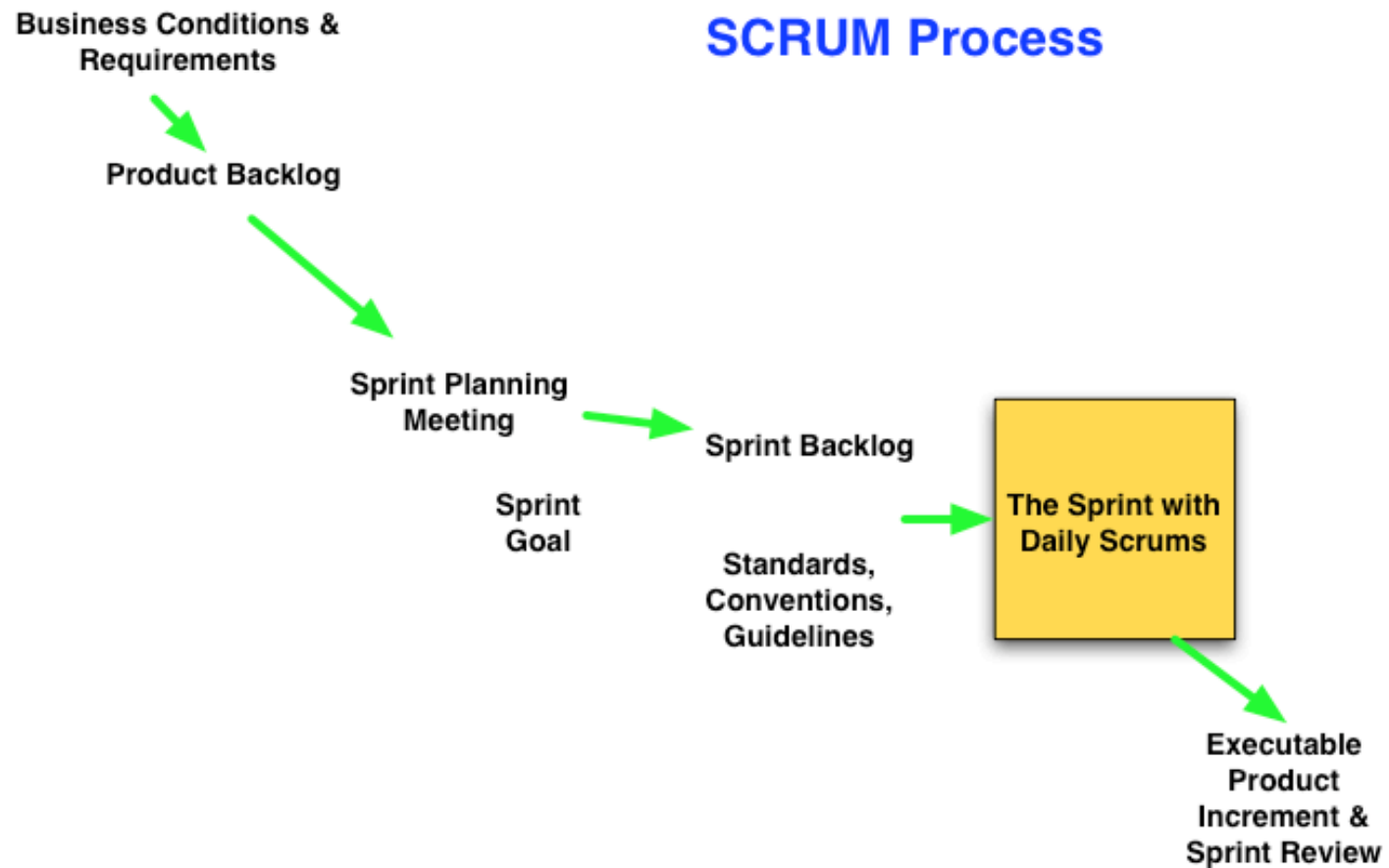
# Key Aspects of XP

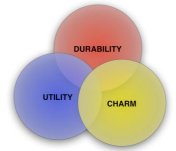
- **Whole team - development + customer**
- **Metaphor** - everyone use common analogy in discussing the system, e.g., desktop metaphor - Scandinavian School
- **The Planning Game** - specify the next step of development and, as project progresses, provides better and better picture of what will be delivered. **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**

# SCRUM





# SCRUM Roles

- Stakeholders
- Product Owner
- SCRUM Master
  - Works with Customers and management, works with the team
  - Key interface
- SCRUM Team
  - Experienced Developer
  - Multidisciplinary
  - - "not my job"
  - 8 at most

# SCRUM Artifacts

- Product backlog - evolving list of business and technical functionality ordered by priority that needs to be in the product
  - Product owner controls it
  - Iterative estimations included on effort
- Sprint goal - an objective met through implementation of Product backlog
  - Wiggle room and test
  - Developed at Sprint Planning meeting
- Sprint backlog
  - Defined and modified only by sprint team
  - A list of tasks it has to meet to achieve the sprint goal
  - 4-16 hours in duration
  - Modified during the sprint

# Sprint Planning Meeting

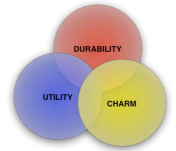
- Actually two meetings
  - Meeting 1 - Product owner meets with stakeholders to define functionality for next sprint. A product backlog is selected and a sprint goal is crafted from results of that meeting
  - Meeting 2 - define sprint backlog to meet sprint goal

# The Sprint

- 30 days in duration
- Ends with a Sprint Review
- Daily SCRUMs
- SCRUM master in control
- Empirical process from process control theory
- No person outside the team can change the scope or nature of the work

# The SCRUM

- Held daily for ideally 15 minutes, all team members must participate
- SCRUM master runs meeting
- 3 questions:
  - What was done in last 24 hrs.
  - What will be done in next 24 hrs
  - Impediments - SCRUM master responsible for resolving impediments
- Visitors are permitted but must be on periphery - not permitted to talk
- May lead to follow-up meetings - SCRUM is informational



## More on the SCRUM

- The SCRUM master makes sure it starts on time and that everything is ready - participants on conference call, ...
- Impediments are boarded - resolving top priority for SCRUM master
- Pigs and chickens
  - Pig and chicken story: "Ham & Eggs"- commitment vs. involvement. Team members are \_\_\_\_\_
- Daily builds -- regression testing

# Sprint Review

- Four hour meeting at end of Sprint
- SCRUM manager provides overview of sprint
- Team presents managers and stakeholders the result of the sprint and it includes demos
- Provides a picture of strengths and weaknesses of the sprint
- Minimal preparation, less than two hours for team, 29 days work, 1 day reporting
- Working and informational meeting
  - Not critical, action oriented

# SCRUM Factoids

- Open office design encouraged
- Best equipment and tools
- XP can be used within - SCRUM emphasizes team operation, XP work unit operation
- Sprints can have abnormal terminations (change of goals -- show stopper) but rarely done because of short increment
- Borrows a lot - "time boxes", ...

# Maintenance vs. Continuing Development

- During the system lifecycle there is a tradeoff on placing resources on **progressive and antiregressive activities**
- Maintenance - Development split is sometimes enforced by the organization and sometimes because of failure to use antiregressive activities or fear of restructuring (due to age)
- Can't avoid it these days - systems under continuous development, iPhone release 3.0

# Software Engineering Institute

- <http://www.sei.cmu.edu/>
- "The SEI promotes the evolution of software engineering from an ad hoc, labor intensive activity to a discipline that is well managed and supported by technology."
- Three themes:
  - Move to the left
  - Reuse everything
  - Never make the same mistake twice - Senator Hollings, "There is no education in the second kick of a mule."

# Capability Maturity Model

- A roadmap for software process improvement (Paulk 1999)
- Describe an evolutionary process from ad hoc to maturity and discipline
- Used in conjunction with the SEI's IDEAL model
  - Initiating the improvement program
  - Diagnosing the current state of practice
  - Establishing the plans for the improvement program
  - Acting on the plans and recommended improvement
  - Learning from it

# CMM (Paulk, 1999)

LEVEL	FOCUS	KEY PROCESS AREAS
5 Optimizing	Continual Process Improvement	Defect prevention, Technology change management, Process change management
4 Managed	Product and process quality	Quantitative process management, Software quality management
3 Defined	Engineering processes and organizational support	Organization process focus, Organization process definition, Training program, Integrated software management, Software product engineering, Intergroup coordination, Peer reviews
2 Repeatable	Project management processed	Requirements management, Software project planning, software project tracking and oversight, Software subcontract management, Software QA, Software configuration management
1 Initial	Competent people	And heroics

# CMM Translation

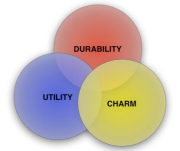
1. Initial - adhoc, chaotic, few processes defined, success is a function of individual effort
2. Repeatable- basic project management tracks costs, schedule and functionality, repeatable processes
3. Defined- Defined, documented organization wide process- all projects use it
4. Managed- Measures of software process and Quality are collected, products and processes are quantitatively understood and controlled using detailed measures
5. Optimizing- Continuous process improvement enabled by quantitative measurement and from testing innovative ideas and technologies

## CMM -> CMMI

- Other CMMs were developed, e.g, Systems Engineering, integrated product development, software acquisition, ...
- Was activity based not results driven
- Biased towards waterfall model - lots of artifacts
- CMM Integration Project was an attempt to integrate and improve

# CMMI Levels

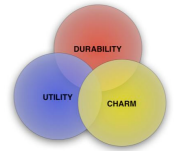
- Level 1 (initial) process maturity represented by unpredictable results: ad hoc approaches depending on the **skill of the team**
- Level 2 (managed) repeatable project performance: process focus is on project level activities
- Level 3 (defined) improving project performance across the organization
- Level 4 (quantitatively managed) improving organizational performance
- Level 5 (optimized) continuous process improvement, rapid adaptability



CMM/CMMI  
Love it?  
Hate it?  
CMM/CMMI what?

# Differences

- TSP has detailed process control, quality and performance metrics
- TSP has large number of scripts, forms roles and exit criteria
- TSP has established reports for tasks and phases and keeps history of activity
- TSP formal/contractual relationship with customer
- XP metrics are product oriented estimating progress and future iterations -- performance and quality responsibility of pairs
- Few guidelines and strict practices
- XP reporting informal and has historian but rather than activity focus it is a post mortem focus (newspaper vs analysis)
- XP collocated, collegial relationship with customer



# Project Management

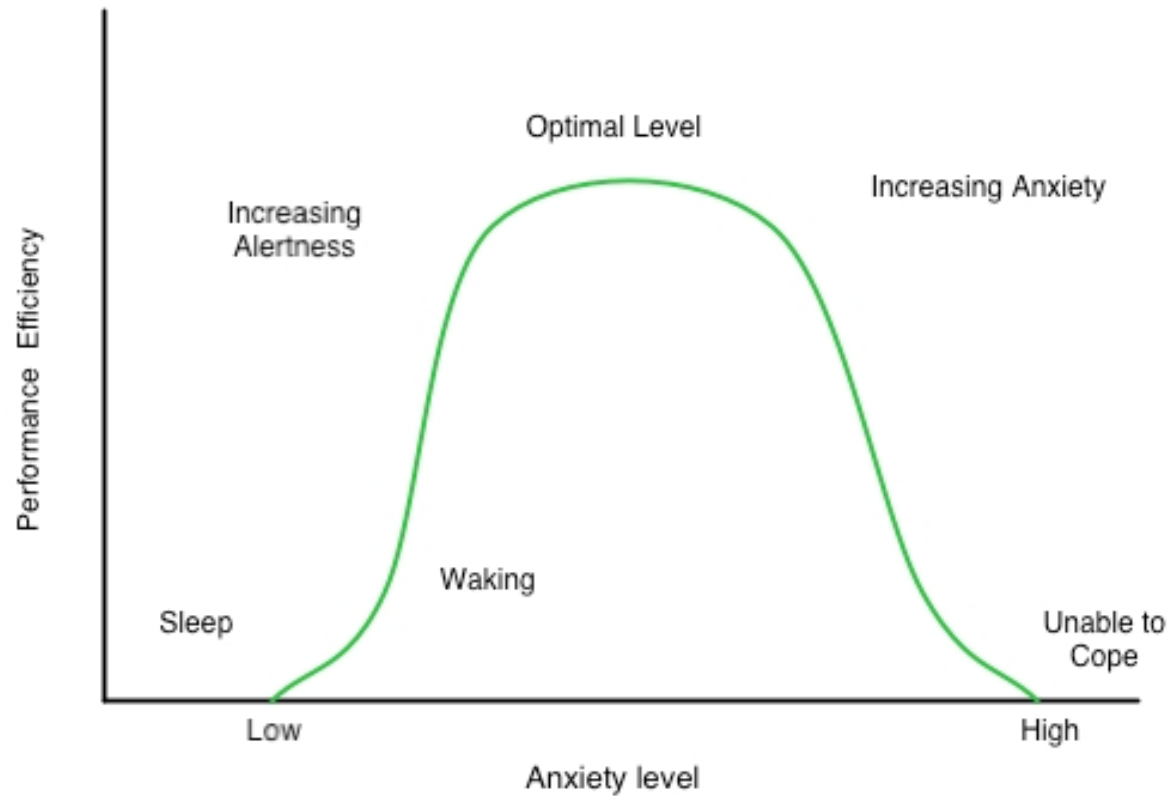
- Focus on the team
- Motivation?
- Communication
- Management heuristics and wisdom

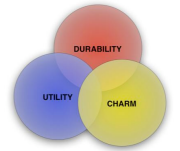
# Why? (Brooks)

- Making things
- Making things useful to others
- Joys of grappling with complexity
- Always learning
- Mind stuff
- (some downside: perfection, control, debugging)
- Applies to most systems, not just large systems, especially today since we are building "large" systems with small teams
- Tracy Kidder, Soul of a New Machine, Pulitzer Prize



# Yerkes Dodson

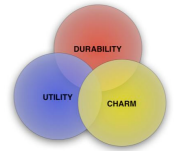




# IT IS ALL ABOUT PEOPLE!

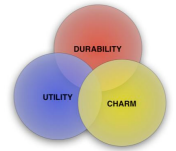
Hardware follows Moore's Law human's often do not.

-Moore's law is the empirical observation that at our rate of technological development, the complexity of an integrated circuit, with respect to minimum component cost will double in about 24 months -  
Wikipedia



# Project Planning and Control

- Control is **THE KEY** for the lead manager
- Requires clearly defined Roles and Responsibilities
- Must be able to measure the progress of the product and the quality and requirements covered in the product
- Communication
- Documentation

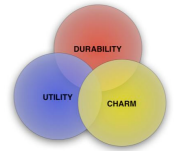


# Why did the Tower of Babel Fall?

- Babel had all the prerequisites for success:
  - Clear mission
  - Adequate manpower
  - Adequate materials
  - Lots of time
  - Appropriate technology
- BUT lacked communication and organization
- **How should teams communicate? In as many ways as possible!** Informal, regular meetings (group and project) and project work book

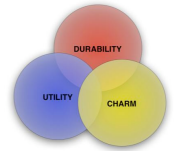
# The Project Workbook

- (easier with the web and wiki's <http://www.wiki.org>)
- Structure is imposed on documents that all documents use
- Control material (number it) but make it available to entire team
- A change summary should highlight what is new



# All Projects Should

- Use a Development Plan Approach (write and follow):
  - What will you do? *Project Description*
  - How will you do it? *Process, Tools, Techniques, Methods...*
  - What do you depend on? *Related Projects, Partners, Teams, Systems, Competitors...*
  - When will you be done? *Schedule*
  - Who will do what? *Roles and Responsibilities*
  - What will you measure, how will you use it? *Metrics*



# Wiki

- A web site on which anyone can contribute pages, a wiki page also can be edited by anyone
- Produced by collaborative software called wiki
- WikiWiki comes from Hawaiian meaning quick or superfast
- Wiki pages have very simple markup and can be edited in web pages
- Links are created using a link pattern initially CamelCase (e.g. TableOfContents)
- Software available on web - google WikiWikiWeb, the 1st wiki.

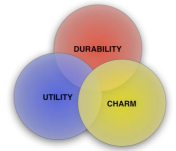
# Chapter 10, The Documentary Hypothesis

- In praise of certain aspects of bureaucracy
- Engineering manager as a flywheel absorbing energy from market and management aka "air cover"
- Documents for a software project:
  - What1? - goals, constraints priorities
  - What2? - product specification, starts as a proposal, ends as a manual and internal documentation
  - When? - Schedule
  - How Much? - Budget
  - Where? - Space allocation (a killer)
  - Who? - organization chart

# Conway's Law

- Organizations must be flexible
- Manager's task - develop a plan and realize it
- Formal Documentation helps:
  - Writing clarifies for all
  - Communicates decisions to others, lightens load for manager whose job is communication

Organizations which design systems are constrained to produce systems which are copies of the communication structure of these organizations.



# The Organization

- Tree like structure of organization
- The producer and the technical/director-architect
  - Producer: assembles team, divides work and establishes schedules
  - Tech/director architect is surgeon - the job worst done by management is to use technical genius not strong on management talent
- Small projects, same person
- Larger projects separate. Producer as boss, Director as right hand man - large projects
  - Establishing director's authority is difficult - symbols / dual ladder
- Director boss, producer right hand man - Brooks okay for small teams - "inhomogeneity of technical understanding" issue

# 3 Little Bears and Documentation



# Brooks 14

## Hatching a Catastrophe

“None love the bearer  
of bad news”

“How does a project get to be a year late?”

“... one day at a time”

- Most software disaster are due to “termites” not “tornadoes”
- Major calamities are easier to handle
- But snow, jury duty, illness, late hardware delivery, ...

# Brooks -14

- Control
  - Have a schedule - I force it!
  - List events, milestones as concrete, specific, unambiguous, measurable events and are "100%" events
  - Chronic schedule slippage is a morale killer at all levels
  - Hustle provides the cushion
  - Get excited about one day slips
  - Critical event/path charts are useful, all tasks and events are not equal
- Every manager needs 2 kinds of information:
  - Exceptions to plan requiring action
  - Status picture for education

# Micro and Macro Software Engineering

- Micro - deals with the individual software engineer or small team.
  - Agile methodology (class 6), much more
- Macro deals with team structure, organization structure, industry structure
  - Much of what we discussed today

# Software Engineering Certification/Licensing

- Always a topic that causes debate
  - Certification is a voluntary process administered by a profession
  - Licensing is a mandatory process administered by a government authority - state level in US
- Y2K problem in 90's stimulated the debate
- Texas has licensing

# Most Engineers are not Licensed

- Nancy Mead, "Issues in licensing and certification of software engineers," SEI

Discipline	Licensed
Civil	44%
Mechanical	23%
Electrical	9%
Chemical	8%
All Engineers	18%

# Certification/Licensing

- Pros
  - Provide companies a basis for hiring
  - It will happen anyway, we should guide the process (control it)
  - Protect public from harm
  - Regularize the field
  - Force CS departments to teach good engineering practices
  - Awareness of professional and legal responsibilities
- Cons
  - Premature - no accepted body of knowledge
  - Beware of the certification bodies - vested interest (general malaise)
  - Many companies (Microsoft ...) are quite competent in knowing what they need, market should certify
  - Issue is decertification - people who do not have degrees - sorry Bill Gates!
  - Licensing (MD) single person does task, teams in software
  - Process takes so long - 10 year old technology

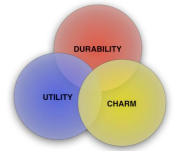
# Survey of **Computing Professionals** in mid '90's

- Value of certification:
  - Very valuable 50.3%
  - Fairly valuable 31.9%
  - Somewhat valuable 16.6%
  - Not valuable 1.2%
- Reasons for seeking certification:
  - Advancement in profession 41.7%
  - Advancement in current job 17.2%
  - Prepare for new job 9.4%
- Perceived results of certification - more credibility with organization, 24.2% and customers, 23.6%

From Mead SEI paper

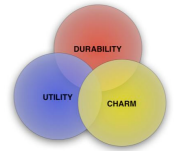
# Why I Waver

- Generally against licensing but:
  - Meager prospects for continuing education in an economically squeezed corporate environment
  - Unknown effect of outsourcing on software profession
    - Regulating our suppliers, but then what about us? Especially the "handlers" - issue of education and experience.
  - Lack of a forum for practitioners and theorists on what should be happening
  - Issue of who controls the licensing if it is inevitable
- However it is rough to legislate the above
- And then there is the issue of Professional Unionization



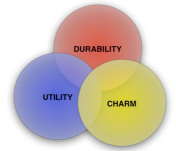
# Software Engineering Ethics

- Book describes disasters due to failures in software engineering.
- Software projects are pressure filled
- Software projects rely on relationships and trust
- IEEE Computer Society and ACM have developed a software engineering code of ethics with eight principles
- Think of the roles software plays in your life - health, transportation, finances, ...



# SE Code of Ethics

1. Public - shall act consistently with the public interest
2. Client and employer - shall act in a manner that is in the best interests of their client and employer and that is consistent with the public interest
3. Product - shall ensure that their products and related modifications meet the highest professional standards possible
4. Judgment - shall maintain integrity and independence in their professional judgment
5. Management - shall subscribe to promote an ethical approach to the management of software development and maintenance
6. Profession - shall advance the integrity and reputation of their profession consistent with the public interest
7. Colleagues - shall be fair to and supportive of their colleagues
8. Self - shall participate in lifelong learning regarding the practice of their profession and promote an ethical approach to the practice of the profession

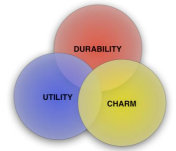


# Thought Problems

- You are part of an off shore development organization that has just been assigned a project from a new company in a new domain. There is a 12 hour time difference. What model?
- You are part of NASA's program for making cost effective interplanetary, multiuse robotics platforms - what CMM level should you chose?

# This Class

- The Class
- Software Engineering
- Software Process Models
- SEI & CMM
- Project management
- Software Engineering certification and responsibility



## Next Time

- Requirements elicitation and representation
- Risk management
- Software project estimation

# Resources

- Futrell, Shafer & Shafer, Quality software project management, Prentice Hall, 2002, ISBN 0-13-091297-2
- van Vliet
- E. Evans. Domain-Driven Design: Tackling complexity in the heart of software, Addison-Wesley, 2004, ISBN 0-321-12521-5
- Van Vliet, H. Software Engineering: Principles and Practice, Wiley, 2000.
- Royce, W. "CMM vs. CMMI: From Conventional to Modern Software Management," Rational Edge, 2002.
- Others embedded in text