Class 1 CS540

Gregg Vesonder

Stevens Institute of Technology

Roadmap- Class 1

- My history
- Class mechanics
- Quantitative Software Engineering
- Software Process Models
- CMM and the SEI
- Ethics of our profession
- Reading: first chapter in BY and chapters 10 & 29 in C
- Reading for next week: chapter 3 in BY, preface and chapter 1 in Brooks, chapter 35 in C

Vesonder's Relevant Bio

- Software for 30+ years
- PhD in Cognitive Psychology - Computer modeling of learning and memory
- [Bell|AT&T] labs for 25 years
- Dozens of projects
- Reviewer and served in Software Technology Center
- Stevens: CS540 (web version too), CS565
- University of Pennsylvania: Software Engineering & Human Computer Interaction
- Your turn -- intro and expectations

Class Mechanics

- Review Syllabus - note dates
- Three texts plus supplementary reading
- 14 lectures + Final
- 2 tests (each 25% of grade)
- Final - 35%
- Participation and Log book - 5% & 10%
- Attendance
- email will be used, vesonder@mac.com
- Web site will have additional resources, especially lecture slides
- Testing will be from books, supplemental readings and lectures
- Blog http://homepage.mac.com/vesonder
- Office hours by appointment

Class Mechanics - 2

- Syllabus is proposed schedule
- For two tests, first half of class is lecture
- Final is complete period
- Note Bernstein and Yuhas book at bookstore by 9/9 (preprints), Constantine (?)

Log Book

- Preferably a bound book
- Contains thoughts and insights about software engineering and how you practice it, especially quantitatively
- Should have at least a paragraph/week (5 entries)
- Review at least two each week in class or on blog- with text copy
- NOT CLASS NOTES!
- Hand in on November 21st, returned at exam
- Method to this - should be part of your professional life, time to start!

Policies

- Cheating is not tolerated - unfortunate that I have to mention this
- On grades …

Views of Software Engineering

- Your view - we will review in the last class

Birth of SE

- The software crisis, NATO conference, autumn 1968, Garmisch, Germany
- Origin of term software engineering
- http://homepages.cs.ncl.ac.uk/brian.randell/NATO/index.html

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

Views of Software Engineering

- Bernstein and Yuhas: "..think like an engineer, especially for software"
  - SE practices make development of software:
    - Less chaotic
    - Reliably repeatable
    - More humane
- Emphasis on simplification, trustworthiness, risk assessment and architecture

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.

Quantitative Software Engineering

- "Quantitative Software Engineering is an analytical approach to producing reliable software products within budget and on time" - Stevens program
- Which matches the IEEE definition:
  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)

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

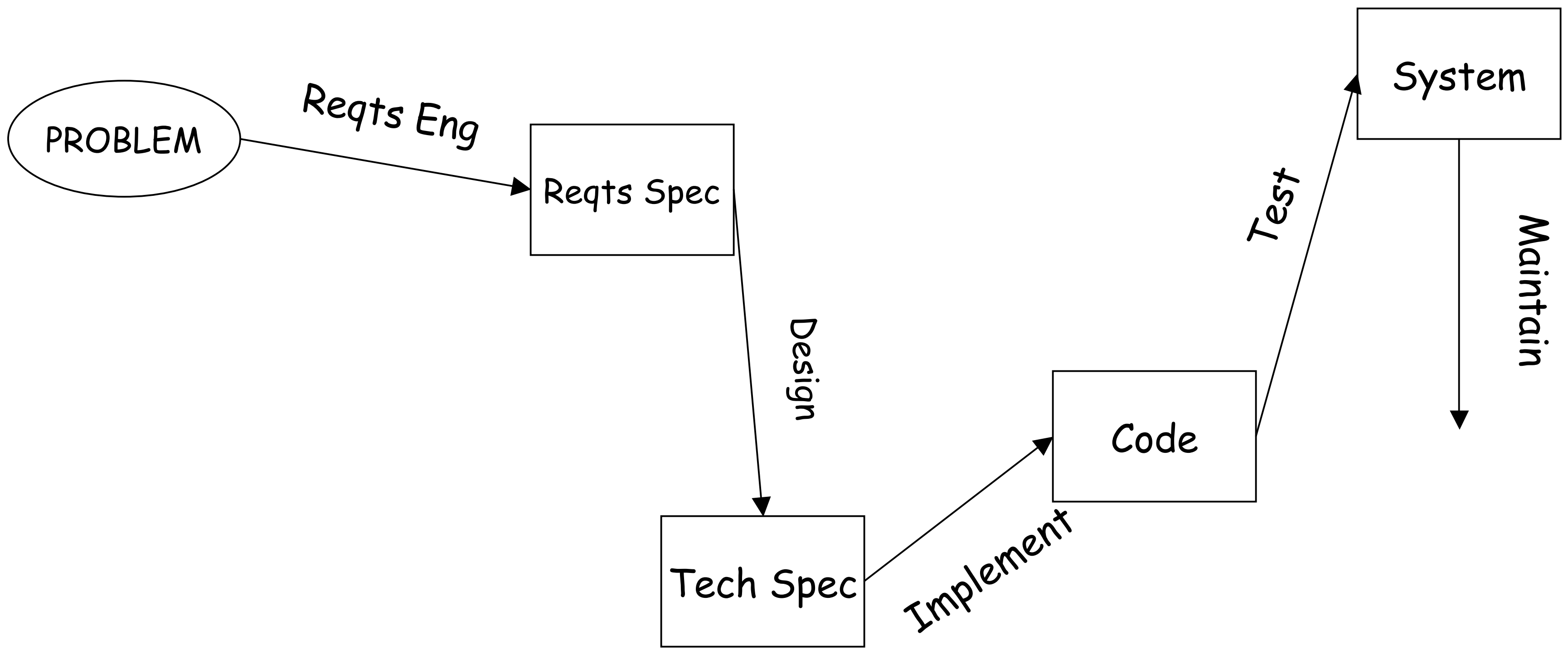
Software Process Models

- 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
  - Prototyping
  - Incremental
  - RAD
  - Spiral

But First

- Code and Fix, Do Until Done Models
- No planning, general idea of product, informal "design" mostly through code
- Code, use, debug, test until ready for release
- No way to tell you are done or if requirements met
- "bankrupt choice born of desperation" (p.16)

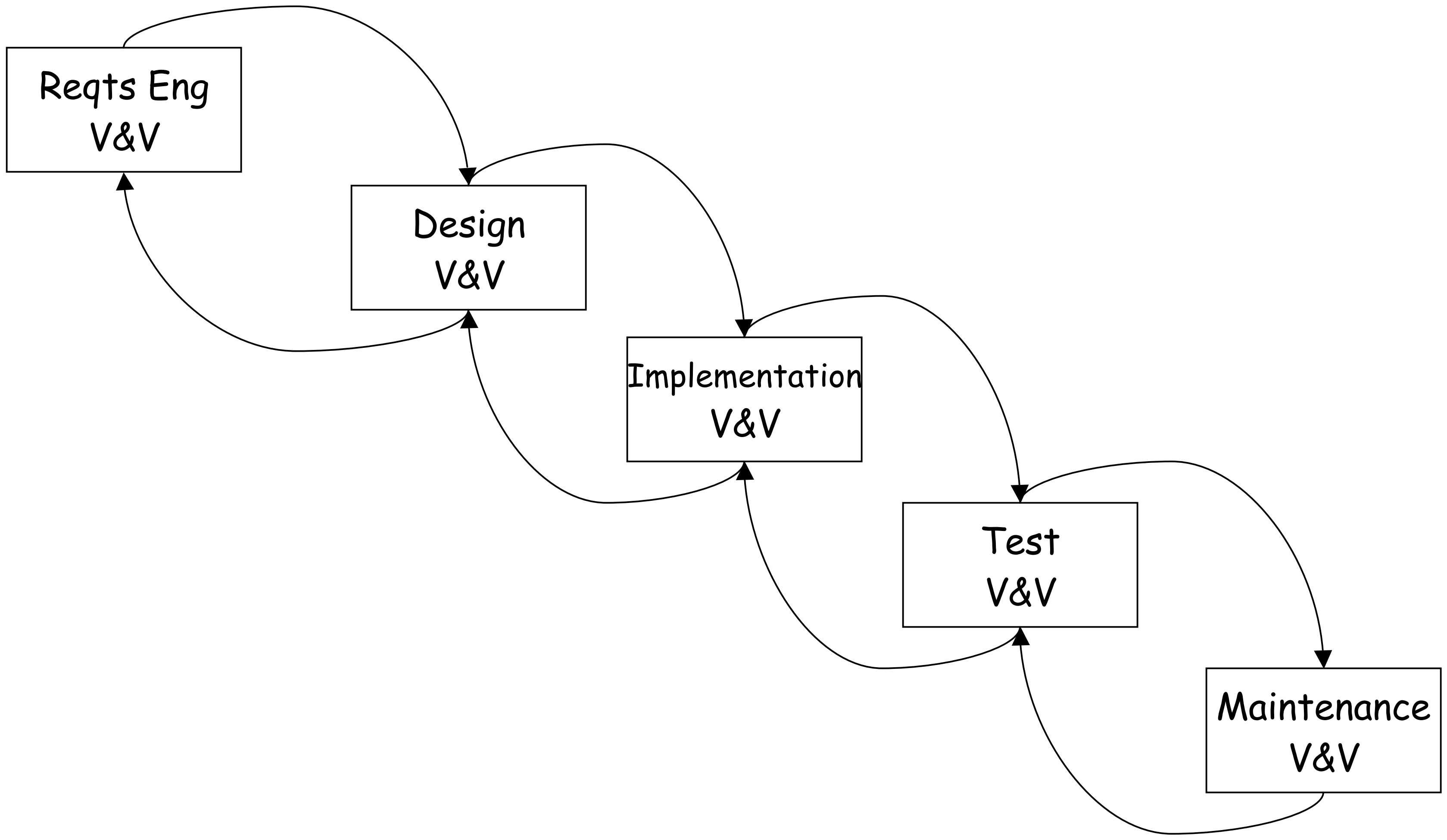
Simplified Model



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 (from van Vliet)

	Design Phase	**Coding Phase**	**Integration Test Phase**	**Acceptance Test Phase**
Integration Test Activity	4.7	43.4	26.1	25.8
Coding Activity	6.9	70.3	15.9	6.9
Design Activity	49.2	34.1	10.3	6.4

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

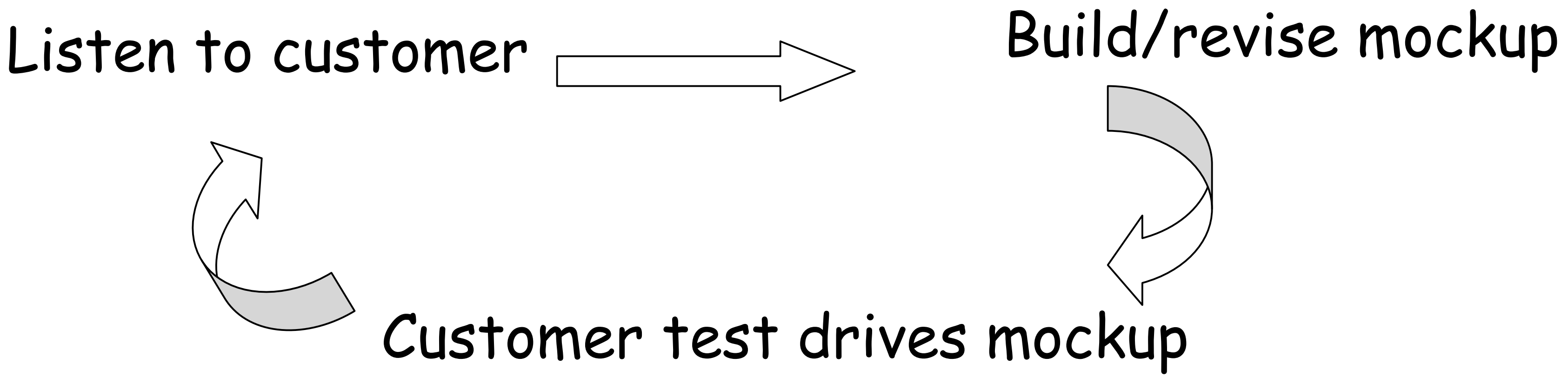
Review of Waterfall

- Not change tolerant
- Difficult for Customer to state all requirements upfront (only 40% to 60% of requirements known initially)- 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,* early days when computer time was expensive
- 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
- Requires experienced developers

Advantages of Proto

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

Disadvantages

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

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

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
- 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
- Customer involvement
- Installation and user training are an explicit part of the process

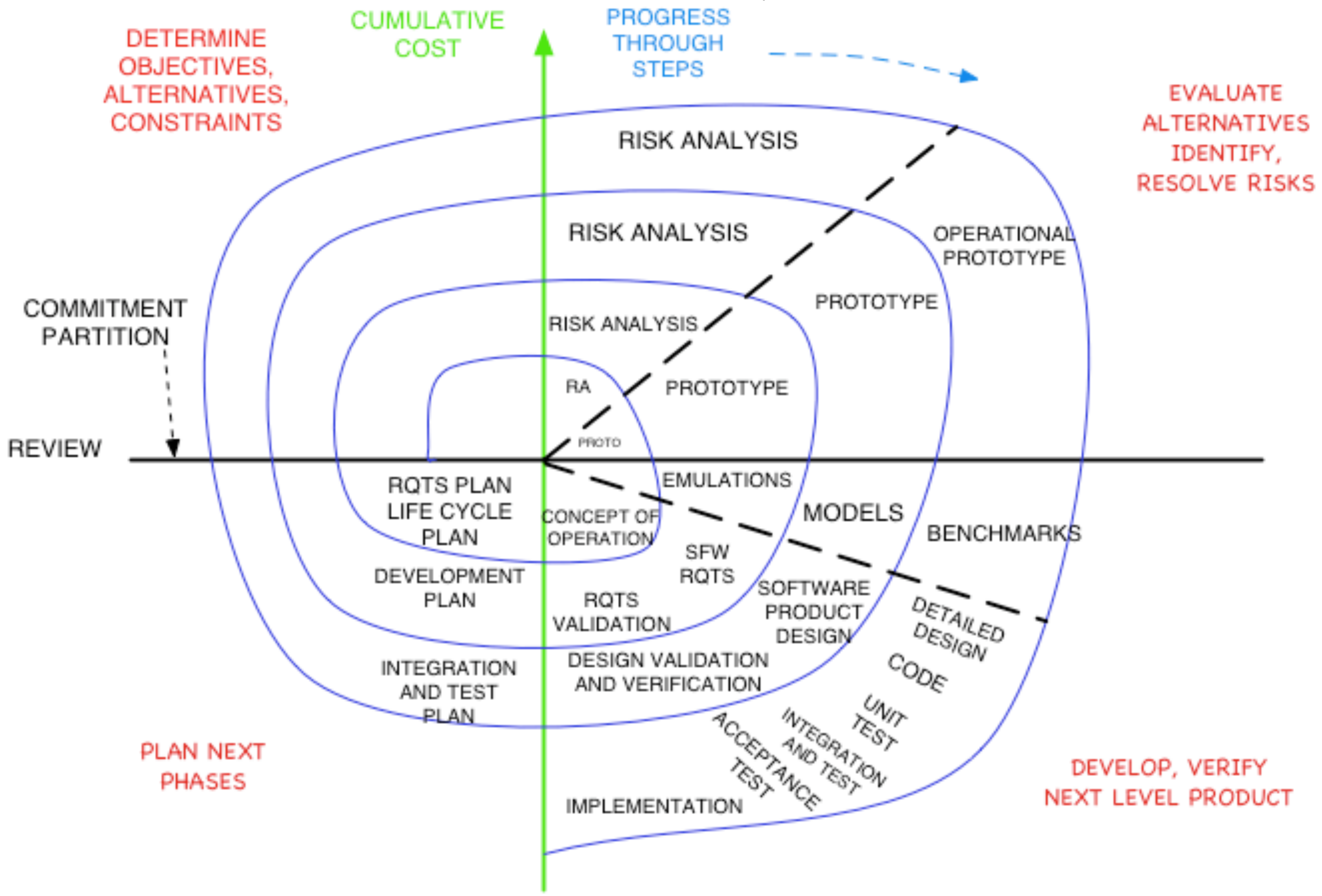
RAD Disadvantages

- 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

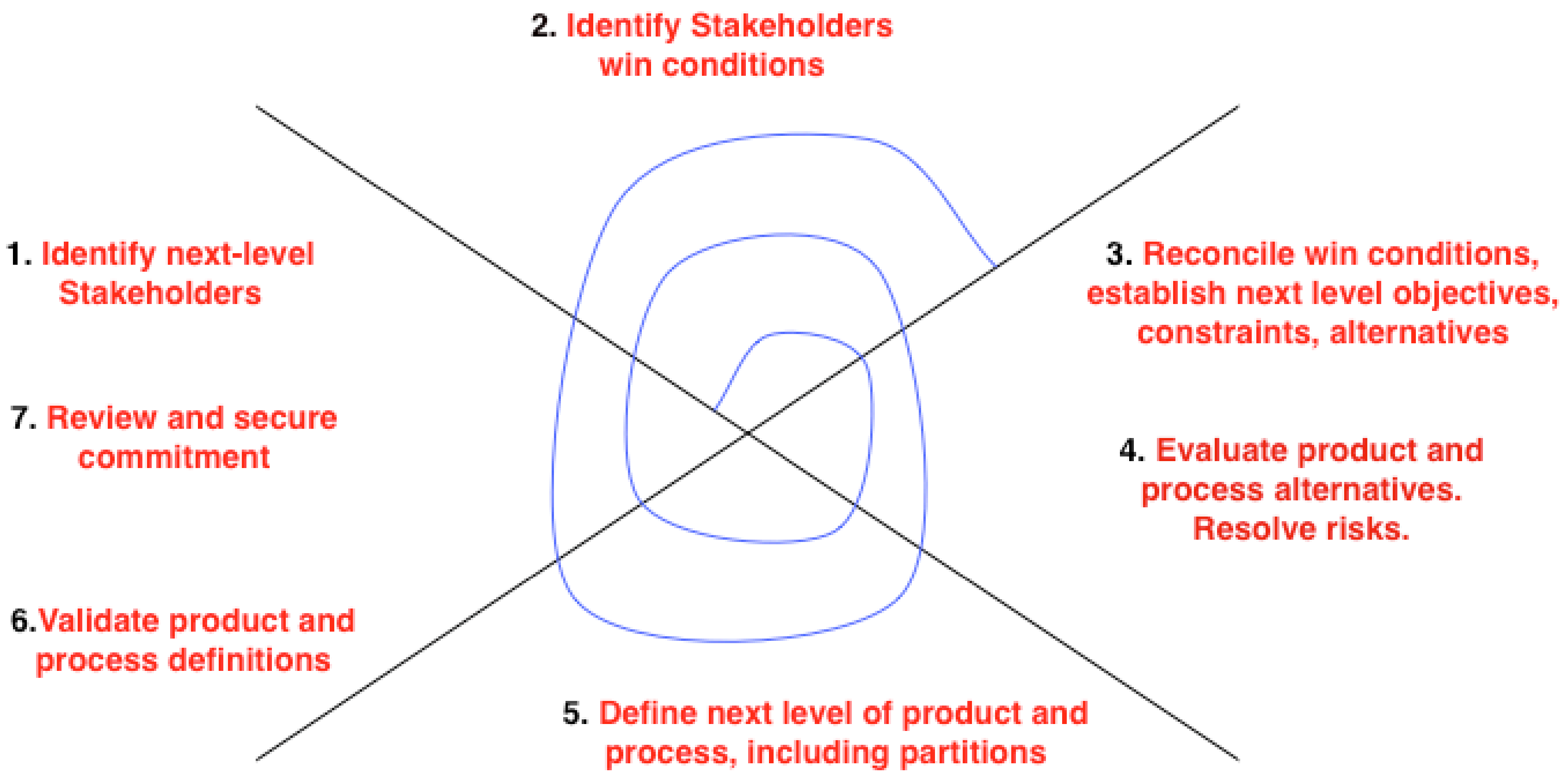
Spiral Development

- 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


2. Identify Stakeholders win conditions
1. Identify next-level Stakeholders
3. Reconcile win conditions, establish next level objectives, constraints, alternatives
7. Review and secure commitment
4. Evaluate product and process alternatives. Resolve risks.
6.Validate product and process definitions
5. Define next level of product and process, including partitions

WinWin Adds

- Life Cycle Objectives - goals for each major software activity
- Life Cycle architecture
- Initial Operation Capability - (site plan+) preparation for software installation/distribution, site preps before install (even for PCs) and assistance required by all relevant parties.

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

All Projects Should

- Use a Development Plan Approach (write and follow):
  - What will you do?
  - How will you do it?
  - What do you depend on?
  - When will you be done?
  - Who will do what?

Requirements Issues

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

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

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)

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

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, … vV provides examples

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

Discussion Points

- 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
- “The more technically competent a team is the more resistant it is to new technology”

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
- Software Engineering and responsibility

Next Time

- Begin Brooks
- Project planning
- Risk Management
- Requirements

Resources

- Futrell, Shafer & Shafer, <u>Quality software project management</u>, Prentice Hall, 2002, ISBN 0-13-091297-2
- Van Vilet, H. <u>Software Engineering: Principles and Practice, Second Edition</u>, Wiley, 2000, ISBN: 0-471-97508-7