CIS 422/522

1. Software Life cycles and Process models I
2. Group Dynamics
3. First team meeting

The Software Lifecycle and Software Process Models

Introduction

View of SE in this Course

- The ***purpose of software engineering*** is to *gain* and *maintain* intellectual and managerial control over the products and processes of software development.
  - *Intellectual control* means that we are able make rational choices based on an understanding of the downstream effects of those choices to deliver a system with the desired capabilities
  - *Managerial control* means we are able to make rational choices about development *resources* to deliver a system on time and within budget
- Begin to consider what this means in practice

Need to Organize the Work

- Nature of a software project
  - Software development produces a set of interlocking, interdependent work products
    - E.g. Requirements -> Design -> Code
  - Implies dependencies between tasks
  - Implies dependencies between people
- Must organize the work such that:
  - Every task gets done
  - Tasks get done in the right order
  - Tasks are done by the right people
  - The product has the desired qualities
  - The end product is produced on time

Addressed by Software Processes

- Developed as a tool for controlling complex software developments
- Answers the “who”, “what”, “when”, etc. questions
  - What product should we work on next?
  - What kind of person should do the work?
  - What information is needed to do the work?
  - When is the work finished?
- Intended use
  - Guidance to developers in what to produce and when to produce it
  - Basis for planning and assessing development progress

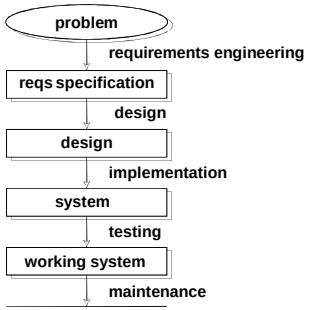
Definitions

- Software Life Cycle: evolution of a software development effort from concept to retirement
- Life Cycle Model: Abstract representation of a software life cycle as a sequence of 1) activities or phases and 2) products (usually graphic)
- Software Process (process model): institutionalized version of a life cycle model defining specific roles, activities, and artifacts

Rationale

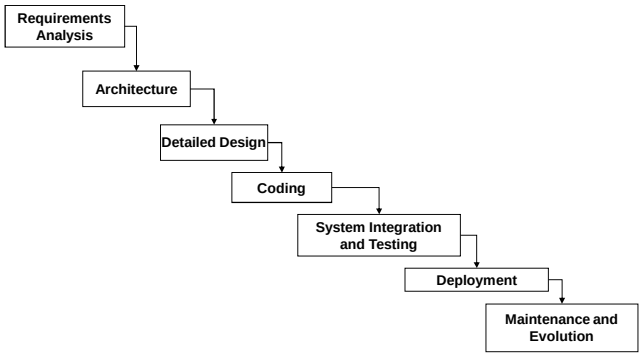
- Developed as a tool for gaining and maintaining control over complex software development processes
- Application of “divide-and-conquer” to software processes and products
  - Goal: identify distinct and relatively independent phases and products
  - Can then address each separately
- Intended use
  - Provide guidance to developers in what to produce and when to produce it
  - Provide a basis for planning and assessing development progress

A Simple Process Model



From van Vliet

A "Waterfall" Model



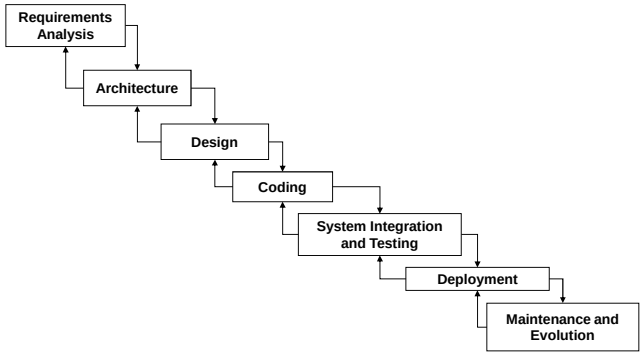
Phases and Products

- Requirements
  - Goal: implementation-independent specification of what the software must do and any constraints on its development
  - Product: Software Requirements Specification (SRS)
- Architecture
  - Goal: decomposition of the problem into components that together satisfy the requirements within the constraints
  - Products: specifications of components, relations, interfaces
- Detail Design
  - Goal: internal design of components (e.g., objects) to identify appropriate algorithms and data structures supporting the interface
  - Products: design documentation, pseudo-code

Phases and Products

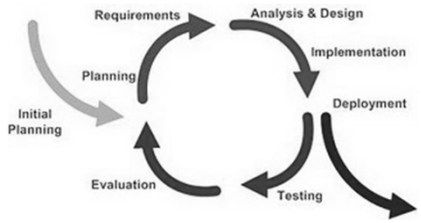
- Implementation
  - Goal: realization of the design in a machine-executable language
  - Product: code
- Testing
  - Goal: validation and verification of the implementation against requirements and design
  - Products: test plan, test cases
- Maintenance
  - Goal: maintain deployed system
  - Products: bug fixes, patches, new versions

Iterative "Waterfall" Model



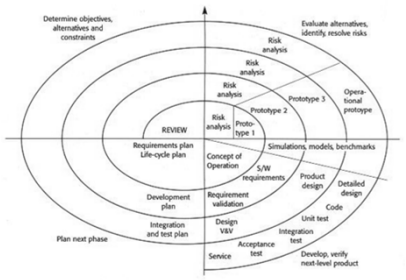
Characteristic Processes: The Iterative Model

- Process viewed as a sequence of iterations, each iteration produces an increment of the working software (sequence of waterfalls)



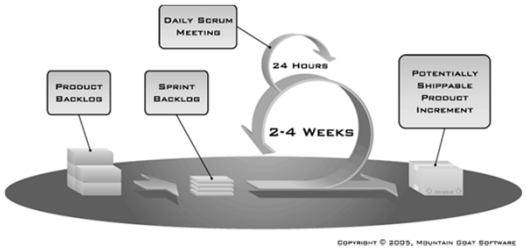
Characteristic Processes: The Spiral Model

- Process viewed as repeating cycles of increasing scale
- Identify risks and determine (next set of) requirements, build next version by extension, increasing scale each time



Characteristic Processes: Agile (scrum)

- Process viewed as nested sequence of builds (sprints)
  - Each build adds small feature set



Also…

- Prototyping models
- RAD models
- Extreme Programming
- Etc., etc.

Why so many models?

Take-away

- For now, important to understand the set of activities that comprise software development
- For next week – consider what kind of process to use for your projects
  - Understand how and why people use different development models
  - Understand how to choose an appropriate model for your developments
  - Understand how to map processes to plans

Teamwork and Group Dynamics

A few tips on effective teamwork, meetings, and presentations

.

What is a Great Team?

- Diverse Skills
  - People skills, communication and writing skills, design skills, implementation skills and knowledge
- Coherence
  - Ability to build and maintain a shared vision
  - Shared expectations
- Mutual Respect and Responsibility
  - You don't *have* to like each other, but you *need* to trust and respect each other — and to earn your teammates trust and respect
  - This is an enduring part of professionalism in the real world

Desired Skill Mix

- At least one person with experience in team projects, preferably with some management experience.
- At least one person with strong skills in programming and program design, preferably including networking.
- At least one person with strong communication skills and good written English.

Team Roles

- Manager: responsible for schedule
- Requirements analysts
- System architect
- Quality control
- Technical writer
  - Technical documentation
  - User documentation (may be different skills)
- User interface designer
- Programmer
- Configuration control (build-master)

Not 1-1 with people. Backup for each role.

Discussion: what are 1) the responsibilities and 2) skill set needed for each?

What do software developers do?

- One way to measure: how do they spend their time?
- IBM study (McCue 78):
  - 50% team interactions
  - 30% working alone
  - 20% not directly productive

Technical excellence is not enough

"Egoless" design

(Weinberg, Psychology of Computer Programming)

- Investing ego in group
- "Letting go" of ego investment in code, design, ideas
  - No winning or losing design debates (focus on improving the product)
  - Once contributed, ideas belong to the group
  - Criticism is aimed at concepts, not people
- The best designers criticize their own designs!
  - Our own assumptions are the hardest to critique
  - Corollary: A good critic is your best ally
    - The hardest lesson to learn but one of the most valuable
- Especially difficult in multi-cultural teams

. . . but we are not egoless people

- Ego investment is normal
  - be aware of it, be in control of it
- Consider the egos of others
  - What are you attacking? Why?
  - What is motivation of the other person?
    - Are they feeling ignored? Not valued?
- Pride in accomplishment is ok, unless it interferes with accomplishment

Consensus decision making

- Consensus is not counting votes
  - Democracy is 51% agreement
  - Unanimity is 100% agreement
  - Consensus is neither
    - It is "buying in" by group as a whole, including those who disagree
- Everyone has their say
- Everyone accepts the decision, even if they don't prefer it
- Usually best approach for peer groups

Consensus takes time and work, but is worthwhile

Conflict

- Can be healthy and productive
- Can destroy a team if not carefully managed
- Manage conflict constructively
  - Soothe and protect egos
    - Everyone's job, but especially the manager's job
  - Keep conflict on a technical level (not personal)
  - Reward conflict resolution

Being a Good Team Member

- Attributes most valued by other team members
  - Dependability
    - When you say you'll do something, you do it
    - Correctly
    - On time
  - Carrying your own weight (doing a fair share of the work)
  - People will overlook almost everything else if you do these

A Word on Managing

- A good manager supplies what is needed for the team to succeed. This includes (but is not limited to)
  - Resources
  - Planning and coordination
  - Pitching in when needed
  - Protection (especially from upper management)
  - Emotional support, etc.
- Good managers are are leaders not dictators
- Good managers are rare

Effective Meetings

Notes on effective meetings

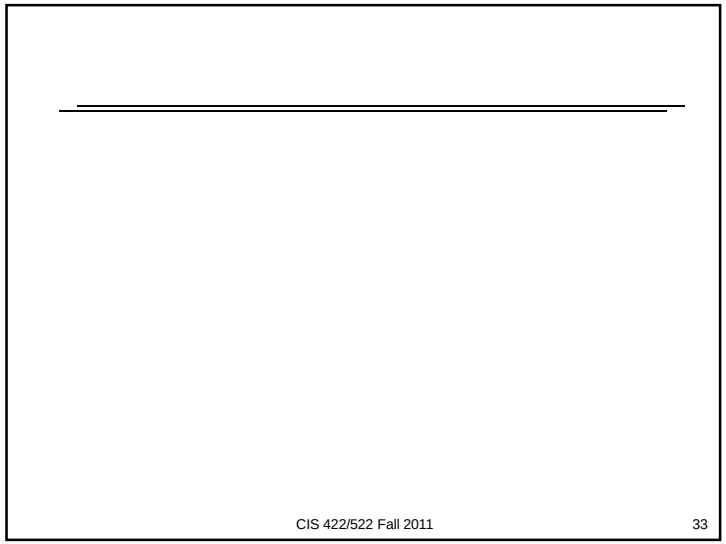
- Only hold meetings if necessary
  - "Necessary" means that the only or most cost effective way to accomplish a goal is by meeting
- Have a goal, and a plan (agenda)
  - Clear meeting objectives
  - Known to all in advance (I.e, distribute via email)
- Plan to goal:
  - Participants - Invite only the necessary people
  - Schedule
  - Intended outcome
- Prepare
  - Cost of wasted time = Time x people x hourly cost
  - Cost of individual prep time is much less

Notes on effective meetings (2)

- Start on time, end on time
- Write down and disseminate the results
  - Leaves an audit trail of decisions
  - Makes people feel included
  - Limits the number of (informational) invitees
- End with concrete, specific action items
  - What must be done
  - Who should do it
  - What the follow-up is

Assignment

- Reading:
  - Text: Chapters 4, and 8
- Project
  - First meeting (in class)
  - Plan and hold at least one project meeting out of class
    - Choose a team name
    - Decide preliminary team roles
    - Identify any initial issues with project
  - Keep meeting notes
    - Agenda
    - Meeting summary
    - Action items
  - Tuesday: quick status report on your decisions (~2 min)


CIS 422/522 Fall 2011
33