ECE450 – Software Engineering II

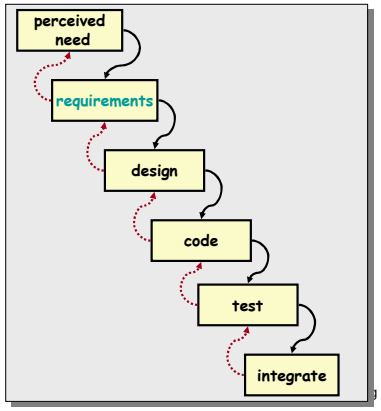
Today: Lifecycles and Methodologies

Lifecycle of Software Projects

- Lifecycle models are useful to compare project management strategies in abstract terms
  - Birds-eye view strategy
  - Detect strengths and weaknesses
  - ... but reality is always more messy

Waterfall Model

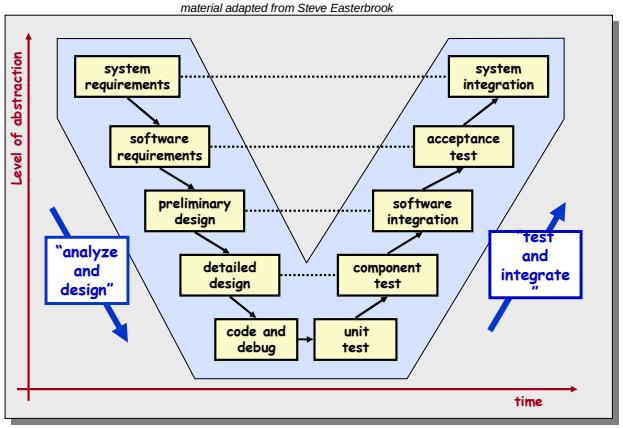
material adapted from Steve Easterbrook



- View of development:
  - A process of stepwise refinement
  - High-level management view
- Problems:
  - Static view of requirements (ignores volatility)
  - Lack of user involvement once spec is written
  - Unrealistic separation of spec from design
  - Doesn't accomodate prototyping, reuse

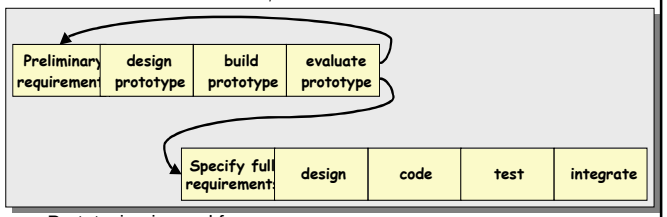
V Model

material adapted from Steve Easterbrook



Prototyping Model

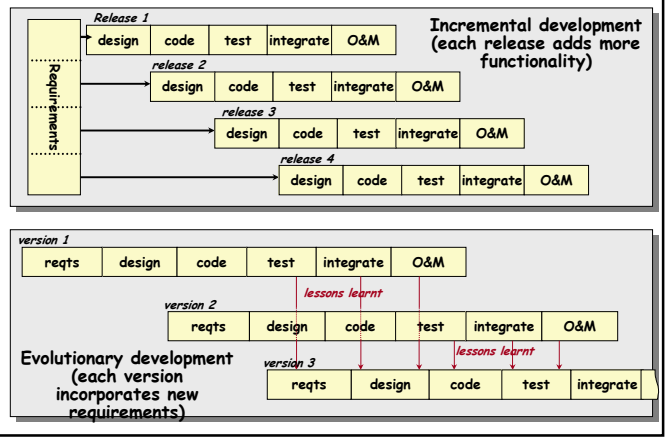
material adapted from Steve Easterbrook



- Prototyping is used for:
  - Understanding the requirements for the user interface
  - Examining feasibility of a proposed design approach
  - Exploring system performance issues
- Problems
  - Users treat the prototype as the solution
  - A prototype is only a partial specification

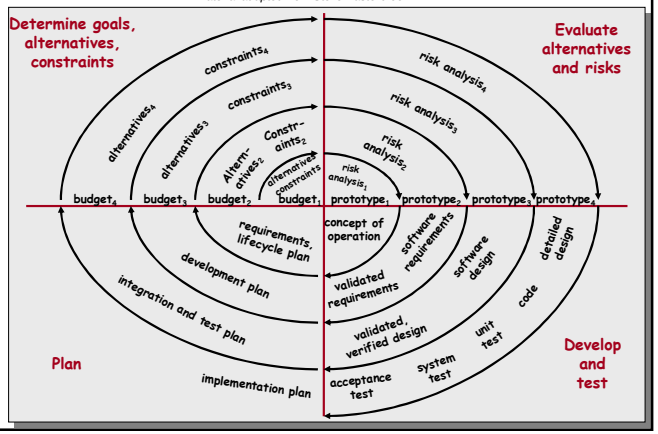
Phased Models

material adapted from Steve Easterbrook



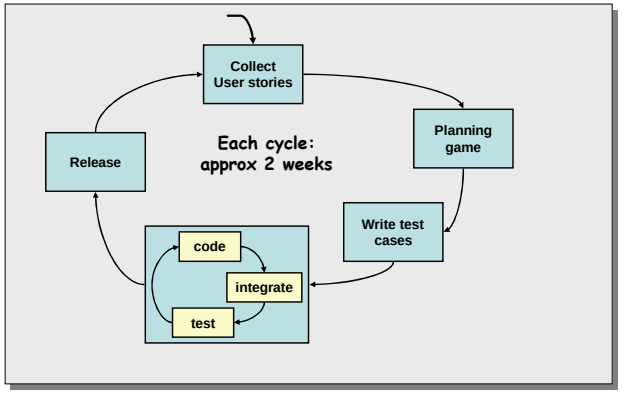
Spiral Model

material adapted from Steve Easterbrook



Agile Model (XP)

material adapted from Steve Easterbrook



Project lifecycle choices

- Which lifecycle model to choose?
  - First of all, CHOOSE ONE!
    - Too many projects drift aimlessly without this kind of strategy
  - Second, if possible, AVOID WATERFALL
    - Most derided, error-prone lifecycle
    - Though still the lifecycle of choice in many corporations
  - Third, prototypes and iterations are good for you
    - Sanity checks
    - Almost never a waste of time/resources
  - Fourth, choose based on context and convenience

Software Methodologies

- Reminder: A **lifecycle** is an abstract description of the life of a project
- A **methodology** is a set of techniques that work well together
- Lifecycles != Methodologies
  - Methodologies are usually (but not exclusively) built upon a lifecycle strategy

Methodology Types

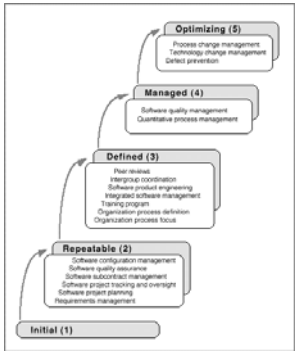
- Main distinction: **Sturdy** vs. **Agile**
- Key difference is how they handle uncertainty
  - ***Sturdy*** approaches attempt to minimize the *amount* of uncertainty
    - Planning, risk prevention
  - ***Agile*** approaches attempt to minimize the *impact* of uncertainty
    - Adaptatability, incremental processes

CMM

- CMM: Capability Maturity Model
  - (now *CMMI*, where *"I" stands for "integration")*
  - Developed by Watts Humphrey and the Software Engineering Institute (SEI) at CMU
  - Five levels
  - Certification process
    - Companies are evaluated as "CMM level 3", for example
  - Mirrors Total Quality Management approaches

CMM (cont)

- Pros:
  - Proven techniques
  - Self-sustained process
  - Required for some software contracts
- Cons:
  - Fear of taking risks
  - Not popular among employees nor stellar companies
  - Doesn't get more rigid than this!
    - Unless you go for ISO



CMM variants: TSP and PSP

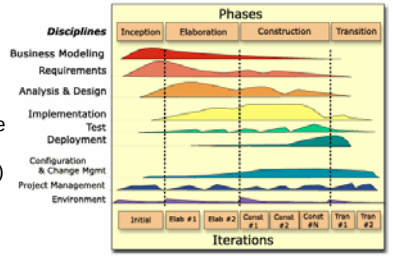
- TSP: Team Software Process
  - Your company isn't keen on the CMM?
    - You can still embrace its processes at the team level
    - Same recipe as CMM, but in smaller scale
- PSP: Personal Software Process
  - No, not PlayStation Portable!
  - Same story as TSP, on an individual scale
    - "*A Discipline for Software Engineering*", Humphrey
  - Worth reading and doing the exercises, at least for self-awareness

Cleanroom

- Best realization of "software engineering is formal methods" concept
  - Main idea: Don't let the bugs in in the first place
    - To be added to product, piece of code must be proven correct
- Pros:
  - Very high-quality software
  - Optimal for mission-critical projects
- Cons:
  - Slow, not cost-effective
  - Good luck finding trained people

RUP

- RUP: Rational Unified Process
  - Propietary process, IBM
  - Characterized by use of UML (Unified Modelling Language)
  - Feels like a matrix evolution on the waterfall model
    - "Phases" and "Disciplines"



RUP (cont)

- Pros:
  - More relaxed, though still “sturdy” approach to software projects
  - Popular in some mid-large software companies
  - Discards naive view of waterfall models
- Cons:
  - Need to train people in new modelling skills
  - Controversy on cost-effectiveness of analysis and modelling
  - Doesn’t work well in changing environments

XP

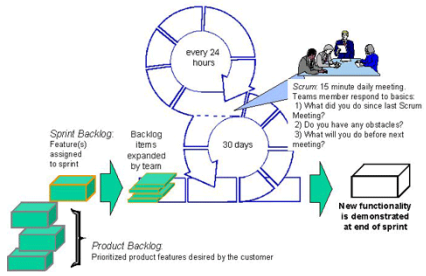
- XP = eXtreme Programming
  - Yes, terrible name
  - Intuition: Requirements changes are inevitable; emphasize adaptability
  - Practices:
    - User Stories
    - Planning Game
    - System Metaphor
    - Test-Driven Development
    - Small Releases
    - Pair Programming

XP (cont)

- The most successful of the agile methodologies
  - Though it’s debatable whether people that *say* they follow XP really *are* doing so
- Pros:
  - Little spending in initial stages, results appear early
  - Change is expected, software adapts faster
  - Short feedback loops
- Cons:
  - No time spent in analysis may mean lots of rework later on
  - No clear end in sight, project may continue forever
  - Pair programming feels awkward for most

SCRUM

- An agile, lightweight “methodology” alternative



SCRUM (cont)

- Pros:
  - Just about the easiest “methodology” to implement
  - Spends little developer time in documentation and meetings
  - 15-minute daily meetings are a great practice
- Cons:
  - Not every customer is agreeable
  - Difficulties of scale
  - Long-term planning concerns

Methodology choices

- THEY ALL WORK
  - Really!
  - They provide a framework for your project plans
  - But you need to be committed to make it work
- Choice depends on personal/company/customer preference
- What about Open Source projects?