Software Process Models

- Taxonomy, Methodology, and Checklists
- Basic Waterfall Models
  - software development activities
  - concurrent development, phase overlap
- Incremental Development Models
- Iterative (spiral) Development Models
- Process Specifications
  - work products

Software Process Terminology

- Vocabulary for S/W development tasks
  - enables more precise communication
  - richer discussions of development activities
- Foundation concepts
  - enables us to better visualize projects
  - enables us to better plan projects
- Taxonomy of concepts
  - invites us to think in terms of system structure
  - invites us to think in terms of relationships

Engineering Methodology

Methodology

a system of principles, practices, and procedures applied to a specific branch of knowledge.

Methodical

characterized by method and orderliness

These are the things that distinguish a "hacker" from an "engineer".

On Check-Lists

Check List

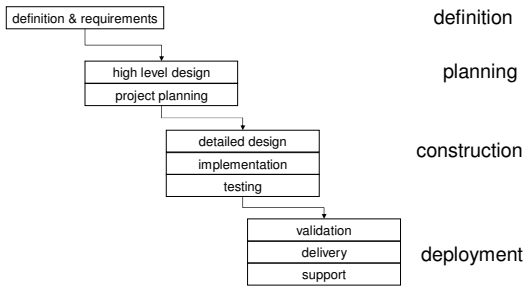
a list of actions to be taken in response to a particular event or situation

- check lists capture "best practices"
  - proven methodology from other projects
- check lists guide us through processes
  - ensure we do all of the required steps
  - ensuring we do steps in the right order
- check lists are not "learning aids"
  - for professionals who can't afford mistakes

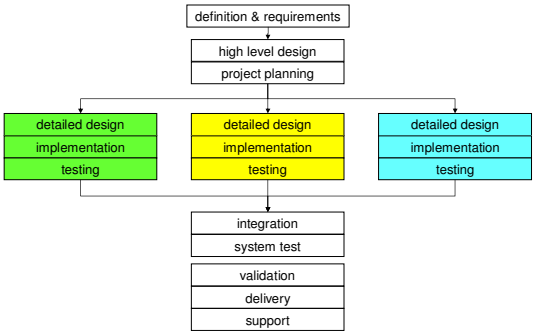
Software Engineering Activties

- understand the problem
  - project definition, requirements development
- plan the solution
  - project planning (risks, resources, schedules)
  - architectural design, modeling, prototyping
  - User Interface & component design
- execute the plan
  - reviews, implementation, testing, integration
  - validation, deployment, support

The Basic Waterfall Model



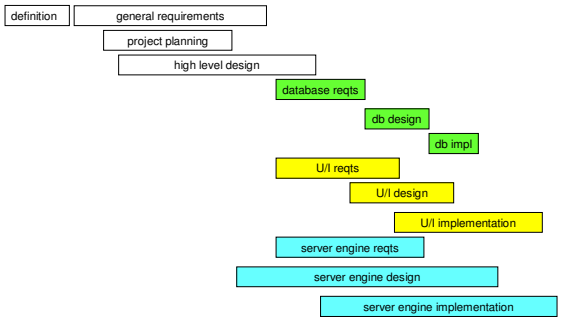
Concurrent Development



Concurrent Development

- most systems require many pieces
  - independent pieces can be built independently
- advantages
  - smaller teams are more efficient
  - smaller projects involve less risk
  - improved resource utilization, earlier finish
- cost
  - resource allocation becomes more complex
  - some problems only emerge after integration

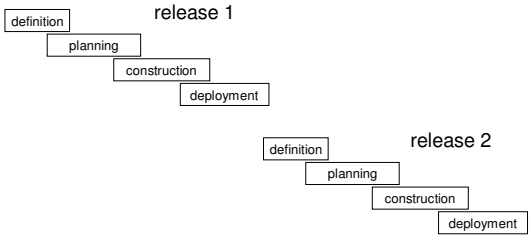
Phase Overlap



Phase Overlap

- phase *n+1* can start before phase *n* ends
  - there are many tasks in phase *n+1*
  - they don't all depend on all of phase *n* tasks
- such overlap has big advantages
  - better resource utilization, earlier completion
- but there are risks
  - if phase *n* action invalidates phase *n+1* work
  - all such dependencies must be tracked
- project planning tools & Gantt charts

The Incremental Delivery Model



The Incremental Model

- there are always requirements we missed
  - they become obvious when product is in use
- new requirements arise
  - technology and the market evolve
- we don't know how to do it all up-front
  - we learn from experience
- insufficient time/people to do everything
  - even if we knew what "everything" was
- new releases mean new sales

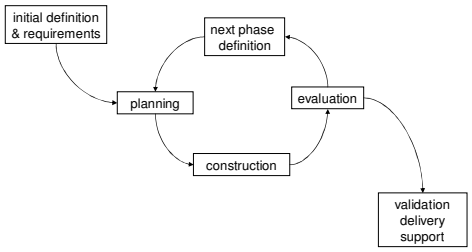
Where these models break down

- the “execute the plan” phase assumes ...
  - we know what product we need to build
    - the customers and their requirements
  - we know what it takes to build the product
    - how to build it, with what resources, how quickly
- these assumptions often fail
  - requirements for **new** products are speculative
  - estimates for **unknown** tasks are fantasy
- plans based on false assumptions are bad

Planning with Poor Information

- Option A: add fudge factors
  - enumerate all of the major uncertainties
  - guess at likely costs implied by each
  - hope that they average out
- Option B: a plan for a plan
  - enumerate all of the major uncertainties
  - plan research/prototype projects to resolve each
  - develop a plan for developing a better plan
- these strategies can be combined

Iterative (spiral) Models



Spiral v.s. Incremental Models

- each incremental iteration is a product
  - it satisfies requirements (for that release)
  - it is tested, documented, and validated
  - it is delivered and supported
- spiral iterations are research projects
  - they start with a premise
  - they build a prototype to test the premise
  - the resulting information feeds future planning
- each iteration proceeds according to a plan

Prototyping addresses ignorance

- Find mistakes before building the real thing
- We aren’t sure what we should build
  - prototype a few alternatives, get feedback
- We aren’t sure how much work it will be
  - identify the parts we don’t know how to build
  - isolate, prototype, and test those mechanisms
  - see what problems arise
- We aren’t sure how well it will work
  - measure a model, simulation or prototype

Why Models Matter

- All projects are not the same
  - different problems, organizations, constraints
  - different models better suit different projects
- Choosing a model sets expectations
  - if model is wrong, expectations won’t be met
  - such projects are usually called failures
- To choose a more appropriate model
  - we must understand their differences
  - we must understand our own situation

Q: What model do I choose?

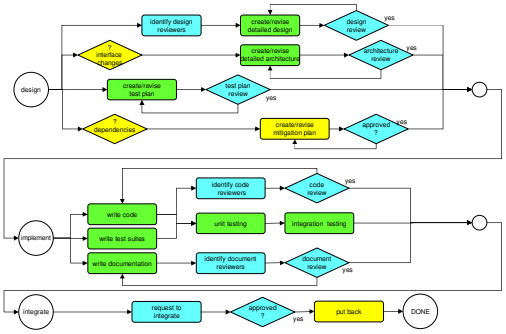
A: The one that most closely fits your problem

- Is this Research or Development?
  - are we trying to define a product or build one?
- Is this a one-off, or a product family?
  - should we plan for incremental delivery?
- Can many parts be built in parallel?
  - are they sufficiently independent?
  - do we have resources for multiple teams?
  - can we manage the added complexity?

Process Specifications

- written descriptions of steps to be performed
  - when carrying out a particular type of project
  - usually a combination of words and diagrams
- they usually describe, for each step,
  - the work that should be performed
  - the acceptance criteria for that work
  - who has the authority to approve it
- they may also specify, for each step
  - required inputs and/or pre-conditions
  - required output (work products)

Typical "Construction" Process



Process Work Products

- the outputs defined by a process
  - specified outputs of development process steps
  - analyses, plans, specs, code, reports, …
  - definitions may be general or very strict
- why do we produce them?
  - they are required inputs to subsequent steps
  - they represent project "mile-stones"
    - they are concrete , measurable, deliverables
    - reviewing them gives us confidence of our progress
    - they are a record of our progress

For the next Lecture

- McConnell 3.2
  - good intro to fitting the process to the project
- Big Requirements Up Front
  - makes the case for agile process
- wikipedia:Agile Process
  - good article on motivations, history, philosophy
- wikipedia:eXtreme Programming
  - good intro to principles & philosophy
- wikipedia:XP practices
  - good overview of key elements of XP (details later)

Supplementary Slides

on real commercial processes

Process Models & Strategy

- Model choice is not just about projects
  - productivity is secondary to staying in business
- Models must support business objectives
  - understand the demands of that business ... find a model that supplies those needs
  - understand the challenges of that business ... find a model that shields us from what we fear
- Process Models for commercial s/w are often as much about business as s/w

A Real Development Process

If you are interested in seeing what a real development process specification looks like, you might want to check out:

http://www.opensolaris.org/os/community/onnv/os_dev_process/

This includes process flow charts, descriptions of work products, and discussions of motivations and principles.

Case Study: Microsoft

- the domain
  - flagship applications like word and excel
- the challenge
  - maximum value in each new release
  - maximize ROI on new feature development
  - maximize release predictability (date/quality)
  - maximize project predictability (cost/success)
- the response
  - a project qualification process

Microsoft Feature Management

- all new projects must create feature value
  - if we can't advertise it, we won't do it
- all proposals must have business cases
  - independent research, product use statistics
  - projects prioritized based on projected revenue
- all projects must be small and complete
  - no project can be larger than two staff weeks
  - no project can depend on other projects
- only fully tested projects will be integrated
  - they had very demanding test standards

Feature Management - benefits

- high value releases with high ROI
  - projects were chosen based on revenue
- high project predictability
  - small projects tend to have fewer side effects
  - small projects are simpler and less risky
- high release predictability
  - rigorous testing requirements reduce breakage
  - independence means we can back out losers
- this helped to ensure business objectives

Feature Management - problems

- It effectively precluded infrastructure projects
  - e.g. network or multi-media integration
- they do not deliver advertisable "features"
  - rather they enable future feature projects
- they are neither small nor independent
  - much new code, much change to existing code
  - all future projects will depend on them
- they are hard to test
  - they are complex, general, and pervasive

Case Study: Sun

- the domain
  - the Solaris Operating System
- the challenge
  - encourage technological innovation
  - avoid breaking customer applications
  - maximize release predictability (date/quality)
  - avoid future support disasters
- the response
  - Architectural Review Committees

SUN: ARC process

- create Architectural Review Committees
  - one for each major technology area
  - staffed by very senior engineers in each area
- create fast-track process for simple projects
  - sponsored cases, auto-approve if unchallenged
- require review/approval for all other projects
  - classify interfaces & ensure sufficient stability
  - ensure conformance w/architectural mandates
  - assess significant support/evolution issues

ARC Process - benefits

- improved release compatibility/quality
  - project integration seldom breaks a release
  - new releases no longer break old applications
- accelerated adoption of new technologies
  - projects were quickly guided in new directions
- significant improvements in product quality
  - numerous support disasters were averted
  - projects benefited from senior engineer review
- this helped to ensure business objectives

ARC Process - problems

- the process was expensive for the company
  - it consumed 25-50% of 30 very senior engineers
  - managers viewed this as development tax
- the process was expensive for projects
  - preparing for a review was time-consuming
  - recommendations made projects larger
  - managers viewed this as extortion
- the process was not applied uniformly
  - different divisions had different processes