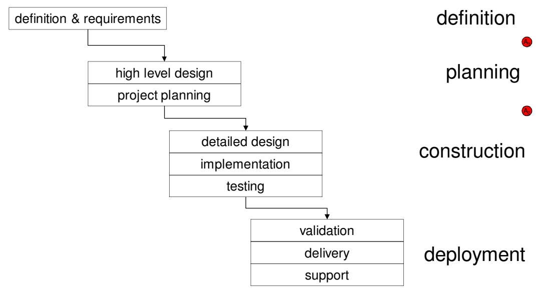
## Software Process Models

- Critique & Defense of “the Waterfall”
- Issues in Waterfall Models
  - concurrent development
  - phase transitions and overlap
- Issues in Evolutionary Models
  - incremental vs. iterative models
  - planned iteration
- Choosing the Right Model



## The Basic Waterfall Model






## The Agile Critique of the Waterfall

- Waterfall strives to satisfy requirements
- “correct” requirements do not exist
  - requirements change over time
  - they were not perfectly understood/captured
  - people make up requirements
  - satisfying wrong requirements is futile
- iterative development is more rational
  - build an initial prototype
  - get concrete feedback from real users
  - improve prototype based on that feedback

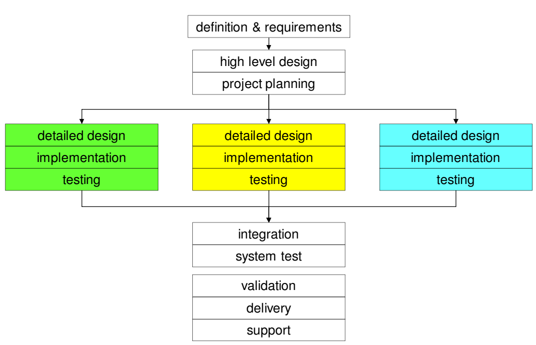


## In Defense of Prescriptive Models

- they capture fundamental truths
  - you can’t build “it” until you know what “it” is
  - things go much better when you have a plan
- a basis for modeling <u>any</u> process
  - basic task break-down and planning template
  - planned progression from one step to next
- some projects really do fit them
  - clear requirements are obtainable
  - technical risk is low
  - agile processes can be seen as extensions



## Concurrent Development

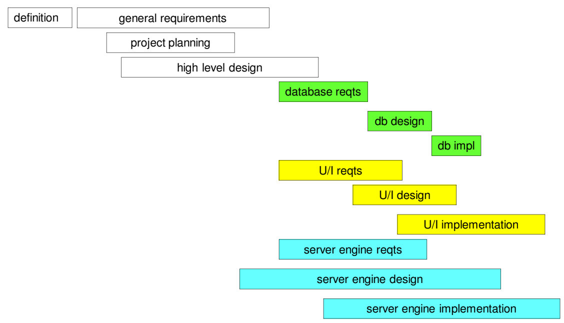





## (Concurrent Development)

- most systems require many pieces
  - <u>independent</u> pieces can be built <u>independently</u>
- 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



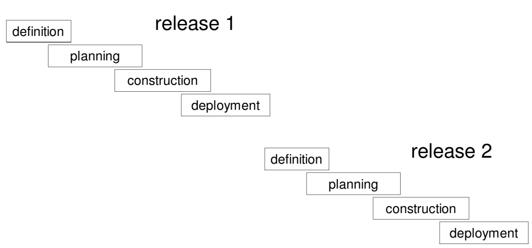



# (Phase Overlap)

- phase *n+1* can start before phase *n* ends
  - there are many tasks in phase *n+1*
  - they don't <u>all</u> depend on <u>all</u> of phase *n* tasks
- such overlap has big advantages
  - better resource utilization, earlier completion
  - experience A impl can influence design of B
- but there are risks
  - if phase *n* action invalidates phase *n+1* work
  - component testing may be done in isolation
  - dependencies must be tracked and managed



# The Incremental Delivery Model






# (The Incremental Model)

- Doing everything in release 1 is a "canard"
  - our requirements are incomplete & imperfect
  - we don't know how to build some pieces
  - insufficient time/people to do everything
- Deliver product in successive releases
  - successive approximations to solution
  - we learn from the experience we gain
  - fewer and smaller tasks in each release
  - sooner delivery, lower cost, lower risk



# Where these models break down

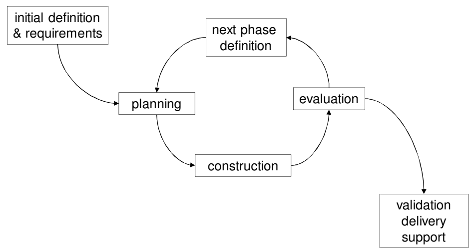
- the "execute the plan" phase assumes ...
  - we know what product we need to build
    - the customers and their requirements
  - we know what it takes to build the product
    - how to build it, with what resources, how quickly
- these assumptions often fail
  - requirements for **new** products are 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
  - this is a plan for developing a better plan

# Iterative (spiral) Models






# (Spiral v.s. Incremental Models)

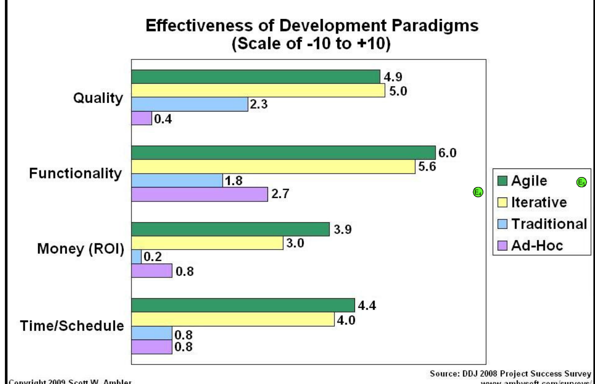
- each incremental iteration is a product
  - it satisfies requirements (for that release)
  - it is tested, documented, and validated
  - it is delivered and supported
- spiral iterations are research projects
  - Goal: answer questions (vs. deliver product)
  - they build a prototype to test the premise
  - the resulting information feeds future planning



# Planned Iteration

- Each iteration has a clear goal
  - we are seeking answers to specific questions
- Each iteration has a plan
  - we know what we are going to do
  - we know how long it will take
  - we know what we will have when we finish
- Each iteration is a commitment point
  - do we still believe in the ultimate goal?
  - is this the right plan to get us there?








# 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
  - plans and designs are predicated on a model
- To choose a more appropriate model
  - we must understand their differences
  - we must understand our own situation



# Sun Tzu

If you know the enemy and know yourself, you need not fear the result of a hundred battles.

If you know yourself but not the enemy, for every victory gained you will also suffer a defeat.

If you know neither the enemy nor yourself, you will succumb in every battle.

## For the next Lecture

- McConnell 3.3-4, 4
  - introduction to the importance of "first things first"
- wikipedia: Requirements Analysis
  - overview of full range of approaches
- Kampe: gathering & analysis of user requirements
  - introduction to the process, guide to project 1A
- Wiegers: Requirements Traps
  - good overview of common mistakes
- Wiegers: Prioritizing Requirements
  - intro to a fundamental planning methodology



## Supplementary Slides

on real commercial processes



## 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)

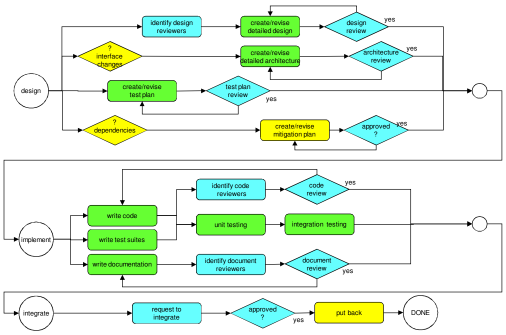


## Examples

- Definition stage – *proposals, requirements specifications, requirements review reports*
- Detailed design – *designs, design review reports*
- Implementation – *software, makefiles, test cases, documentation, code review reports, test reports*
- Validation – *bug reports*, *test results, alpha/beta reports*
- Deployment – *installation statistics*, *bug reports, call reports*
- Process Paperwork – *request and approval forms*



## 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

## 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



## 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



## Keys to Incremental Development

- each increment must be useful
  - not all subsets of functionality are useful
  - if it is not useful, nobody will use it
- each increment must be build-able
  - we must know how to build it
  - we must have the time and resources
- need a plan to sustain the effort
  - can we fund successive approximations
  - can we retain internal/external commitment



## 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

# 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?