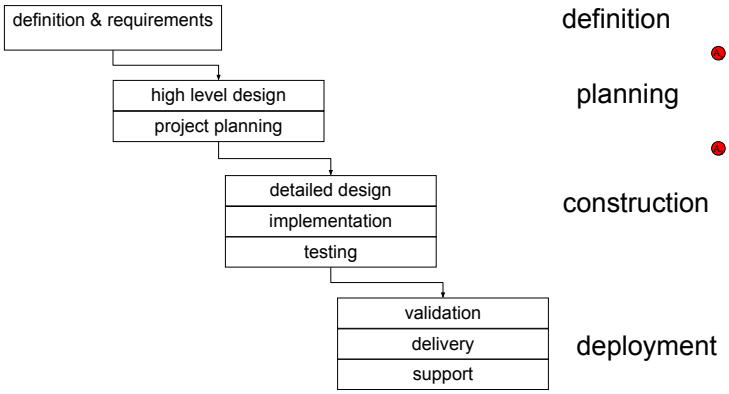
Software Process Models

- 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



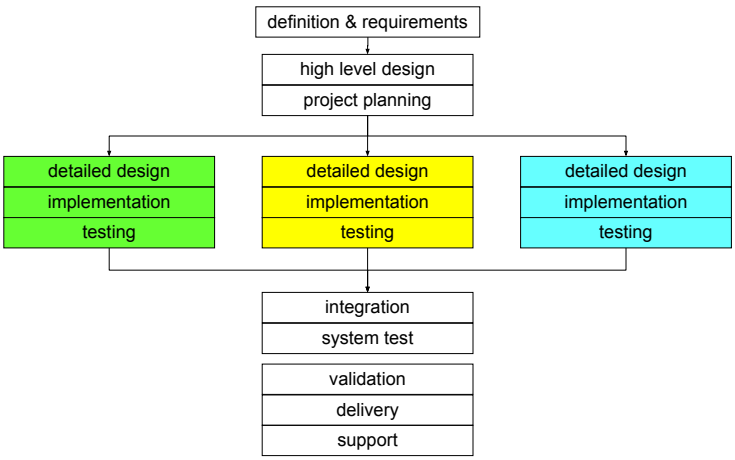
The Agile Critique of the Waterfall

- Waterfall strives to satisfy requirements
- complete/correct requirements don’t 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

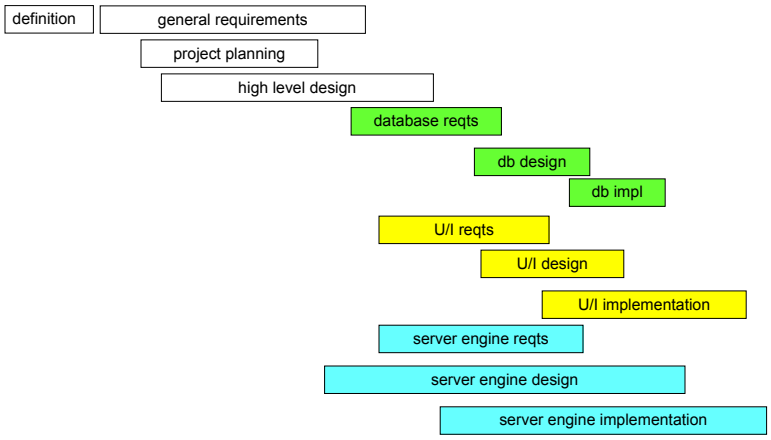
Concurrent Development



(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

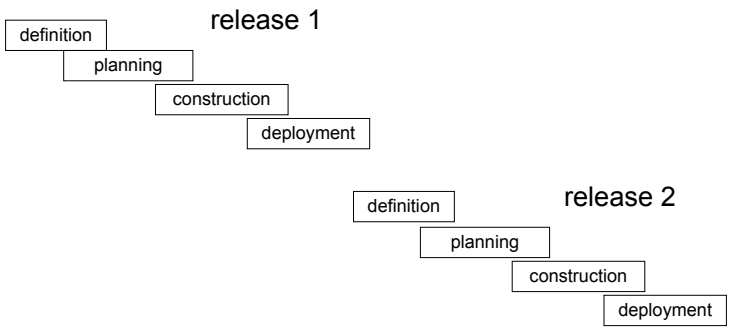
Phase Overlap



(Phase Overlap)

- phase *n+1* can start before phase *n* ends
  - there are many tasks in phase *n+1*
  - they don't all depend on all 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 R1 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

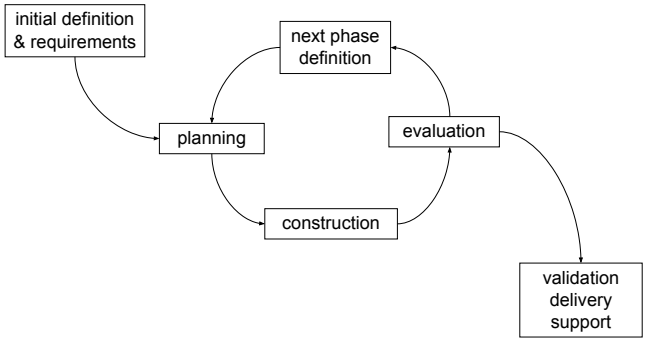
Where these break down

- 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, tools/resources, how much time
- these assumptions often fail
  - requirements for **new** products are speculative
  - estimates for **unknown** tasks are fantasy
  - surprises under **new/unfamiliar** technologies
- 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
- Option C: admit this is a research project

Iterative (spiral) Models

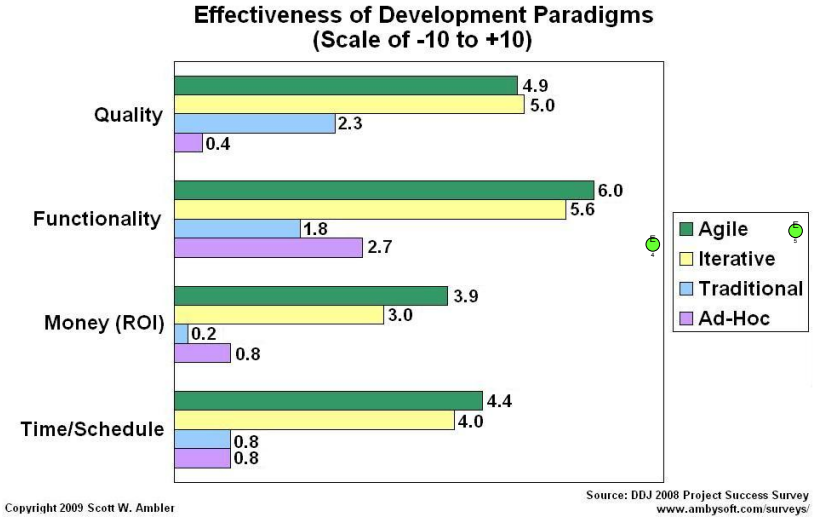


(Spiral v.s. Incremental Models)

- 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 clear goals
  - 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?

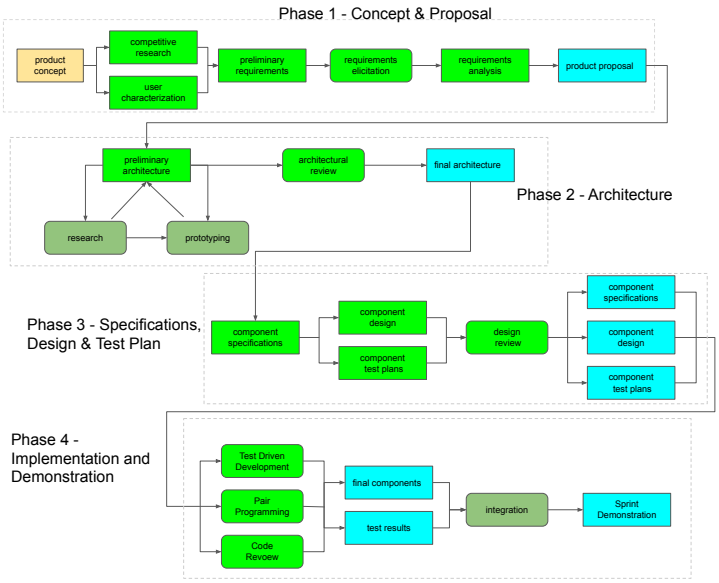


Why Models Matter

- 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

Very Different Problems

- Clarity of Requirements
  - clear: formal assertions, defined processes
  - speculative: ease of use
- Obviousness
  - a familiar problem vs. new frontiers
- Tools and Technologies
  - understanding of capabilities and use
- Project scope
  - incremental features vs. a new system
  - one small team vs. many sub-contractors



Your Projects in this Course

- Clarity of Requirements
  - you will develop preliminary requirements
  - they are unlikely to be defined for you
- Obviousness of Solution
  - complexity is required (to learn SWE skills)
- Tools and Technologies
  - you will almost surely work w/new tools
  - you will be unfamiliar w/capabilities and use
- Project Scope
  - you are building a new system from scratch
  - small (good communication) team

Team Exercise (1st iteration)

- Share your thoughts on
  - general product concepts
  - relevant team experiences and strengths
  - implementation tools and technologies
- Assess
  - how well it is likely to meet P2-4 reqts
  - how clear the design is
  - how well you understand the tools
- Punch-lines
  - what is under control
  - what is frightening (needed iterations)

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
  - look at the Project 1 Requrements section as well
- Wiegers: Requirements Traps
  - good overview of common mistakes
- Wiegers: Prioritizing Requirements
  - intro to a fundamental planning methodology

Supplementary Slides

on real commercial processes

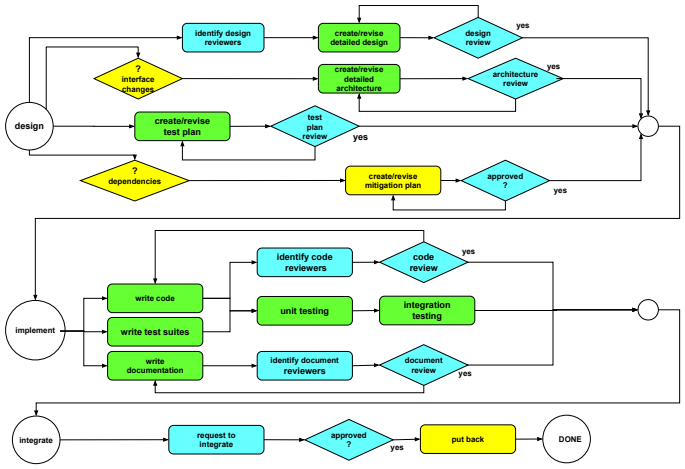
Process Specifications

- 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*

a typical formal 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

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?

Discussion Slides

Deeper

- How do we know that we are ready to move from the definition to the planning phase?

  In general, we can start the planning process as soon as our requirements seem to be credible and stable (converged).

  In novel projects, there may be many iterations of definition, requirements, and preliminary planning before real commitments are made.

Deeper

- How do we know that we are ready to move from the planning to the construction phase?

  In general, we can start the construction process when we believe the plans to be adequate (to meet the requirements) and complete.

  A good test for completeness is whether or not the people who will do the work understand what is to be done and believe that they how to do it.

Deeper

- Are there situations that would seem to demand parallel development?

  Parallel development is the first choice when the schedule requires work to be completed faster than would be possible with sequential scheduling.

  If there are multiple independent teams available to work on the project, this too would seem to be begging for parallel development.

Deeper

- Why might smaller teams be more efficient?

  Better communication and coordination, fewer misunderstandings and conflicts.

- Why might smaller tasks be lower risk?

  A smaller task is more easily understood, and problems are more easily anticipated.

- How might this improve efficiency?

  Lower overhead for communication, and fewer mistakes.

Deeper

- What would make parallel development (for nominally independent tasks) difficult?

  When multiple tasks require the same resources (typically people, but possibly labs or special equipment). This doesn't preclude parallel development, but necessitates careful scheduling.

Deeper

- Why does parallel development often lead to problems when the pieces are combined?

  The requirements and interface definitions for the independently developed components may have been ambiguous, and differently interpreted by the different teams.

  Problems encountered in the implementation of one component may require changes to the interfaces of another component to fix.

Deeper

- How can design enable or preclude parallel development?

  If the interfaces between components are simple and clean, it may be easy to develop them independently.

  If the interfaces and interactions between components are complex, it may be necessary to develop them together.

Deeper C_1

- Give an example where open requirements would not preclude the start of planning?

All requirements don't affect all components. We don't need complete requirements for A before we can start B.

Some requirements have few design implications. We may consciously choose to delay the fixing of those requirements (to get started sooner, or to give us more time to develop the right requirements).

Deeper C_2

- How do you know if it is safe to start phase n+1 before phase n is complete?

You must be able to "put a fence around" the missing information.

This means you need a complete understanding of what information is missing, and some estimate of the range within which it will fall.

These become the assumptions upon which the phase n+1 work is predicated. They must be monitored carefully.

Deeper C_3

- How might an action in the design of component A invalidate work done in the implementation of component B?

We allowed B's implementation to move forward because we believed it to be independent of the design of A. In the process of designing A, we found a problem, that could only be fixed by changing the specifications of component B.

We were wrong when we said that the design of B was independent of the design of A.

Deeper C_4

- Is it possible to thoroughly test one component, if components with which it interacts are still incomplete?

It is often possible to simulate missing components in order to test the existing ones. Such simulations can actually provide better testing than the real components would.

This does not happen by accident, but must be anticipated when the high level architecture is created.

Deeper D_1

- What kinds of things could you reasonable defer to a later release?
  - *advanced features that require more time*
  - *quality or performance enhancements*
  - *support for new protocols or technologies*

It boils down to a question of whether or not you can satisfy user demands with a subset of the proposed work.

A release doesn't have to do everything, but it does have to be useful.

Deeper D_2

- Examples of useful subsets
  - a home music server could be a very successful product, even if advanced features like "go find me other music I will probably like" are not available in the first release.
- Examples of useless subsets
  - a home music player that doesn't support mp3 or whose web interface doesn't work with Internet Explorer is a non-starter.

Deeper

- If you can choose what subset you want to build, doesn't that guarantee that your product will be buildable?
  - *There may be a hundred different features from which you can choose ... but some of those features may be absolute requirements for success. If only one of those requirements is something you can't build, then you cannot build a viable product.*
  - *This could happen if there are already products in the market, and you can only enter the market if you provide a capability that the existing products lack.*

Deeper

- How can we fund successive incremental approximations towards our ultimate goal?
  - *If we are developing a commercial product, we may be able to sell release* n*, and generate a revenue stream that justifies investment in release* n+1*.*
  - *If we are doing research, we may be able to use results obtained from version* n *to bolster our claim that we can obtain even greater results from version* n+1*.*

Deeper

- How could succesful release n create support for release n+1?
  - *By attracting customers and proving the demand for the envisioned product.*
- How could successful release n erode support for release n+1?
  - *It may turn out that release* n *solves the customer's needs, and there is no demand for the enhanced features planned for* n+1*.*

Deeper

- What if first release is not well received?
  - *Perhaps we chose the wrong feature subset.*
  - *Perhaps we implemented it poorly.*
  - *Perhaps we misunderstood the goal or market.*
  - *Perhaps this is evolution in action, and we should give up on this product and move on to something more productive.*
  - *We learned this lesson sooner, and at a lower cost than we would have, had we tried to build the whole thing.*

Deeper

- Why would anybody ever choose option A?
  - *The situation doesn't allow enough time to do option B.*
  - *They don't understand the unknowns well enough to plan attacks on them.*
  - *They lack the experience and courage to "explain things" to people higher up the food chain.*

Deeper

- Why might we have to use a spiral of product simulations to clarify user requirements?
  - *We are creating a new service or appliance. We think we know what users will want to do, but we don't know how proposed interfaces will work out, or what operations they will need that we haven't yet thought of.*

Deeper

- Why might we have to use a spiral prototyping cycles to reduce technical risk?

We are designing something complex and novel. We have a lot of ideas, but we honestly haven't got a clue how they are going to work. We should prototype some of our proposals to see how difficult they are and how well they work before committing to a particular design.

Deeper

- The advantages of agile/iterative?

Getting more frequent feedback results in fewer mistakes being made, and enables mistakes to be found and fixed sooner. It also aborts wrong directions sooner, saving considerable effort … and allowing that effort to be spent on features that will have value.

Deeper

- The advantages of agile over iterative?

They are not huge because agile methods are iterative. But the key characteristic of agile methods (that might not be present in other iterative approaches) is a high degree of customer involvement … and more feedback is likely to yield a better result.

Deeper

- What might happen if management was promised an incremental release model, but the project was actually a spiral?

This happens all the time. Engineering is going through prototyping cycles (whether they know it or not) and management is expecting a product to be released soon.

Such projects usually fail when management cuts off the funding for them.

Deeper

- What could happen if management planned on concurrent development, but did not bother to tell the designers about this?

Concurrent development assumes a high degree of independence between the components. If the designers were not told of the importance of such independence, they might adopt a design that gained other advantages at the expense of reduced independence.

This would preclude successful concurrent development efforts.