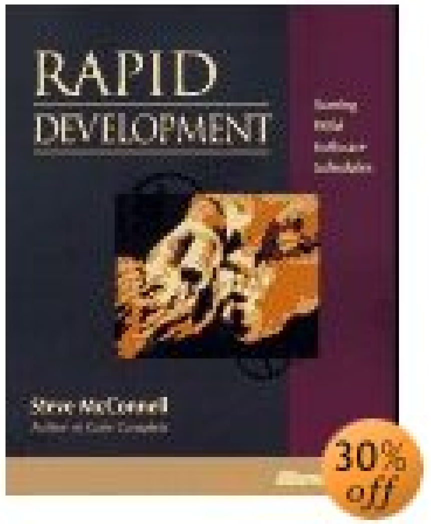
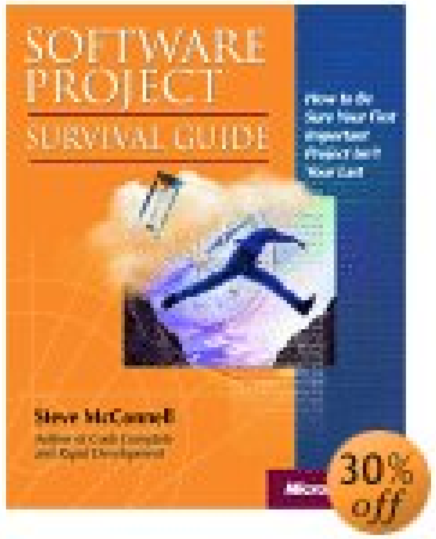
Lifecycle Planning





Rapid Development &
Software Project Survival Guide
Steve McConnell
Dave Root
(Developed with Mel Rosso-Llopart)

Topics

- Who am I to be talking to you?
- Lifecycle Defined
- Benefits of lifecycle models
- Cover eleven different models
  - Benefits and disadvantages
- Choosing an appropriate model
  - Filling in a comparison table between the models

We have a lot to cover, it will go fast

My Background
(my “I love me” slides)

- Teaching at CMU for 7 years
  - 3 yrs Leadership/ethics
  - 4 yrs SE
- Retired U.S. Navy Officer
  - Aviator
  - Top Gun graduate
  - Projects

Background

- Degrees in CS (Berkeley), Education (Chapman) and IT (CMU)
- Currently
  - Full time Lecturer
  - Associate Director of DE
  - Academic interest in
    - distributed learning
    - **agile processes**
- Other interests
  - Motorcycles, flying, Tennis, retiring….

Lifecycle Defined

Note: You must define terms

“Every software-development effort goes through a ‘lifecycle,’ which consists of all the activities between the time that version 1.0 of a system begins life as a gleam in someone’s eye and the time that version 6.74b finally takes its last breath on the last customers machine.”

Steve McConnell, Rapid Development, 1996

“The goal is often not to achieve what you said you would do at the beginning of the project, but to achieve the maximum possible within the time and resources available.”

Roger Sherman, Microsoft, 1995

What is a Life Cycle?

- Websters (1892):
  - "The series of stages in form and functional activity through which an organism passes between successive recurrences of a specified primary stage."

- Reifer (1997): (product)
  - "Period of time that begins when a software product is conceived and ends when the product is retired from use."

What is a Life Cycle?

Tony Lattanze

- The *software lifecycle* is the *cradle to grave* existence of a software product or software intensive system
    - includes initial development, repairs, and enhancement, and decommission
- Management of the entire lifecycle of a software intensive system requires a deeper knowledge than basic in-the-small development intuition and experience

More on What...

- Lifecycle models attempt to generalize the software development process into steps with associated activities and/or artifacts.
  - They model how a project is planned, controlled, and monitored from inception to completion.
- Lifecycle models provide a starting point for defining what we will do.

So...What is a Process?

(remember this for the process lectures)

- A process is a sequence of steps performed for a given purpose.

Websters:

"a series of actions or operations conducing to an end."

The concept of software process is rarely presented in undergraduate education.

Process ≠ Lifecycle

- Software process is not the same as life cycle models.
  - process refers to the specific steps used in a specific organization to build systems
  - indicates the specific activities that must be undertaken and artifacts that must be produced
  - *process definitions* include more detail than provided lifecycle models
- Software processes are sometimes defined in the context of a lifecycle model.

Benefits of a Lifecycle Model

REPEATABLE!

- Streamline project
- Improve development speed
- Improve quality
- Improve project tracking and control
- Minimize overhead
- Minimize risk exposure
- Improve client relations

Life Cycles

- Ad Hoc
- Classic (waterfall)
- Prototype
  - Throw away and evolutionary
- RAD
- Incremental
- Spiral
- Design to Schedule
- Evolutionary delivery
- COTS

This list is not all inclusive... there are more .... maybe

Look at with respect to:

- Scope, time, resources, quality
- Stakeholders
- Requirements volatility
- Environments
  - Business / market
  - Cultures
  - Moral, legal constraints
- And More...

Ad Hoc
"Hobbyist"

- Legacy
- Code – Test – Code – Test……..
  - Becomes a mess, chuck it, start over
- Design (high level) – Code – Test – Code – Test…..
  - (Reality was Code - Test – Code – Test – Document the resulting design)
- Maintenance Phase: Test – Code - Test

Waterfall Model

also called traditional

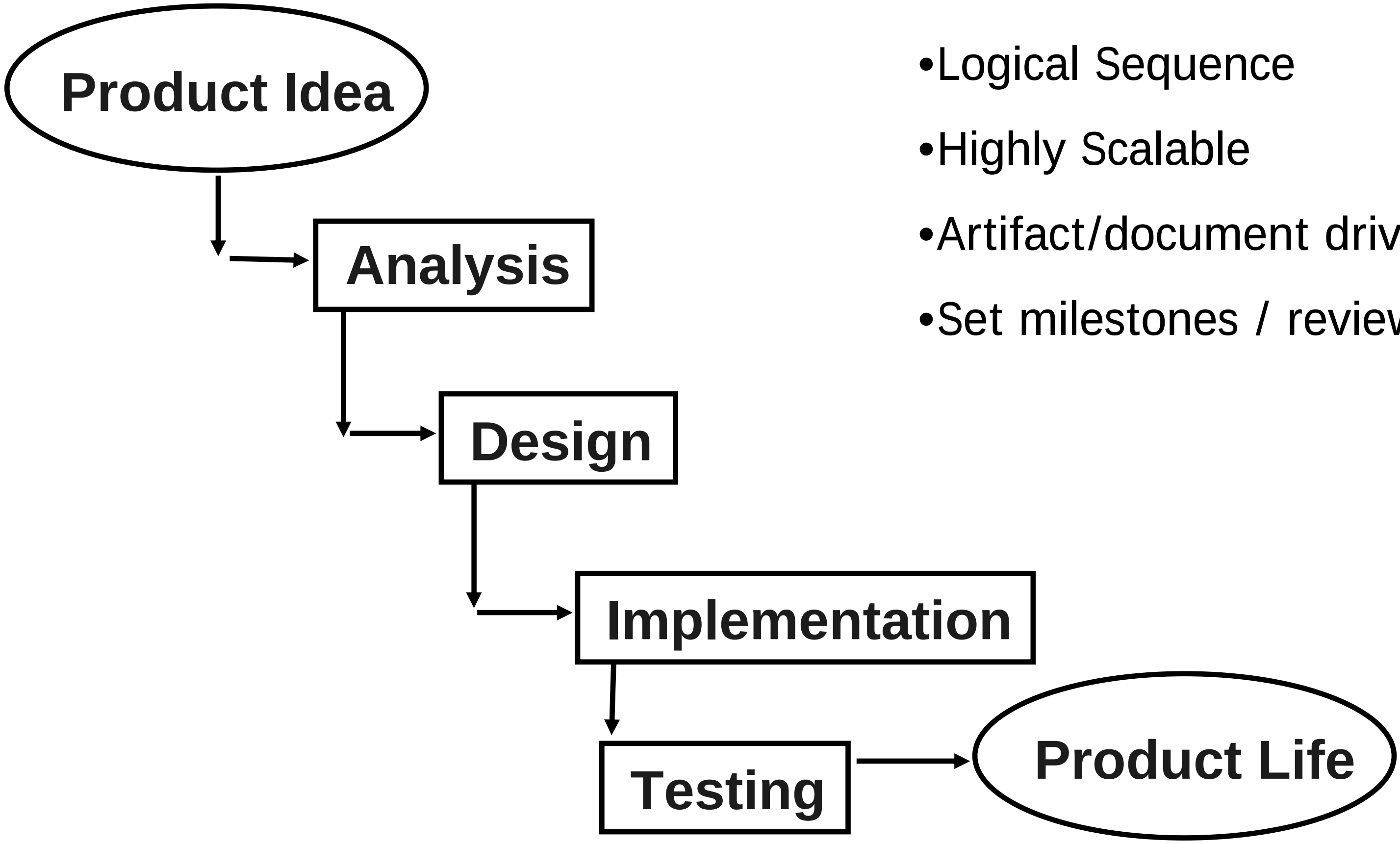
- First proposed in 1970 by W.W. Royce
- Development flows steadily through:
  - requirements analysis, design implementation, testing, integration, and maintenance.

Note: Royce advocated iterations of waterfalls adapting the results of the precedent waterfall.

Waterfall Model

- Technology had some influence on the viability of the waterfall model.
  - slow code, compile, and debug cycles
- Reflected the way that other engineering disciplines build things.
- Formed the basis of the earliest software process frameworks
- Waterfall is still used today (but no one will admit it)

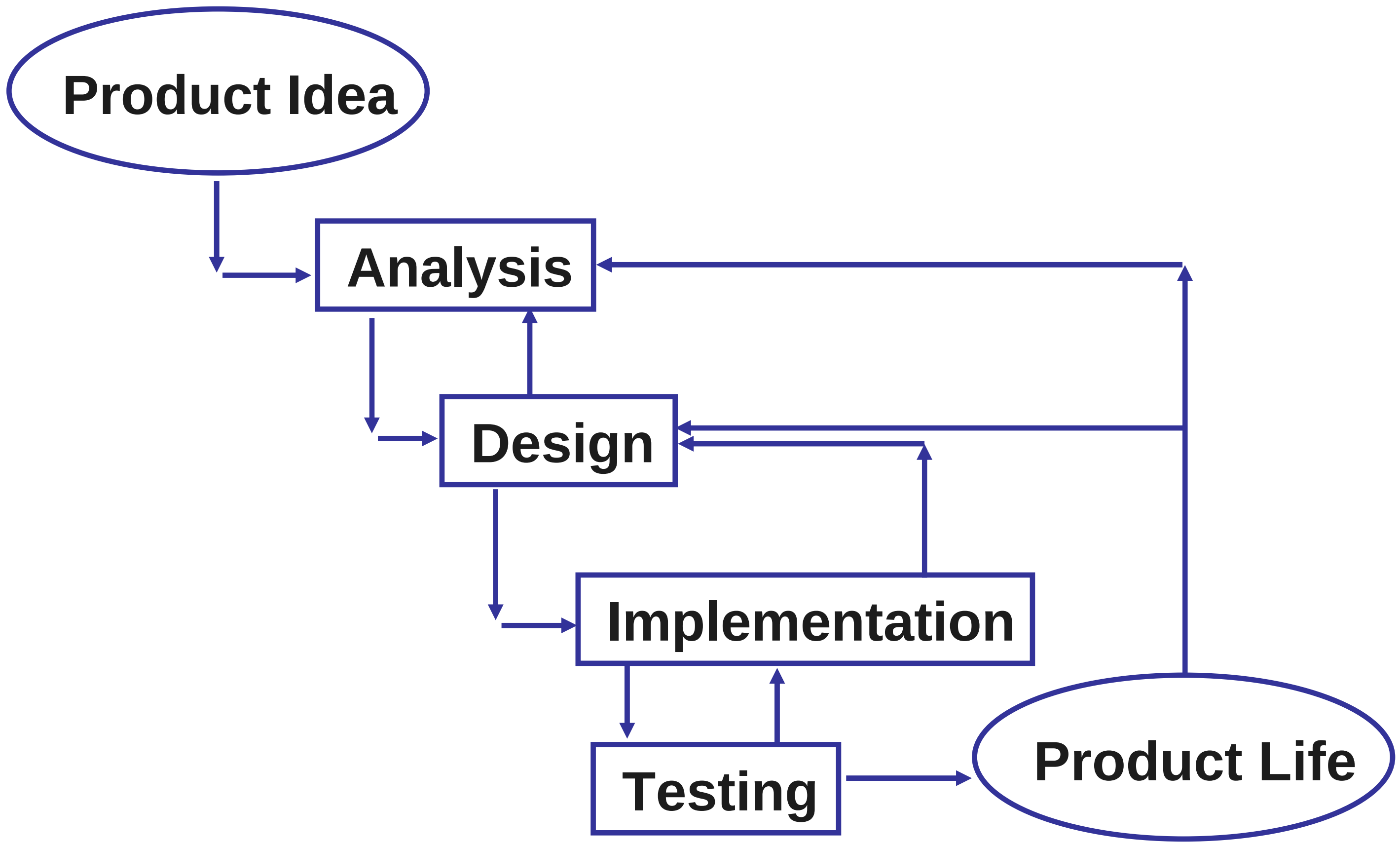
Waterfall (linear) (Classic) Model Intent



Benefits:

- •Logical Sequence
- •Highly Scalable
- •Artifact/document driven
- •Set milestones / review points

Waterfall Model: Reality



Waterfall Problems

- Need for "big specification" (requirements)
- Inflexible
- Increasing use of resources?
  - Oops
    - Go back to a previous step
    - Progressively more costly
- No results till end
- Possible cost of cascading bugs
- Importance of secondary artifacts
- Where appropriate?

From Chris Kemerer……
Reality of Waterfall

1. Enthusiasm
2. Disillusionment
3. Panic & Hysteria
4. Search for the Guilty
5. Punishment of the Innocent
6. Praise & Honors for the non-participants

Throw away Prototype

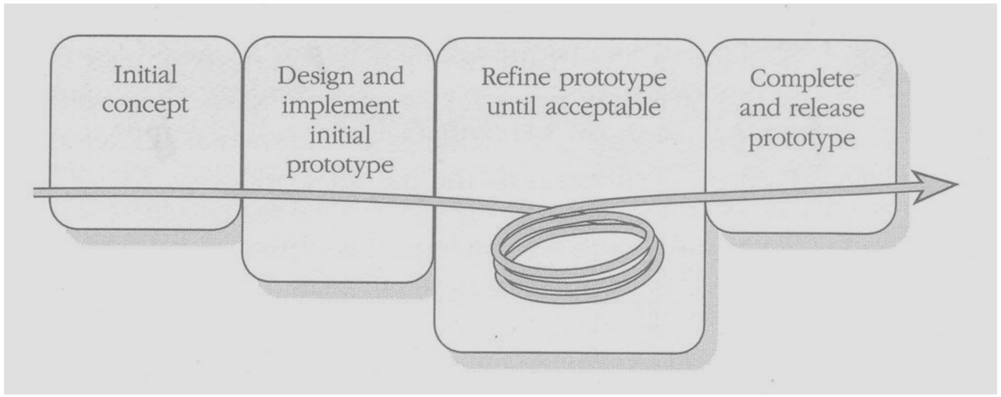
- Proof of concept – It can be done
- End point unknown!
- Goal is domain knowledge increase
- Disadvantages
  - Seen as project completion
    - But not robust
  - Quality

Evolutionary Prototype

- Keep something
- Different than incremental?
- The evolutionary development model can be distinguished from the prototyping model in that
  - a final product is typically specified
  - the product features are *evolved* overtime to some predetermined final state

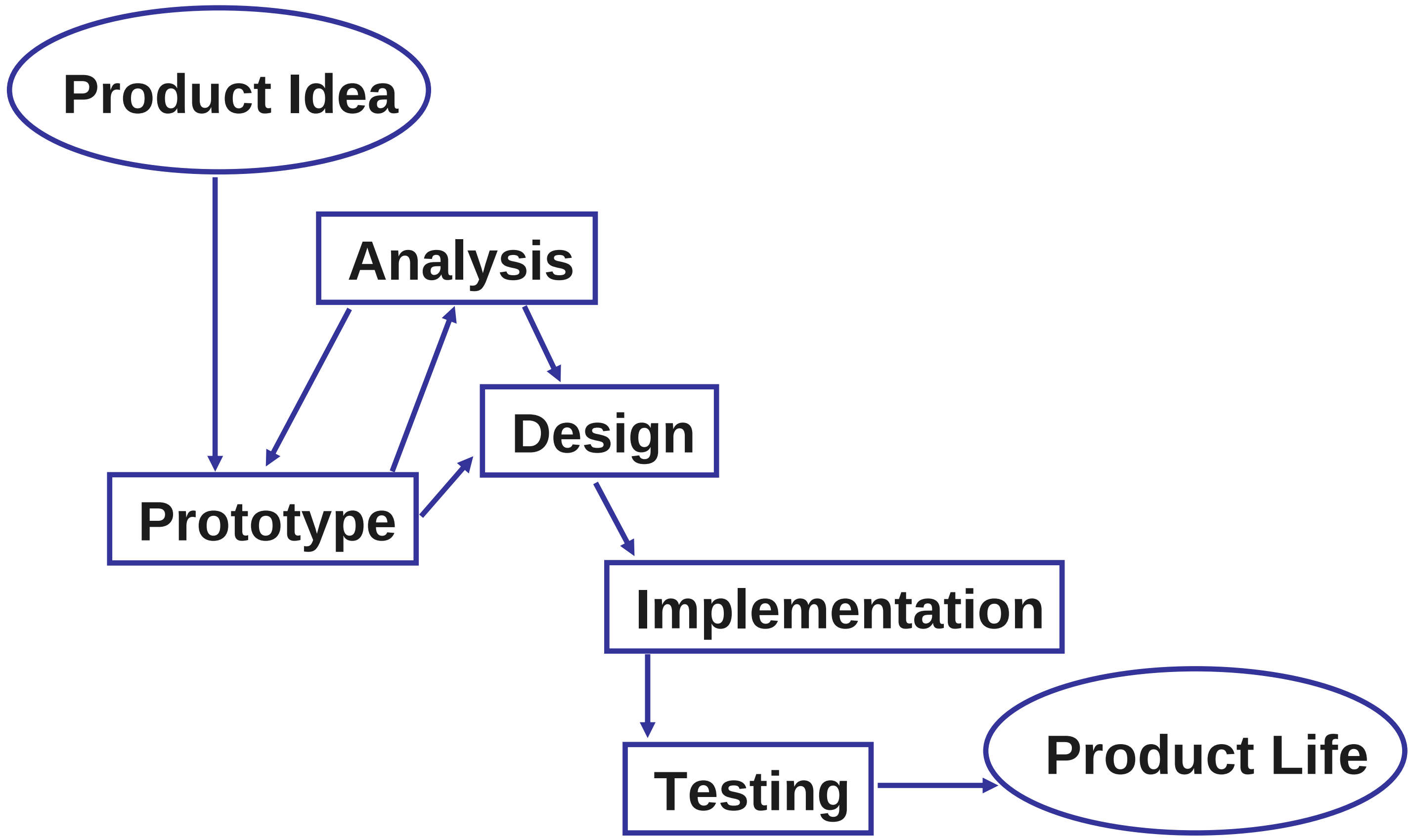
Evolutionary Prototyping

- Develop system concept as the project progresses
- Begin with the most visible aspects
- Prototype

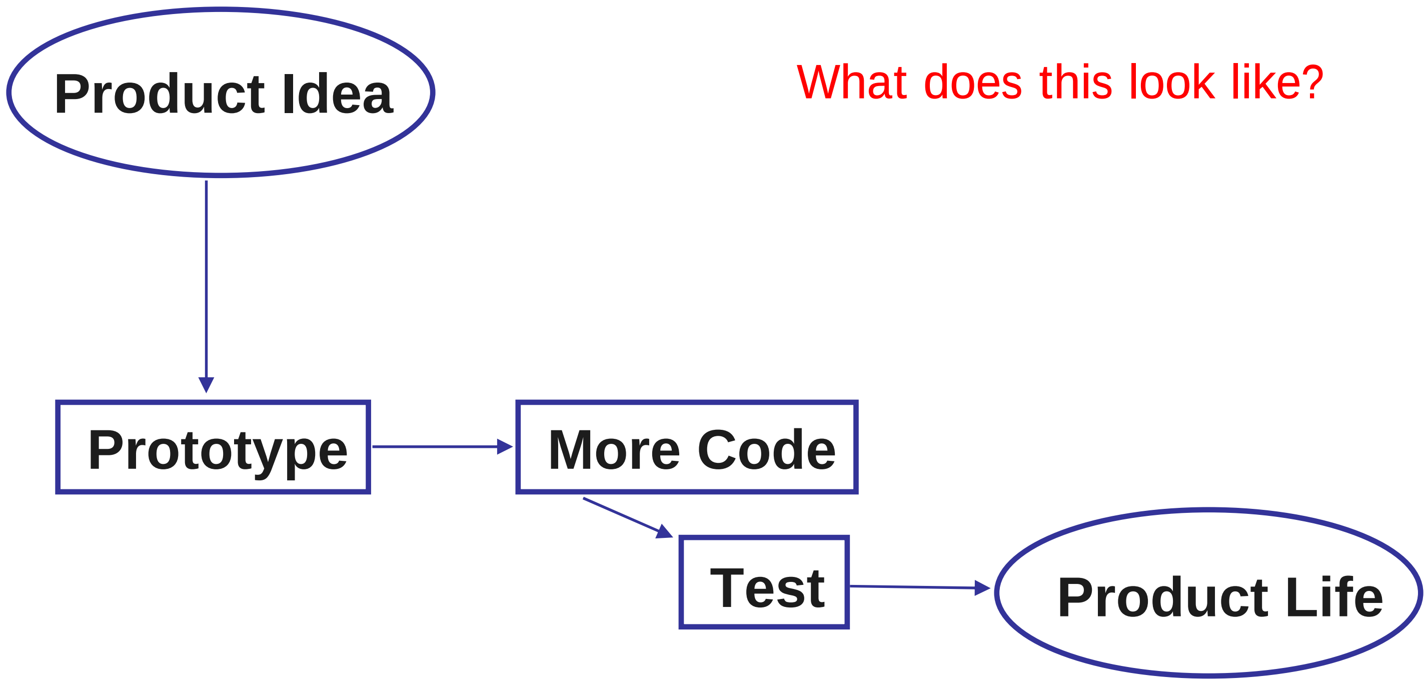

Initial concept
Design and implement initial prototype
Refine prototype until acceptable
Complete and release prototype

Rapid Development, 1996

The Rapid Prototype Model



A Common Misuse of the Rapid Prototype Model



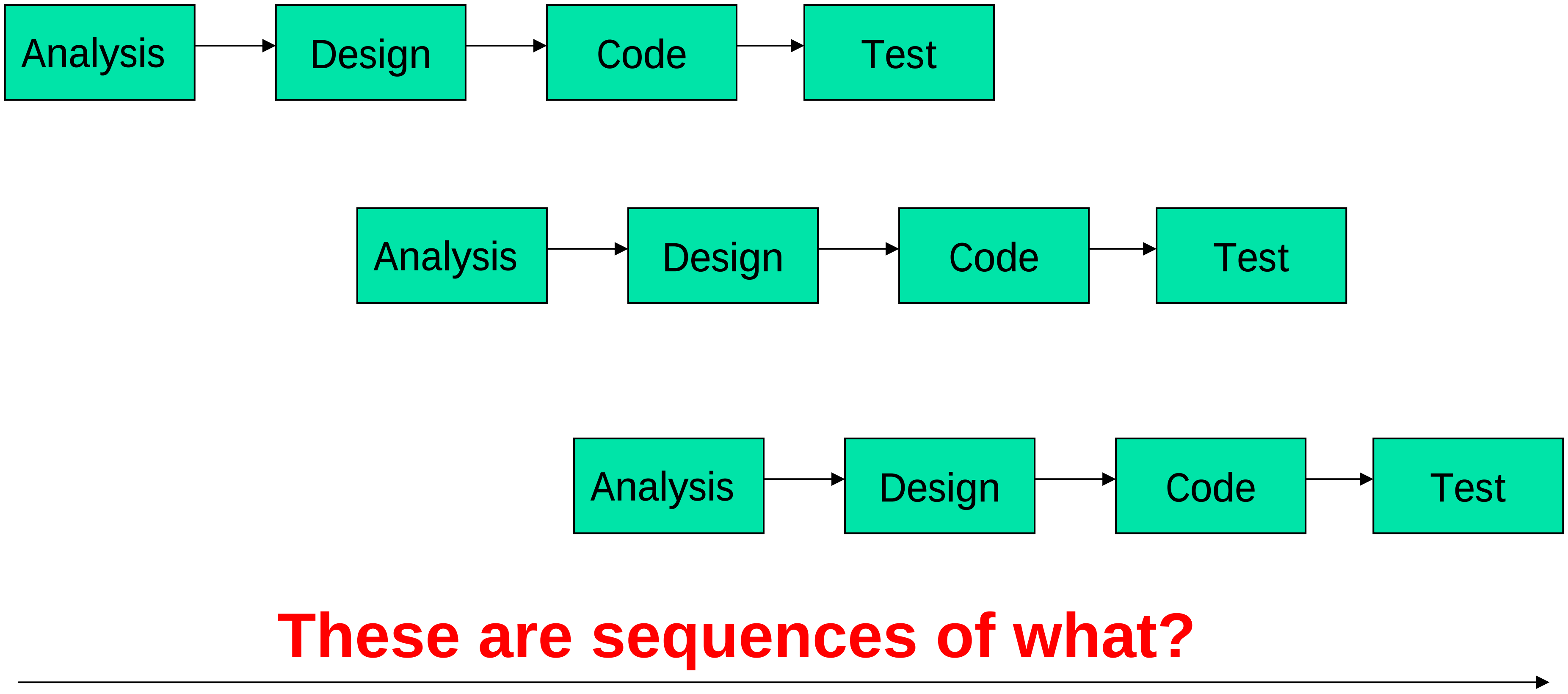
What does this look like?

Incremental
Typical Agile Model

- The incremental model prescribes *developing* and *delivering* the product in *planned* increments.
  - The product is *designed* to be *delivered* in *increments*.
  - Each increment provides (in theory) more functionality than the previous increment.
- How is this different from Evolutionary?

Incremental Model

(what “blocks” are missing?)



These are sequences of what?

Incremental / Agile methods

- Customer centric – customer expectation management
  - Deliver every increment
  - Develop the product with tight customer involvement
  - Use customer needs to drive priorities for the project
- Similar in many aspects to “rapid prototyping”
- Use “small team” integration to handle many project issues
  - Scrum and XP are primary examples

Agile Advantages

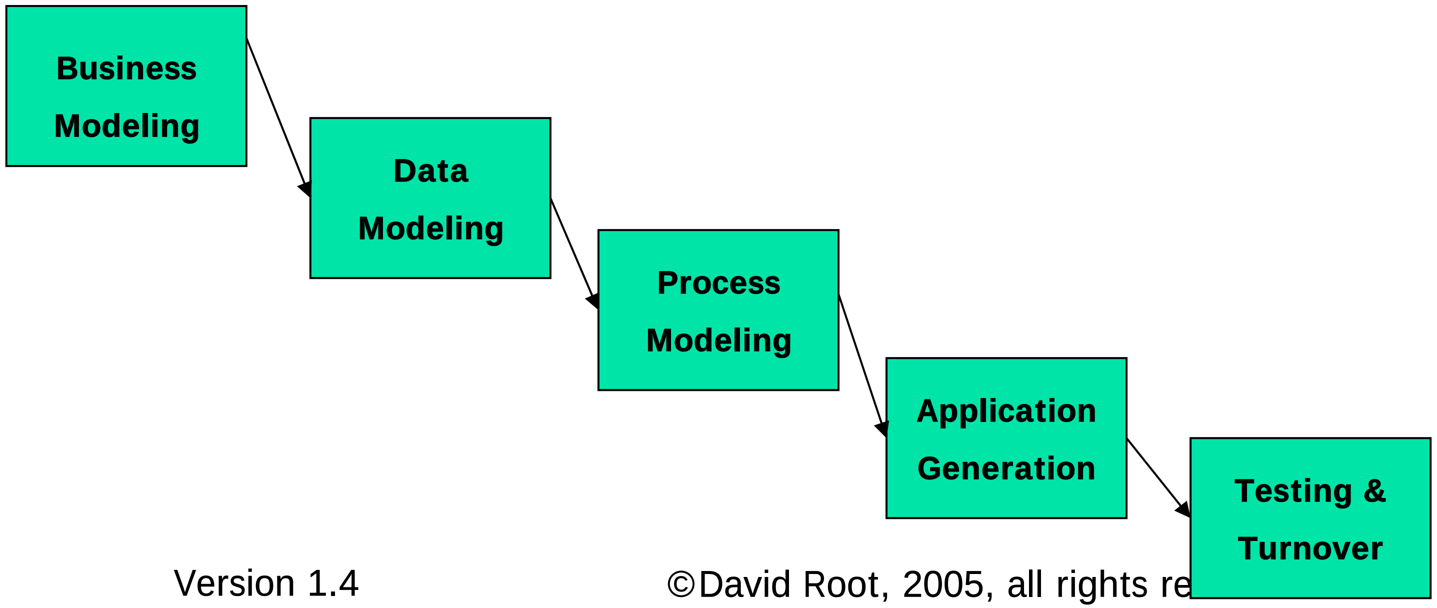
- Highly Flexible – volatile requirements
- Works on what is important for the customer
- Primary artifact is Code
- Short increments reduce failure impact
- Use teaming controls to improve quality

Disadvantages

- Team co-location required - maybe
- Scope is looked at in “short cycles”
  - “What can be done in a week”
- Drive towards product only focus
  - Maintenance issues
  - Documentation – Legal, safety of life
  - Design on the fly
- Scalability

Rapid Application Development (RAD)

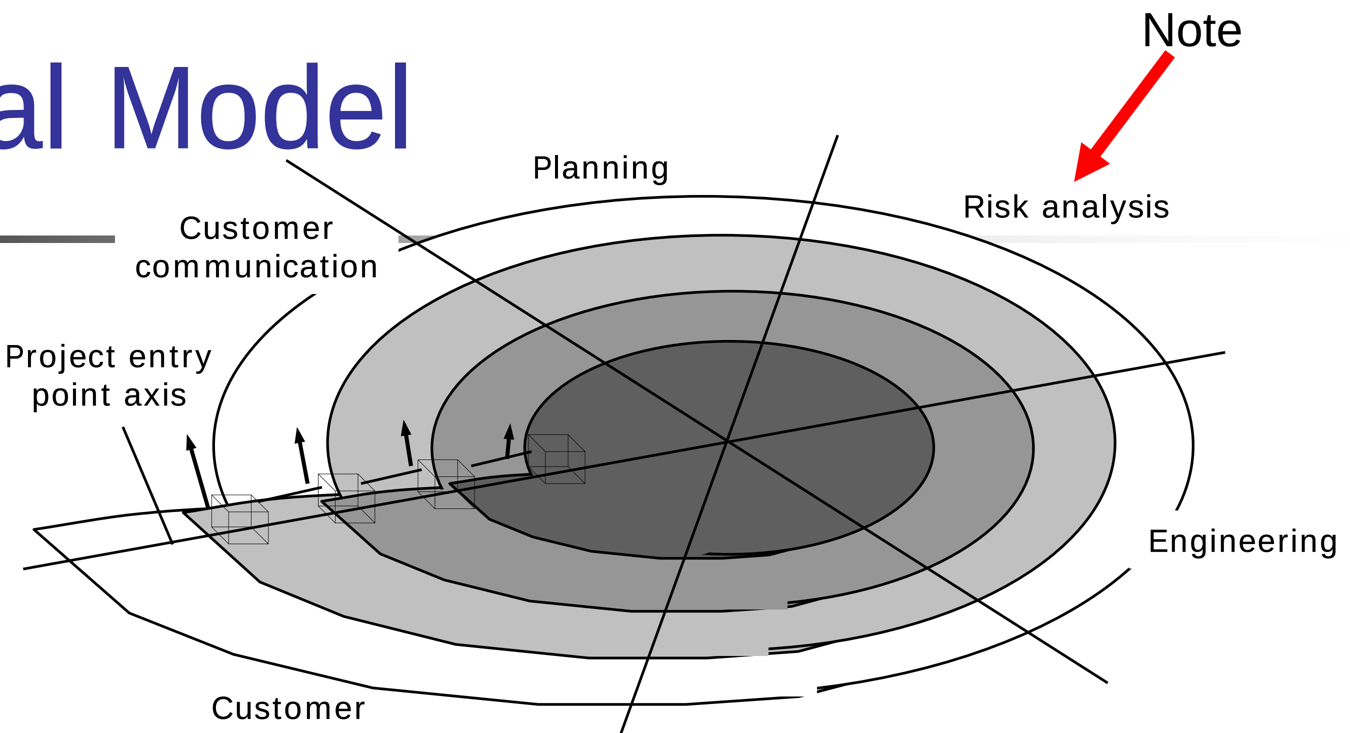
- Incremental
- <u>60-90 days</u> per release
- Information Systems
- 4th Generation Techniques



Spiral Model

- The spiral model
  - First defined by Barry Boehm
  - combines elements of:
    - evolutionary, incremental, and prototyping models
  - First model to explain
    - why iteration matters
    - How iteration could be used effectively
  - the term *spiral* refers successive iterations outward from a central starting point.

Spiral Model



☐	Product maintenance projects
▨	Product enhancement projects
▨	New product development projects
■	Concept development projects

Roger S. Pressman's "Software Engineering, a Practitioners Approach"

Spiral Model

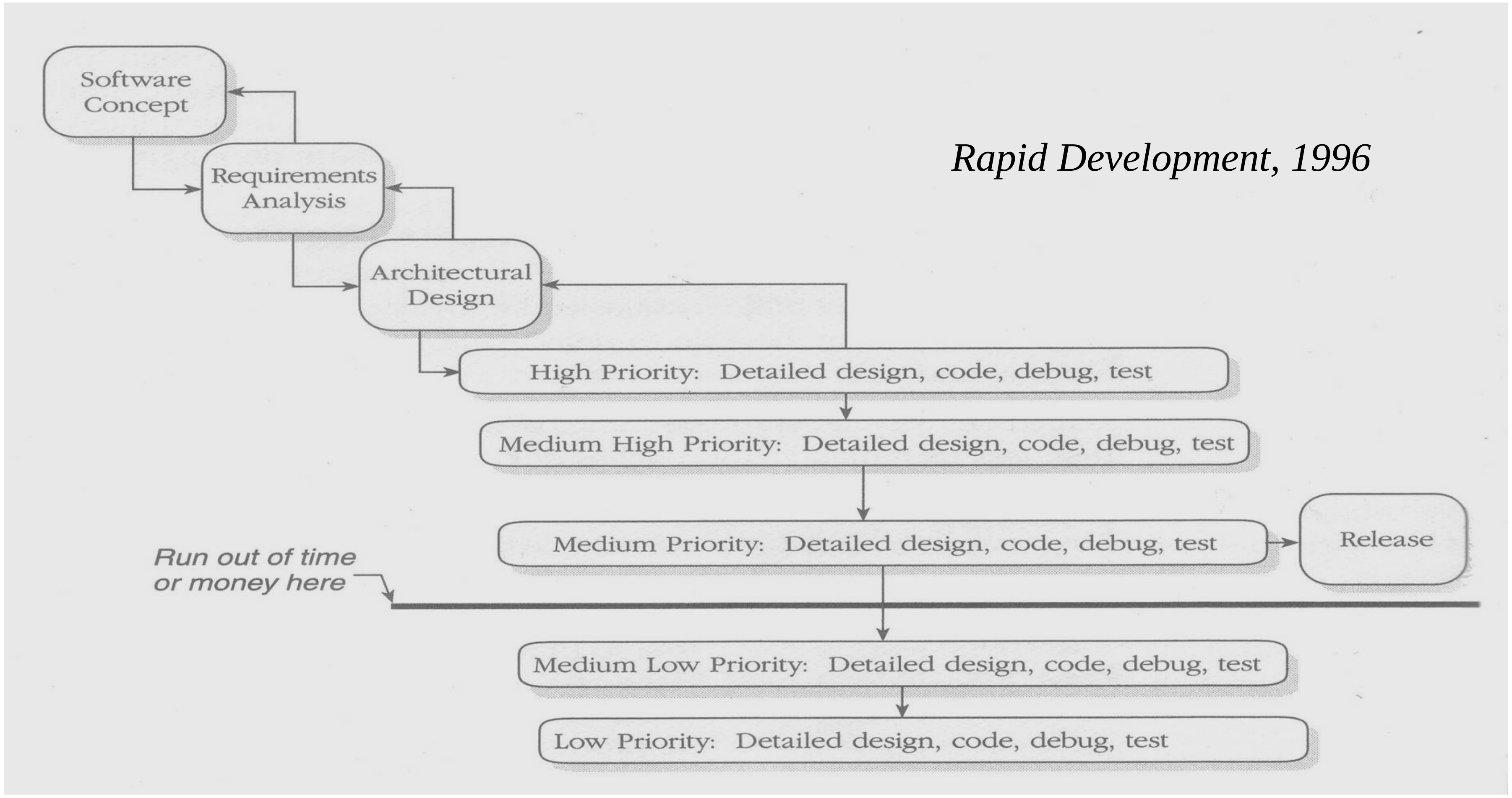
- The goal is to identify risk and focus on it early.
- In theory, risk is reduced in outer spirals as the product becomes more refined.
  - Cost/time increases reduce risk
- Each spiral
  - starts with design goals
  - ends with the client reviewing the progress thus far and future direction
  - was originally prescribed to last up to 2 years
- Flexible

Possible Applications

- High risk projects
  - Poorly understood requirements
  - Poorly understood architecture
  - Potential performance problems
  - Problems in the underlying technology
- Combine with other lifecycle models
  - Terminate with waterfall or other lifecycle
  - Incorporate other lifecycle models as iterations

Design-to-Schedule

- Prioritize features
- Unsure if final release will be reached


Software Concept
Requirements Analysis
Architectural Design
High Priority: Detailed design, code, debug, test
Medium High Priority: Detailed design, code, debug, test
Medium Priority: Detailed design, code, debug, test
Release
Run out of time or money here
Medium Low Priority: Detailed design, code, debug, test
Low Priority: Detailed design, code, debug, test

Rapid Development, 1996

Design-to-Schedule Benefits

- Ensure product release for a particular date
- Most important features completed first
- Useful for project parts not on the critical path

Design-to-Schedule Disadvantages

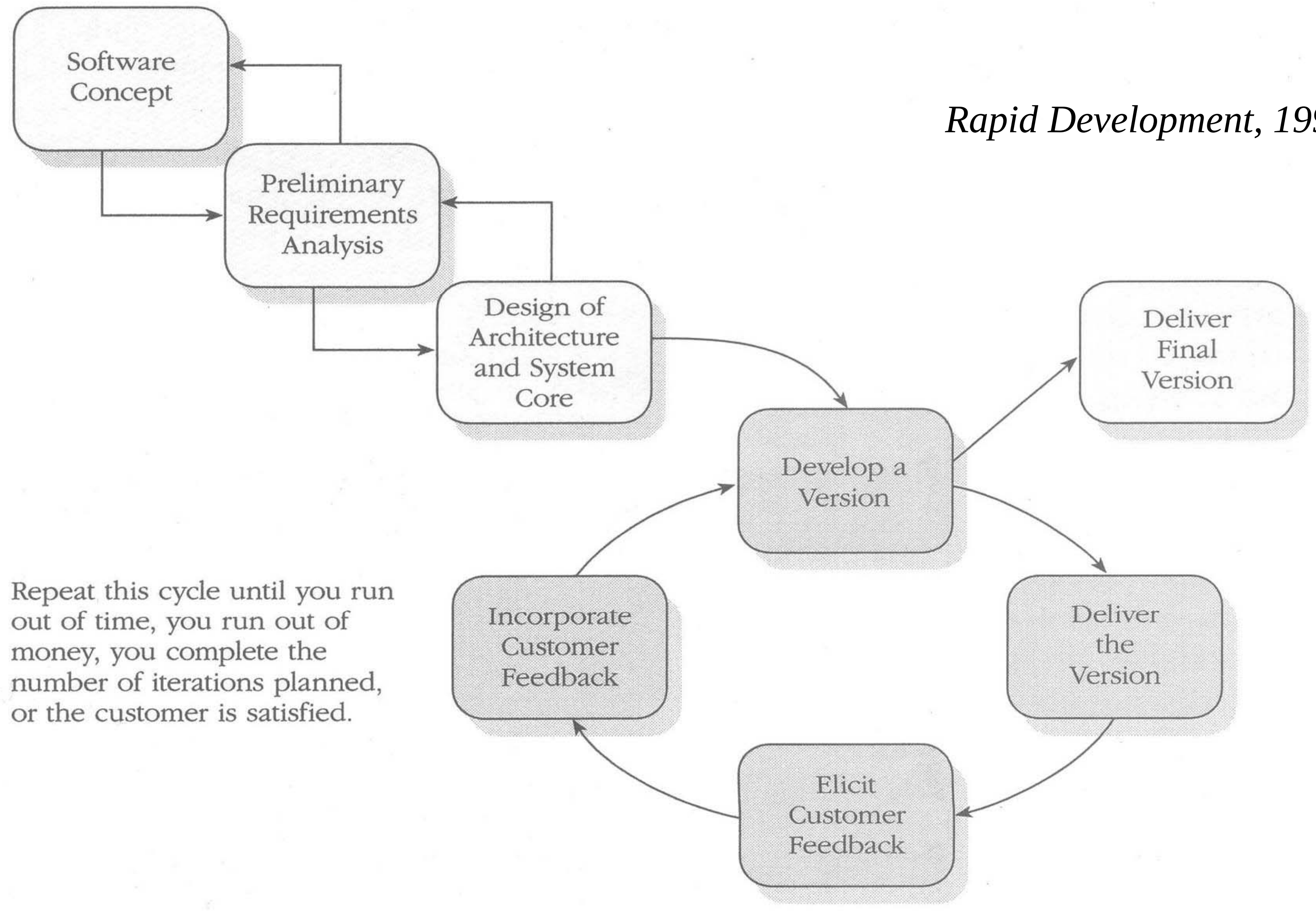
- Wasted effort specifying unfinished stages
  - Could complete one or more stages if time was not wasted specifying several unfinished stages

Evolutionary Delivery

- Similar to Evolutionary Prototyping
- Refine version based upon customer feedback
- Emphasizes core of the system

Evolutionary Delivery

Rapid Development, 1996


Software Concept
Preliminary Requirements Analysis
Design of Architecture and System Core
Develop a Version
Deliver Final Version
Deliver the Version
Elicit Customer Feedback
Incorporate Customer Feedback

Repeat this cycle until you run out of time, you run out of money, you complete the number of iterations planned, or the customer is satisfied.

Evolutionary Delivery

- Benefits
  - Can accommodate customer requests
  - Allows a degree of midcourse changes
  - Provides tangible results
- Disadvantages
  - Requires careful planning
  - May lead to Code-and-Fix development
- Use for Exceptionally time-sensitive projects

Commercial Off-the-Shelf Software

- Cycle
  - Identify Possible ones
  - Check Library
  - Use (if they exist)
  - Build new ones (if they don't
  - Put new ones in Library
- But: Software rarely matches ideal software
  - Design concessions
  - Cost concessions
  - Schedule concessions

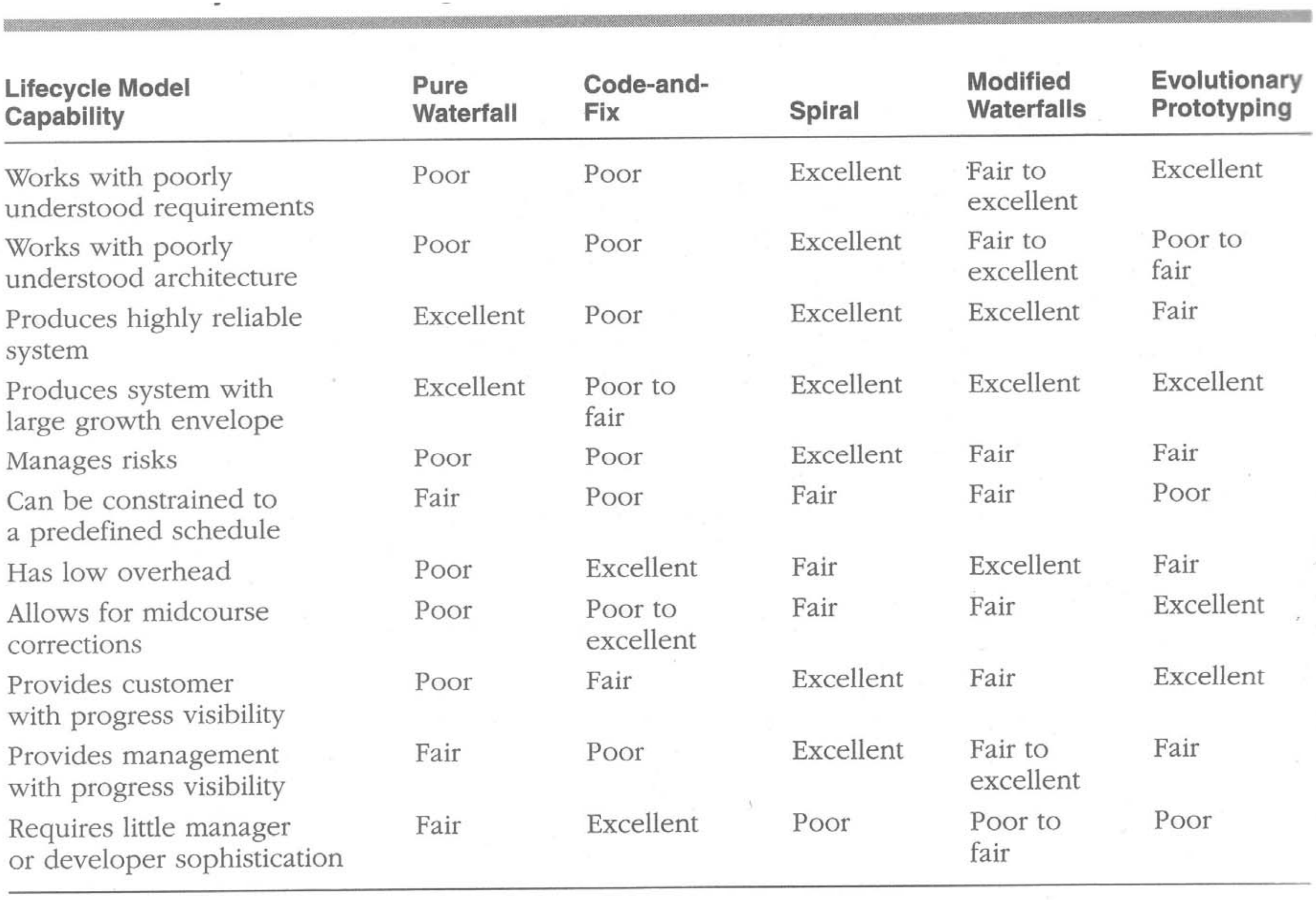
Choosing: Criteria to consider

- Requirements understood? Volatile?
- Scope of project
- External constraints?
- Need for design / architecture
- Quality
- Future revisions?
- How much risk can you accept
- Schedule / Resource constraints

Choosing An Appropriate Lifecycle (cont.)

- Need to provide visible progress to customers
- Need to provide visible progress to management
- How sophisticated (complicated) is the model

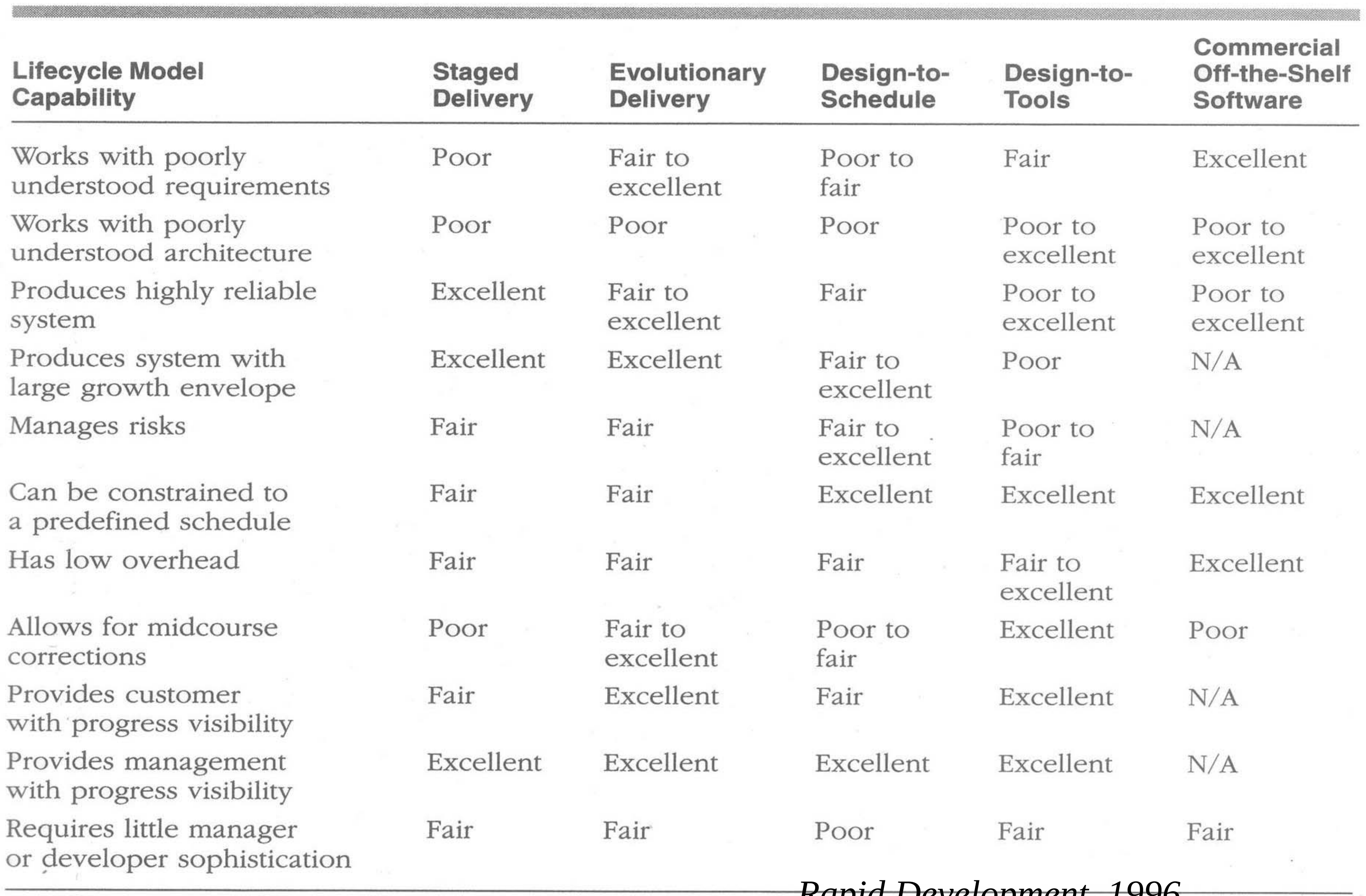
Strengths & Weaknesses



Lifecycle Model Capability	Pure Waterfall	Code-and-Fix	Spiral	Modified Waterfalls	Evolutionary Prototyping
Works with poorly understood requirements	Poor	Poor	Excellent	Fair to excellent	Excellent
Works with poorly understood architecture	Poor	Poor	Excellent	Fair to excellent	Poor to fair
Produces highly reliable system	Excellent	Poor	Excellent	Excellent	Fair
Produces system with large growth envelope	Excellent	Poor to fair	Excellent	Excellent	Excellent
Manages risks	Poor	Poor	Excellent	Fair	Fair
Can be constrained to a predefined schedule	Fair	Poor	Fair	Fair	Poor
Has low overhead	Poor	Excellent	Fair	Excellent	Fair
Allows for midcourse corrections	Poor	Poor to excellent	Fair	Fair	Excellent
Provides customer with progress visibility	Poor	Fair	Excellent	Fair	Excellent
Provides management with progress visibility	Fair	Poor	Excellent	Fair to excellent	Fair
Requires little manager or developer sophistication	Fair	Excellent	Poor	Poor to fair	Poor

Rapid Development, 1996

Strengths & Weaknesses - 2



Lifecycle Model Capability	Staged Delivery	Evolutionary Delivery	Design-to-Schedule	Design-to-Tools	Commercial Off-the-Shelf Software
Works with poorly understood requirements	Poor	Fair to excellent	Poor to fair	Fair	Excellent
Works with poorly understood architecture	Poor	Poor	Poor	Poor to excellent	Poor to excellent
Produces highly reliable system	Excellent	Fair to excellent	Fair	Poor to excellent	Poor to excellent
Produces system with large growth envelope	Excellent	Excellent	Fair to excellent	Poor	N/A
Manages risks	Fair	Fair	Fair to excellent	Poor to fair	N/A
Can be constrained to a predefined schedule	Fair	Fair	Excellent	Excellent	Excellent
Has low overhead	Fair	Fair	Fair	Fair to excellent	Excellent
Allows for midcourse corrections	Poor	Fair to excellent	Poor to fair	Excellent	Poor
Provides customer with progress visibility	Fair	Excellent	Fair	Excellent	N/A
Provides management with progress visibility	Excellent	Excellent	Excellent	Excellent	N/A
Requires little manager or developer sophistication	Fair	Fair	Poor	Fair	Fair

Rapid Development, 1996

Strength and Weakness - 3

Lifecycle model	**Agile methods**		
Poorly understood requirements	Good		
Poorly understood Architecture	Poor		
Produces highly reliable system	Fair		
Produce system with large growth	Fair		
Manage Risks	Good		
Can be schedule constrained	Good		
Has Low overhead	Good		
Allows midcourse corrections	Good		
Customer visibility	Excellent		
Management visibility	Fair		
Mgmt or developer sophistication	Poor		

Common Errors in Choosing

- Tailoring project to fit lifecycle
  - Looking for a recipe
  - NO! Tailor lifecycle
  - Imbedded lifecycles
- Supermarket approach
  - Pick and choose?
- It must be repeatable!

Questions?