CSE 403
Lecture 2

Software Lifecycle Models

Thanks to Marty Stepp, Michael Ernst, and other past instructors of CSE 403
http://www.cs.washington.edu/403/

Lecture outline

- The software lifecycle
  - evaluating models
- Lifecycle models
  - code-and-fix
  - waterfall
  - spiral
  - evolutionary prototyping
  - staged delivery
  - design-to-* (schedule, tools, etc.)

Big questions

- What is a software lifecycle model? When and why should we use such models?
- How do we decide which model is the "best" one to use?
- Briefly describe each of these models:
  - code-and-fix, waterfall, spiral, evolutionary prototyping, staged delivery, design-to-schedule, etc.
- What are some benefits and drawbacks of each model?

How complex is software?

- Measures of complexity:
  - lines of code
    - Windows Server 2003: 50 MSLoC
    - Debian 5.0: 324 MSLoC *(61 years to type at 50wpm!)*
  - number of classes
  - number of modules
  - module interconnections and dependencies
  - time to understand
  - # of authors

  - … many more

Ad-hoc development

- **ad-hoc development**: no formal process (aka "code and fix")
  - Sounds great! No learning required.

- drawbacks?
  - some important actions (design, testing) may go ignored
  - not clear when to start or stop doing each task
  - does not scale well to multiple people
  - not easy to review or evaluate one's work
  - code didn't match user's needs (no requirements!)
  - code was not planned for modification, not flexible

- Key observation: The later a problem is found, the more expensive it is to fix.

The "Software Lifecycle"

- **software lifecycle**: The entire **process** of creating a software product from an initial concept until the last user stops using it.
  - often divided into "phases" although the ordering may vary:
    - Requirements Analysis & Specification
    - High-level (Architectural) Design
    - Detailed (Object-oriented) Design
    - Implementation, Integration, Debugging
    - Testing, Profiling, Quality Assurance
    - Operation and Maintenance
    - other possibilities: Risk Assessment, Prototyping
  - goals of each phase:
    - mark out a clear set of steps to perform
    - produce a tangible document or item (aka "artifact" or "deliverable")
    - allow for review of work
    - specify actions to perform in the next phase

Some lifecycle models

- **code-and-fix**: write some code, debug it, repeat (i.e., *ad-hoc*)
- **waterfall**: standard phases (req., design, code, test) in order
- **spiral**: assess risks at each step; do most critical action first
- **evolutionary prototyping**: build an initial small requirement spec, code it, then "evolve" the spec and code as needed
- **staged delivery**: build initial requirement specs for several releases, then design-and-code each in sequence
- **agile development**: iterative, adaptive, incremental improvement done by self-organizing cross-functional teams

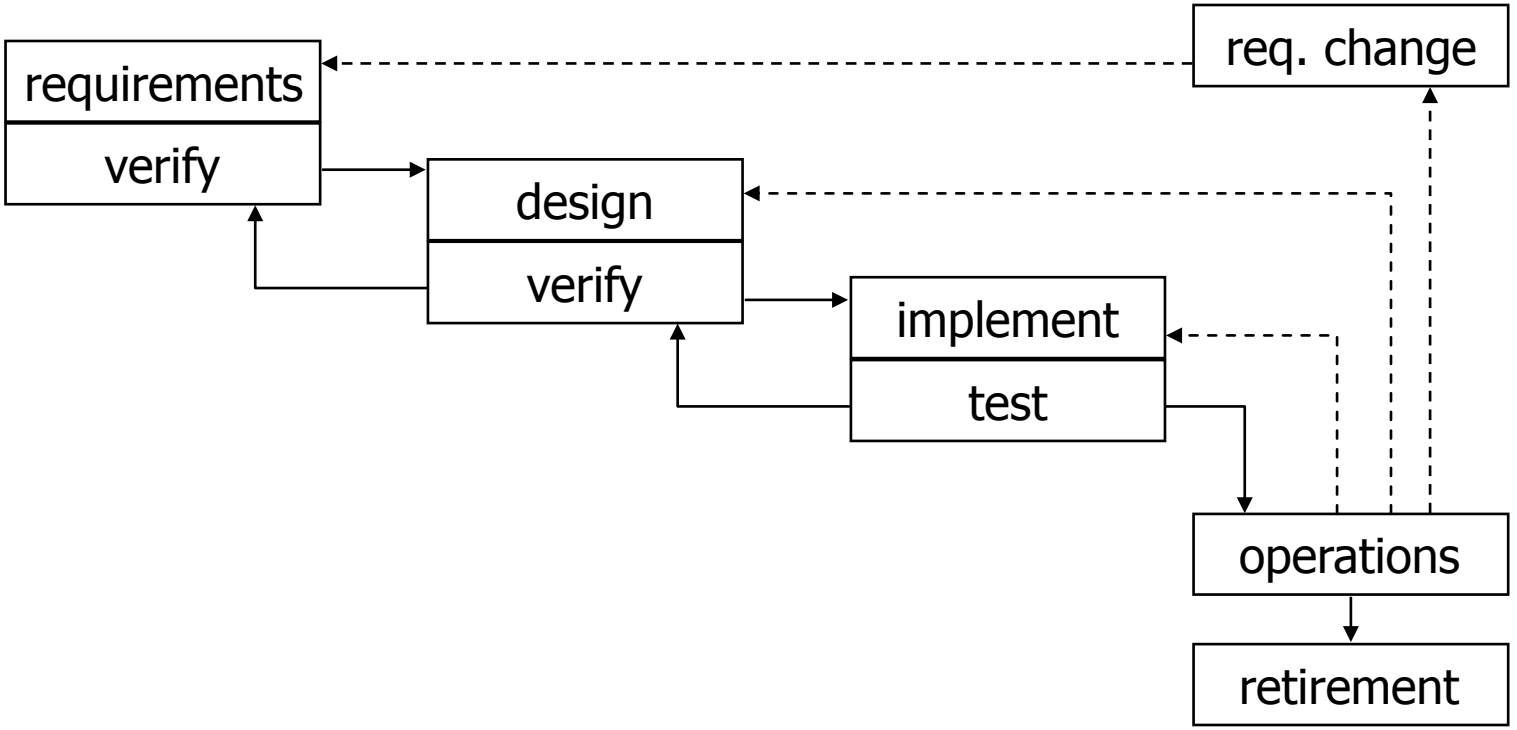
Benefits/limits of models

- benefits of models 
  - structures workflow, decomposes workflow, helps us understand/manage process

- limitations of models 
  - can lead to compromises and artificial constraints
  - risk of overemphasizing process (not the end in itself)

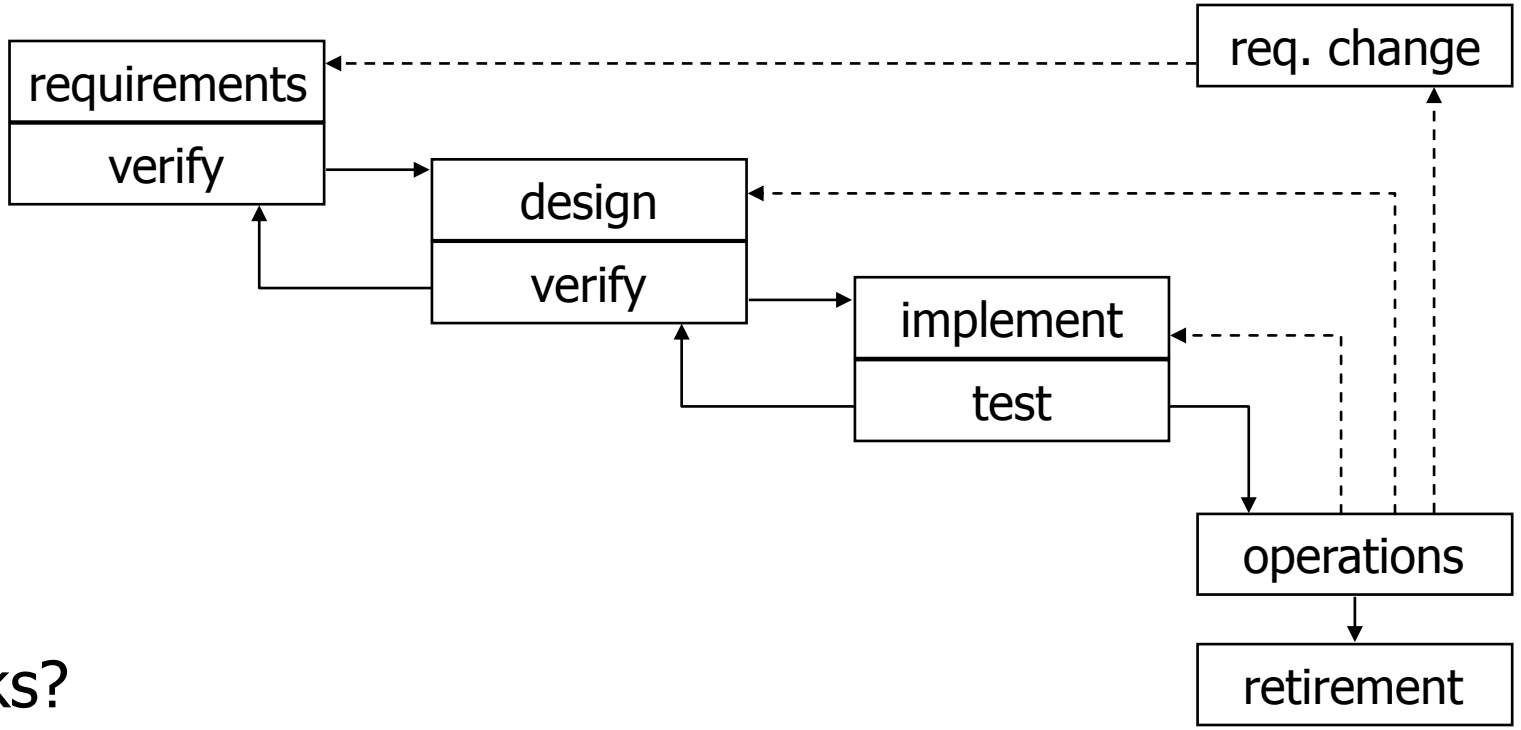
- ways of evaluating models 
  - risk management, quality/cost control, predictability, visibility of progress, customer involvement/feedback

Waterfall

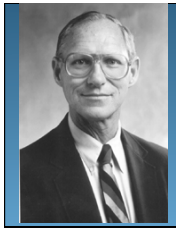


- benefits?
  - formal, standard; specific phases with clear goals
  - clear divisions between phases
  - good feedback loops between adjacent phases
  - supports inexperienced teams

Drawbacks of waterfall



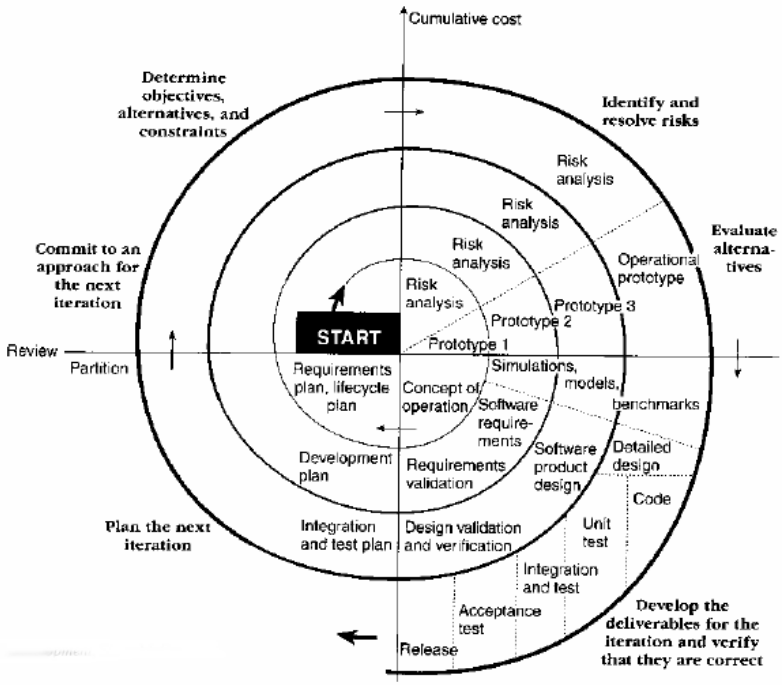
- drawbacks?
  - assumes requirements will be clear and well-understood
  - requires a lot of planning up front (not always easy)
  - rigid, linear; not adaptable to change in the product
  - costly to "swim upstream" back to a previous phase
  - nothing to show until almost done ("we're 90% done, I swear!")



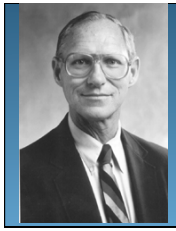
Spiral

Barry Boehm, USC

- steps taken at each loop:
  - determine objectives and constraints
  - identify **risks**
  - evaluate options to resolve risks
  - develop and verify deliverables



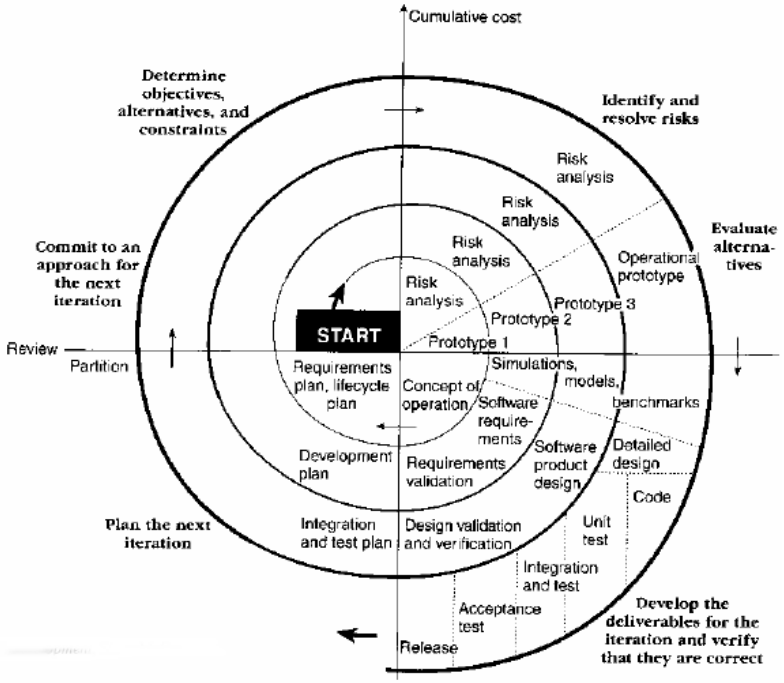
- benefits?
  - provides early indication of unforeseen problems
  - always addresses the biggest risk first
  - accommodates changes, growth
  - eliminates errors and unattractive choices early



Drawbacks of spiral

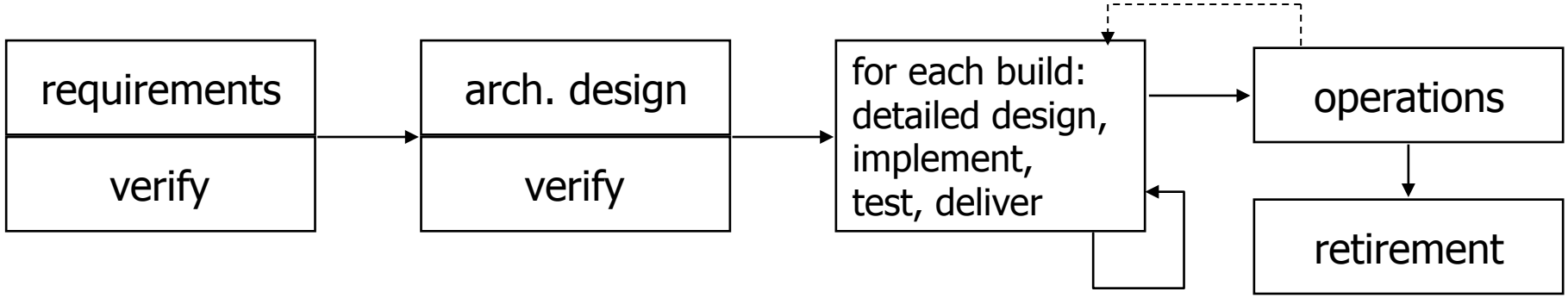
Barry Boehm, USC

- steps taken at each loop:
  - determine objectives and constraints
  - identify **risks**
  - evaluate options to resolve risks
  - develop and verify deliverables



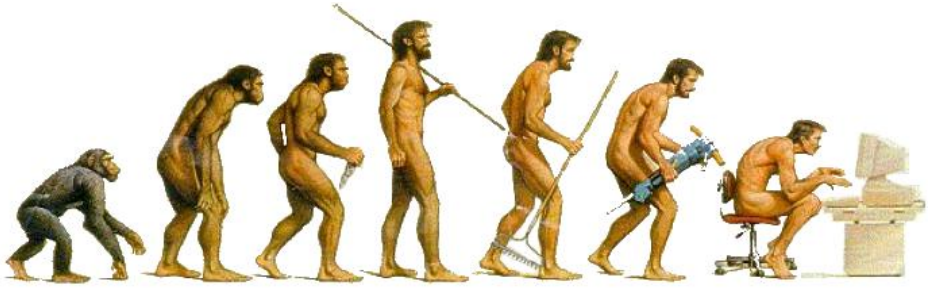
- drawbacks?
  - relies on developers to have risk-assessment expertise
  - perhaps over-focuses on risk and "putting out fires"; other features may go ignored because they are not "risky" enough
  - complex; how do you actually follow this?
  - works poorly when bound to an inflexible contract

Evolutionary prototyping

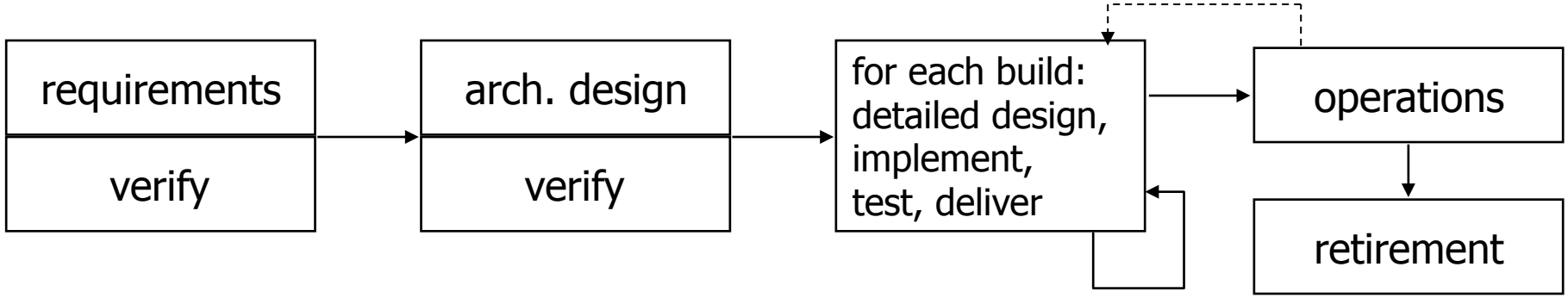


- build initial requirements, design/code it, "evolve" as needed

- benefits?
  - produces steady signs of progress, builds customer confidence
  - useful when requirements are not well known or change rapidly
  - customer involvement ("What do you think of this version?")



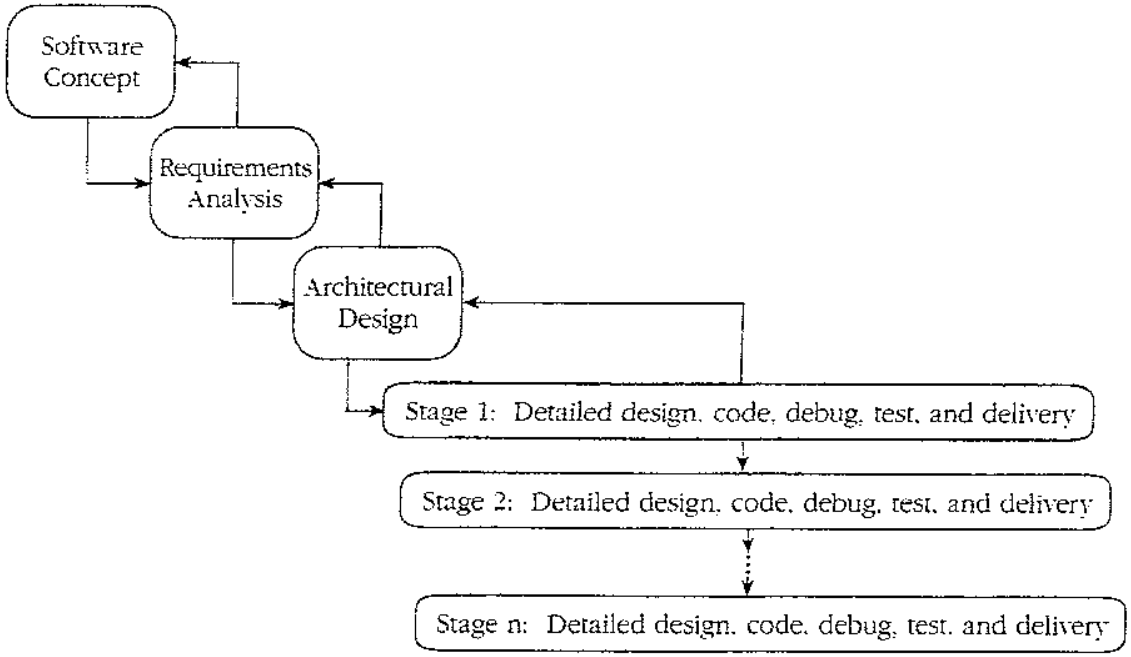
Drawbacks of evol. proto.



- drawbacks?
  - requires close customer involvement
  - assumes user's initial spec will be flexible
  - unclear how much iteration/time will be needed to finish
    - hard to estimate schedule or feature set
  - fails for separate pieces that must then be integrated
  - temporary fixes become permanent constraints
  - bridging; new software trying to gradually replace old

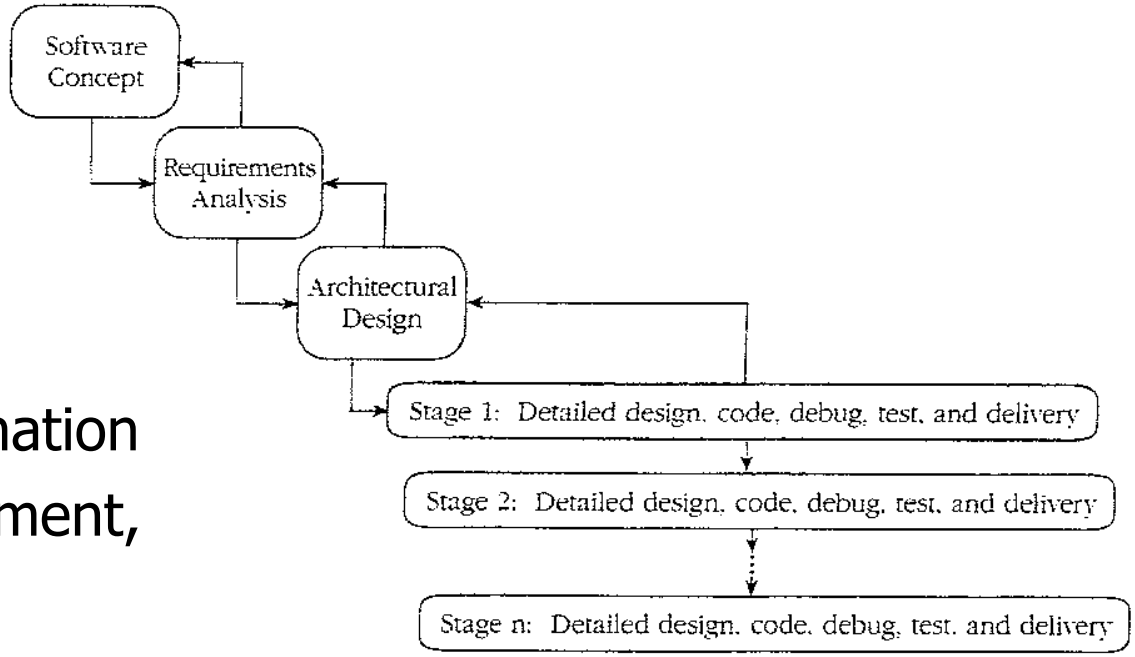
Staged delivery

- Waterfall-like beginning
- Short release cycles



- benefits?
  - *can ship at end of any release cycle* during implementation
  - from the outside (to customers) it looks like a successful delivery even if it is not the final goal the team aimed for
  - intermediate deliveries show progress and lead to feedback
  - problems visible earlier due to earlier integration

Drawbacks of staged delivery

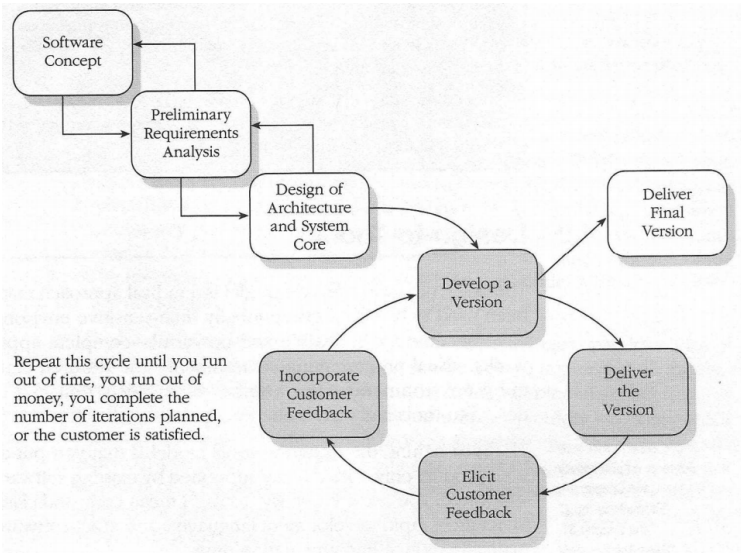


- drawbacks?
  - Requires tight coordination with docs, management, marketing
  - Product must be decomposable

- How does staged delivery differ from evolutionary prototyping?
  - In staged delivery, requirements are better known ahead of time rather than discovered by customer feedback on each release.

Evolutionary delivery

- **evolutionary delivery**
  - a hybrid between evolutionary prototyping and staged delivery



- difference from evo. prototyping
  - focuses on low-level systems first
  - evo. prototyping focuses on visible aspects (front-end)

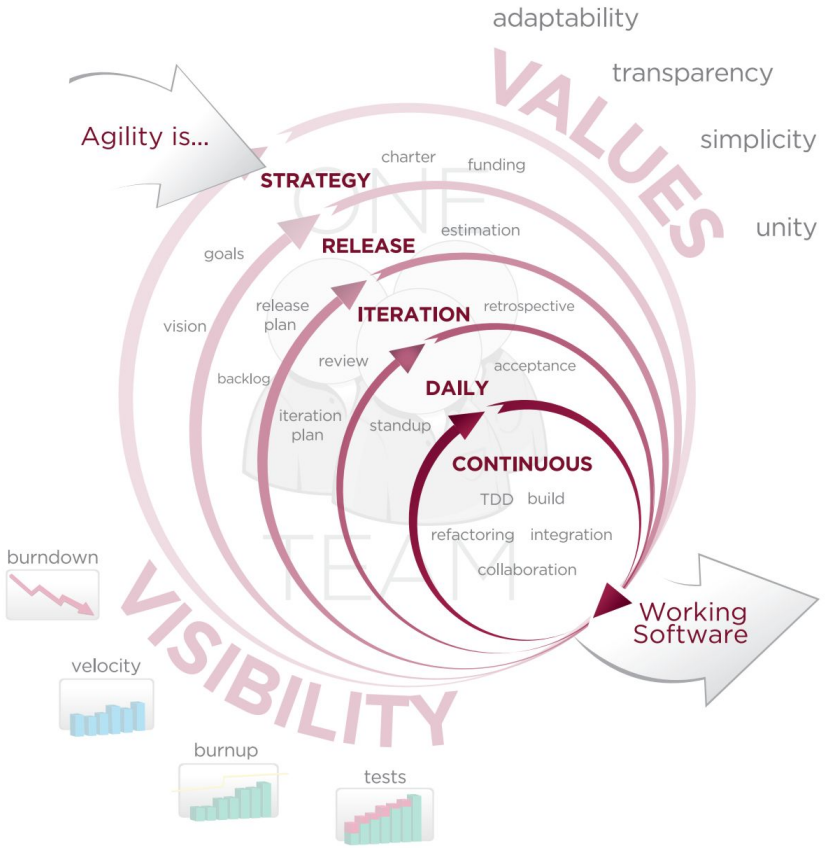
Design-to-*

- **design-to-schedule**
  - useful when you absolutely need to ship by a certain date
  - similar to the staged delivery model
    - but less flexible because of the fixed shipping date
  - requires careful prioritization of features and risks to address
  - not recommended
- **design-to-tools**
  - a model where the project only incorporates features that are easy to implement by using or combining existing components
  - reduces development time at cost of losing control of project
  - not recommended
- **off-the-shelf software**: don't build it, just purchase it (...)

Agile development

- **agile software development**: An adaptive, iterative process where teams self-organize and build features dynamically.
  - Extreme Programming
  - Scrum

- values:
  - **Individuals and interaction**
    over processes and tools
  - **Working software**
    over documentation
  - **Customer collaboration**
    over contract negotiation
  - **Responding to change**
    over following a plan



Agile Manifesto

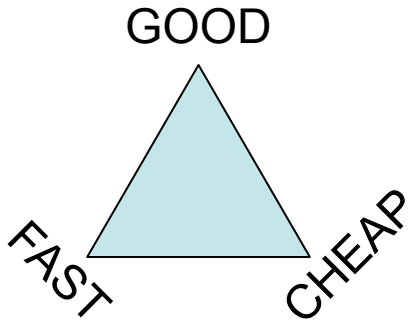
- The 12-point Agile Manifesto:
  - customer satisfaction by **rapid delivery** of useful software
  - welcome **changing requirements**, even late in development
  - working software is delivered **frequently** (weeks rather than months)
  - **working software** is the principal measure of progress
  - sustainable development, able to maintain a **constant pace**
  - close, **daily co-operation** between business people and developers
  - **face-to-face** conversation is the best form of communication
  - projects are built around **motivated individuals**, who are trusted
  - continuous attention to **technical excellence** and good design
  - **simplicity**
  - **self-organizing** teams
  - regular **adaptation** to changing circumstance
- Works well when used with **small teams of experts** who can handle a bit of uncertainty, chaos, change

What's the "best" model?

- Consider
  - The task at hand
  - Risk management
  - Quality / cost control
  - Predictability
  - Visibility of progress
  - Customer involvement and feedback

- Often a combination of models is used in practice

- Aim for good, fast, and cheap
  - But you can't have all three at the same time



Model category matrix

- Rated 1 (low) – 5 (high) in each of the categories shown:

	Risk mgmt.	Quality/ cost ctrl.	Predict-ability	Visibility of progress	Customer involvement
code-and-fix	1	1	1	3	1
waterfall	3	4	4	1	2
spiral	5	5	3	3	3
evolutionary prototyping	3	3	2	5	5
staged delivery	4	5	4	4	4
design-to-schedule	4	3	5	3	2

Model pros/cons

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