

CSE 403: Software Engineering, Winter 2016

courses.cs.washington.edu/courses/cse403/16wi/

Software Lifecycle

Emina Torlak

emina@cs.washington.edu

Outline

- Essential tasks of development
- What is a software development lifecycle?
- Why do we need a lifecycle process?
- Five basic lifecycle models and their tradeoffs
- Evaluating models
- Summary

Essential tasks of development

Requirements

Design

Implementation

Testing

Maintenance

Essential tasks of development

Requirements

Design

Implementation

Testing

Maintenance

Each phase requires different tools, knowledge, skill-set. A lot of ways to split responsibilities!

Essential tasks of development

Requirements

Design

Implementation

Testing

Maintenance

Each phase requires different tools, knowledge, skill-set. A lot of ways to split responsibilities!

How are these related?
What is a good order?

The software lifecycle

Requirements

Design

Implementation

Testing

Maintenance

Software lifecycle is a series of phases through which software is produced:

- from conception to end-of-life
- can take months or years to complete

The software lifecycle

Requirements

Design

Implementation

Testing

Maintenance

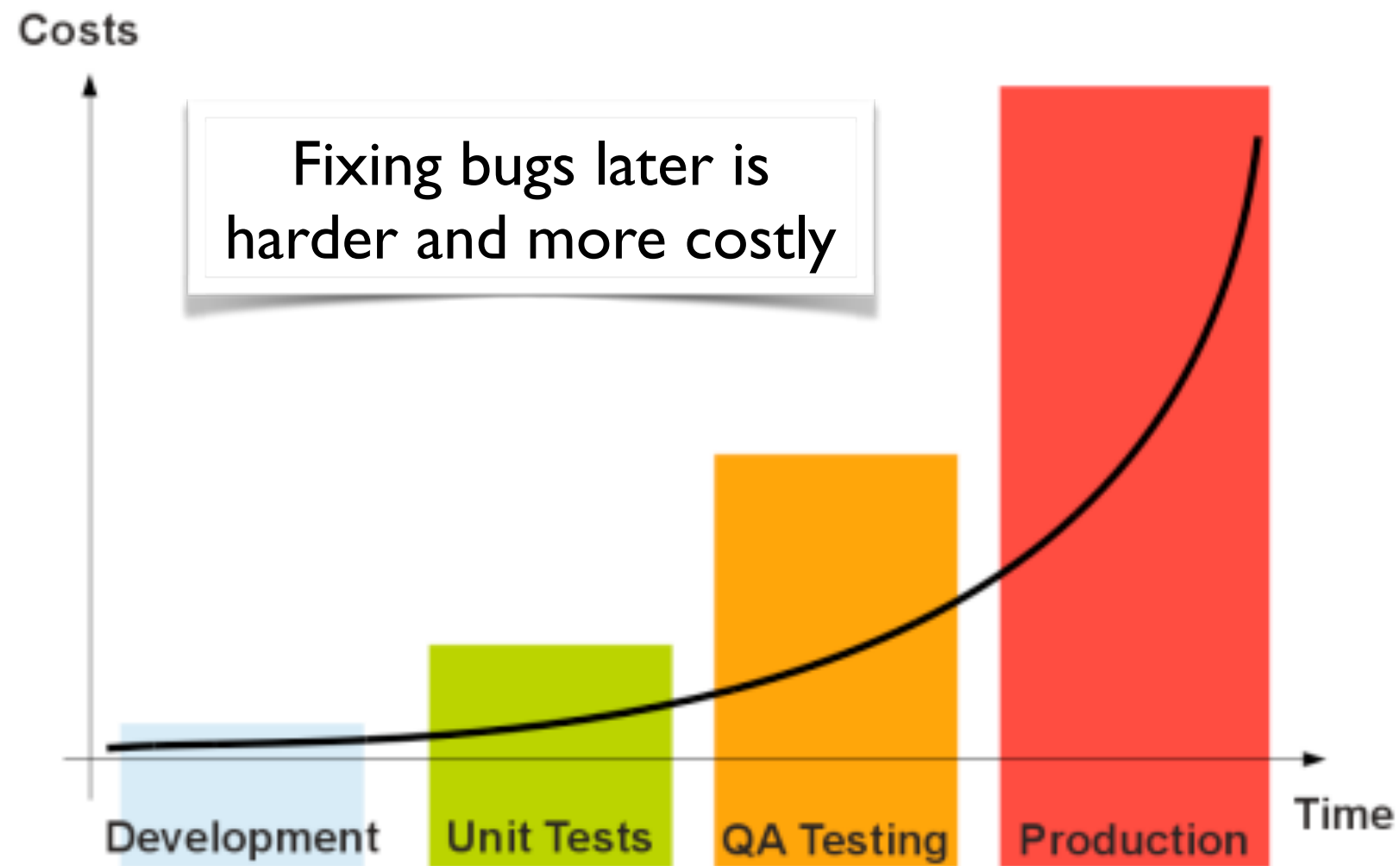
Software lifecycle is a series of phases through which software is produced:

- from conception to end-of-life
- can take months or years to complete

Goals of each phase:

- mark out a clear set of steps to perform
- produce a tangible item
- allow for review of work
- specify actions to perform in the next phase

Why do we need models and process?



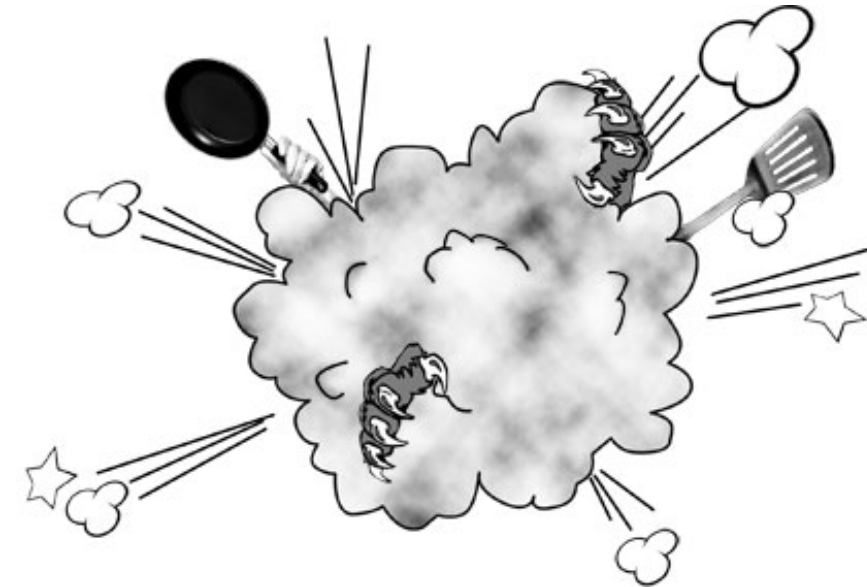
Why do we need process?



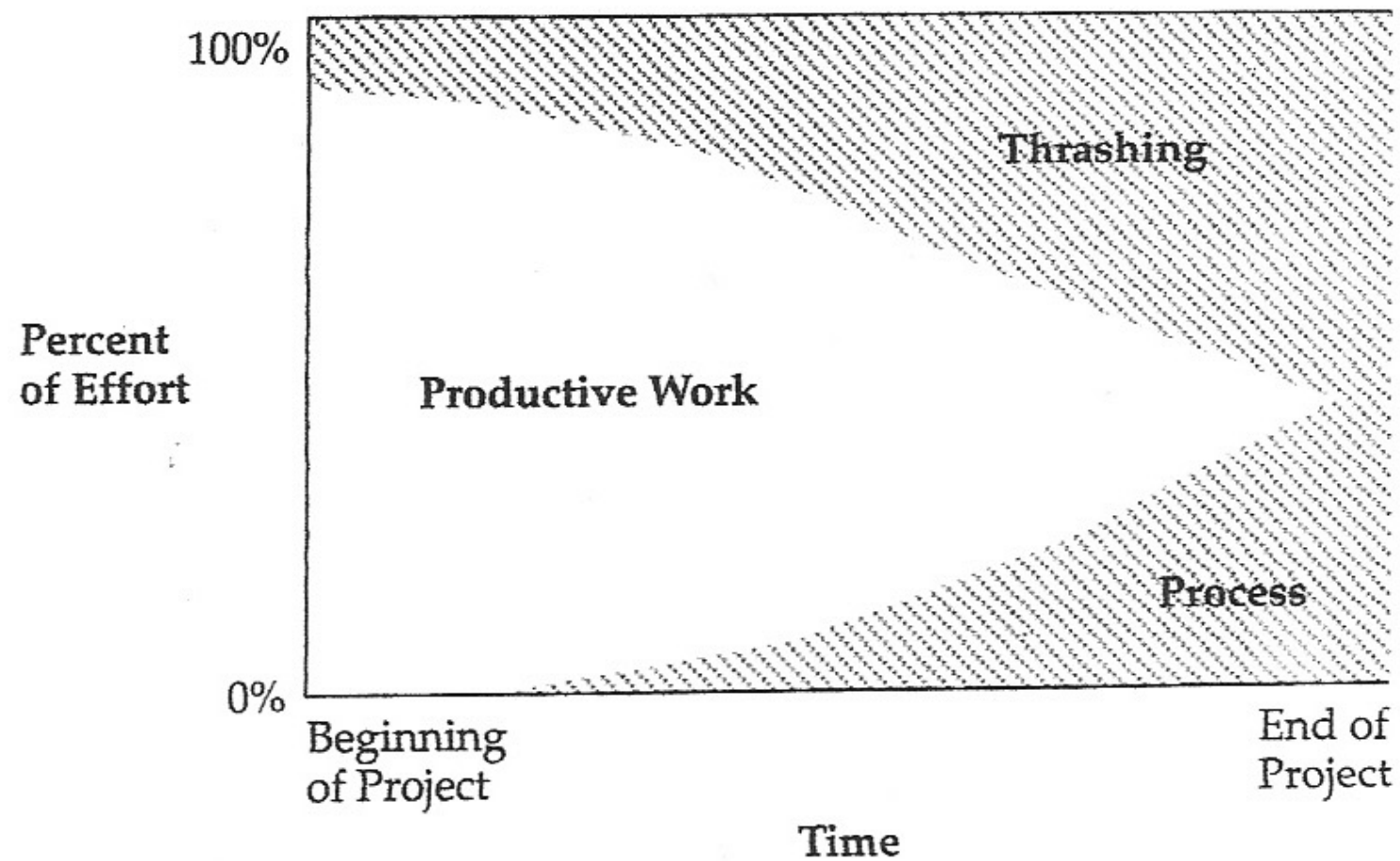
A bug in the requirements.

Life without software process

- Advantages:
 - nothing to learn or plan!
 - work on whatever is interesting, ignore the rest.
- Disadvantages:
 - may ignore some important tasks (testing, design)
 - not clear when to start or stop doing each task
 - scales poorly to multiple people
 - hard to review or evaluate one's work
 - code may not match user's needs (no requirements!)
 - code was not planned for modification, not flexible



Project with little attention to process

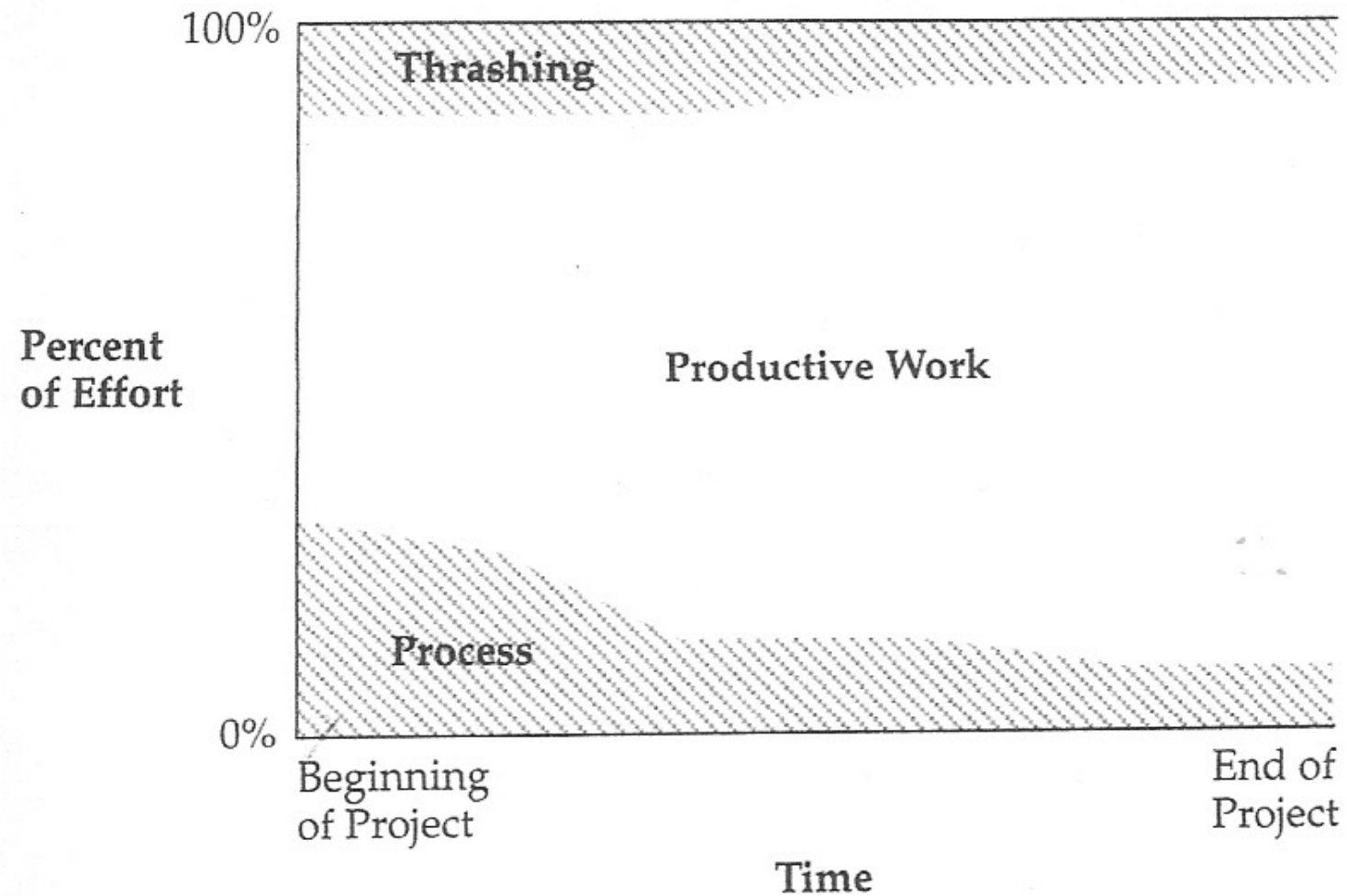


Survival Guide, McConnell, p. 24

Life with software process

- Advantages:
 - Provides structure in which to work
 - Forces you to think about the big picture and to follow steps to reach it
 - Without it, you may make decisions that are locally optimal but globally misdirected
 - It is a management tool
- Disadvantages:
 - Can lead to compromises and artificial constraints
 - Risk of over-emphasizing process rather than the product itself!

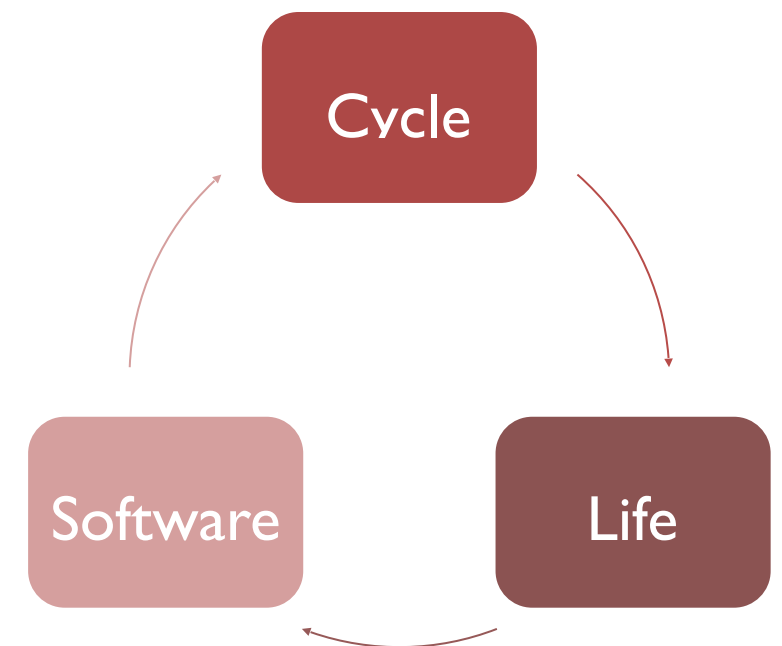
Project with early attention to process



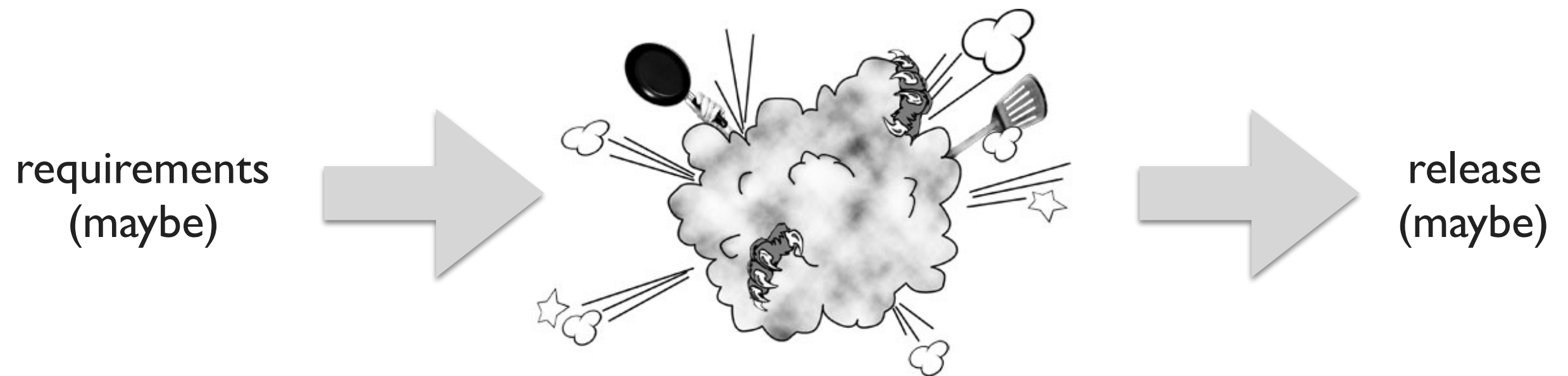
Survival Guide, McConnell, p. 25

Some lifecycle models

- **Code-and-fix:** write code, fix it when it breaks
- **Waterfall:** perform each phase in order
- **Spiral:** triage/figure out riskiest things first
- **Staged delivery:** build initial requirement specs or several releases, then design-and-code each in sequence
- **Evolutionary prototyping:** do the next easiest thing that could possibly lead to feedback



Code-and-fix model



Code-and-fix model

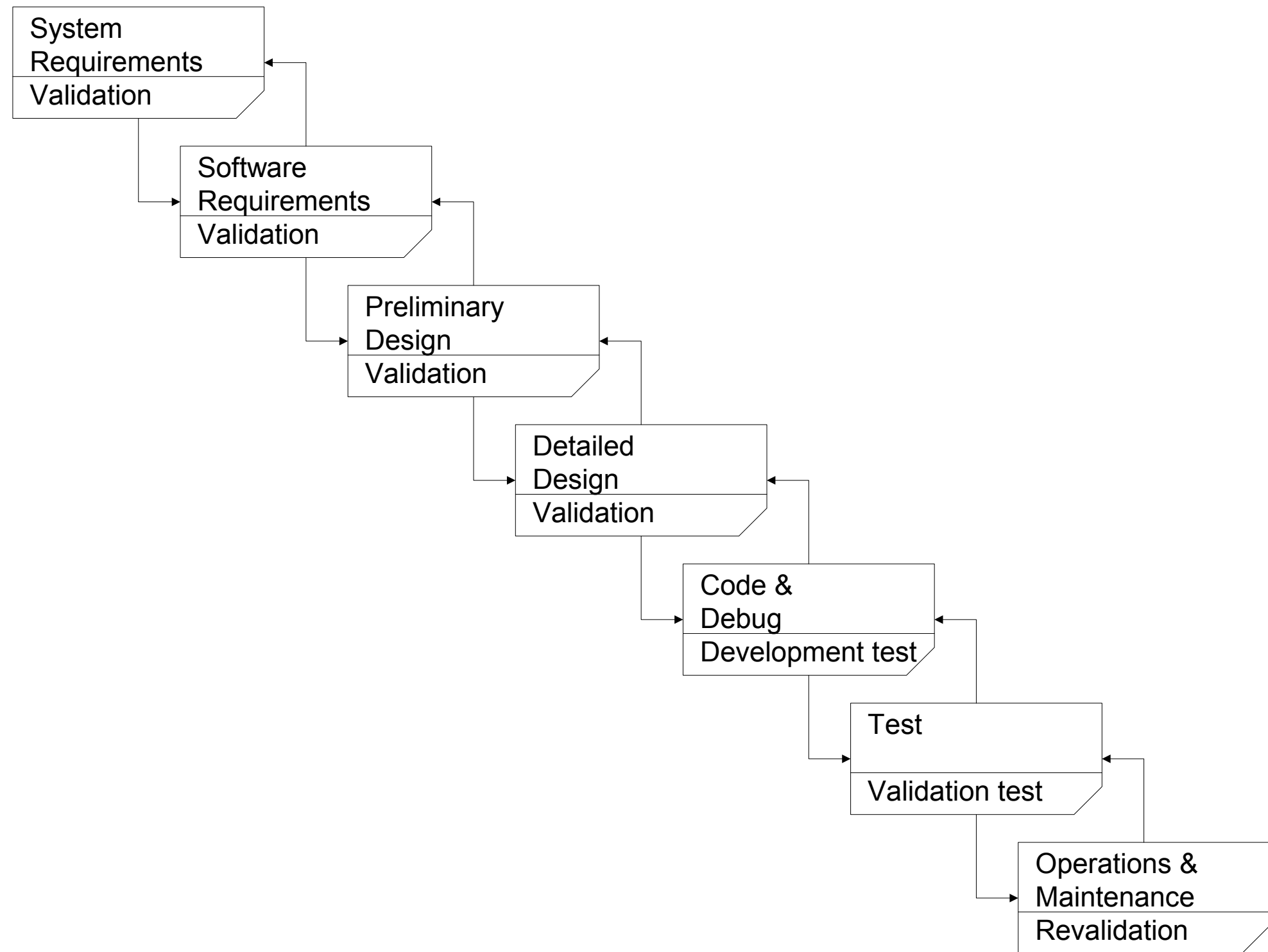
- Advantages:
 - Little to no overhead, see progress quickly
 - Applicable for very small projects and short-lived prototypes
- Disadvantages:
 - No way to assess progress, quality, or risks
 - Unlikely to accommodate changes without a major design overhaul
 - Unclear delivery features (scope), timing, and support

Code-and-fix model

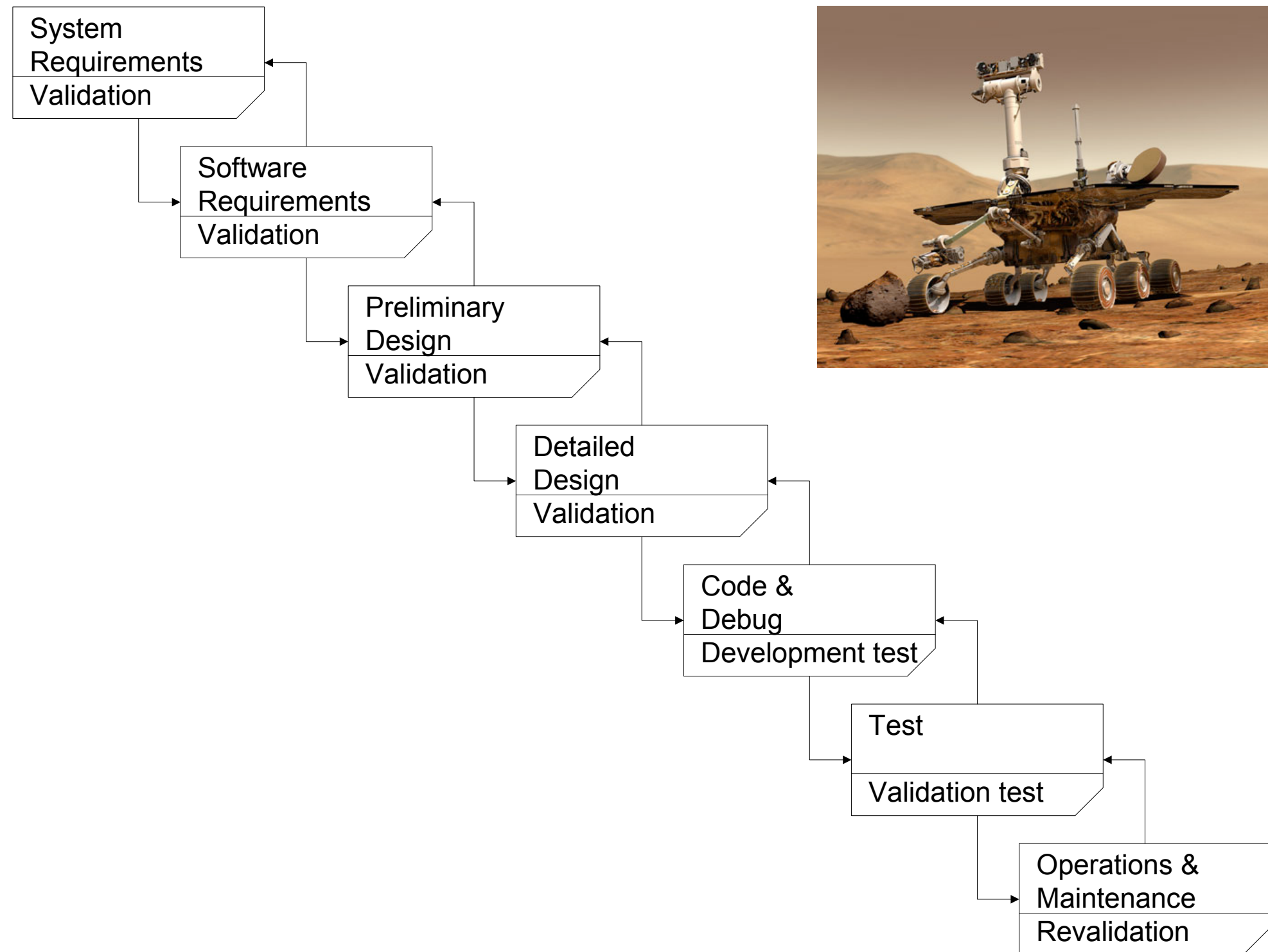
- Advantages:
 - Little to no overhead, see progress quickly
 - Applicable for very small projects and short-lived prototypes
- Disadvantages:
 - No way to assess progress, quality, or risks
 - Unlikely to accommodate changes without a major design overhaul
 - Unclear delivery features (scope), timing, and support



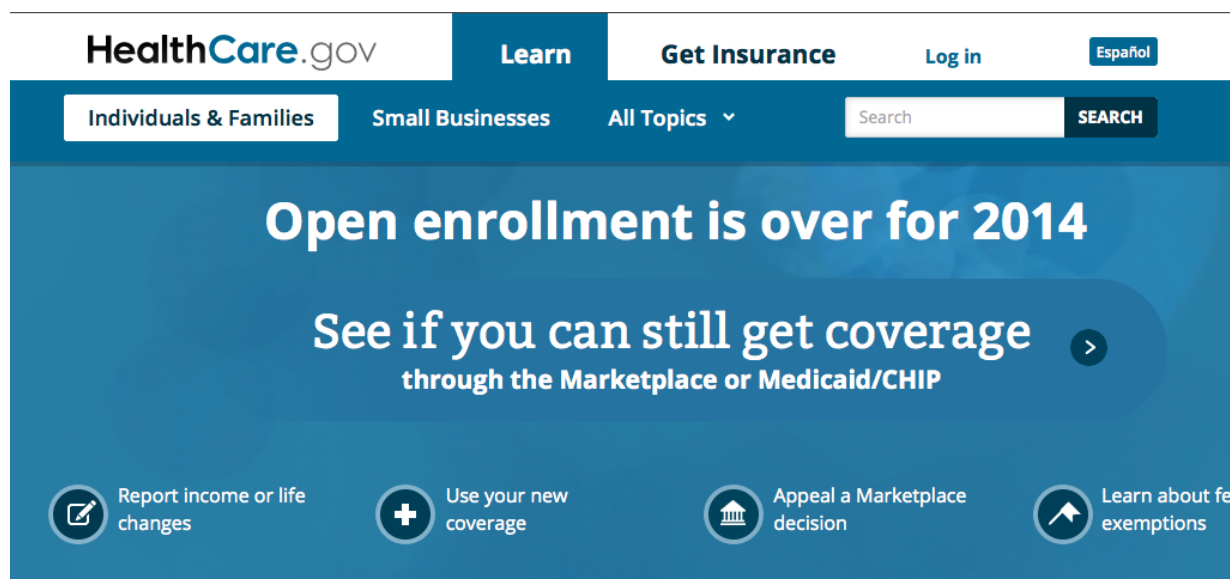
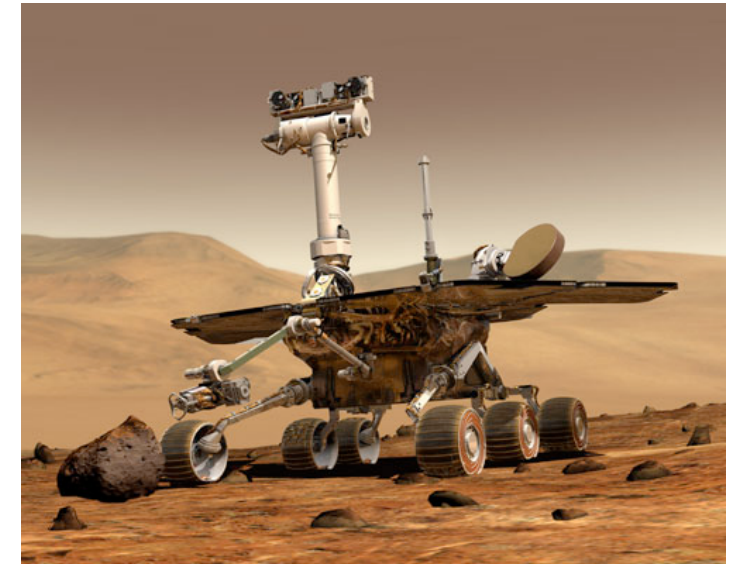
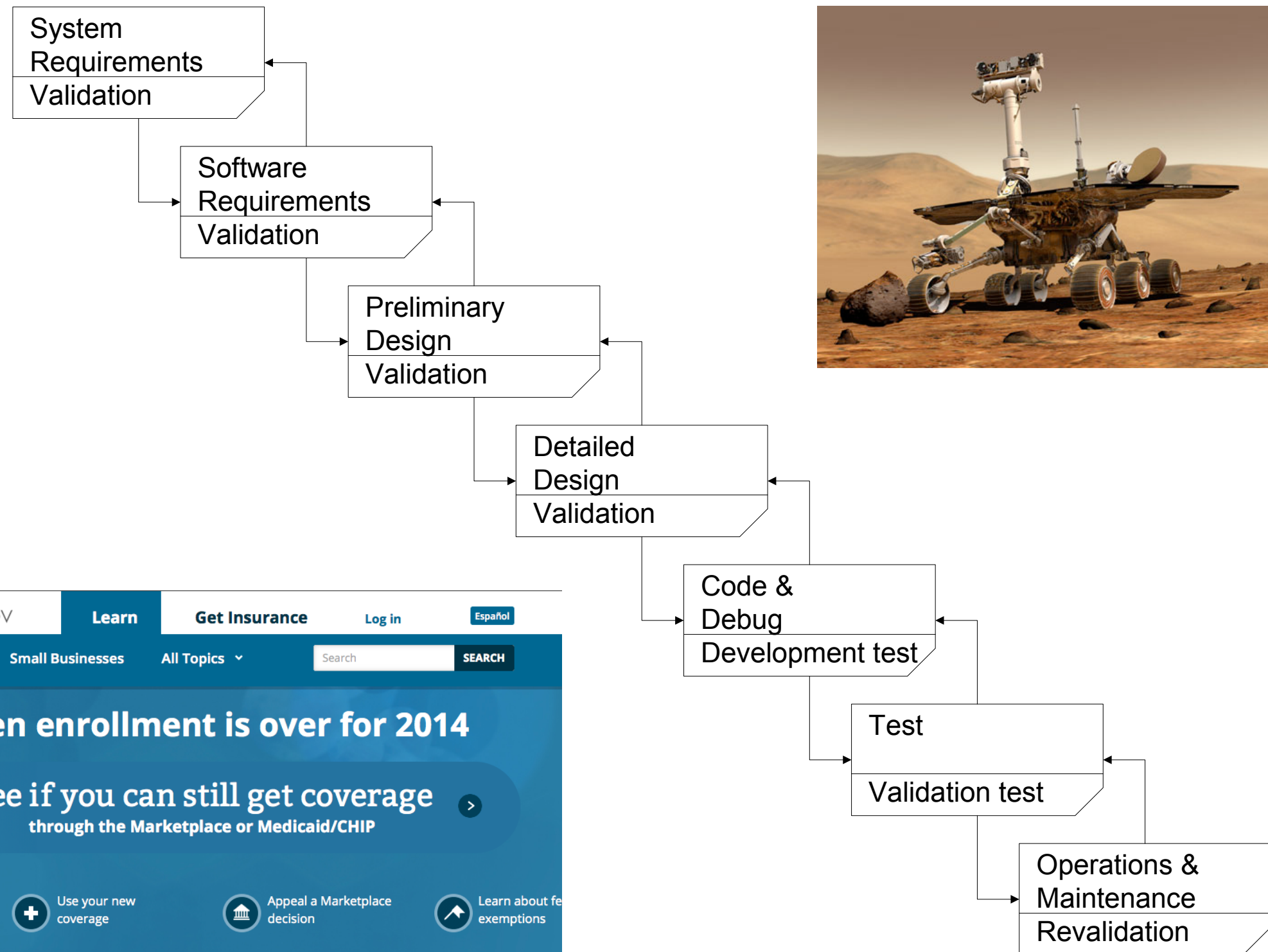
Waterfall model



Waterfall model



Waterfall model



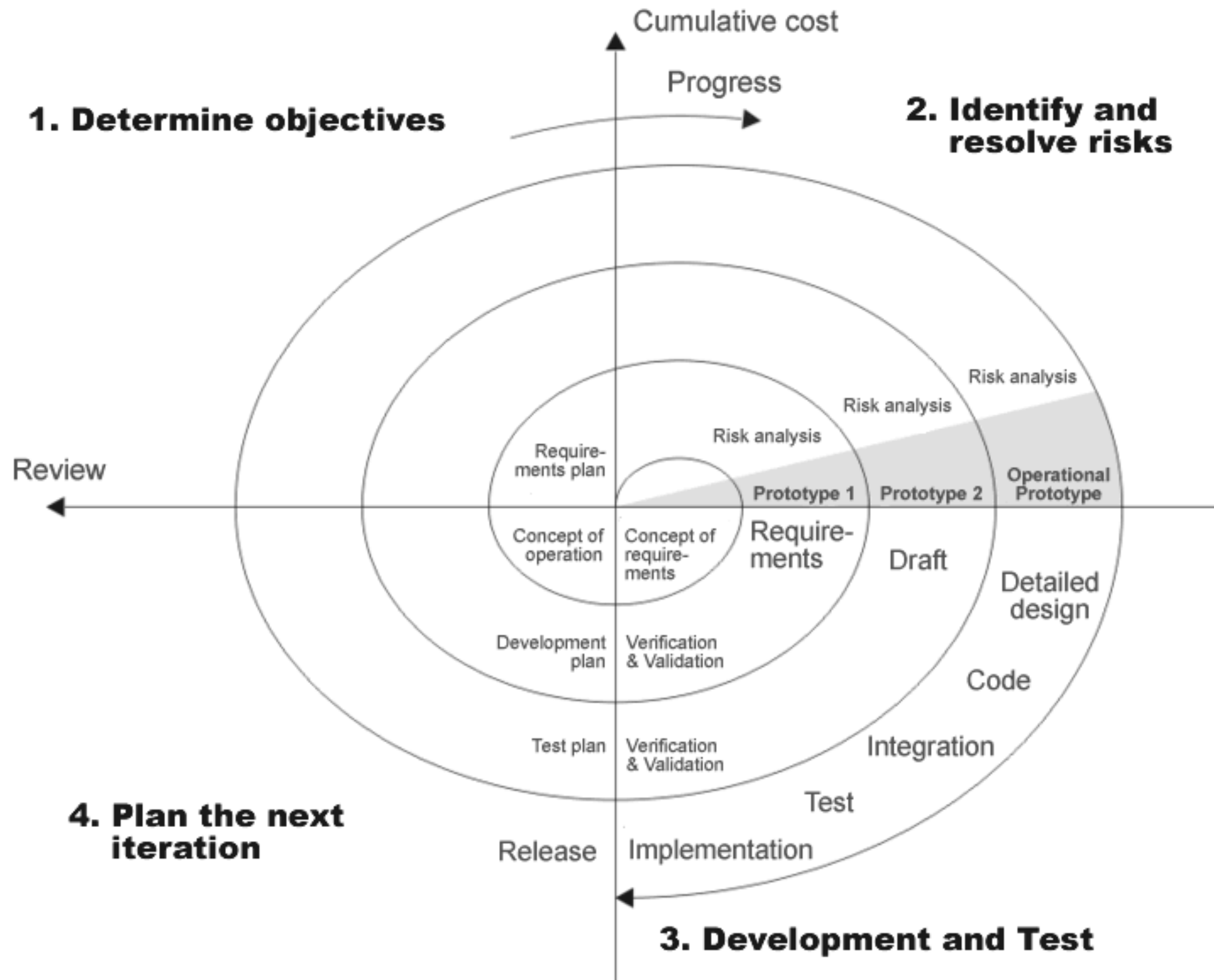
Waterfall model advantages

- Suitable for projects that are very well understood but complex
 - Tackles all planning upfront
 - The ideal of no midstream changes equates to an efficient software development process
- Supports inexperienced teams
 - Orderly, easy-to-follow sequential model
 - Reviews at each stage determine if the product is ready to advance

Waterfall model disadvantages

- Requires a lot of planning up front (not always easy)
 - assumes requirements will be clear and well-understood
- Rigid, linear; not adaptable to change in the product
 - costly to "swim upstream" back to a previous phase
- No sense of progress until the very end
 - no code to show until almost done
- Integration occurs at the very end
 - defies “integrate early and often” rule
 - solutions are inflexible, no feedback until end
- Delivered product may not match customer needs
 - phase reviews are massive affairs
 - inertia means change is costly

Spiral model (risk oriented)



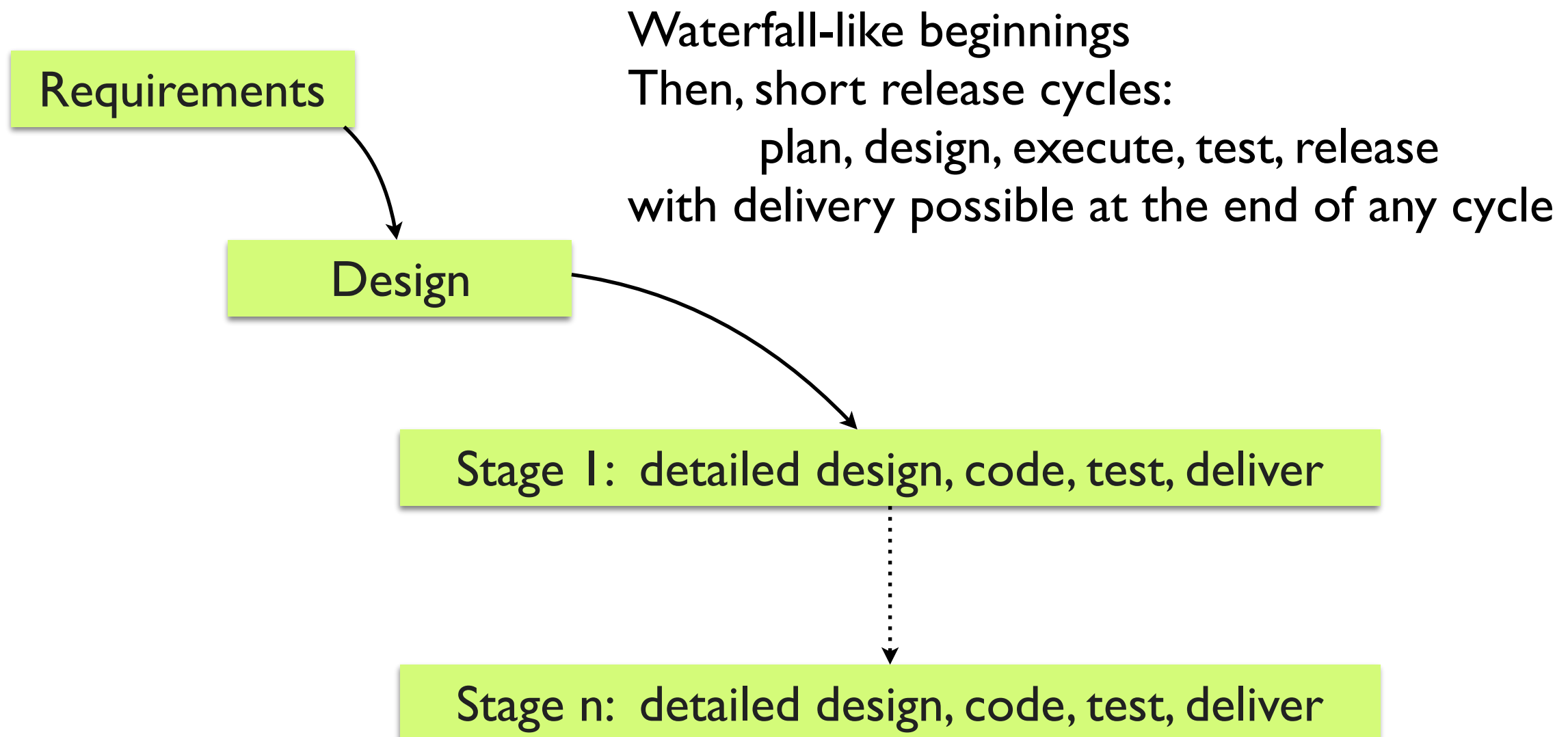
Spiral model advantages

- Especially appropriate at the beginning of the project, when the requirements are still fluid
- Provides early indication of unforeseen problems
- Accommodates change
- As costs increase, risks decrease!
- Always addresses the biggest risk first

Spiral model disadvantages

- A lot of planning and management
- Frequent changes of task
 - But, get to stick with one product feature/goal
- Requires customer and contract flexibility
- Developers must be able to assess risk
 - Must address most important issues

Staged delivery model



Staged delivery model advantages

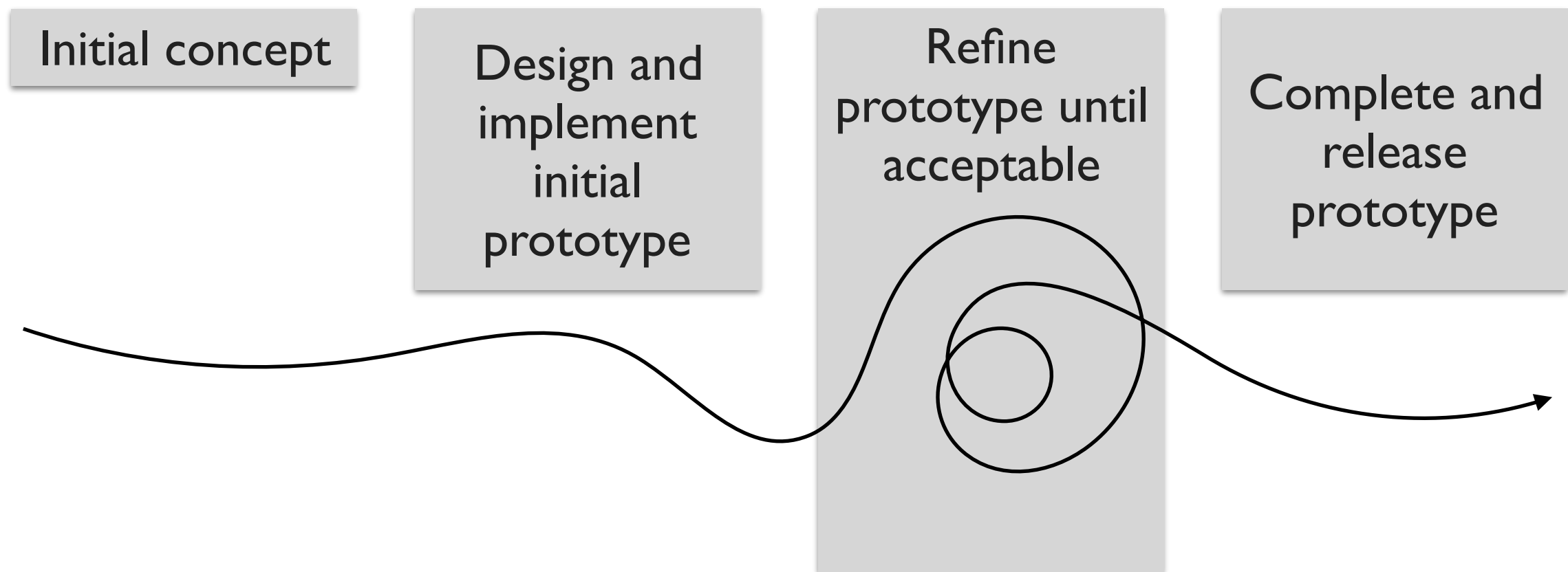
- Can ship at the end of any release cycle
 - Looks like success to customers, even if not original goal
- Intermediate deliveries show progress, satisfy customers, and lead to feedback
- Problems are visible early (e.g., integration)
- Facilitates shorter, more predictable release cycles

Very practical, widely used and successful

Staged delivery model disadvantages

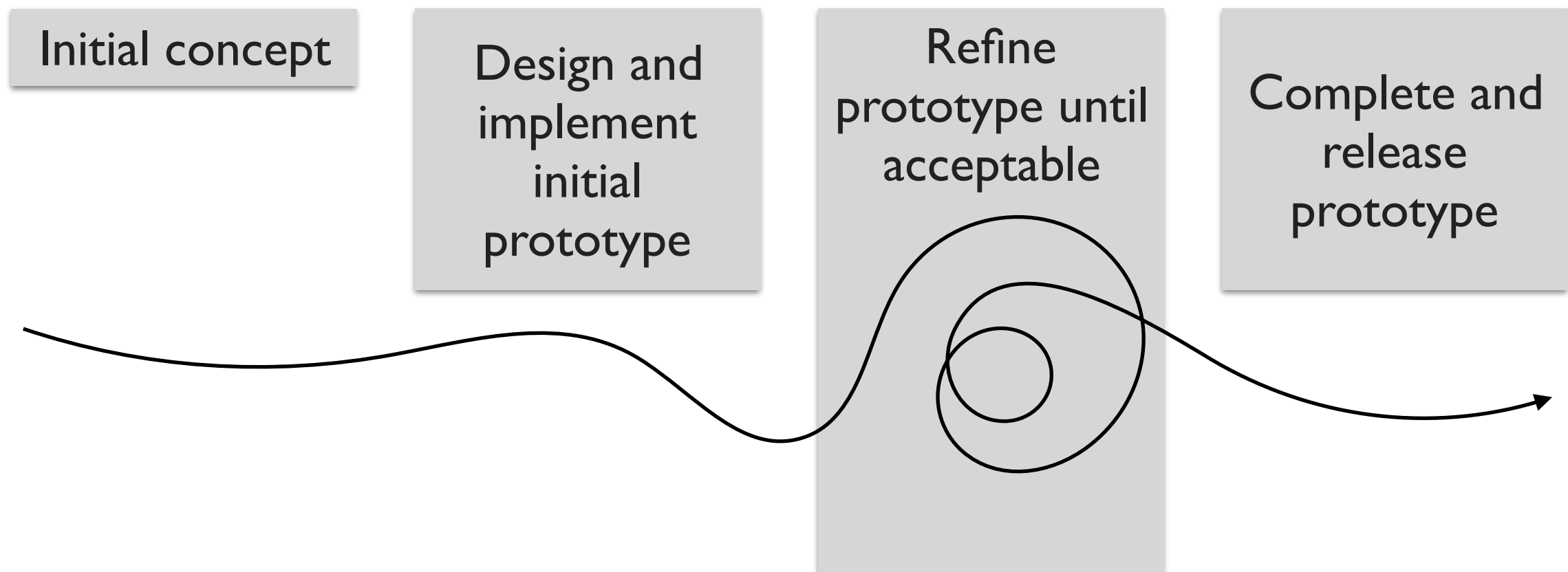
- Requires tight coordination with documentation, management, marketing
- Product must be decomposable
- Extra releases cause overhead

Evolutionary prototyping



Develop a skeleton system
and evolve it for delivery

Evolutionary prototyping



Develop a skeleton system and evolve it for delivery

Different from staged delivery in that requirements are not known ahead of time. Discovered by feedback.

Evolutionary prototyping advantages

- Addresses risks early
- Steady signs of progress build customer confidence
- Useful when requirements are unknown or changing
- Participatory design / useful feedback loops

Very practical, widely used and successful

Evolutionary prototyping disadvantages

- Requires close customer involvement
- Assumes user's initial spec is flexible
- Problems with planning
 - especially if the developers are inexperienced
 - feature creep, major design decisions, use of time, etc.
 - hard to estimate completion schedule or feature set
 - unclear how many iterations will be needed to finish
- Integration problems
 - fails for separate pieces that must then be integrated
 - bridging; new software trying to gradually replace old
- Temporary fixes become permanent constraints
- Requires low friction deployment and experimentation

More models: embracing or fighting timelines

- Fit-to-schedule
 - “We will ship on a certain date and cut until it fits”
 - similar to the staged delivery model
 - but less flexible because of the fixed shipping date
 - requires careful prioritization of features and risks
- Fit-to-features/quality
 - “We’ll ship the product when it is ready”
 - Trade predictable schedules for quality control

Why are there so many models?

- The choice of a model depends on the project circumstances and requirements.
- A good choice of a model can result in a vastly more productive environment than a bad choice.
- A cocktail of models is frequently used in practice to get the best of all worlds. Models are often combined or tailored to environment



What's the best model?

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

What's the best model?

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

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

Model category matrix (on a scale from 1 to 5)

	Risk mgmt.	Quality/ cost ctrl.	Predict- ability	Visibility of progress	Customer involvement
Code-and-fix	1	1	1	3	2
Waterfall	2	4	3	1	2
Spiral	5	5	3	3	3
Evolutionary prototyping	3	3	2	5	5
Staged delivery	3	5	3	3	4
Fit-to-schedule	4	3	5	3	2

What's the best model for ...

- A system to control anti-lock braking in a car
- A hospital accounting system that replaces an existing system
- An interactive system that allows airline passengers to quickly find replacement flight times (for missed or bumped reservations) from terminals installed at airports
- A mobile app for finding romantic partners

Summary

- Software lifecycle models as management tools
 - System for organizing effort among workers
 - Forces planning and seeing consequences
 - Splits work into smaller, tractable units
 - Supports feedback and accurate timescales
 - Processes support production at scale
- Limitations of lifecycle models
 - Can lead to artificial design constraints
 - Risk of overemphasizing process over results
 - Models only approximate actual practices

