

# Software Development Process

October 6, 2022

Byung-Gon Chun

(Slide credits: George Candea, EPFL)

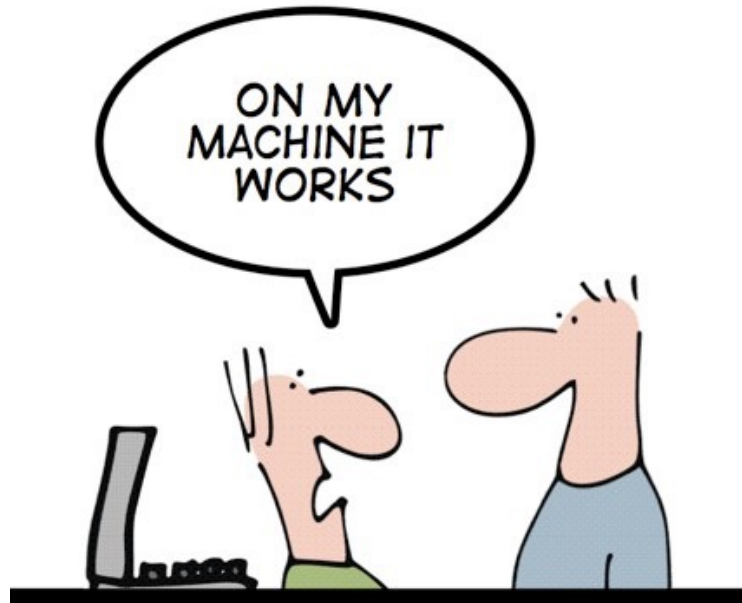
*WHEN YOU HEAR THIS:*



*YOU KNOW YOU'RE IN A  
SOFTWARE PROJECT*

*JUST IN CASE YOU'RE STILL NOT  
SURE WHETHER YOU'RE IN A  
SOFTWARE PROJECT*

*WAIT UNTIL YOU HEAR THIS:*



# Objectives

- Understand phases of software development
- Use processes to improve product
  - Classic processes
  - Modern processes
- Understand which work and which don't

# Development Phases

1. Market research
2. Requirements gathering and analysis
3. Specification, planning, design
4. Implementation and testing
5. Deployment
6. Support and maintenance

# Market Research

- Talk to users, “the field” and internal staff
  - *existing/potential/lost customers, software developers*
- Evaluate economic/social feasibility
  - *Worth doing? Good fit for distribution channels? What effect on existing product lines (cannibalism)?*
- Evaluate cost + time needs/assumptions

# Requirements Analysis

- Define product
  - *information, function, behavior, performance, interfaces*
  - *understand the goals and use cases*
- Customers do not always know what they want
  - If I'd asked my customers what they wanted,  
they'd have said a faster horse. (Henry Ford)
- Analysis: what to keep and what to toss away
  - *prune and document requirements specifications*

# Planning and Design

- Turn requirements specification into design specification
- Specify...
  - *class architecture, data structures, algorithmic details, etc.*
- Describe as **rigorously** as feasible
  - *e.g., safety-critical software use formal languages*



# Implementation & Testing

- Write the code
  - *typically takes ~15% of total cycle*
- Test and integrate
  - *typically takes ~50% of total cycle*
- Document internal designs
- Remember...
  - *the later a bug is discovered, the harder it is to fix*

# Deployment, Support, Maintenance

- Package code into a product or service
  - *may involve porting to many platforms*
  - *e.g., Oracle runs on many platforms*
- Install at the customers' sites
- Deployment in SaaS is easier!
- Detect and fix bugs



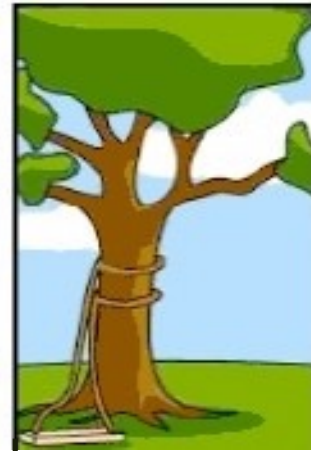
How the customer explained it



How the Project Leader understood it



How the Analyst designed it



How the Programmer wrote it



How the Business Consultant described it



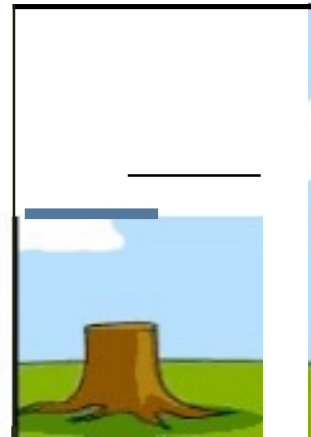
How the project was documented



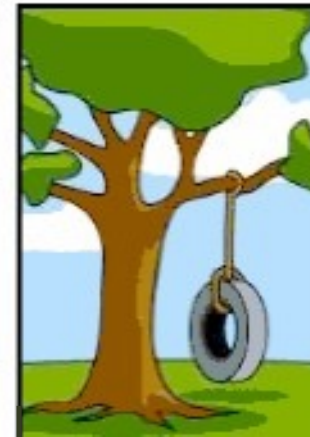
What operations installed



How the customer was billed

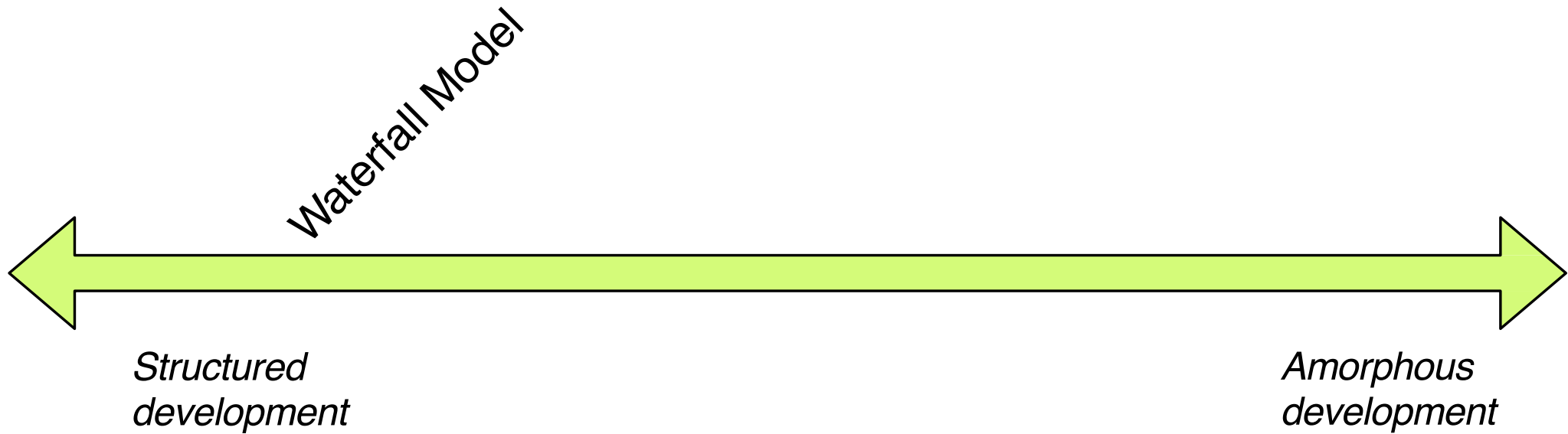


How it was supported

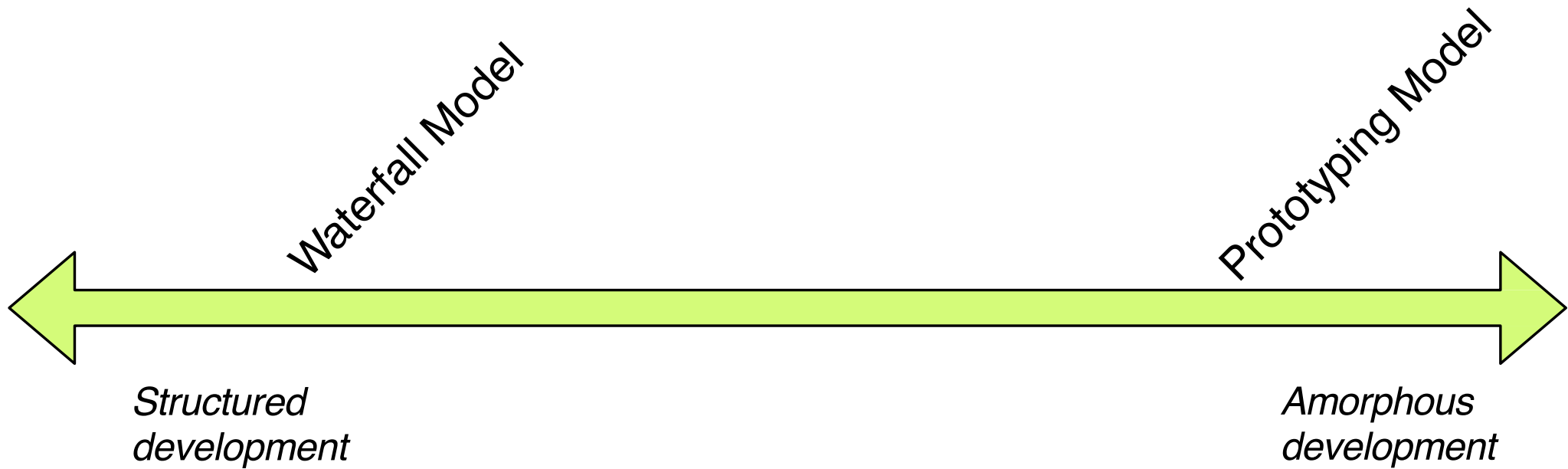


What the customer really needed

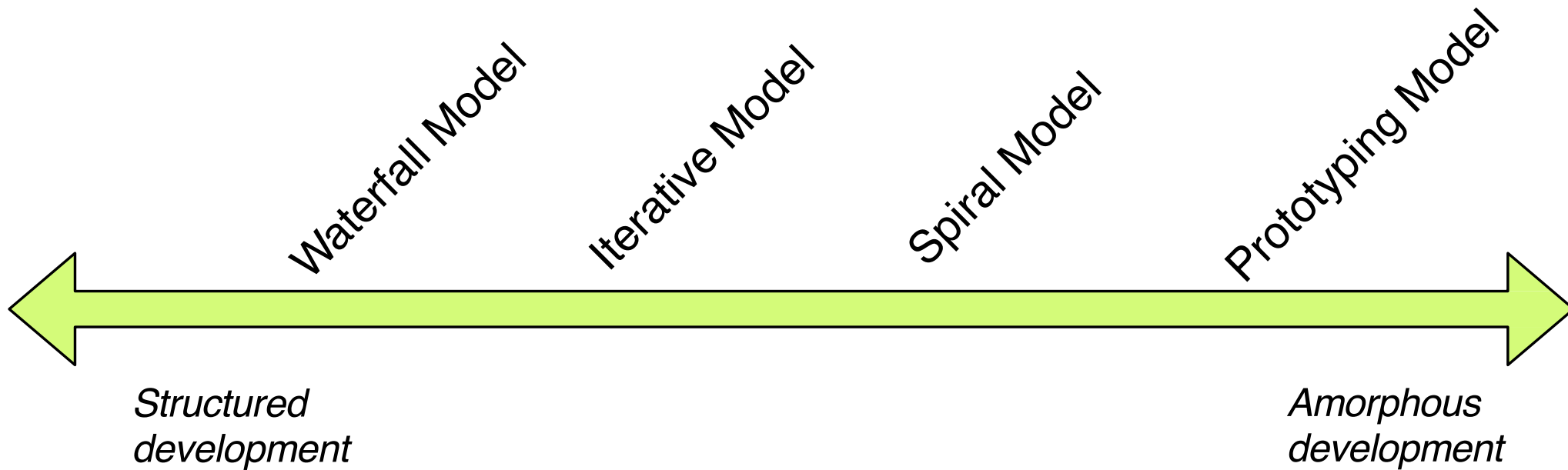
# Software Development Process



# Software Development Process

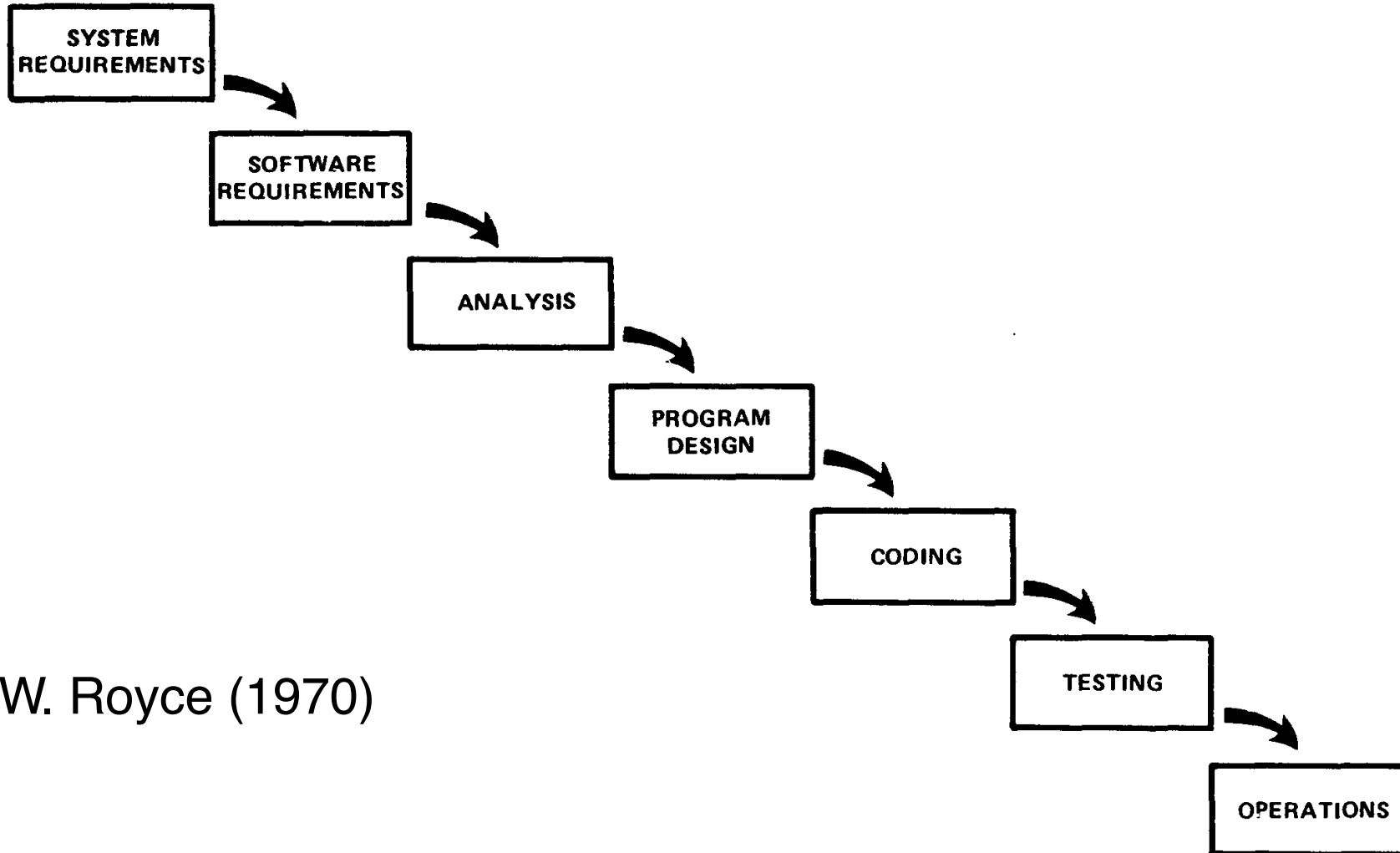


# Software Development Process



# **The Waterfall Model**

# The Waterfall Model

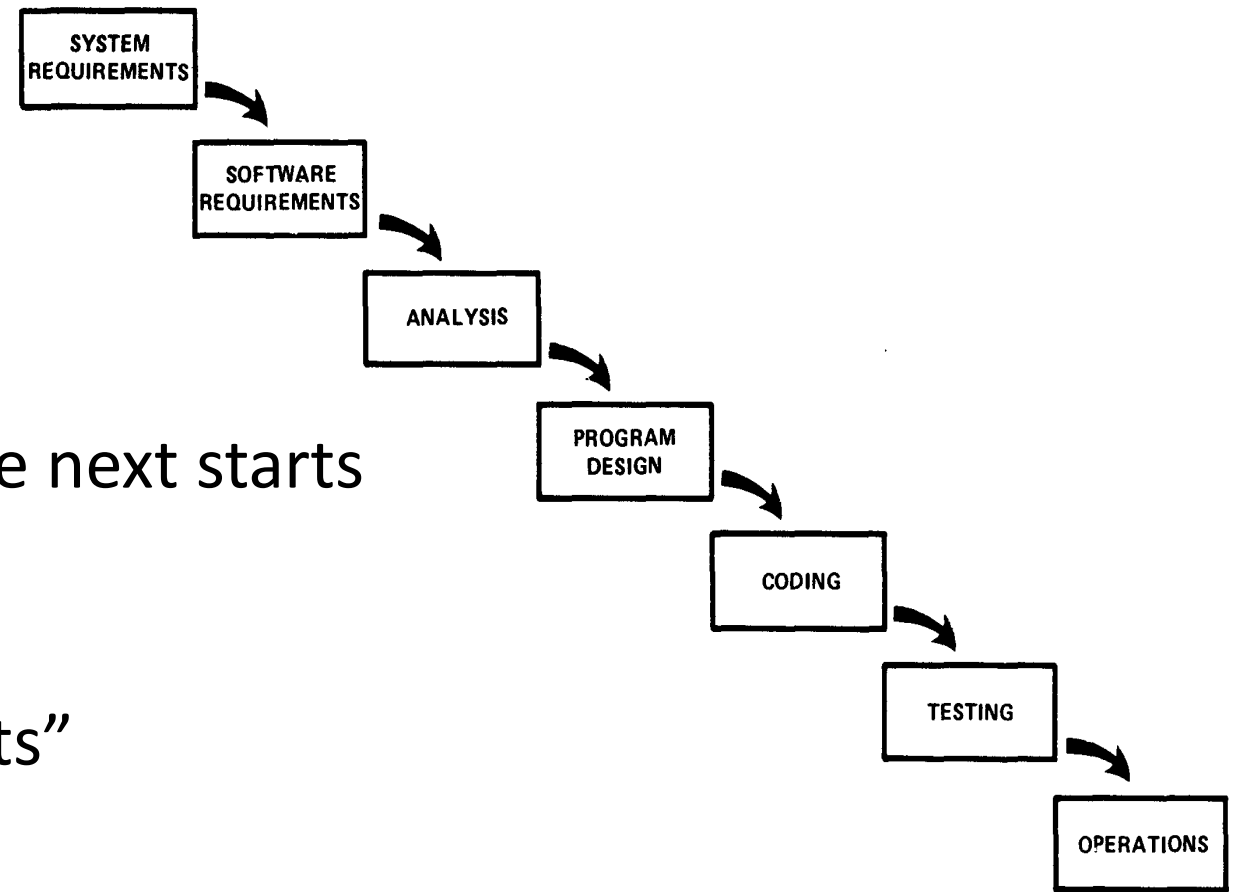


- Winston W. Royce (1970)



# Characteristics

- Key feature: linear, sequential
  - each stage completes before the next starts
  - documentation and review at each phase transition
  - specifications serve as “contracts”
  - “freeze dates”
- Model still used (in various forms)
  - *e.g., mandated at U.S. DoD and NASA until 1994*
  - *e.g., Oracle, IBM*



# Waterfall Strengths

- Early validation (can save 50x-200x in cost!)
  - *enforces stability of requirements*
- Structure + discipline
  - strong control over process
  - good for inexperienced or new staff
  - mitigates risk of departing staff
  - clear progress metrics, good resource usage

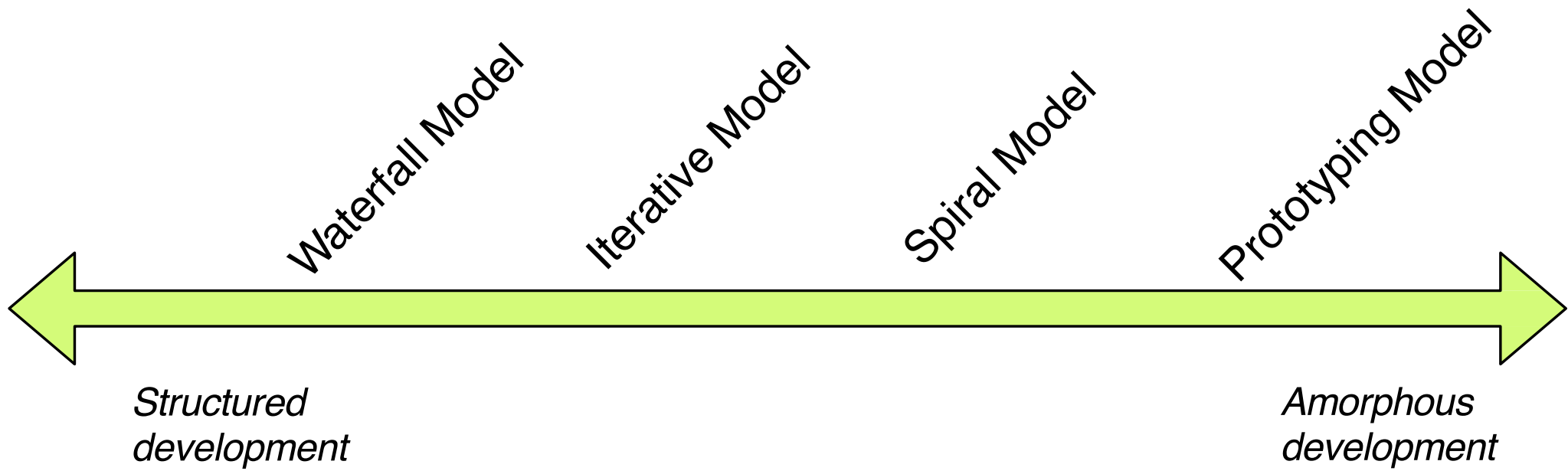
# Waterfall Weaknesses

- Requirements must be known upfront
  - *perfecting a phase before moving on is unrealistic*
  - *many problems can only be discovered by doing*
- Inflexible → slow, costly, cumbersome
  - *cost/benefit ratio (e.g., lots of documentation)*
- Customer does not get to preview the product
  - *product validation is delayed for a long time*
  - *promotes gap between users and developers*

# When to Use?

- Objectives + solution are clear
  - *product definition can be stabilized*
  - *mature technology, no risk of surprises*
  - *done before*
- Inexperienced project manager or team
- Large, complex projects (enterprise)
  - *project subject to formal approvals*

# **Prototyping Model**



# Prototyping Model

- Purely iterative (exact opposite of waterfall)
  - iterate until it's done
  - a.k.a. evolutionary model
- Typical sequence:
  - build a prototype
  - Show to end users (use as a way to refine requirements)
  - User evaluates and suggests improvements or decides it's almost ready to release
  - Developers refine prototype
  - Repeat loop

# Prototyping Strengths

- Copes with users' difficulty of expressing needs
- Encourages user participation → higher satisfaction
- Resolves unclear objectives, helps make tradeoffs
- Can exploit knowledge gained in earlier iterations
- Encourages innovation and flexible design
- Get a functional application quickly



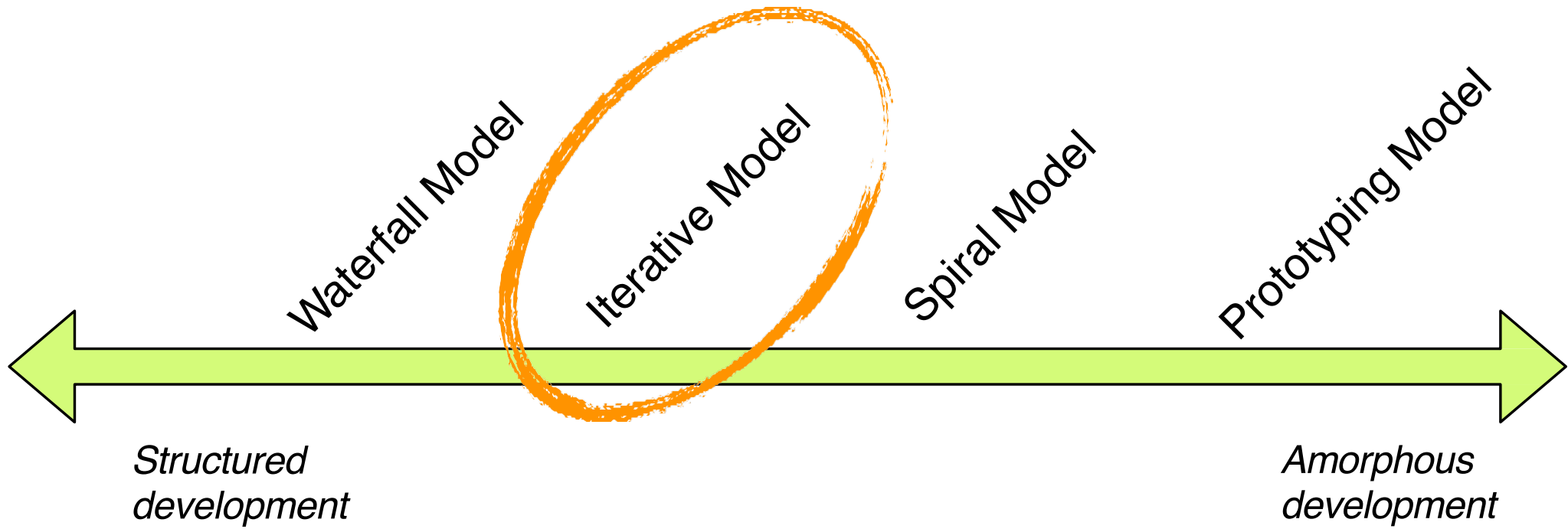
# Prototyping Weaknesses

- Difficult to manage
- Can lead to poor design - evolution not always good
  - *unsuitable for poorly skilled staff*
  - *prototypes are by definition “throwaway” models*
  - *prototypes can unwittingly become “final product”*
- Incomplete problem analysis
  - *only superficial needs are addressed*
  - *hasty early rejection, or may iterate forever*
- Lose perspective of reason behind design choices

# When to Use?

- Short-lived demos, UIs, etc.
  - *user-exposed aspects are more important than reliability*
- User requirements unclear, or likely to change
  - *use iteration to mitigate risk*
- User or customer not very knowledgeable
- High pressure for immediate implementation
- Experienced team
- Flexible designs "for the future" are not essential

# **Incremental Model**



# Incremental Model

- Waterfall with a divide-and-conquer strategy
  - Break project down into smaller parts
  - Combine linear model with iterative approach, to reduce project risk
- 3 Approaches
  - Sequence of mini-waterfalls; each release adds more functionality to the product
  - Break down into mini-waterfalls to be pursued in parallel (must design interfaces carefully)
  - Do a waterfall up to (and including) design, then do iterative prototyping

# Incremental Model: Strengths

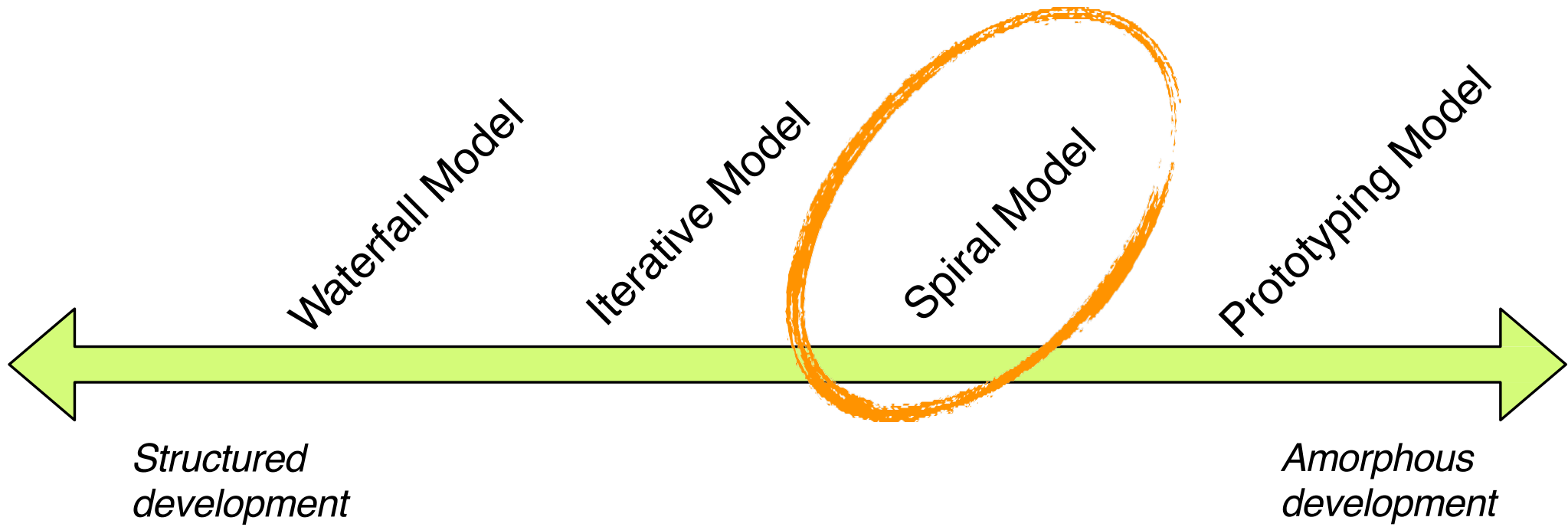
- Can exploit knowledge from prior increments
- Better control (documentation, review, etc.)
- Customer gets important functionality early
- Mitigates integration risks early (through increments)
- Can go into production sooner
- Can accommodate changing requirements

# Incremental Model: Weaknesses

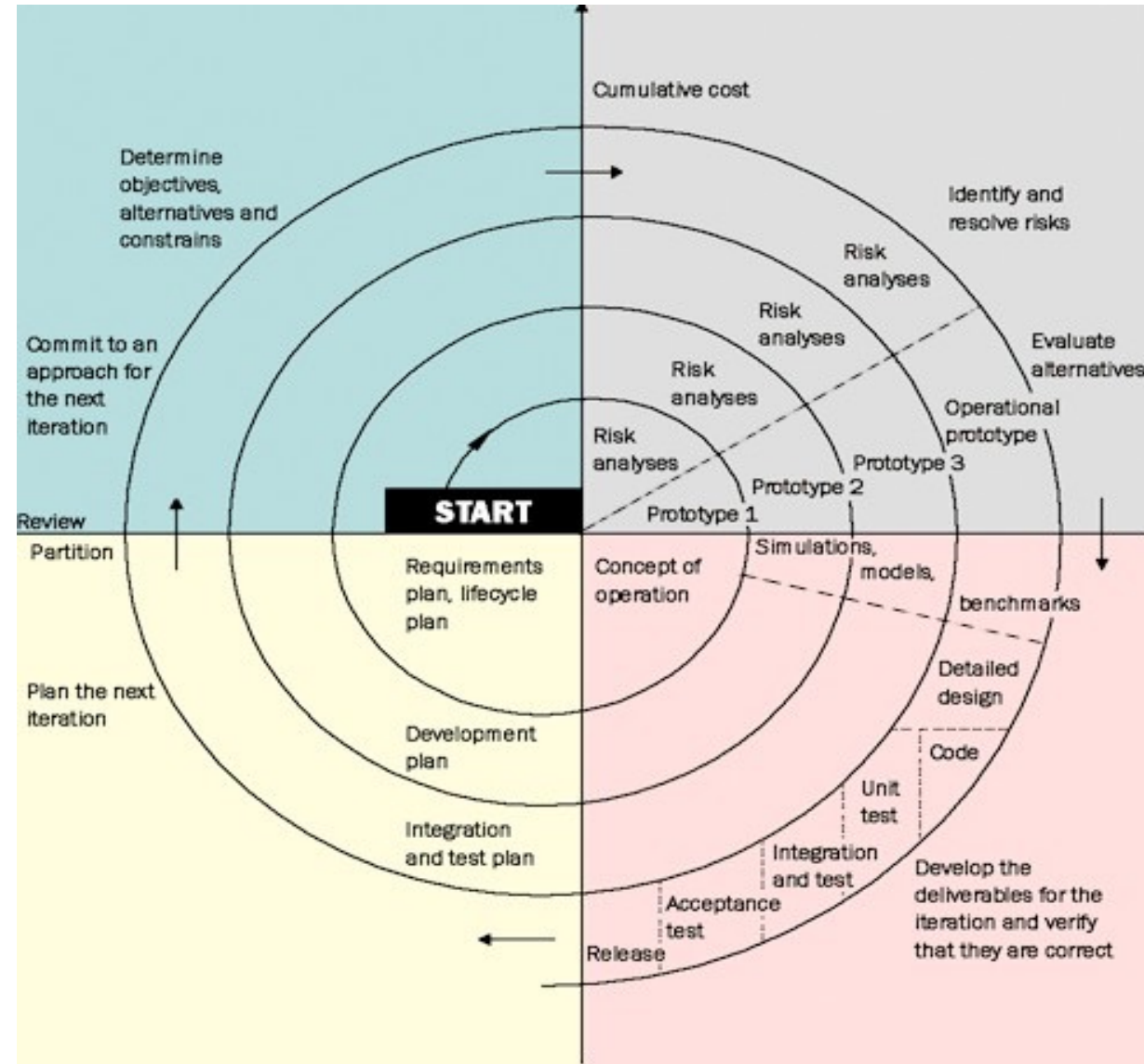
- Mini-waterfalls do not encourage thinking ahead about the big picture
- Must define good interfaces, or else integration will not work
- Temptation to defer difficult functionality till later
- Not all requirements upfront => incompatibilities revealed later

# Spiral Model



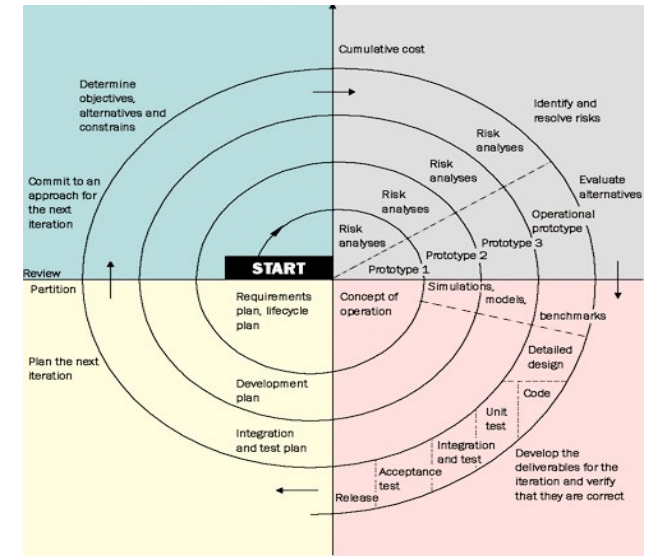


# Spiral Model



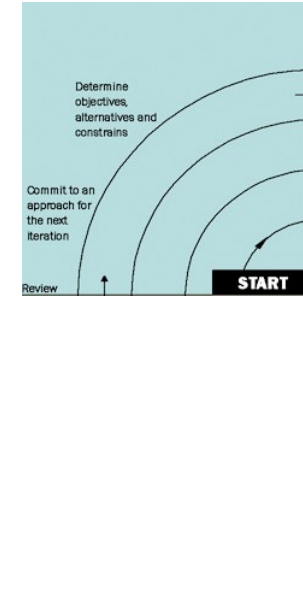
# Spiral Model

- Barry Boehm (1988)
  - *main goal: reduce risk in every phase*
- Four quadrants:



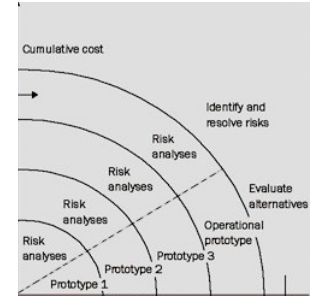
# Spiral Model

- Barry Boehm (1988)
  - *main goal: reduce risk in every phase*
- Four quadrants:
  - *Objectives, Alternatives, Constraints*
    - Objectives: functionality, performance, interfaces, etc.
    - Alternatives: build, reuse, buy, sub-contract, etc.
    - Constraints: cost, schedule, interface, etc.



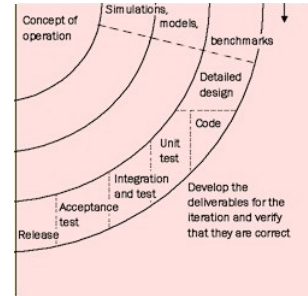
# Spiral Model

- Barry Boehm (1988)
  - *main goal: reduce risk in every phase*
- Four quadrants:
  - *Objectives, Alternatives, Constraints*
    - Objectives: functionality, performance, interfaces, etc.
    - Alternatives: build, reuse, buy, sub-contract, etc.
    - Constraints: cost, schedule, interface, etc.
  - *Evaluate, Identify & Resolve Risks*
    - E.g., lack of experience, new tech, tight schedule, etc.



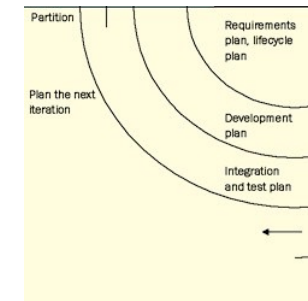
# Spiral Model

- Barry Boehm (1988)
  - *main goal: reduce risk in every phase*
- Four quadrants:
  - *Objectives, Alternatives, Constraints*
    - Objectives: functionality, performance, interfaces, etc.
    - Alternatives: build, reuse, buy, sub-contract, etc.
    - Constraints: cost, schedule, interface, etc.
  - *Evaluate, Identify & Resolve Risks*
    - E.g., lack of experience, new tech, tight schedule, etc.
  - *Develop & Verify*
    - Design, review it, write code, inspect code, test product



# Spiral Model

- Barry Boehm (1988)
  - *main goal: reduce risk in every phase*
- Four quadrants:
  - *Objectives, Alternatives, Constraints*
    - Objectives: functionality, performance, interfaces, etc.
    - Alternatives: build, reuse, buy, sub-contract, etc.
    - Constraints: cost, schedule, interface, etc.
  - *Evaluate, Identify & Resolve Risks*
    - E.g., lack of experience, new tech, tight schedule, etc.
  - *Develop & Verify*
    - Design, review it, write code, inspect code, test product
  - *Plan Next Phase*



# Spiral Strengths

- Identifies high risks early
- Users involved in each step -- rapid prototyping
- Explicit consideration of alternatives and constraints
- Design can evolve
- Good for large, complex projects
- Maintenance included in the spiral



# Spiral Weaknesses

- Lots of time spent on evaluating risks, objectives, ...
- Complex
  - *especially if you include other development methodologies*
- No firm deadlines, can spiral forever
- Developers must be re-assigned during non-dev
- If sw dev is contractual, client must evaluate all risks before signing contract
  - *hard to do if using the spiral*

# When to Use?

- Medium-high-risk projects
- Large-medium size systems
- Can tolerate resource consumption
- Formal approval process already required
- Robust implementation more important than functionality
- Significant changes expected

# **Agile Development**

# Agile Methods

- Builds upon iterative development
  - *mid-90s, in reaction to heavyweight methods*
  - *leverage iterative dev (uncover problems early)*
- Adds a people-centric viewpoint
- Principle: “if something is good, do it a lot”
  - *frequent feedback (instead of planning)*
  - *communicate regarding impediments*
  - *use prototypes to learn more about requirements*

# Agile Methods

- Scrum
- Adaptive Software Development
- Feature Driven Development
- Crystal Clear
- Extreme Programming
- Rapid Application Development
- Rational Unify Process

# Scrum

- Emerged in mid-80s, formalized in 1995
  - *OOPSLA paper by Ken Schwaber and Jeff Sutherland*
- Growing fast
  - *Yahoo!, Microsoft, Google, Facebook, Motorola, SAP, Cisco, General Electric, Lockheed Martin, ...*

# Scrum

- Basic structure = ***Sprint***
  - *1-4 weeks (rigidly fixed length)*
  - *one after each other*
  - *working product at the end of each sprint*
- Cross-functional teams of 3-9 people
- Meet daily

**Inputs from  
Customers, Team,  
Managers, Execs**



**Product Owner**

  
**Manager**



**The Team**

  
**Scrum Master**



**Daily Standup  
Meeting**



**Sprint Review**



**Potentially  
Shippable Product**



**Sprint  
Retrospective**

1	List of
2	requirements
3	prioritized by
4	business value
5	(highest value
6	at top of list)
7	
8	

**Product  
Backlog**

Team selects  
starting at top  
as much as it  
can commit  
to deliver by  
end of Sprint

**Sprint  
Planning  
Meeting**

**Task  
Breakout**

**Sprint  
Backlog**

**1-4 Week  
Sprint**

**No Changes**

(in Duration or Deliverable)



Inputs from  
Customers, Team,  
Managers, Execs



**Product Owner**

  
**Manager**

  
**Scrum Master**



  
**Daily Standup Meeting**

- Product owner
  - Represents customer's or end-user's interests
  - Maximizes business value
  - Translates needs into a priority list
  - Equivalent to traditional product manager

1	List of
2	requirements
3	prioritized by
4	business value
5	(highest value
6	at top of list)
7	
8	

**Product Backlog**


**Sprint Planning Meeting**

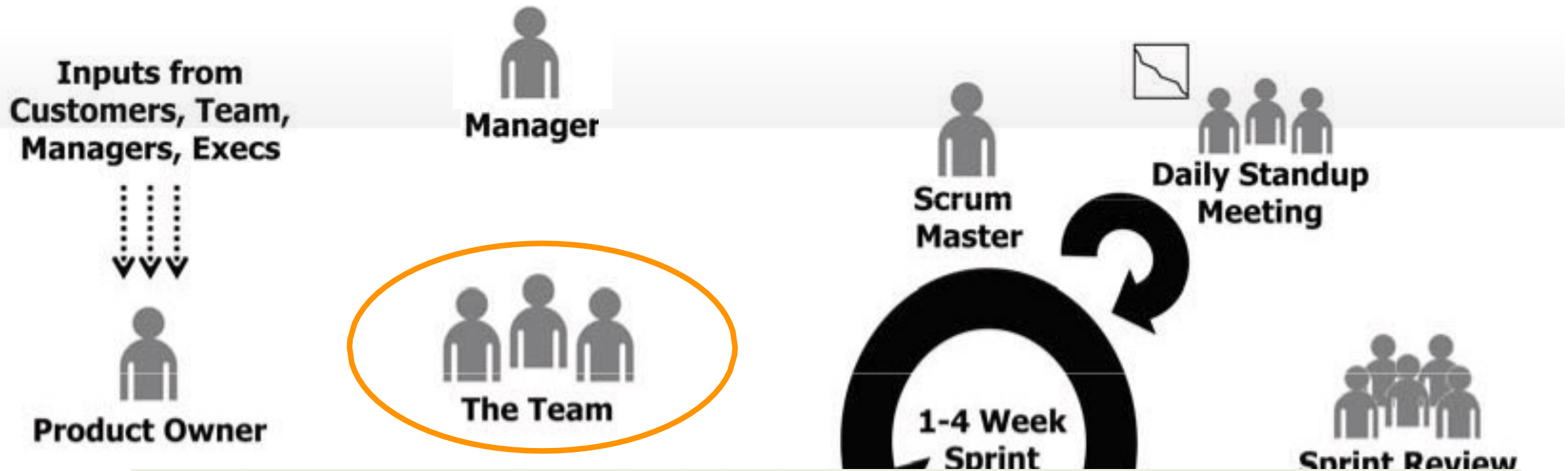
**Sprint Backlog**

**No Changes**  
(in Duration or Deliverable)

**Potentially Shippable Product**

  
**Sprint Retrospective**

  
**Review**



- The Team

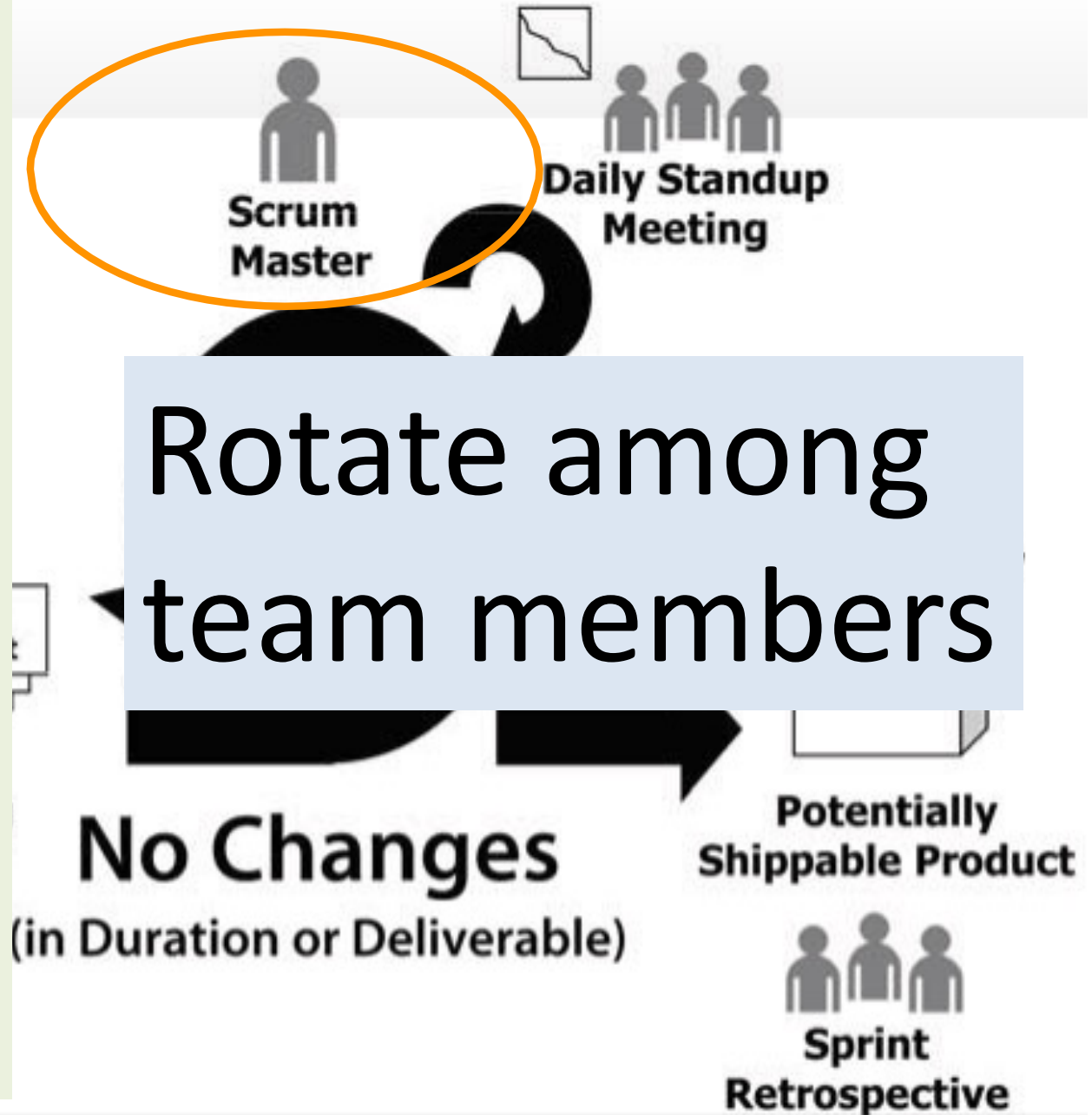
- Builds the product
- Cross-functional (developers, UI designers, testers, analysts, ...)
- Self-managing -> autonomous + accountable
- Small (5-10 people)
- For large projects, form several Scrum teams

1	Li
2	requi
3	prior
4	busin
5	(high
6	at to
7	
8	

Pro  
Bac

Retrospective

- Scrum Master
  - Role = ensure team's success
  - **NOT** a manager of the team
  - Protects team from outside interference (e.g., may push back on product owner)
  - Helps resolve impediments
  - Background can be varied: management, engineering, design, testing, etc.
  - Scrum Master could be a member of the team
    - But must be different from Product Owner



Inputs from  
Customers, Team,  
Managers, Execs

⋮



- Manager
  - Not a nanny, but a guru
  - Mentors, coaches, helps problem-solve
  - Often is an ex-Scrum Master



**Inputs from  
Customers, Team,  
Managers, Execs**



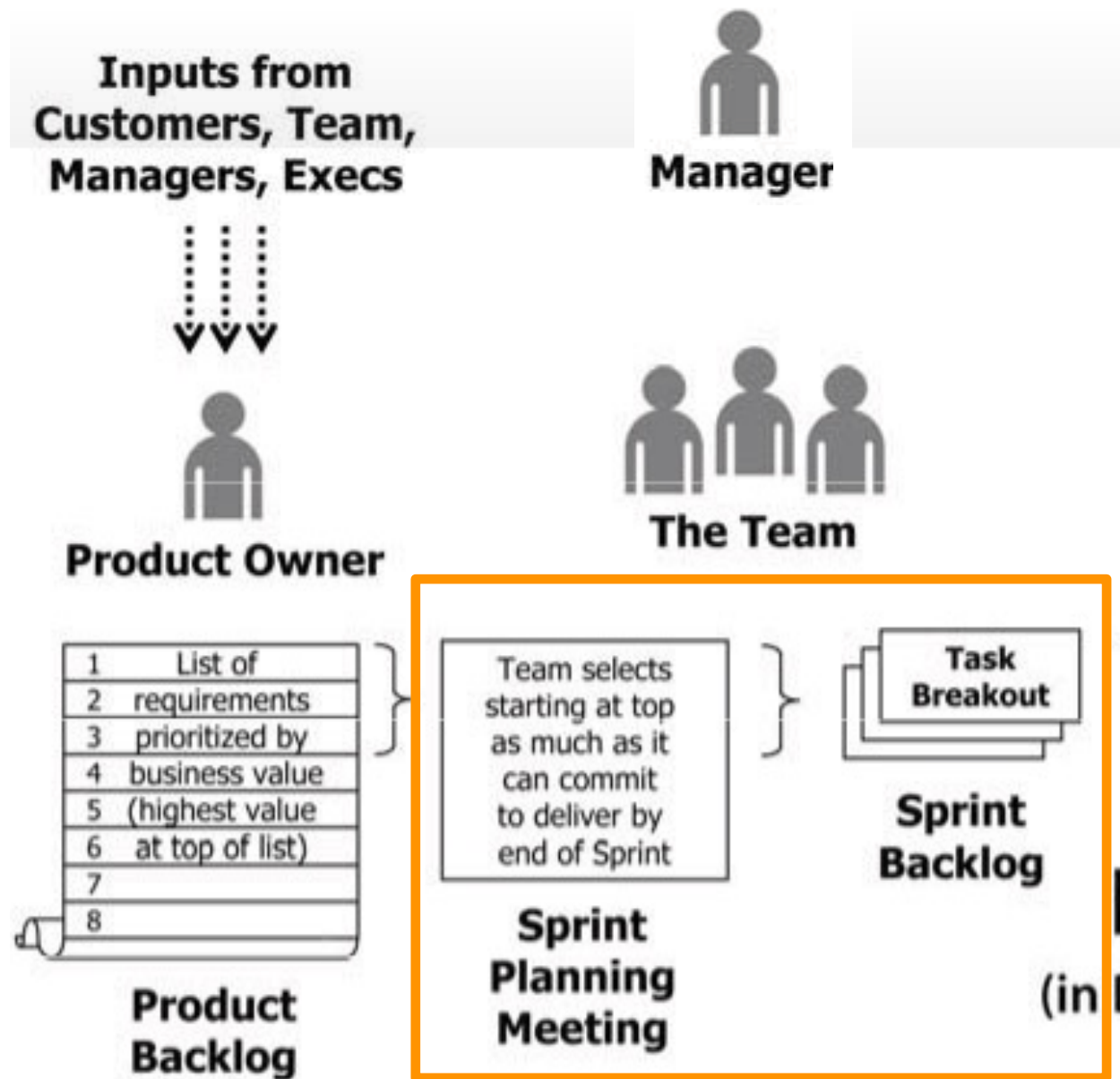
**Product Owner**

1	List of
2	requirements
3	prioritized by
4	business value
5	(highest value
6	at top of list)
7	
8	

**Product  
Backlog**

- Product Backlog
  - Product Owner articulates product vision
  - To-do list prioritized by value to customer
  - Contains
    - Features
    - Development requirements
    - Research/investigate tasks
    - Bugs
  - Articulated in terms of “user stories”
  - Evolves over time (change is a given!)
  - Is the definite view of “everything that could be done by the team ever, in order of priority”
- Team provides time estimates
  - Product Owner uses them to prioritize Backlog





- **Sprint Planning Meeting**
  - First step of every Sprint
  - Product Owner and Team review Product Backlog
    - Gives team insight into the thinking of the customer
  - Team selects items they can complete in this Sprint (starting from the top)
  - Team breaks each item down into tasks, thus producing the Sprint Backlog
  - Once Team has committed, no changes to Product Backlog

- Inputs from
- Daily Scrum
    - Stand-up meeting at fixed time every day
    - Meeting lasts < 15 minutes
    - Each team member reports 3 items:
      - What (s)he has done since last meeting
      - What (s)he will do until next meeting
      - Any blocks or impediments
      - No discussion, just reporting
    - After meeting
      - Scrum Master resolves impediments and updates progress sheet
- Backlog Meeting



# Sprint Duration

- A Sprint is never extended
- If goals not meet, team must own up to it
  - *takes some experience to determine how long tasks will take → over time, team gets better at it*
- Pick one Sprint duration and stick to it
  - *helps team improve their estimates*



**Inputs from  
Customers, Team,  
Managers, Execs**

  
**Manager**

  
**Scrum**

  
  
**Daily Standup  
Meeting**

- Sprint Review
  - Demo the product (<30 minutes prep)
  - Participants: Team + ScrumMaster + Product Owner + customers + stakeholders + experts ...
  - Anyone can ask questions
  - Meeting lasts as long as necessary

  
**Product Owner**

1	List of
2	requiremen
3	prioritized b
4	business val
5	(highest val
6	at top of lis
7	
8	

**Product  
Backlog**

**Sprint  
Planning  
Meeting**

**Backlog**

**No Changes**  
(in Duration or Deliverable)

  
**Sprint Review**

  
**Potentially  
Shippable Product**

  
**Sprint  
Retrospective**

**Inputs from  
Customers, Team,  
Managers, Execs**

  
**Manager**

  
**Scrum**

  
  
**Daily Standup  
Meeting**

  
**Product Owner**

1	List of
2	requiremen
3	prioritized b
4	business val
5	(highest val
6	at top of lis
7	
8	

**Product  
Backlog**

- Sprint Retrospective
  - Participants: Team + ScrumMaster + Product Owner
  - Facilitated by neutral outsider (e.g., other ScrumMaster -> good for cross-pollination)
  - Product Owner updates the Product Backlog (feeds into next Sprint Planning Meeting)
  - No downtime between Sprints, maintain the pace

  
**Sprint Review**

  
**Potentially  
Shippable Product**

e)

  
**Sprint  
Retrospective**

**Inputs from  
Customers, Team,  
Managers, Execs**



**Product Owner**

  
**Manager**

  
**The Team**

  
**Scrum Master**

  
**Daily Standup Meeting**

**1-4 Week  
Sprint**

  
**Sprint Review**

1	List of
2	requirements
3	prioritized by
4	business value
5	(highest value
6	at top of list)
7	
8	

Team selects  
starting at top  
as much as it

**Task  
Breakout**

- Once Product Owner decides product is ready, do a final Release Sprint

  
**Potentially  
Shippable Product**

  
**Sprint Retrospective**

(in Duration or Deliverable)

**Inputs from  
Customers, Team,  
Managers, Execs**



**Product Owner**



**Manager**



**The Team**



**Scrum  
Master**



**Daily Standup  
Meeting**



**Sprint Review**



**Potentially  
Shippable Product**



**Sprint  
Retrospective**

1	List of
2	requirements
3	prioritized by
4	business value
5	(highest value
6	at top of list)
7	
8	

**Product  
Backlog**

Team selects  
starting at top  
as much as it  
can commit  
to deliver by  
end of Sprint

**Sprint  
Planning  
Meeting**

**Task  
Breakout**

**Sprint  
Backlog**

**1-4 Week  
Sprint**

**No Changes**

(in Duration or Deliverable)