

# Agile development

Pedro Cabalar

Departamento de Computación  
Facultad de Informática  
University of Corunna, SPAIN

# Agile Development

- Main bibliographic source:

*Agile & Iterative Development – A Manager's Guide*, Craig Larman, Addison-Wesley, 2004.

# Manufacturing vs Development

Compare these 2 building tasks:

## 1. Building mobile phones on an assembly line

- Unambiguous specifications
- Reliable estimation of tasks and scheduling





## 2. Building a custom house

- Initially, the owners are not exactly sure of what they want
- They may clarify their mind as they see the design, the costs, time, etc.



# Manufacturing vs Development

Predictable Manufacturing	New Product Development
	
We can: first complete specifications and then build	Difficult to create unchanging specs from the beginning
Cost and effort can be <b>reliably estimated</b> at the beginning	Estimations are only possible after some <b>empirical</b> data increasingly available
We can identify, define and <b>schedule all the activities</b> (PERT, Gantt, etc )	We need adaptative steps driven by <b>build-feedback cycles</b>
Unpredictable <b>change</b> is rare and <b>unexpected</b>	<b>Creative adaptation to change</b> is the norm

# Software development

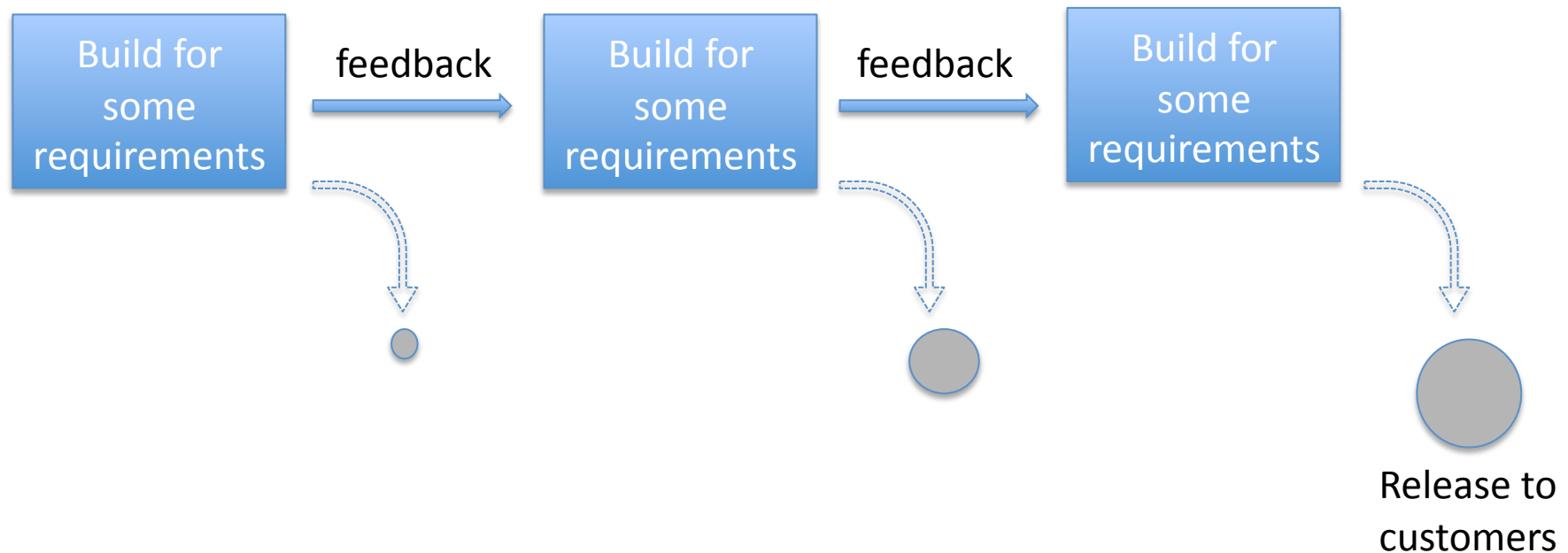
- Software construction mostly fits in the case of **new product development**
- However, SW development methodology was originally focused on a **mass manufacturing problem...**

# Classical waterfall methods



# Iterative & Evolutionary

- **Iterative development:** the overall lifecycle is composed of several **iterations** in sequence.



# Iterations

- Each iteration generates an **iteration release**:
  - release = stable, integrated and tested system, collecting all the SW across the teams
  - Most releases are internal. The final iteration release is the complete product released externally
  - Recommended duration: **1-6 weeks per iteration**
- **Incremental development**: in each iteration the system is tuned, but it also **grows** with new features



# Iterative planning

- Iterative **planning**: what to do in the next iteration?
- **Risk-driven**: riskiest features first
- **Client-driven**: the client **decides** the most valuable feature to build next
- Advantage: the client steers the project using the **latest insight**, not just an initial speculation

# Iterative planning

What to do if **we cannot meet the end date** for an iteration? Three solutions:

1. Slip the end date
2. Work more hours
3. Reduce the scope

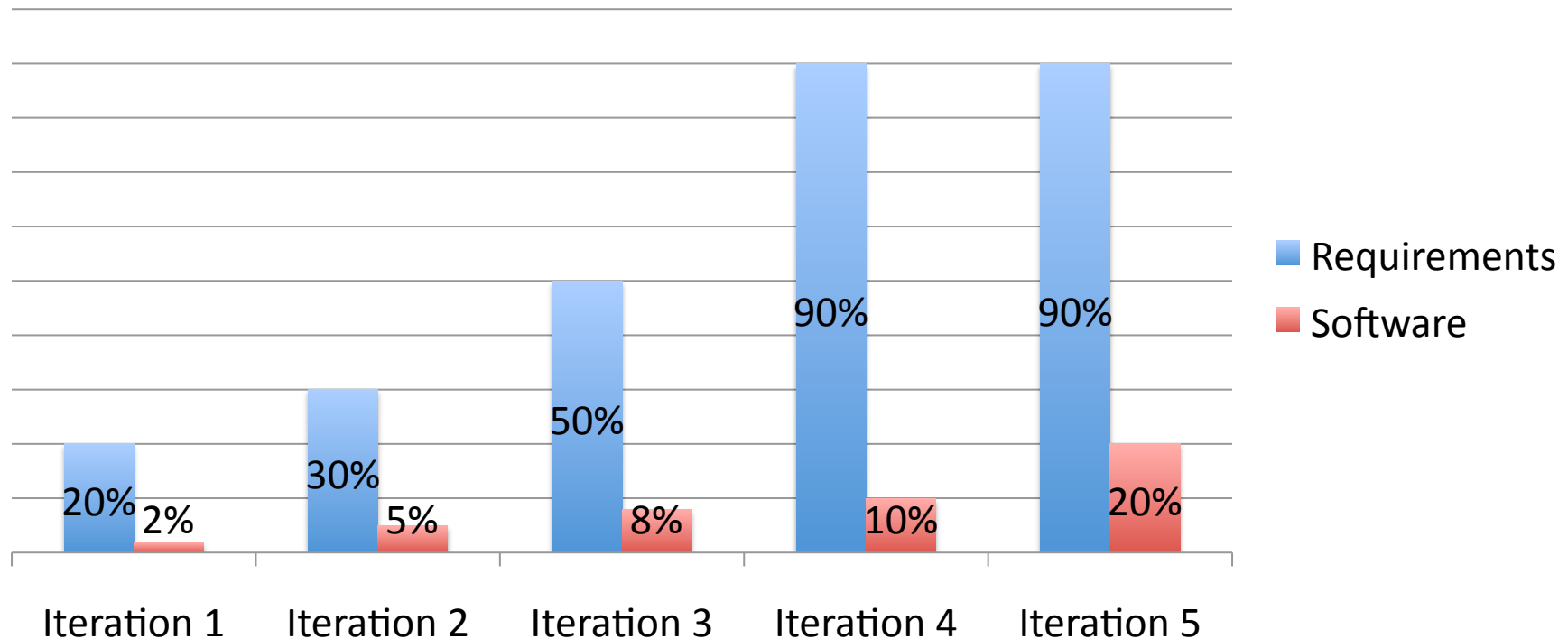
1. **NO. Timeboxing:** the end date cannot be changed!
2. **NO. Respect:** do not add pressure due to bad planning
3. **YES! Reduce the scope:** lower priority features moved to the wish list

# Evolutionary requirements

- Embrace **change** but not chaos: changes occur from iteration to iteration
- A point of **stability** is needed:  
*during an iteration, no changes from external stakeholders*
- **Requirements workshop** at each iteration:  
used to expand or refine requirements.

# Evolutionary requirements

- Requirement analysis is **evolutionary**, but not unbounded. Usually, 90% of requirements are obtained in the first 3 or 4 iterations. Example:



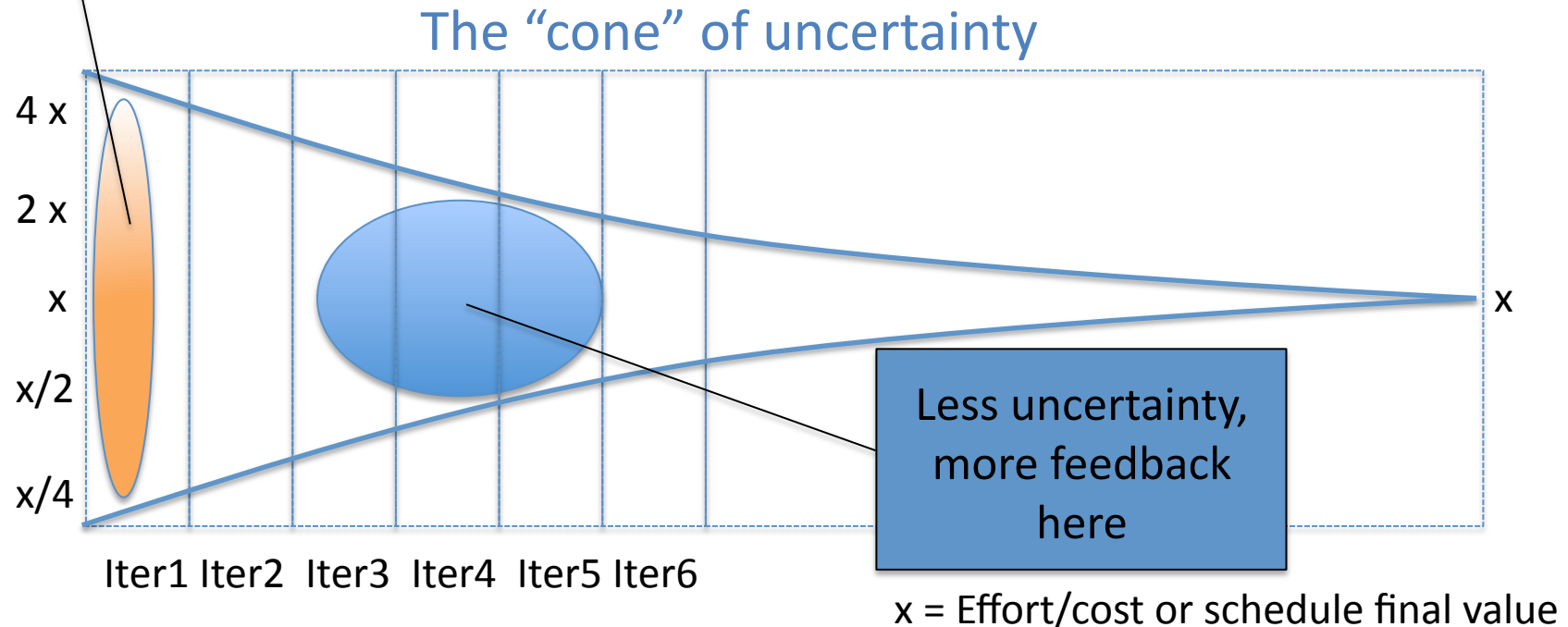
# Evolutionary requirements

- Evolutionary ≠ “no early requirements”
- First iterations encourage clarifying things like:
  - “top ten” high level requirements
  - Architectural influential factors (load, usability, internationalization, ...)

Bad moment  
for estimations

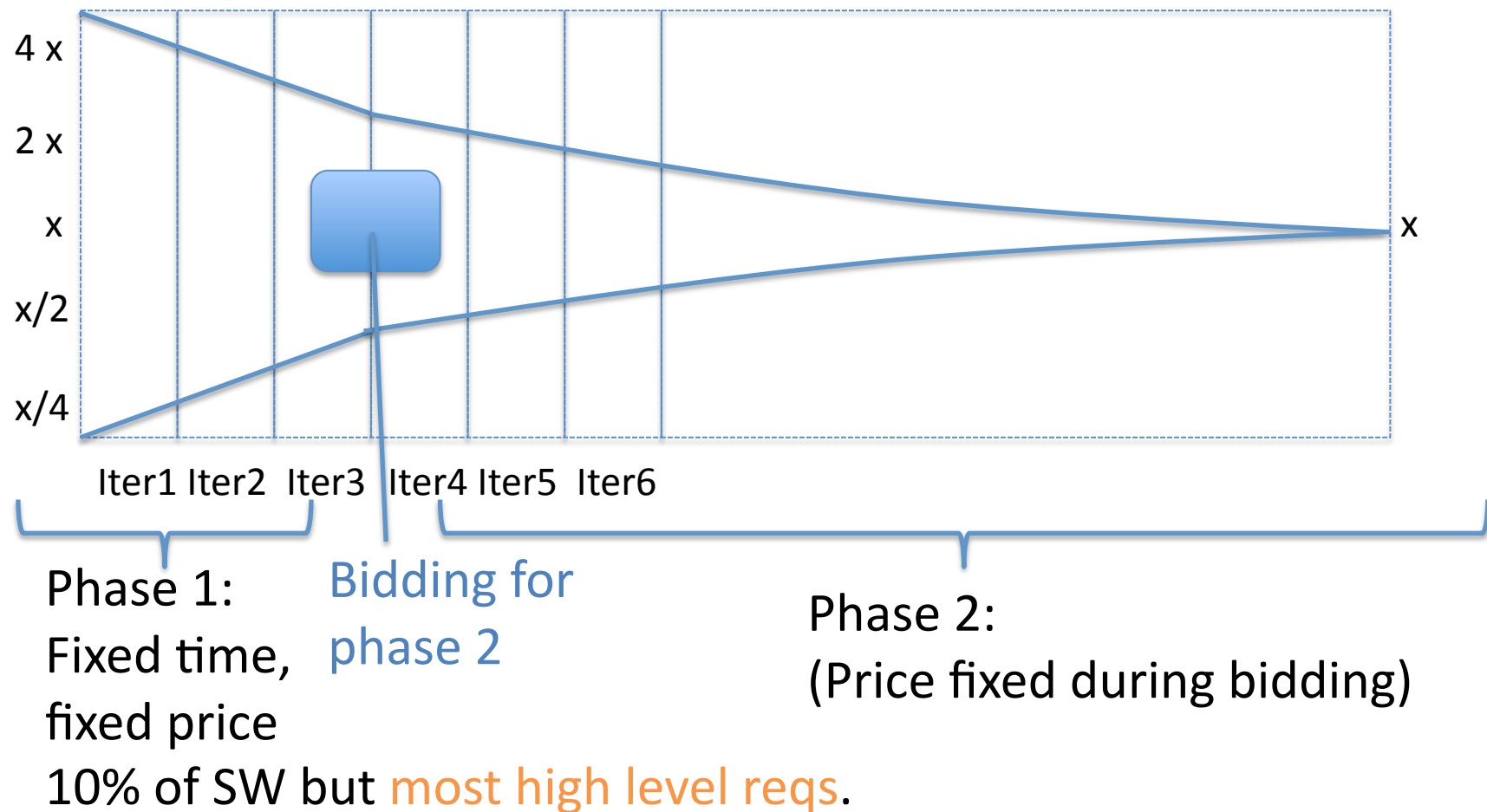
# Adaptative planning

- Initial phase has **high uncertainty**. Difficult to make a good planning or cost estimation.
- **Defer expectation** for good estimates until a few iterations (10-20% of the project)



# Adaptative planning

- Sometimes contracts are split into two phases



# Delivery

- Delivery = is an iteration release put in production (marketplace)
- Promote and schedule **incremental deliveries**, with **expanding capabilities**.  
Example: plan deliveries each 6 months
- Promote **evolutionary deliveries**: use **feedback** from users of installed products.



# Iterative & Evolutionary methods

## Iterative methods

- Evo (Evolutionary project management) (1960s)
  - Iterations of 1-2 weeks
  - Evolutionary delivery at each iteration
  - Adaptative client/value-driven planning
  - Numeric measures for progress and quality

# Iterative & Evolutionary methods

## Iterative methods

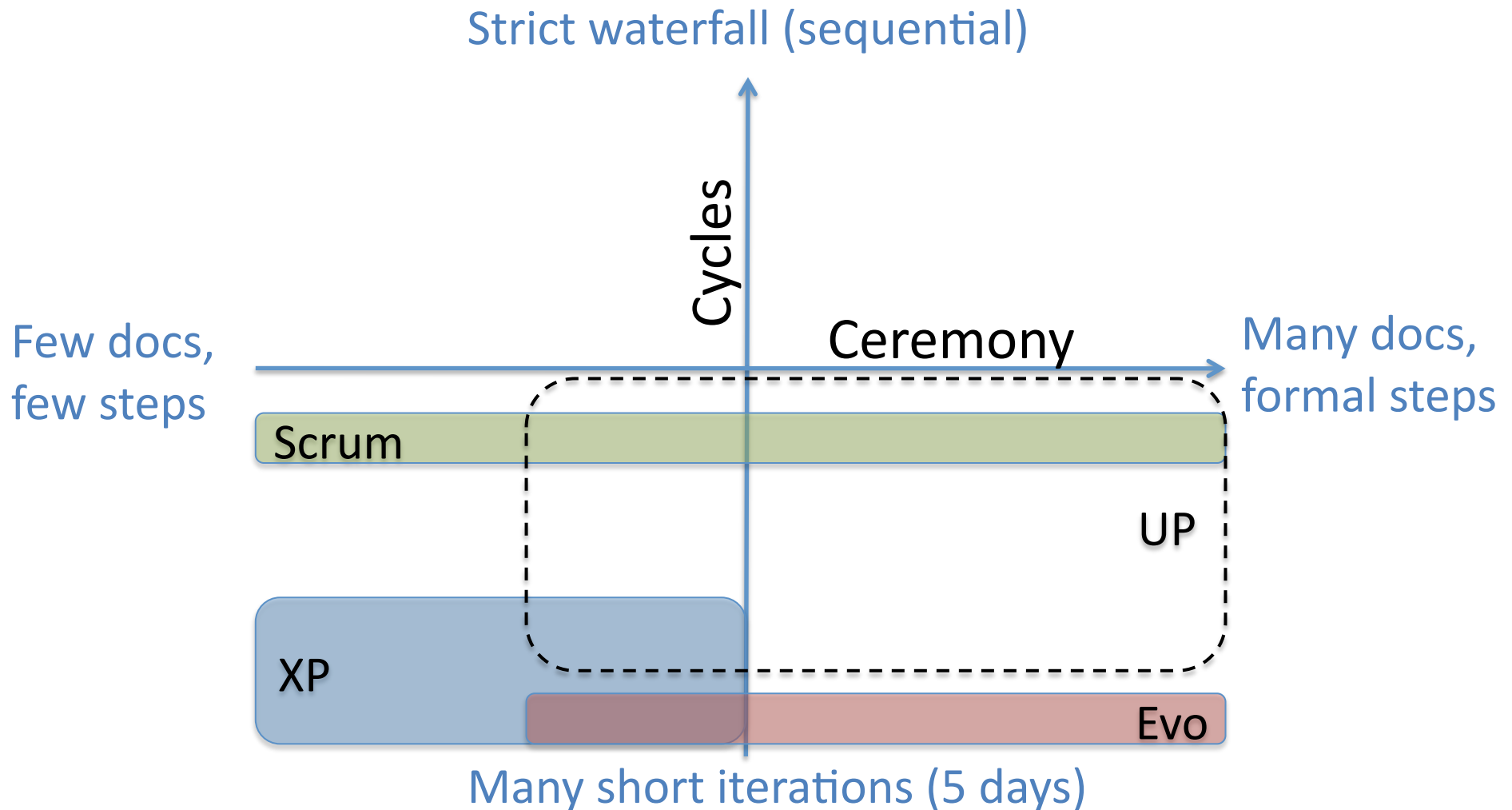
- UP (Unified Process) /RUP (Rational UP) (mid 1990s)
  - Short timeboxed iterations
  - **Elaboration phase**: early iterations fix the high-risk & high-value elements that are core for architecture
  - **Construction phase** iterations build the remainder
  - We can use up to 50 **artifacts** or workproducts (vision, use-case model, risk list, etc) classified by **disciplines** (requirements, design, project management, implementation, etc).

# Iterative & Evolutionary methods

## Agile methods

- They were conceived in the 1990's. Some well-known examples are:
  - Dynamic Systems Devel. Method, DSDM (1994)
  - Scrum (1995)
  - Crystal Clear (1996)
  - Extreme Programming, XP (1996)
  - Feature Driven Development, FDD (1997)

# Classification of methods



# Classification of methods

- Cockburn scale used to show adequacy of methods with respect to **criticality** and **staff**

Criticality (defects cause loss of ...)	Number of people			
	1-6	≤20	≤40	≤100
Life (L)	L6	L20	L40	L100
Essential Money (E)	E6	E20	E40	E100
Discretionary Money (D)	D6	D20	D40	D100
Comfort (C)	C6	C20	C40	C100

# Classification of methods

- An example: method measuring for an L100 project has nothing to do with a C6 project.

Criticality  
(defects cause loss of ...)

Number of people  
1-6    ≤20    ≤40    ≤100

Life (L)	L6	L20	L40	L100	...
Essential Money (E)	E6	E20	E40	E100	
Discretionary Money (D)	D6	D20	D40	D100	
Comfort (C)	C6	C20	C40	C100	

# Agile methods



# Agile methods

- They share
  - Short timeboxed iterations
  - *Embrace change*: agility, rapid and flexible response to change
  - Adaptive, evolutionary refinement of plans
- Although conceived in the 1990s, they receive the name of “*agile methods*” in 2001. A group of 17 developers meet at the Rocky Mountains, Utah and publish the

## *Agile Manifesto*

- It contains the following 12 principles ...



# The Agile Manifesto

1. Customer satisfaction = highest priority

☞ Keypoint: early and **continuous** delivery of valuable SW

2. Welcome changing requirements even in late development

☞ Greater customer's competitive advantage

3. Deliver **working SW frequently**

☞ Weeks better than months!

# The Agile Manifesto

4. Business people <--> developers  
daily cooperation
5. Build projects around motivated individuals
  - ☞ Give them environment and support,  
then trust them to get the job done
6. The best form of communication is  
face-to-face conversation
  - ☞ Documentation is relegated to a 2nd level

# The Agile Manifesto

7. The primary measure of progress is **working software**
8. Promote **sustainable development**
  - 👉 Be able to maintain a constant pace (like running a marathon)
9. Continuous attention to **technical excellence** and **good design**

# The Agile Manifesto

10. **Simplicity** – the art of maximizing the amount of work not done – is essential
11. The best architectures, requirements and designs emerge from **self-organizing teams**
12. At **regular intervals**, the team **readjusts** and tunes to become more effective

# Agile Practices

- I. Individuals and interactions over process and tools
  - People is the most important ingredient of success
  - A “strong player” need not be an ace programmer, but an average one that works well with others
  - First, build a good team, then let them configure their environment and tools

# Agile Practices

## II. Working software over comprehensive documentation

- Documenting software is essential, but **too much is worse** than too little
- Too much docs = effort to **sync with the code**, or they become big lies that lead to confusion
- A document needs to be **short** and **salient**:  
Short = <30 pages, salient = overall design and highest-level structures

# Agile Practices

## III. Customer collaboration over contract negotiation

- A closed contract is an early source of failure
- Requirements, schedule, costs usually become obsolete before the project is completed
- Involve customer feedback on a regular, frequent basis
- The best contracts govern the way in which customers and developers will work together

# Agile Practices

## IV. Response to change over following a plan

- SW projects cannot be planned for far future: changing environment, requirements, estimations, ...
- Plans degrade with time: PERT, Gantt diagrams change not only in durations but also in structure
- Solution: make flexible plans, detailed for next 2 weeks, rough for the next 3 months, and crude beyond that