Chapter 6

Iterative, Evolutionary and Agile

Iteration

- The most import aspect of OOA/D
- An *agile* practice

en.wikipedia.org/wiki/Agile_software_development

- vs *waterfall* – early and repeated programming, feedback, adaptation
- *waterfall* is the opposite – big up front requirements investment

What is UP?

- development organized is a series of short, fixed-length mini-projects called *iterations.*
- each mini-project results in tested, integrated, executable partial programs
- each mini-project has its own requirements analysis and design, implementation and testing phases
- complete project implemented by successive enlargement and refinement
- feedback and adaptation at the end of a mini-project is crucial.
- both *iterative and incremental* as well as *iterative and evolutionary*

Example (three week iteration):

- Monday AM, kick-off meeting, clarify tasks and goals
- meanwhile reverse engineer last iteration’s code into UML diagrams
- Monday PM, whiteboard work in pairs, UML diagrams captured with digital camera, some pseudocode, notes
- next three weeks – coding, testing, design, integration, daily builds
- NOTES:
  - requirements/design part of each iteration
  - after three weeks, something should execute
  - output is NOT experimental or throw-away, this is not prototyping

Handling Change

- Embrace it

  I have always found that plans are useless, but planning is indispensable.

  - Dwight Eisenhower

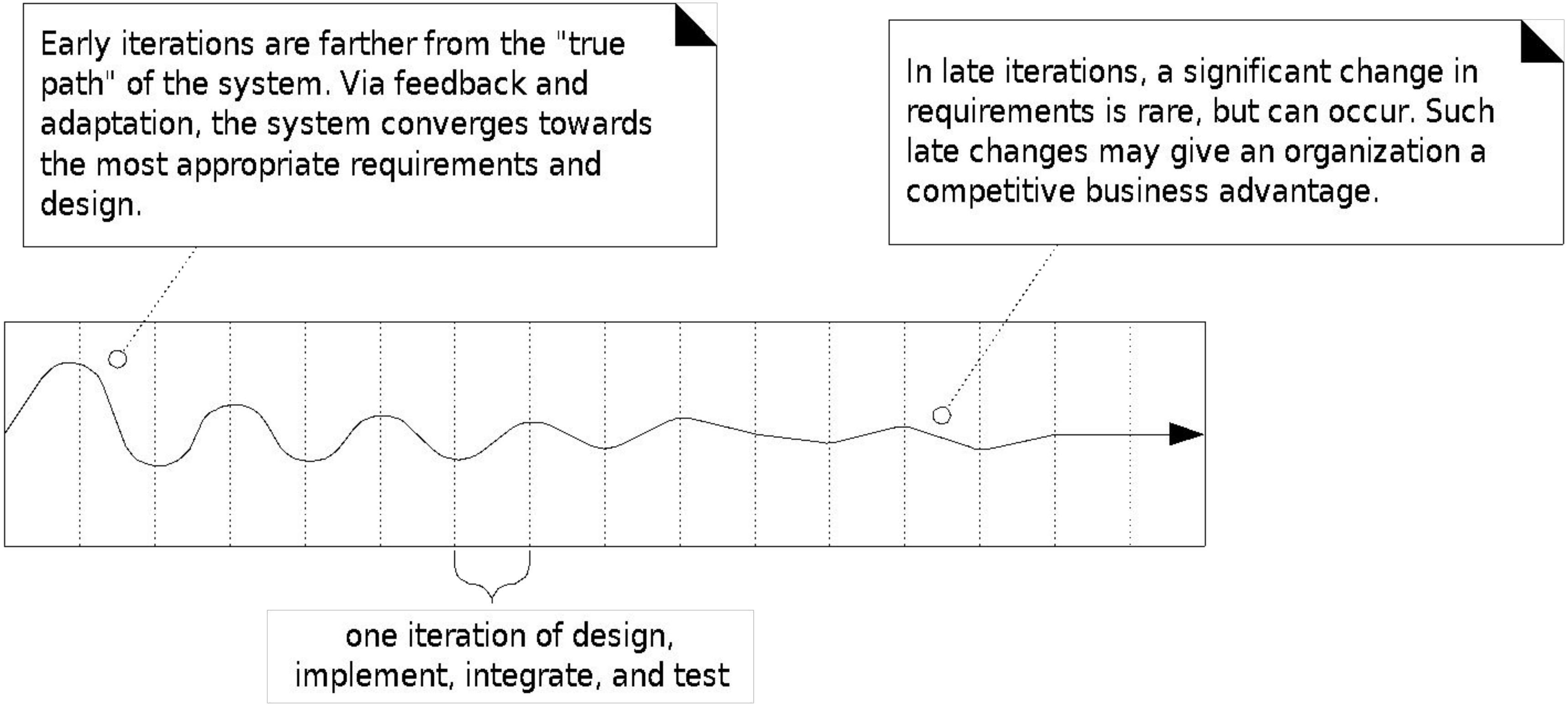
- Don’t try to avoid change by being “complete” up front
- adaptation drives success

- should not be uncontrolled (feature creep), must be disciplined
- keep to small subset of requirements, quick coding, early feedback, be ready for change

Value of Feedback

- Can’t be underestimated. Yes, some new features will be added but mostly it will clarify requirements
- Example, load testing could show fundamental approach is not scalable.
- Early on, you see more deviation caused by feedback, later on, less

Fig. 2.2



Early iterations are farther from the "true path" of the system. Via feedback and adaptation, the system converges towards the most appropriate requirements and design.

In late iterations, a significant change in requirements is rare, but can occur. Such late changes may give an organization a competitive business advantage.

one iteration of design, implement, integrate, and test

Benefits:

- less project failure, better productivity, fewer defects
- early mitigation of high risks
- early visible progress
- early feedback
- managed complexity (bite-sized chunks)
- learned in one iteration, used in next

How long should an Iteration be?

- usually two to six weeks
- regardless, fix the time (**timeboxed**)

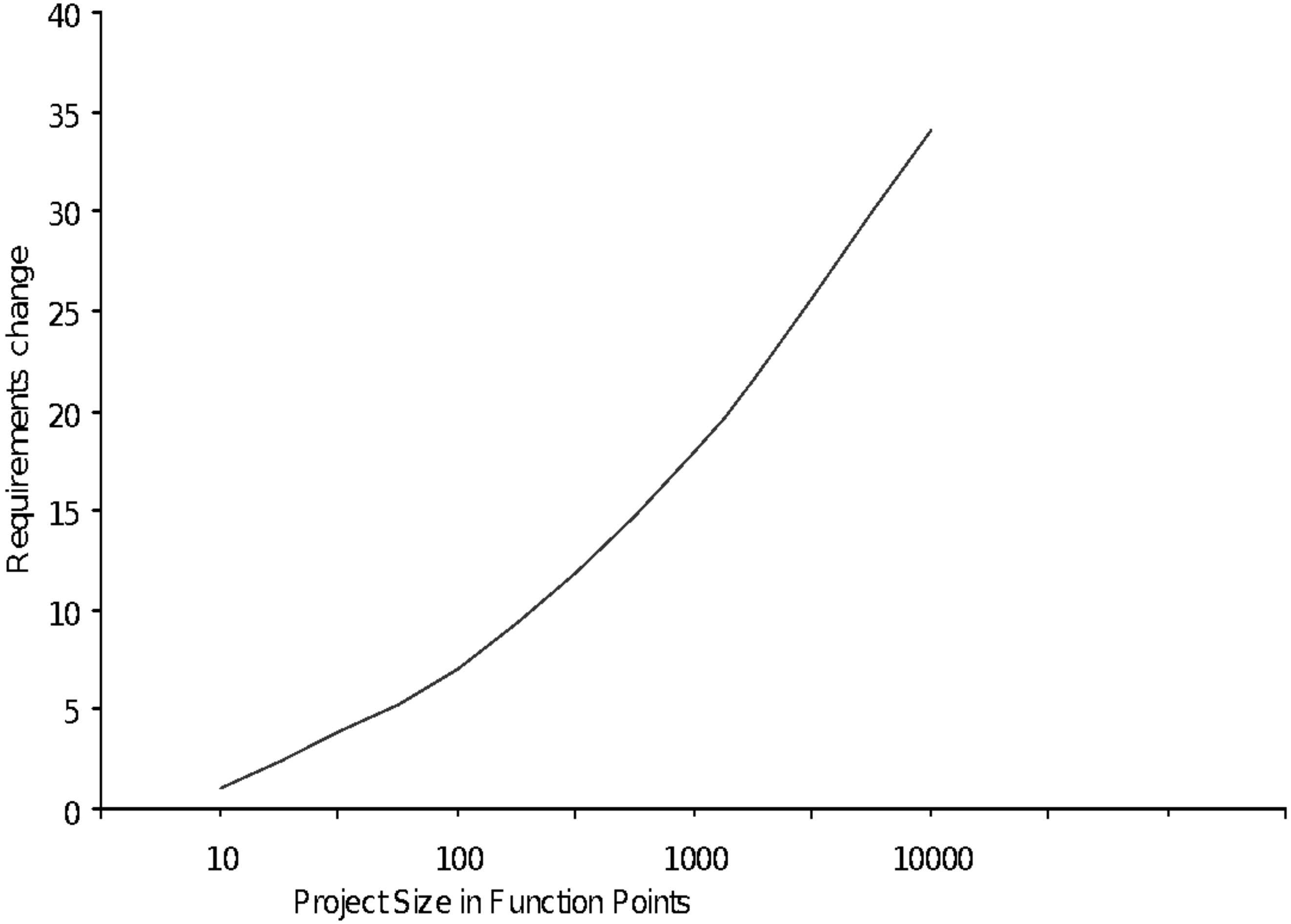
What about Waterfall?

- decidedly sequential
- requirements before programming
- too much time invested in UML diagrams that change
- high failure rates, defect rates
- 45% of requirements never used
- schedule varies up to 400%
- need to avoid "waterfall" thinking in our project (let's write all the use cases first) or (let's design all our classes with UML before we code)

Why Waterfall Fails:

- false assumption: specifications are predictable and stable and can be correctly defined at the start
- typical is 25% change in requirements, more in big projects
- change is the only constant so feedback and adaptation are essential

Fig. 2.3


40
35
30
25
20
15
10
5
0
Requirements change
10
100
1000
10000
Project Size in Function Points

How to do Iterative:

- Analysis and design are essential starting points, just don’t try to be complete
- A recipe:
  - before 1st iteration meet with business people and decide list of use-case names, non-functional features
  - pick 10% of use-cases with these qualities
    - significant, high business value, high risk
  - do detailed analysis on these 10%
  - select a subset of the 10% for design and implementation (3-week timebox)
  - list the tasks of this iteration

How to do Iterative (2):

- Iteration 1:
  - days 1 & 2: modeling and design work in pairs, whiteboarding, UMP, use a war room
  - after day 2: put on your programming hat – program, test, integrate
  - various levels of testing – unit, acceptance, load, usability, …
  - 1 week to go: check if iteration goals met; if not, scale down expectations (create "to do" list). **NOTE: You will almost certainly only have a fraction of coding done**.
  - 4 days to go: freeze code,
  - 3 days to go: demo
  - rest: get ready for next iteration (next slide)

How to do Iterative (3):

- After Iteration 1:
  - do second requirements workshop; review results of last iteration, identify another 10% of the requirements that you will work on (most significant, etc) and analyze them in detail
  - at this point up to 25% of requirements are detailed
  - last day: planning day for next iteration
- Iteration 2:
  - do it as before
- Repeat for a couple more iterations
  - we now have 80% of detailed requirements but only 10% of code

How to do Iterative (4):

- After 4 Iterations:
  - 20% of iterations done
  - in UP this is called the end of the **elaboration phase.**
  - time to estimate what is needed to complete detailed requirements (should be good estimates)
  - no more requirements; 3-week iterations until you are done.

What is Risk-Driven and Client-Driven Planning?

- **Risk-Driven:** early iterations driven by high risk requirements
- **Client-Driven**: early iterations driven by what client cares most about

What are Agile Methods and Attitudes?

- Agile development uses timeboxed, iterative and evolutionary approaches.
- Use adaptive approaches, incremental delivery, encourage speed and flexibility
- Some practices:
  - common project workroom,
  - self-organizing teams

What is Agile Modeling?

- Secret of Modeling: to understand not to document
- The purpose of UML is to give you well-defined language in which to develop your understanding
- together these make up “agile modeling”

Consequences:

- not about avoiding modeling
- modeling to support understanding
- don’t use UML everywhere; too time consuming
- follow the “simplest tool” approach
- work in pairs; switch roles
- create models in parallel (use case || sequence diagram)
- use “good enough” notation; reduce UML
- realize your understanding will be imperfect; fill in the details later
- developers should do their own design; letting others design for you is “waterfall”

What is Agile UP?

- UP is big; pick a small subset of activities you like best
- UP is iterative and evolutionary so agile by nature
- follow agile modeling practices
- not a detailed plan for entire project; Iteration Planning is done one iteration at a time

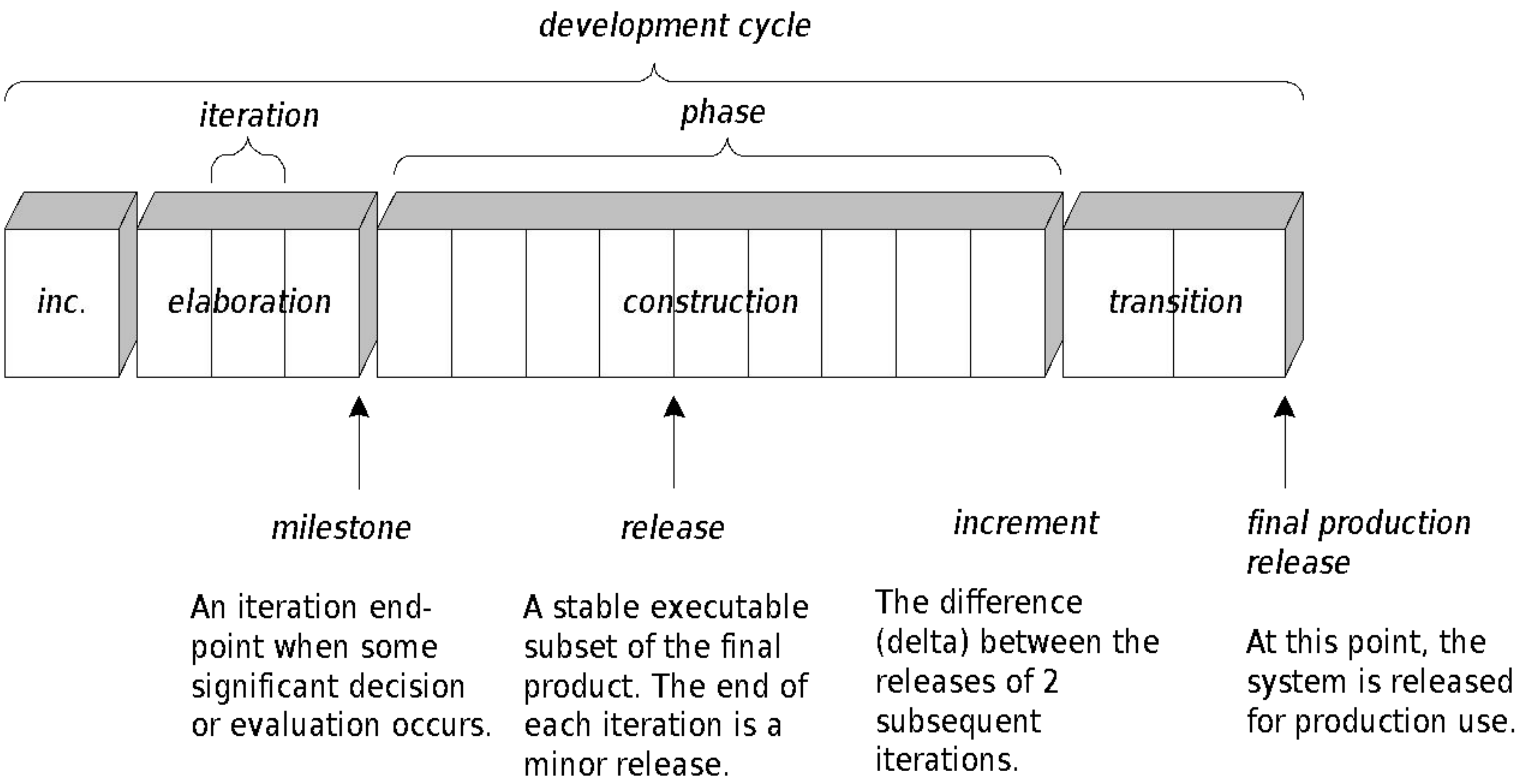
Other UP Practices:

- essential idea: short timeboxed iterative, evolutionary and adaptive programming
- Also:
  - tackle high-risk, high-value issues first
  - engage users at feedback time
  - build the core early
  - continuous verify; test early and often
  - use use-cases
  - use visual modeling
  - manage requirements (no feature creep)
  - create a change request mechanism

What are the UP Phases?

- Inception:
  - approximate vision, business case, scope, estimates
- Elaboration:
  - refined vision, core implemented iteratively, attack high risks, most requirements identified
- Construction:
  - fill in the details through iteration
- Transition:
  - beta tests and deployment

Fig. 2.6



milestone

An iteration end-point when some significant decision or evaluation occurs.

release

A stable executable subset of the final product. The end of each iteration is a minor release.

increment

The difference (delta) between the releases of 2 subsequent iterations.

final production release

At this point, the system is released for production use.

What are the UP Disciplines?

- In UP, a type of activity is a **discipline**; an **artifact** is a work product

- Business Modeling:
  - The Domain Model artifact
- Requirements:
  - Use-Case Model, Supplementary Specifications
- Design:
  - Design Model

UP Disciplines (Fig. 2.7)

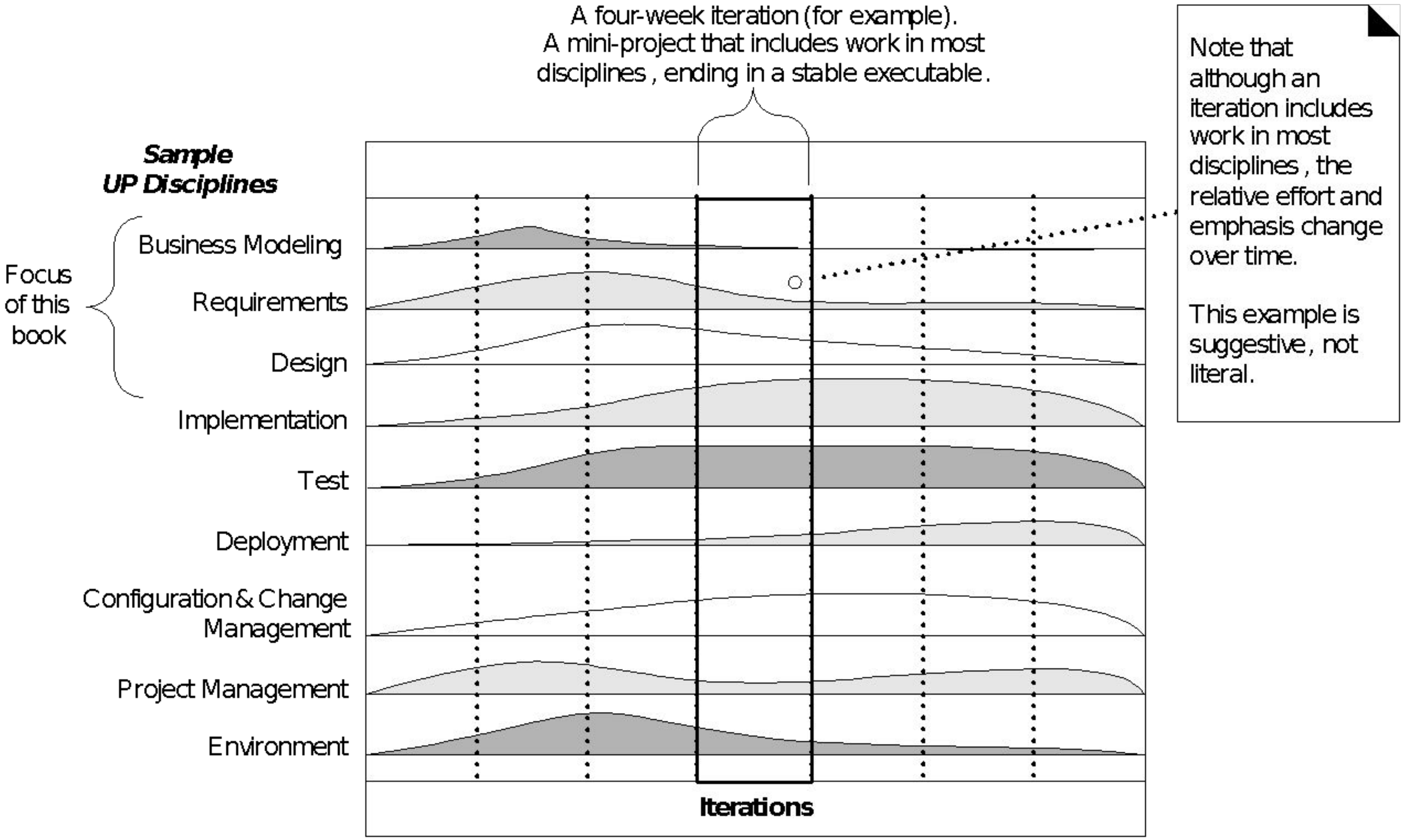
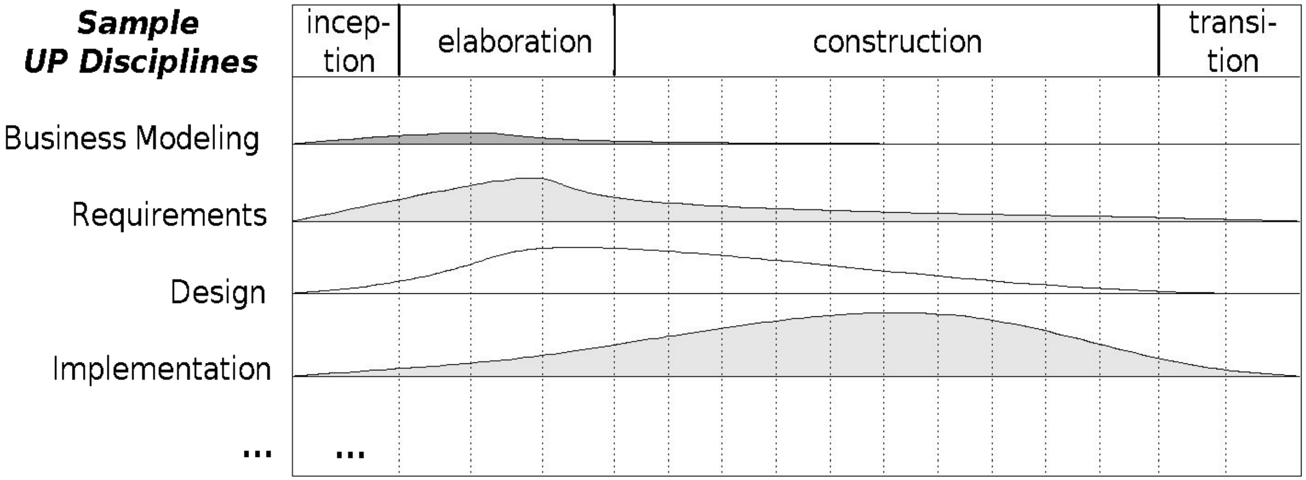

A four-week iteration (for example).
A mini-project that includes work in most disciplines, ending in a stable executable.
Sample UP Disciplines
Focus of this book
Business Modeling
Requirements
Design
Implementation
Test
Deployment
Configuration & Change Management
Project Management
Environment
Iterations
Note that although an iteration includes work in most disciplines, the relative effort and emphasis change over time.
This example is suggestive, not literal.

Fig. 2.8



The relative effort in disciplines shifts across the phases.

This example is suggestive, not literal.

How to Customize the Process?

- Almost everything is *optional* (not the code)
- The GOAL is to have a good understanding of the system being developed.