



# Announcements

---

Today: start on the Unified Process  
(A software life cycle model)

HW 4: email problems

...if you sent the HW and it was returned,  
please send it again

Note HW5 due next \*time\*

# The Unified Process



---

- Reading for today's material: chapter 2
- Background on life cycle models
  - Waterfall model – classic (see Fig.)
  - Fountain model – OO version of waterfall (Fig.)
  - Incremental model – loop-based
- Software *development* process
  - Just the development *part* of the life cycle
  - Unified Process – based on incremental life cycle
  - Let's look at the Unified Process more...

# Iterative and Incremental Development



---

- Using the Unified Process means building software
  - ...iteratively
  - ...incrementally
- (See Figure 2.3)
- The traditional spec, design, implementation, etc., phases map fuzzily to Fig. 2.3



# Models, Ideologies, and Extremes I

---

- The relatively rigid waterfall idea often isn't the best way
  - Often, you can't *lock in* specs and design
  - Waterfall works best for *standard* systems
  - Developing *novel* systems needs more flexibility
- Incrementalism is somewhat the opposite of the waterfall approach



# Models, Ideologies, and Extremes II

---

- If the waterfall idea has limits...
  - Is incrementalism the perfect answer?
- Like any model, philosophy, etc...
  - ...incrementalism has limits too!
- Example: temperate fruit system
  - (Story from Schach, 5<sup>th</sup> ed., p. 497)
- (UP fans would say UP could handle this)



# Models, Ideologies, and Extremes III

---

- A good philosophy:
  - Models, ideologies, philosophies, etc., *have limits*
    - (We'll leave politics and religion alone, but draw your own conclusions)
  - (Does this philosophy have limits?)



# Unified Process: More Motivation

---

- Recall: using the Unified Process means building software
  - ...iteratively
  - ...incrementally
- Specs often change as projects proceed
  - ...often, clients can't reliably specify until they see the context – a working program
  - ...work *with* that fact, instead of fighting it



# Unified Process: Some Details

---

- Iterate a specify-design-implement loop
- For each iteration:
  - Specify some (“a small subset of”) requirements
  - (“Quickly”) design, implement, and test them
  - Get feedback from users, performance tests, etc.
- Next iteration:
  - Do more requirements
  - Or, redo the last iteration to get it right



# Does the System Stagger about Uncontrollably??

---

- Build-and-fix life cycle model
  - Yes...that's why software engineering started
- Waterfall life cycle model (& others)
  - No (actually a little) staggering allowed!
- Incremental life cycle model
  - (like Unified Process)
  - 'Controlled staggering' is the way to go
  - (See Figure 2.2)



# Some Advantages of Iterative Development

---

- Reduce risk of major backtracking
- Backtracking is expensive!
  - That's why the build-and-fix model is bad
- (See illustrative figure)
- Lesson: do the most risky things first
  - ...i.e., in an early iteration



# Advantages of Iterative Development II

---

- Everyone (you & client) can see progress
- Get feedback from clients early & often
- Breaks overall task into manageable bites
  - ...avoid “analysis paralysis”
- Each iteration holds lessons for the next
  - ...the process itself improves
  - CQI = ‘Continuous Quality Improvement’
  - Can use Capability Maturity Model (see Fig.)
  - ISO 9001 is another approach to process quality



# More on the Iterations

---

- Too short iterations –
  - Not enough is done to get meaningful feedback
- Too long iterations –
  - Backtracking risks increase
  - Start losing the benefit of iterative development
- Just right iterations
  - 2-6 weeks is usually reasonable
  - Huge teams – up to 6 months
  - Example: Canadian air traffic control system
    - 150 programmers
    - 6 month iterations
    - ...but subteams can have shorter sub-iterations

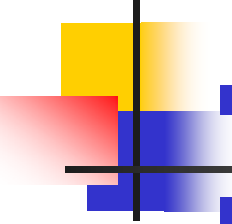
# Unified Process – Best Practices



---

- Use timeboxed iterations
  - ...i.e. scheduled iteration completion time
  - ...but this can vary for different iterations
- OO (is this intrinsic to incremental model?)
- Do the risky things in the early iterations
- Keep clients involved
- Settle basic architecture early
- Employ use cases and UML diagramming

# Grouping Iterations into Phases

- 
- See Figure 2.3 again
  - Phases impose some structure
  - Some iterations are different from others!
  - Phase 1: Inception
    - often only one iteration
    - Develop vision and project scope
    - Rapid prototyping (not reusable!) is good
    - Really, the inception phase is an admission:
      - Iterative development can be taken too far
      - (Recall earlier note on extremism)



## Phase 2: Elaboration

---

- Iteratively develop core of system
- Resolve riskier parts of project
  - If project not doable, find out soon not late
  - (Cf. the spiral life cycle model – see Fig.)
  - Resolve project scope and resource estimates
- Take-away lesson – iterations in the elaboration phase are *non-routine*



## Phase 3: Construction

---

- The iterations are more routine
- Each iteration results in a better program
- Hopefully, no major surprises occur
- (See Figure 2.3 again)



# Phase 4: Transition

---

- System is moved into being deployed
- Includes
  - Beta (final) testing
  - Bug fixes
  - Tuning
  - Training
  - Etc. – Larman devotes about 1/2 page max in entire book
- Appears to show limits of iterative concept

# Unified Process Disciplines

## Rational Unified Process: Overview

Artifacts

Examples

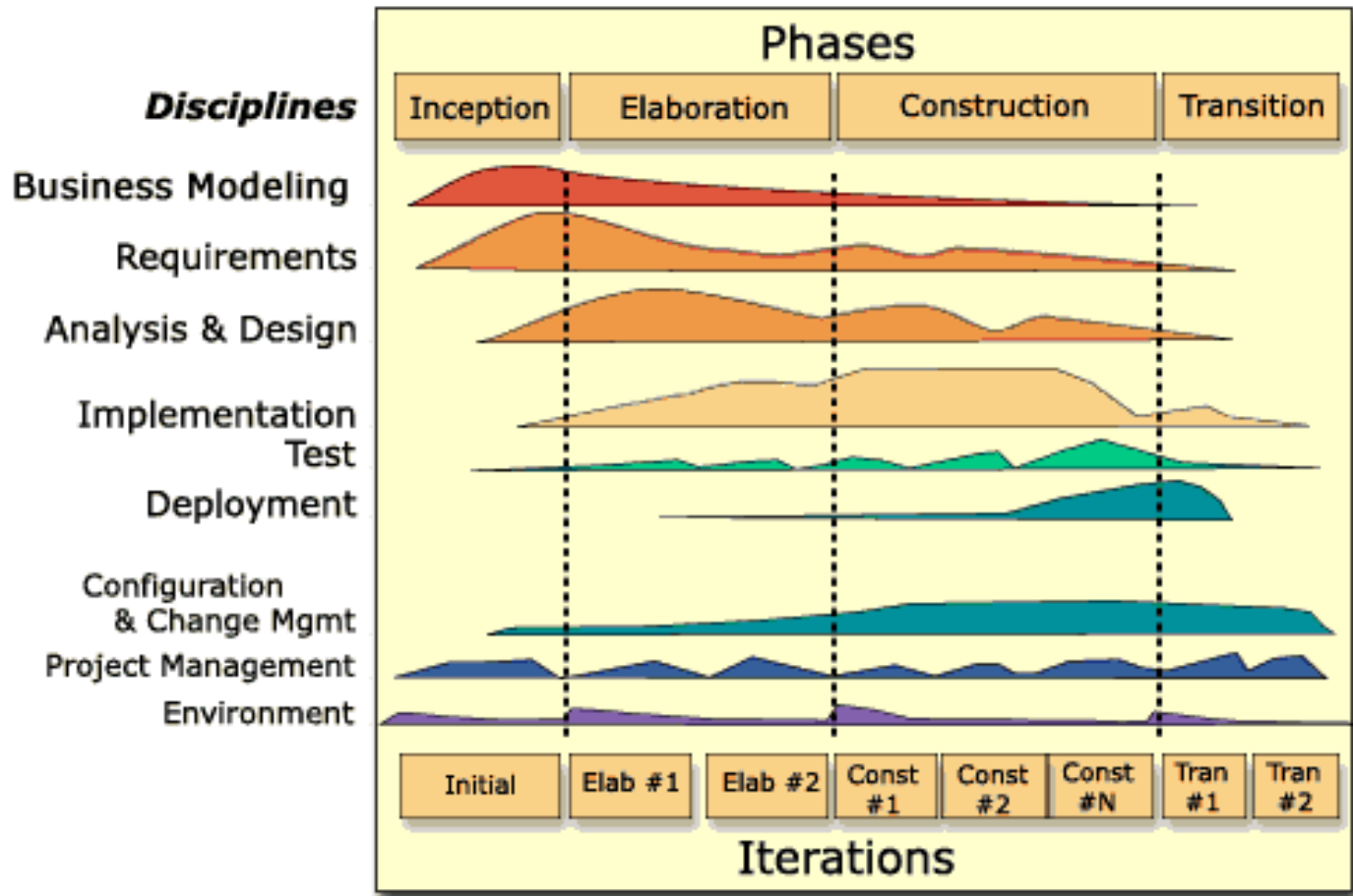
Roles

Roadmaps

Site Map

Oddly, this is slightly different from Figure 2.4...

(2<sup>nd</sup> – 4<sup>th</sup> are focus of book & course)





# Unified Process: More Comments

---

- Unified Process contains many, many activities and artifacts
  - Activity – something you *do*
  - Artifact – a work *product*
- Many of these are “optional”
  - ...use the ones appropriate to the project
- Larman’s medicine analogy –
  - UP is like a medicine cabinet – use what’s needed
  - (Same as the many options in UML)