Génie Logiciel et Gestion de Projets

Software Processes
Focus on Extreme Programming

Roadmap

- Process, Method, Methodology??
- What is a software process?
- Software Process Models
- Methodologies:
  - RUP
  - Focus on Extreme Programming

Process, method, methodology??

Process, method, methodology

- Process or method refers to
  - the particular tasks carried out and
  - how they are organised over the project life cycle
- Methodology refers to the general framework and principles used to organise the tasks
- Some authors use method to mean methodology
- Some authors use process for both

Methodology

- comprises
  - an approach to software engineering
  - a series of techniques and notations that support the approach
  - a lifecycle model to structure the development process
  - a unifying set of procedures and philosophy.

Software Process???

Software Process

- Process = Set of activities that lead to a software product
  - lots of processes exist
  - share some fundamental activities

- Product = What is delivered to the customer:
  - Requirements specification,
  - system,
  - all documentation, manuals etc.

Software Process Activities

Requirements collection	Establish customer's needs
Analysis	Model and specify the requirements ("what")
Design	Model and specify a solution ("how")
Implementation	Construct a solution in software
Testing	Validate the solution against the requirements
Maintenance	Repair defects and adapt the solution to new requirements

Software Process Models

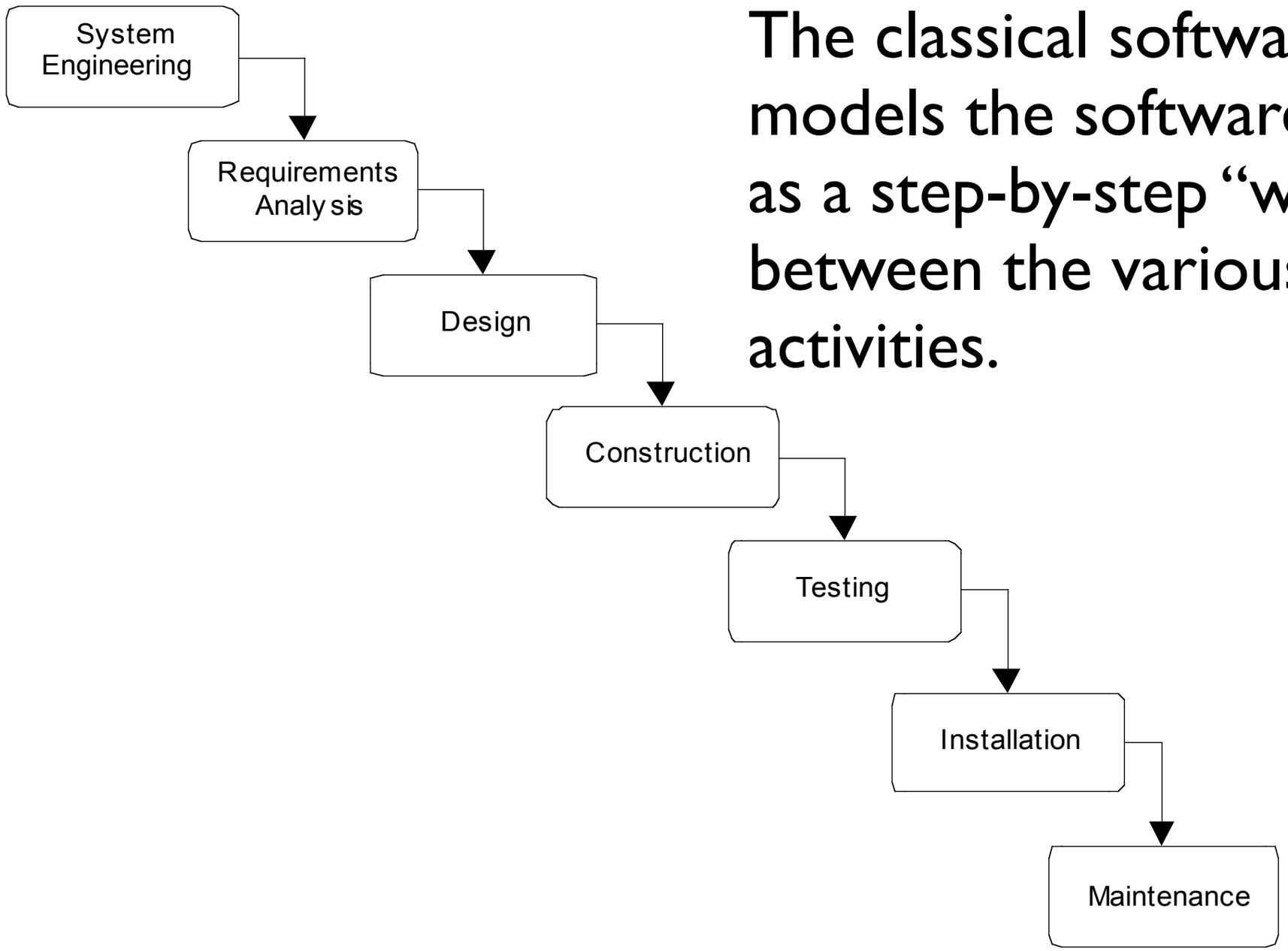
Five of them

Software Process Models

Abstract representation of a software process

- Waterfall model
- Prototyping
- Iterative and incremental development
- Boehm's spiral model
- Agile Development

Waterfall Development



The classical software lifecycle models the software development as a step-by-step "waterfall" between the various development activities.

Waterfall Model

- Sequential ordering of phases
- After each phase: reports are 'signed of' by partners
  - heavyweight process
    - next phase cannot start without ending the previous one
      in practice: certain overlap between phases
- starts with a Software Requirements Document
  - almost impossible to change

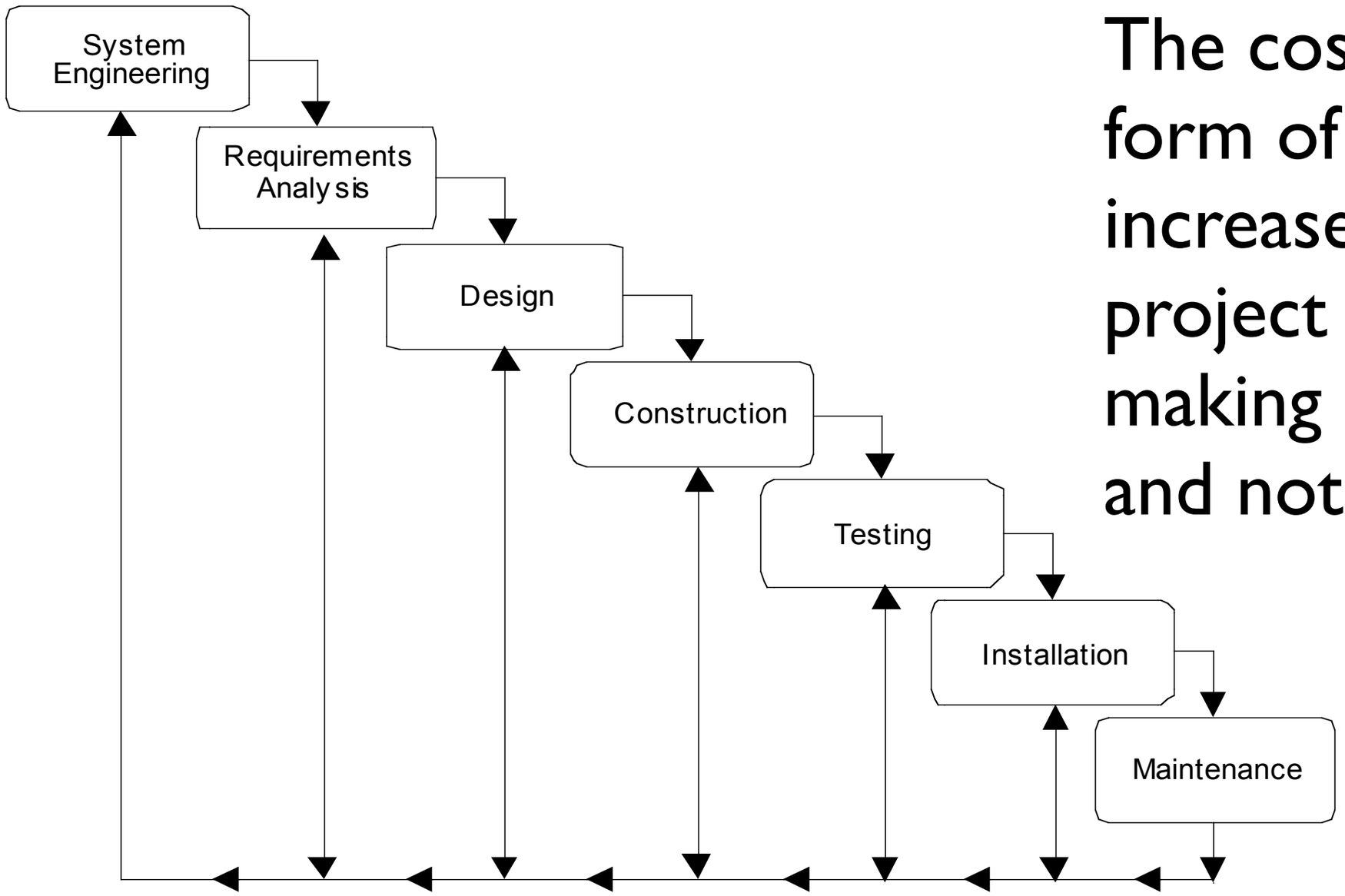
Advantages

- The waterfall model is popular for upper management, because
  - *visible*: it is easy to control project progress.
- documentation produced after each phase,
- sequential planning.

Disadvantages

- The waterfall model is unrealistic for large projects, because:
  - *Complete*: a customer cannot state all requirements explicitly.
  - *Idealistic*: in real projects iteration occurs (but tools and organization obstruct).
  - *Time*: A working version of the system is only available late in the project.
  - *Change*: it is very difficult and costly to adapt to changes in the requirements.

Waterfall process with backflow



The cost of this form of iteration increases as the project progresses making it impractical and not effective.

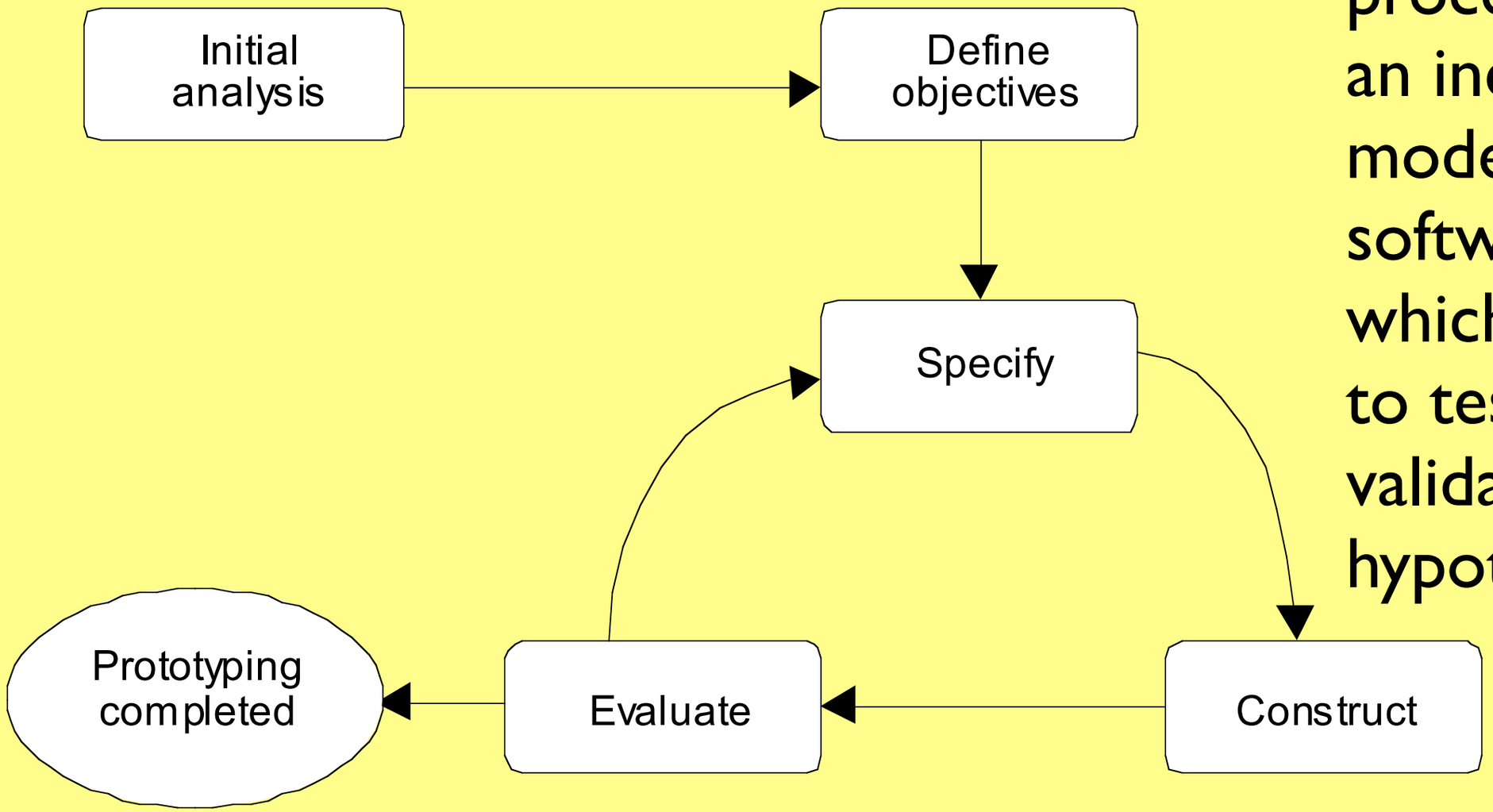
Challenges

- Writing good and clear requirements is hard
- And ... they change all the time!
  - initial set was not complete
  - business changes during project
  - technology changes during project
  - gain understanding during later phases
  - ....
- Evolutionary Development takes this into account

Note: Delivery versus Process

- Delivery = giving the product to the customer
- Can be:
  - either when it is completely finished
  - or incrementally: delivered in several pieces
- The process itself can be sequential or iterative
  - sequential: every phase once, following another one
  - iterative: possibly several loops of phases

Prototyping



Prototyping is the process of creating an incomplete model of the future software program which can be used to test, explore or validate a hypothesis.

Throwaway Prototyping

- A throwaway prototype is intended to validate requirements or explore design choices.
  - quickly
  - finally discarded

- Examples
  - UI prototype - validate user requirements
  - rapid prototype - validate functional requirements
  - experimental prototype - validated technical feasibility

Evolutionary Prototyping

- An evolutionary prototype is intended to evolve in steps into a finished product.
  - grow, don't build [Broo87a]: "grow" the system redesigning and refactoring along the way

Advantages

- Early demonstrations of system functionality help identify any misunderstandings between developer and client
- Client requirements that have been missed are identified
- Difficulties in the interface can be identified
- The feasibility and usefulness of the system can be tested, even though, by its very nature, the prototype is incomplete.

Disadvantages

- The client may perceive the prototype as part of the final system
- The focus on a limited prototype can distract developers from properly analyzing the complete project.
- Prototyping requires significant user involvement
- Excessive development time of the prototype

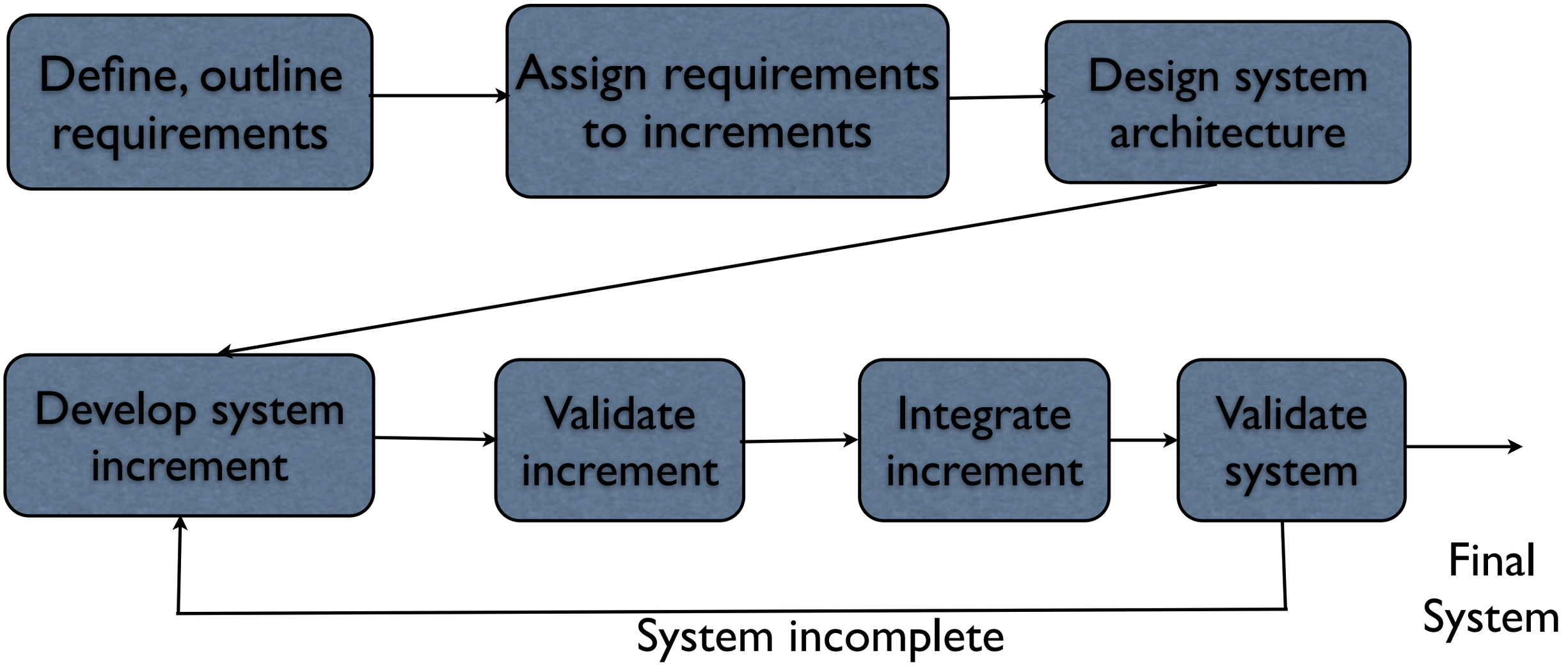
Iterative Approaches

- Incremental Delivery
- Boehm's Spiral Model
  phases spiral outwards, within space some process is chosen to best address risk
- Agile Development
  delay requirements and design decisions until they make sense

Incremental Delivery

- Customers identify services for system
- classify according to importance
- define delivery increments (part of functionality)
- repeat increment:
  - detailed requirements + development
  - maybe specify requirements for next increments
  - deliver to customer

Incremental Development



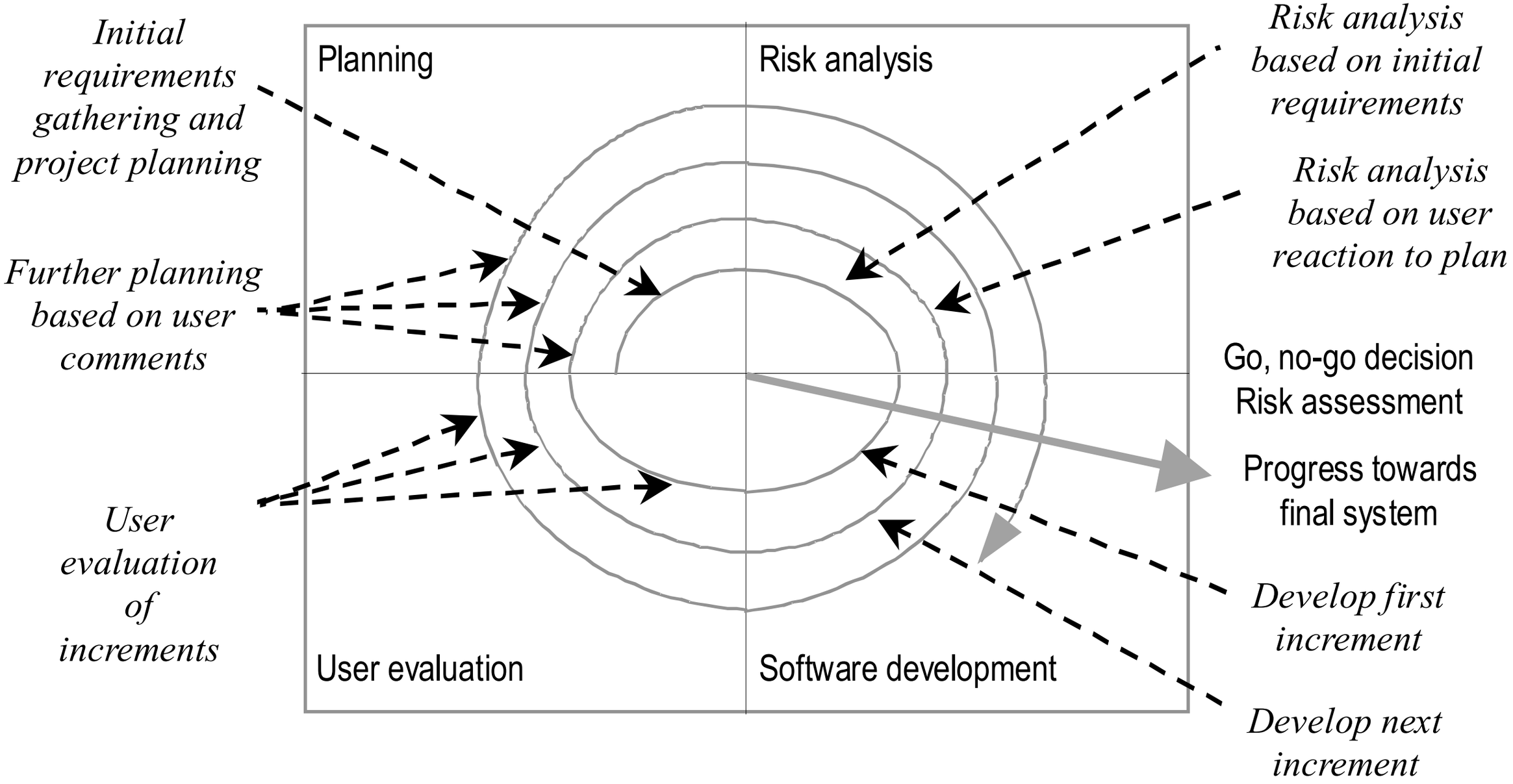
Advantages

- Customers get partial, functional pieces of the application
- Customers can use the early increments to refine requirements for following increments
- Less risk of overall failure
- Because highest priority services delivered first, they automatically receive more testing

Disadvantages

- It can be difficult to map requirements to increment of right size
  - because
    - increments should be relatively small
      - each increment should deliver something of value to client
- What basic facilities that can be reused in different parts of the system are necessary?
  - not known, since not all requirements are available

Spiral Model



Boehm's Spiral model

- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents one phase in the process
- Each loop is split into 4 sectors:
  - planning: define objectives and risks
  - risk analysis: analyze and reduce risks
  - engineering: development and validation
  - evaluation and planning: project is reviewed and next phase of the spiral is planned

Agile Development

Manifesto for Agile Software Development

We are uncovering better ways of developing software
by doing and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

The Manifesto for Agile Software Development

Advantages

- Customer is involved
  - Increments are of value to customer
  - discrepancies are discovered early-on
- Problem of changes in requirements are tackled by the process

Disadvantages

- Customer needs to agree to participate actively
  - difficult in practice:
    - managers are used to sequential approaches
    - needs to involve people
- Writing contracts is difficult
  - customer pays for time, not requirements
- Quality of code can more easily degrade
- Less documentation (system changes quickly)

Agile approaches

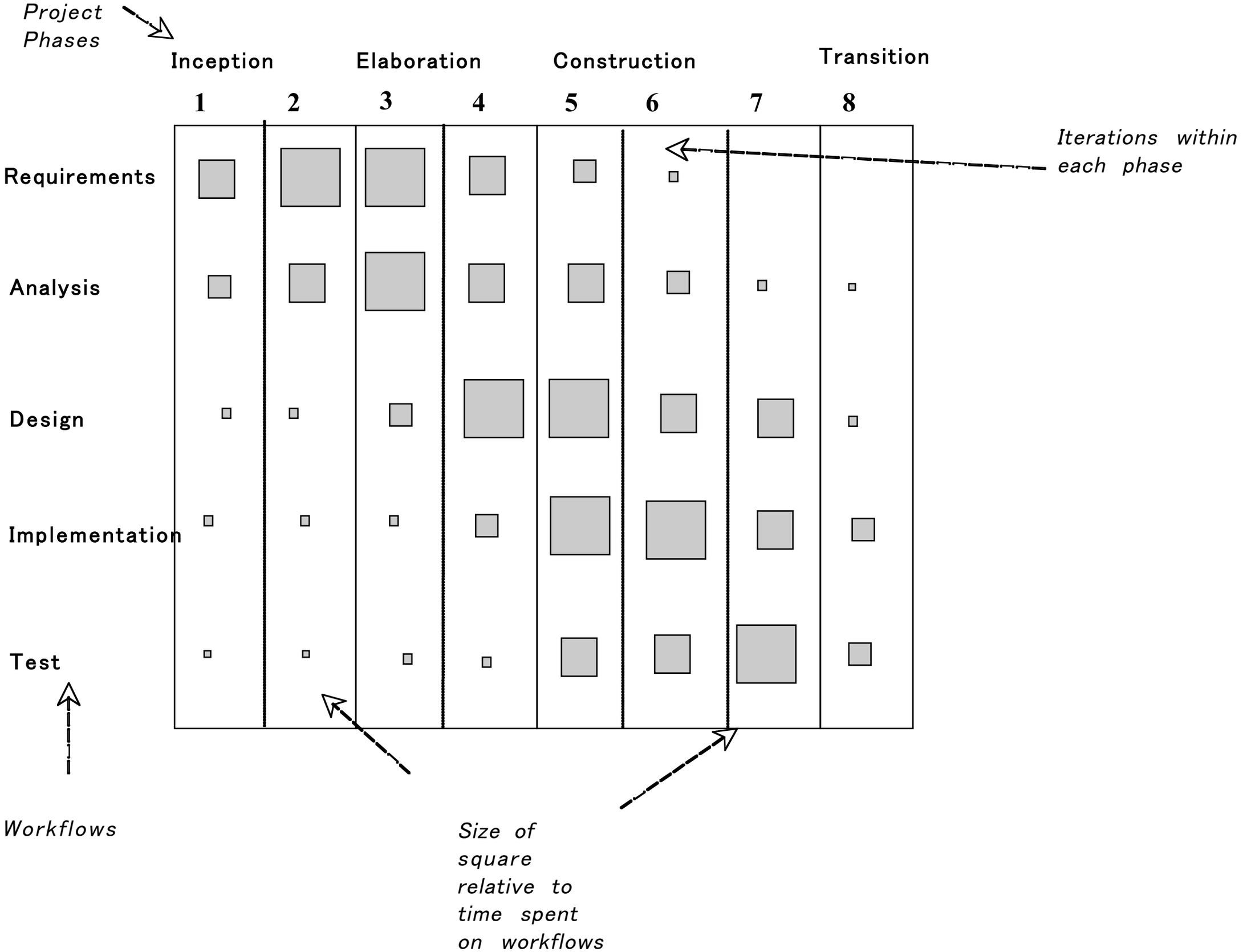
- Became quite popular recently
- Examples
  - extreme programming (2000)
  - Scrum (2001)
  - Crystal (2001)
  - Feature Driven Development (2002)
  - ...
- We'll look at extreme programming

Unified Process

Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009

The Unified Process

- A process model derived from the work on the UML and associated processes.
- The different RUP phases:
  - Inception: determine scope and purpose of the project,
  - Elaboration: requirements capture and determine structure of the system,
  - Construction: build the software,
  - Transition: deals with product installation and rollout.


Project Phases
Inception
Elaboration
Construction
Transition
1
2
3
4
5
6
7
8
Iterations within each phase
Requirements
Analysis
Design
Implementation
Test
Workflows
Size of square relative to time spent on workflows

Extreme Programming

Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009

Extreme Programming (XP)

- Point of XP: coping with change and uncertainty
- Based on number of practices:
  - small, frequent releases of the system,
  - full-time engagement of customer,
  - pair programming, collective ownership of the code, sustainable development,
  - regular system releases, test-first development, continuous integration,
  - constant refactoring, simplest thing that can work.

Driving Metaphor

Driving a car is not about pointing the car in one direction and holding to it; driving is about making lots of little course corrections.

"Do the simplest thing that could possibly work"

Customer-Developer Relationships

- *A well-known experience*: The customer and the developer sit in a boat in the ocean and *are afraid of each other*

Customer fears	Developer fears
They won't get what they asked for	They won't be given clear definitions of what needs to be done
They must surrender the control of their careers to techies who don't care	They will be given responsibility without authority
They'll pay too much for too little	They will be told to do things that don't make sense
They won't know what is going on (the plans they see will be fairy tales)	They'll have to sacrifice quality for deadlines

- Result: a lot of energy goes into protective measures and politics instead of success

The Customer Bill of Rights

You have the right to an overall plan	To steer a project, you need to know what can be accomplished within time and budget
You have the right to get the most possible value out of every programming week	The most valuable things are worked on first.
You have the right to see progress in a running system.	Only a running system can give exact information about project state
You have the right to change your mind, to substitute functionality and to change priorities without exorbitant costs.	Market and business requirements change. We have to allow change.
You have the right to be informed about schedule changes, in time to choose how to reduce the scope to restore the original date.	XP works to be sure everyone knows just what is really happening.

The Developer Bill of Rights

You have the right to know what is needed, with clear declarations of priority.	Tight communication with the customer. Customer directs by value.
You have the right to produce quality work all the time.	Unit Tests and Refactoring help to keep the code clean
You have the right to ask for and receive help from peers, managers, and customers	No one can ever refuse help to a team member
You have the right to make and update your own estimates.	Programmers know best how long it is going to take them
You have the right to accept your responsibilities instead having them assigned to you	We work most effectively when we have accepted our responsibilities instead of having them thrust upon us

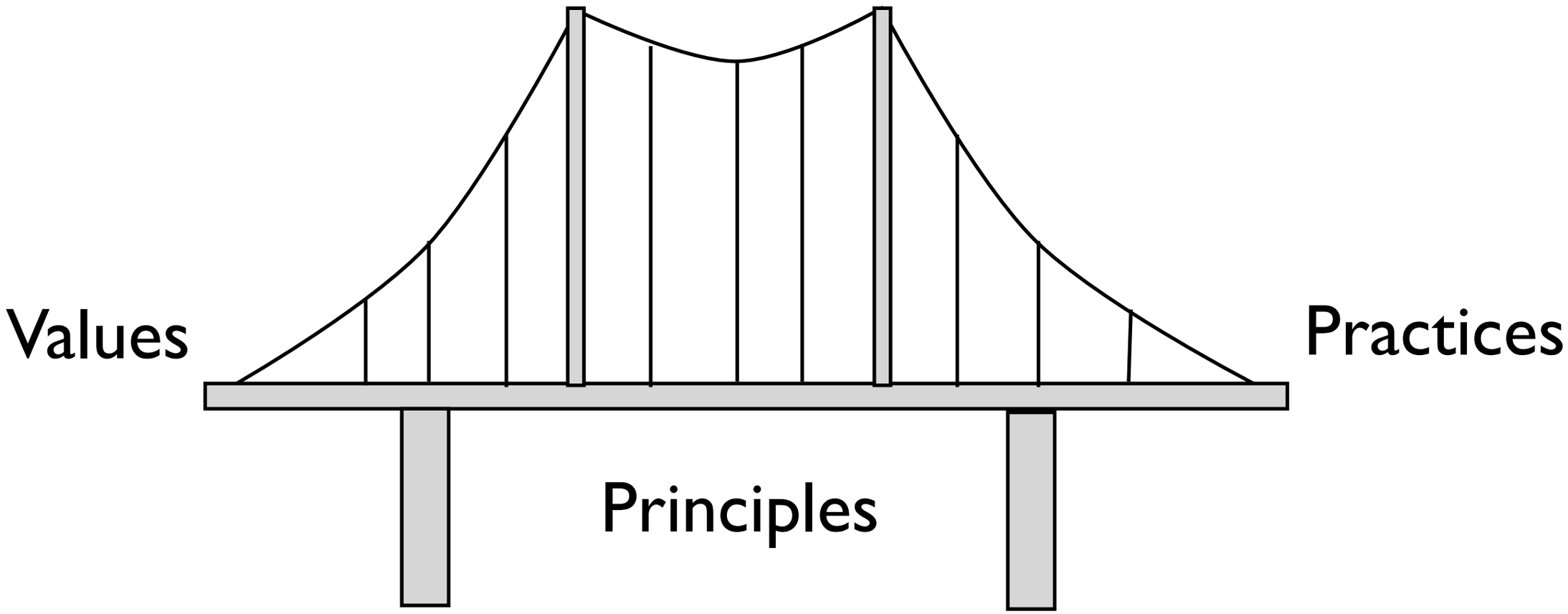
Separation of Roles

- Customer makes business decisions
- Developers make technical decisions

Business Decisions	**Technical Decisions**
Scope	Estimates
Dates of the releases	Dates within an iteration
Priority	Team velocity
	Warnings about technical risks

- The Customer owns “what you get” while the developers own “what it costs”.

Describing XP



Basic XP Values

- Communication
  - communicate problems&solutions, teamwork
- Simplicity
  - eliminate wasted complexity
- Feedback
  - change creates the need for feedback
- Courage
  - effective action in the face of fear
- Respect
  - care about you, the team, and the project

Principles

Humanity, Economics, Mutual Benefit, Self-Similarity, Improvement, Diversity, Reflection, Flow, Opportunity, Redundancy, Failure, Quality, Baby Steps, Accepted Responsibility

Will not detail them -- they govern what the practices tend to accomplish

So, on to the practices!

Primary Practices

Sit Together

Whole Team

Informative Workspace

Energized Work

Pair Programming

Stories

Weekly Cycle

Quarterly Cycle

Slack

Ten Minute Build

Continuous Integration

Test-First Programming

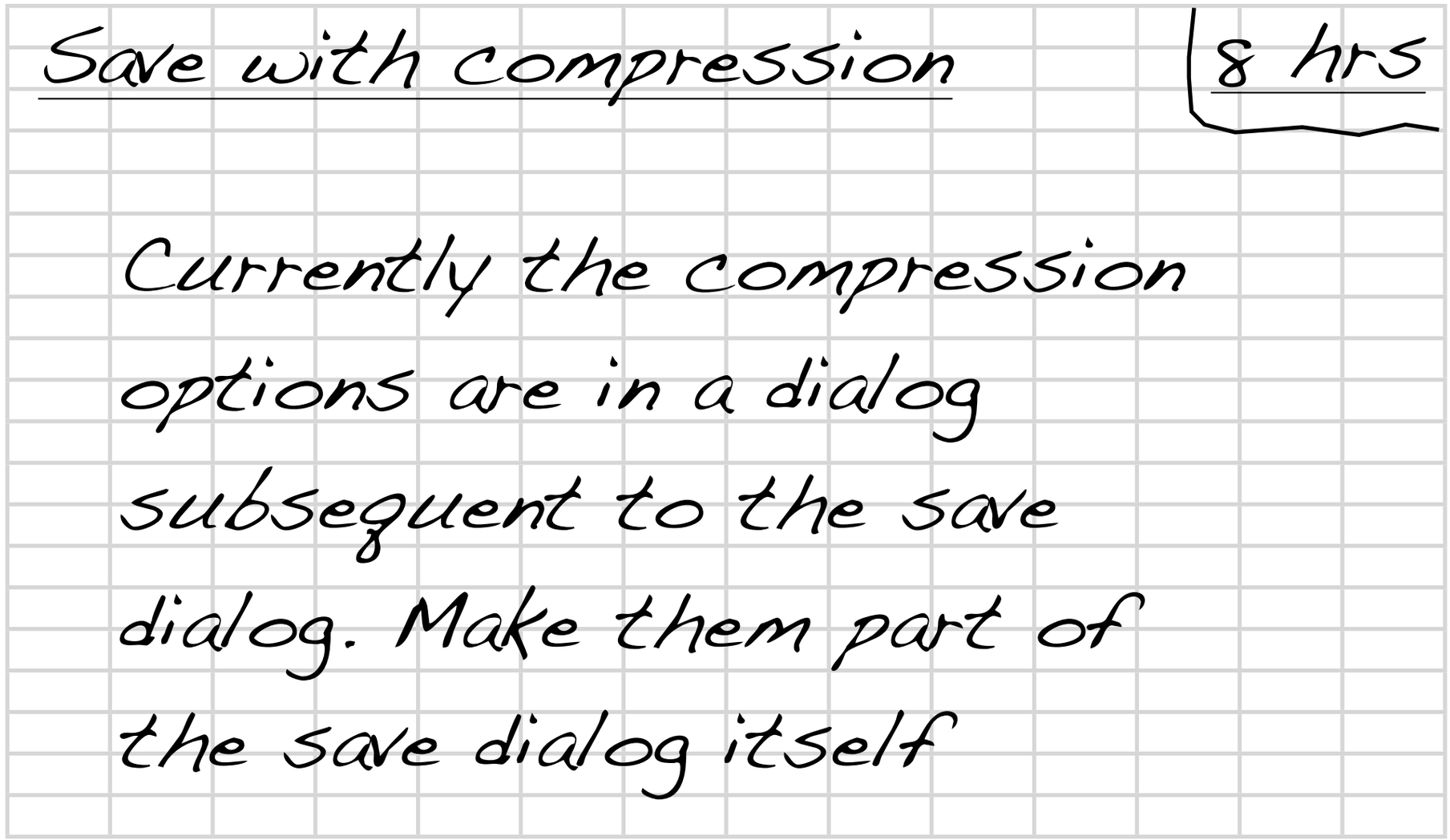
Incremental Design

Stories

plan using units of customer-visible functionality

name

estimate



index card

short description

Another example

173. Students can purchase parking passes.

Priority: 8
Estimate: 4

7 more User Stories

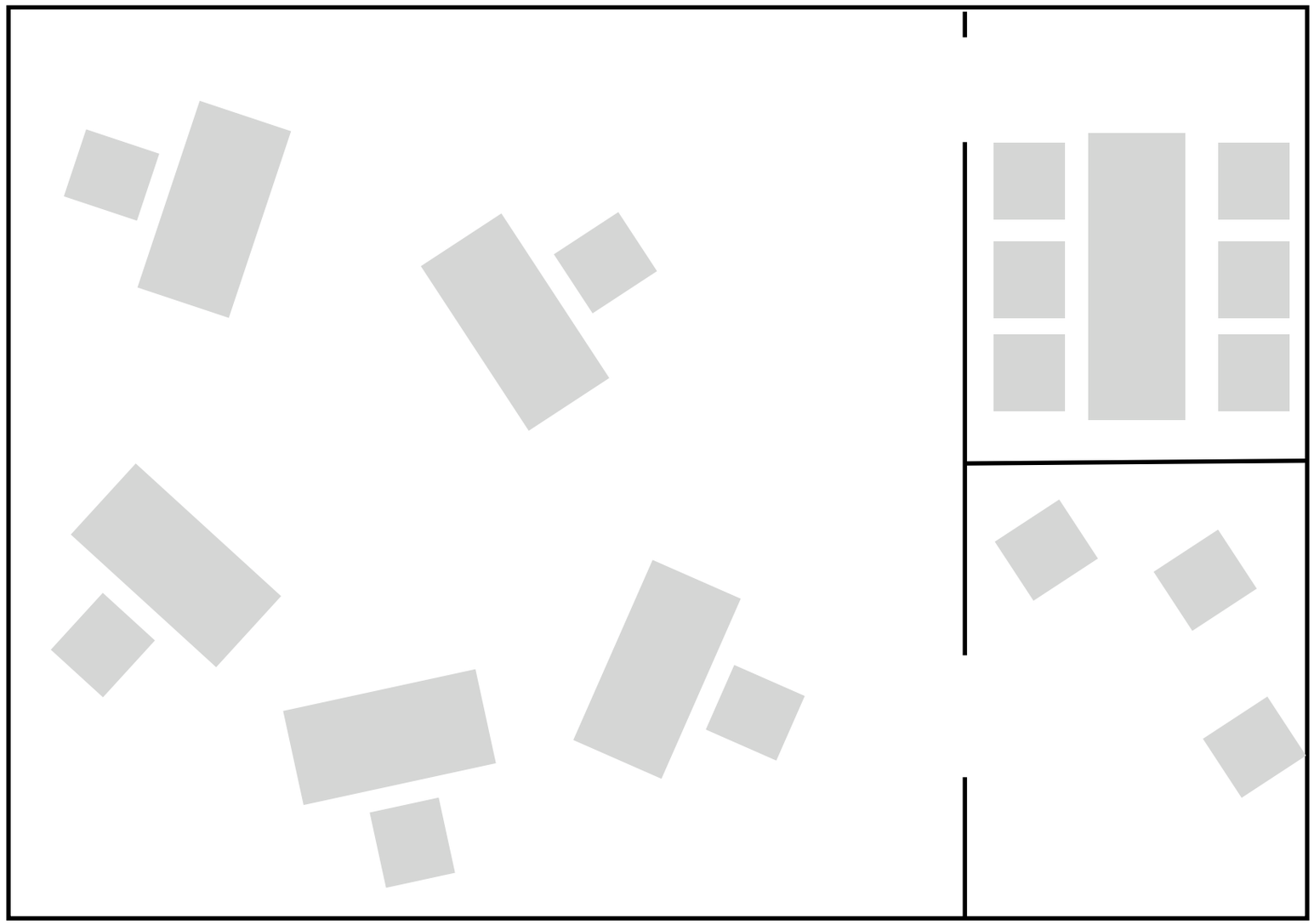
- Students can purchase monthly parking passes online.
- Parking passes can be paid via credit cards.
- Parking passes can be paid via PayPal ™.
- Professors can input student marks.
- Students can obtain their current seminar schedule.
- Students can only enroll in seminars for which they have prerequisites.
- Transcripts will be available online via a standard browser.

User Stories vs. Use Case

- Not the same artifact!

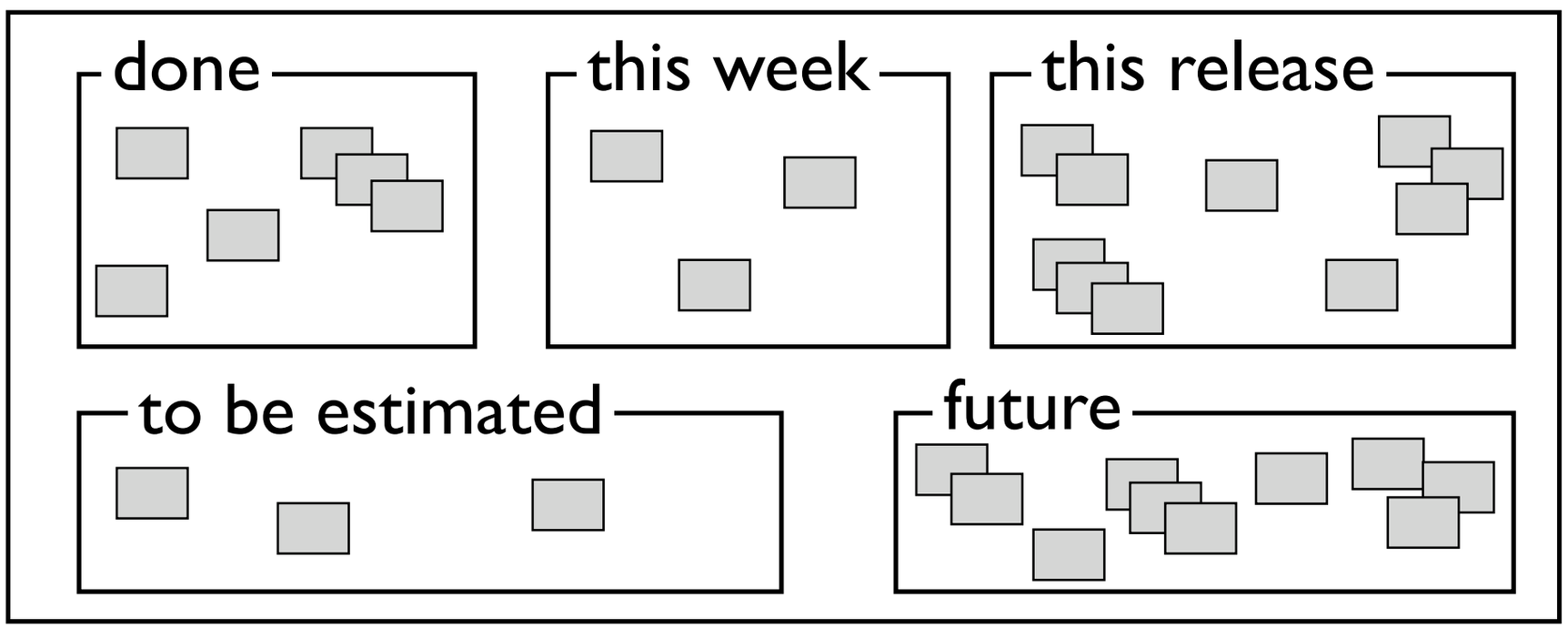
Sit Together

Develop in an open space big enough for the team



Informative Workspace

- Workspace = about your work
  - 15 seconds to convey how project is going
  - shows important, active information
  - drinks & snacks available, and clean



Pair Programming

- Write all production programs with two people sitting at one machine
  - make enough room, move keyboard and mouse
- Pair programmers:
  - keep each other on task
  - brainstorm refinements to the system
  - clarify ideas
  - take initiative when partner is stuck (less frustration)
  - hold each other accountable to practices

Pair programming and privacy

- Sometimes you might need some privacy
  - then go work alone
    - come back with the idea (NOT the code)
      - quickly reimplemented with two
      - benefits the whole team, not you alone

Pair Programming

- Rotate pairs frequently
  - every couple of hours, at natural breaks in development
  - with a timer, every 60 minutes (or 30 minutes for difficult problems)

Pair Programming and Personal Space

- Not everybody likes to sit close!
- Observe personal hygiene and health
- Sexual feelings are not in best interest of the team
  - even when mutual
- When uncomfortable pairing with somebody, talk about it with someone safe
  - chances are that you are not the only one
  - everybody needs to feel comfortable

Weekly Cycle

- Plan work one week at a time.
- Do this on a meeting at the begin of each week

  1. Review progress.
  2. Let customers pick a week's worth of stories to implement this week.
  3. Break the stories into tasks. Team members sign up for tasks and estimate them.

- Start writing tests that will run when the stories are completed

Ten-Minute Build

- Automatically build the whole system and run all of the tests in ten minutes
  - longer: will not be used (and errors result)
  - shorter: not enough time to drink coffee
- Note: if it takes longer than 10 minutes:
  - maybe only rebuild changed part or test changes
  - But: introduces errors. Only do this when necessary
- Lowers stress: "Did we make a mistake? Let's see."

Continuous Integration

- Team Programming = Divide, Conquer, *Integrate*
- Integrate and test changes after no more than a couple of hours
  - integration typically takes long
  - when done at the end, risks the whole project when integration problems are discovered
  - the longer you wait, the more it costs and the more unpredictable it becomes

Using Continuous Integration

- Synchronous
  - After a task is finished, you integrate and run the tests
  - Immediate feedback for you and your partner
- Asynchronous
  - After submitting changes, the build system notices something new, builds and tests the system, and gives feedback by mail, notification, etc.
  - Feedback typically comes when a new task is started
  - Pair programmers might have been switched already

Test-first Programming

- Write a failing automated test before changing code
- Addresses many problems:
  - Scope creep: focus coding by what the code should do, not on the "just in case" code
  - Coupling and cohesion: If it's hard to write a test, there is a design problem (not a testing problem)
  - Trust: clean working code + automated tests
  - Rhythm: gives focus on what to do next
    - efficient rhythm: test, code, refactor, test, ...

Incremental Design

- Invest in the design of the system every day. Strive to the design of the system an excellent fit for the needs of the system that day
  - Completely opposite to lots of other practices
    - Waterfall and similar approaches
- Can work with XP because of the other practices
  - Automated tests, continuous integration, ...
- Note: you *need* to invest in design!
  - not just implement story after story after story...

Corollary Practices

Real Customer Involvement

Incremental Deployment

Team Continuity

Shrinking teams

Root-Cause Analysis

Shared Code

Code and Tests

Single Code Base

Daily Deployment

Negotiated Scope Contract

Pay-Per-Use

Stages in XP Project

- Initiation
  - User Stories
- Release Planning
- Release (each Release is typically 1 -6 months)
  - Iteration 1 (typically 1 -3 weeks)
  - Iteration 2

    :
  - Iteration n

- User stories capture requirements
  - replaces use cases
  - made by the customer
  - used to create time estimates for release planning

- User stories are used to create acceptance tests
  - verify proper implementation of story

Design and XP

- XP says: "Design is daily activity", and "Use the simplest solution that works"
  - incremental design

When to design in XP

- Just enough to get feedback for next iteration
  - deferred until decision can be used immediately
- Therefore
  - deploy software sooner
  - make decisions with certainty
  - avoid living with bad decisions
  - maintain pace of development

Summary

- Development phases can be structured according to software process models
- Two big groups: specification based vs. iterative
  - Waterfall (specification based): useful for big projects in known/stable domains
  - Iterative approaches can support changing requirements
    - looked at extreme programming in detail

References

- Ian Sommerville. Software Engineering. Eighth Edition. 2007. ISBN: 9780321313799
- eXtreme Programming Explained: Embrace Change -- Second edition. Kent Beck. Addison-Wesley Pub Co; ISBN: 0-321-27865-8;
- Extreme Programming, précis et concis. Traduction de Michel Casabianca. O'Reilly, 2005.