Theme I
Software Processes

Focus on Extreme Programming

Learning Goals

- How did agile methods come about?
- What are the principles of agility?
- How are agile processes carried out?
- Can agile processes be combined with non-agile ones?

- In conclusion => understanding of the main ideas of agile development methods and XP in particular.

Table of content

- Definition Software Process
- Waterfall
- Iterative and incremental development
- Agile development
  - principles
  - cycle
  - integration with non-agile processes

Software Process

- Software project composed of activities or steps
  - Requirements, design, implementation, testing, deployment, maintenance.
- Activities organized into phases
- A software process:
  - prescribes the order and frequency of phases
  - specifies criteria for moving from one phase to the next
  - defines the deliverables of the project

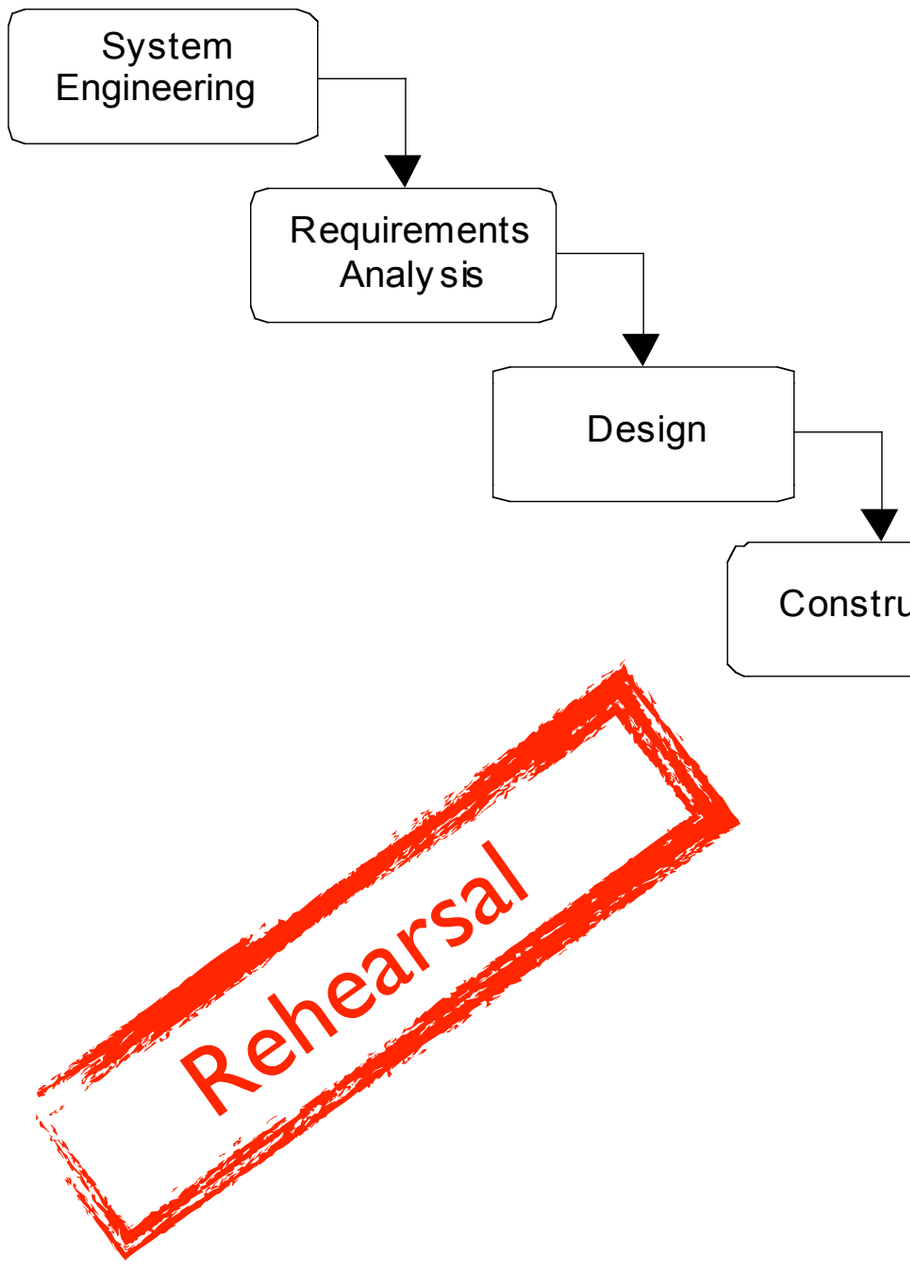
  - Consider your last project or programming exercise. How were the activities/steps defined, executed and controlled?

Think/pair/share

Enumerate the software processes you know, define them using one sentence and indicate the ones you have used in the past.

Énumérez les processus de logiciel que vous connaissez, définissez les processus en une phrase et indiquez les processus que vous avez déjà utilisé.

Waterfall Development



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

Iterative and Incremental

- **Iterative**
  - repeated execution of the waterfall phases, in whole or in part, resulting in a refinement of the requirements, design and implementation
- **Incremental**
  - operational code produced at the end of an iteration
  - supports a subset of the final product functionality and features
- Artifacts evolve during each phase
- Artifacts considered complete only when software is released

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

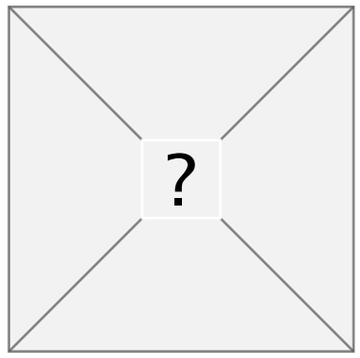
Agile Principles

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Agile Principles

- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Cycle



[Agile Methods]

Obtain **high-level requirements**

Obtain **requirements** for next period's* segment

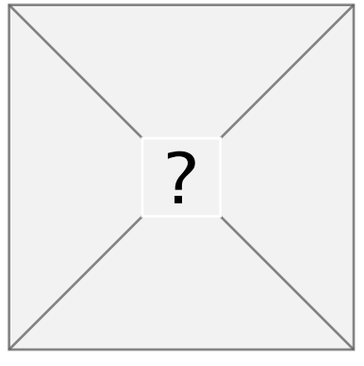
Refactor to clean up

Refactor to accommodate new requirements

Modify **code** and **test** code base to handle additional requirements

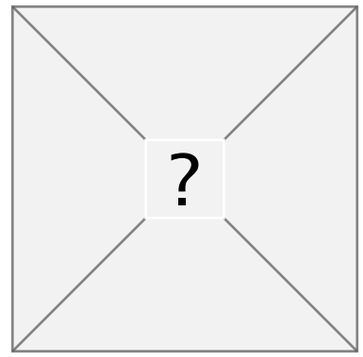
* Typically 1-6 weeks

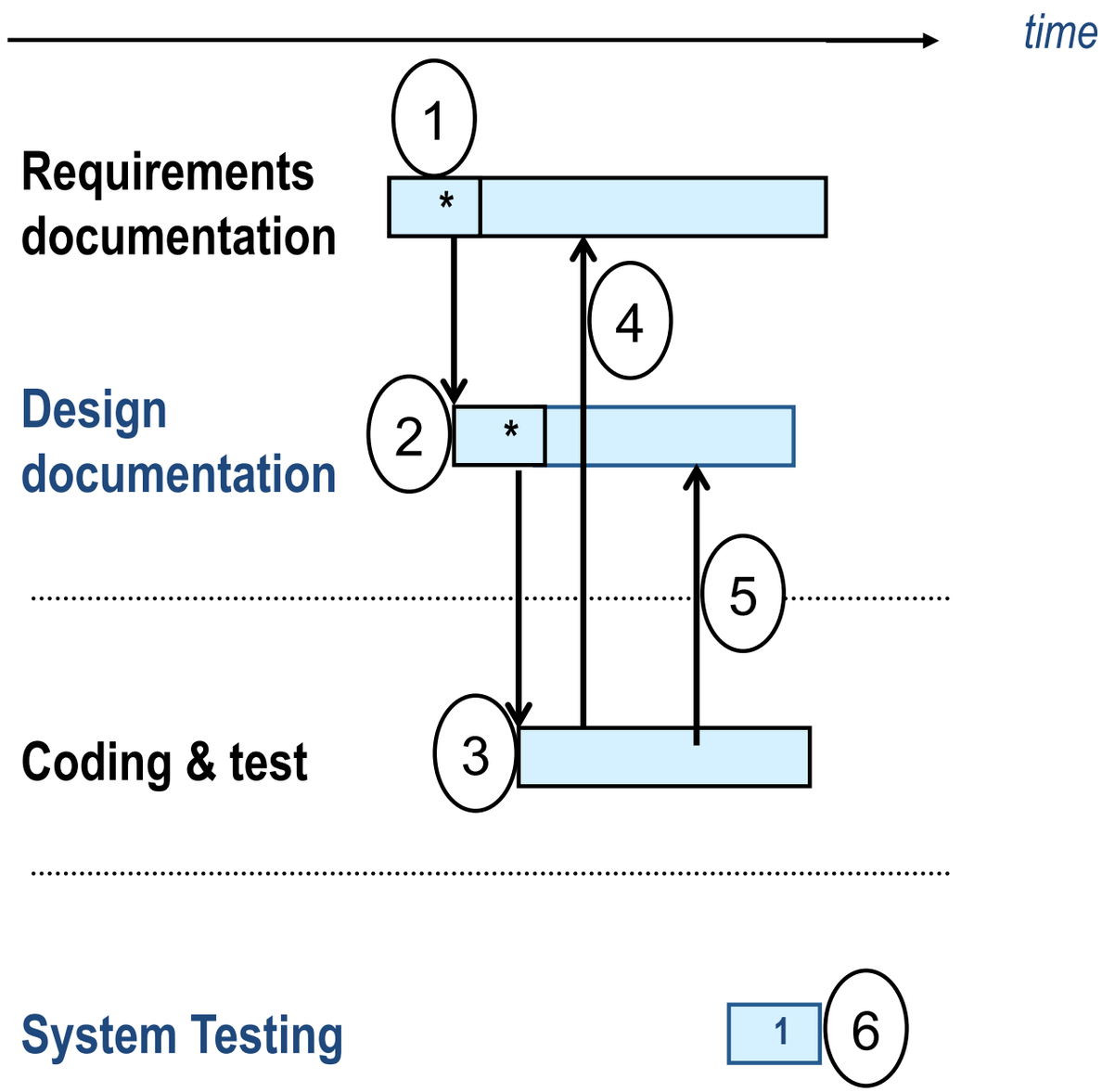
Integrating Agile with Non-Agile Processes



- Regardless of development process used:
  - need to make trade-offs in deciding how extensively to pursue a phase before moving to another phase.
  - E.g. how much effort to spend on planning a software enterprise.

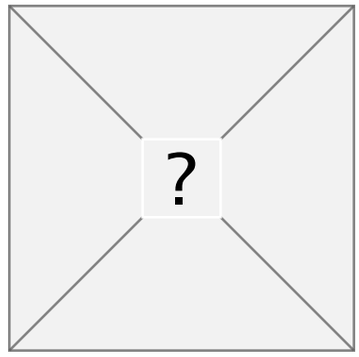
Integrating Agile with Non-Agile Methods: Non-Agile-driven





* High level

Integrating Agile with Non-Agile Methods: Agile-driven



Initial agile development

Requirements documentation

Design documentation

Coding & test (including agility?)

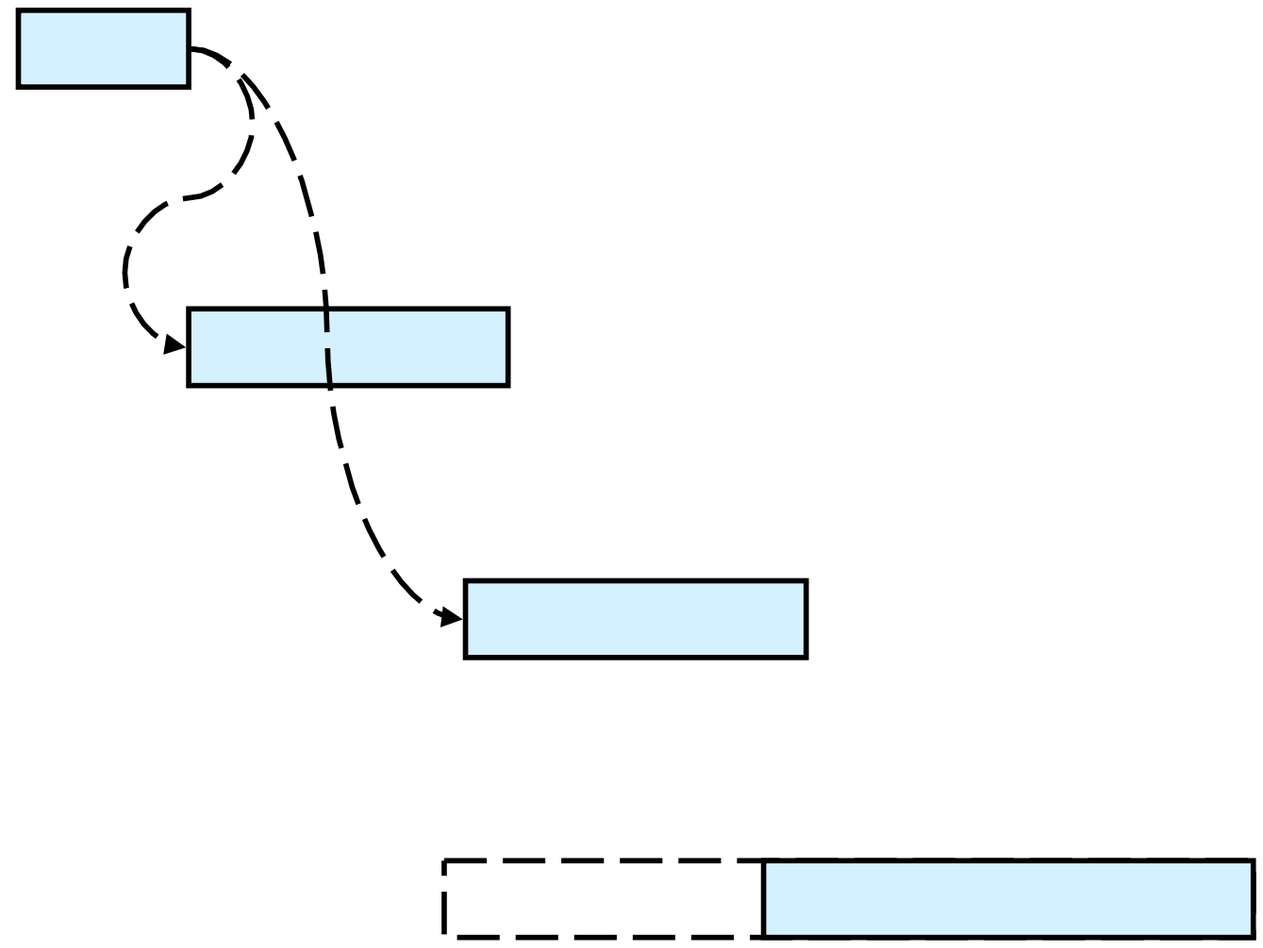
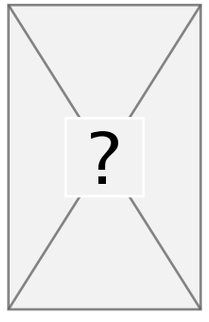


Table of content

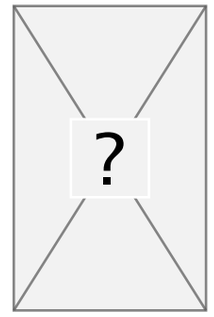
- Definition eXtreme Programming
- Basic values
- Principles
- Practices
  - pair programming
  - stories
  - test-first programming
  - continuous integration

Extreme Programming (XP)



- Point of XP: coping with change and uncertainty
- **software product = f(c,t,s,q)**
- parameters:
  - cost ≈number of staff on the project
  - time ≈ time to delivery deadline
  - scope ≈ amount of functionality
  - quality ≈ reliability

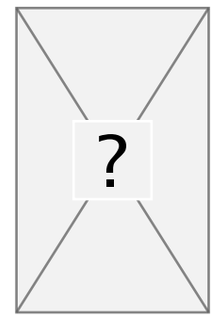
Extreme Programming



- Software Product = f(c, t, s, q)

- **Question:**
  Which parameters are fixed by the management?
- **Question:**
  Which parameter is left for developers to tweak in order to adjust the workload?

Extreme Programming



- Software Product = f(c, t, s, q)

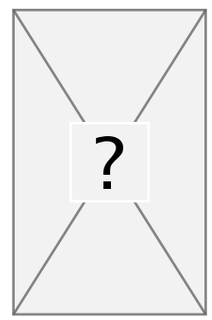
- **Question:**
  Which parameters are fixed by the management?

  Cost, Time, Scope

- **Question:**
  Which parameter is left for developers to tweak in order to adjust the workload?

  Quality

Extreme Programming



- The Agile/XP answer:

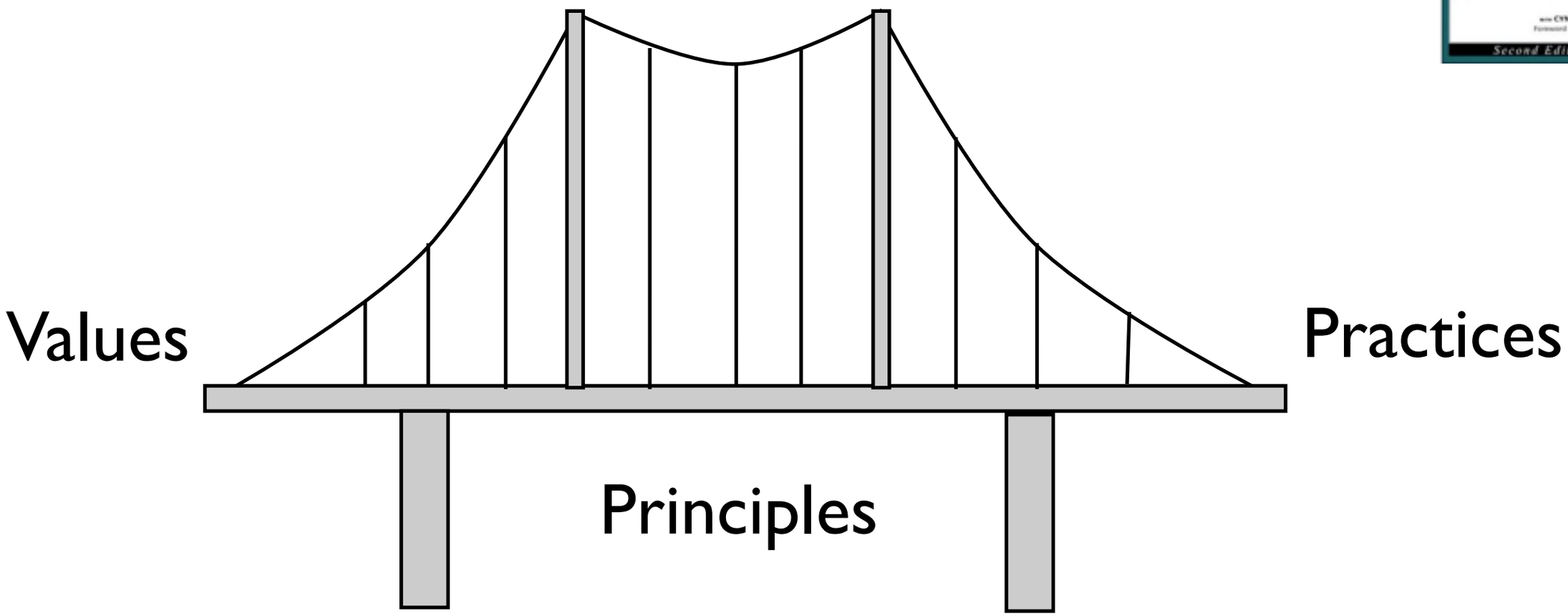
Projects fix [price, time, quality]

Open parameter [scope]

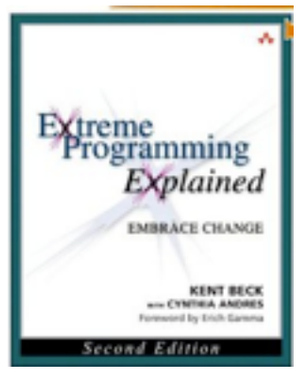
Meaning:

- Scope control important
  - On site customer / user feedback
  - Rapid release cycles
- Quality control important
  - Testing & user feedback
  - Test-driven development

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

Energized Work

Pair Programming

Stories

Automated Testing

Small Releases

Weekly Cycle

Quarterly Cycle

Ten Minute Build

Continuous Integration

Incremental Design

Collective Code Ownership

Coding Standards

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

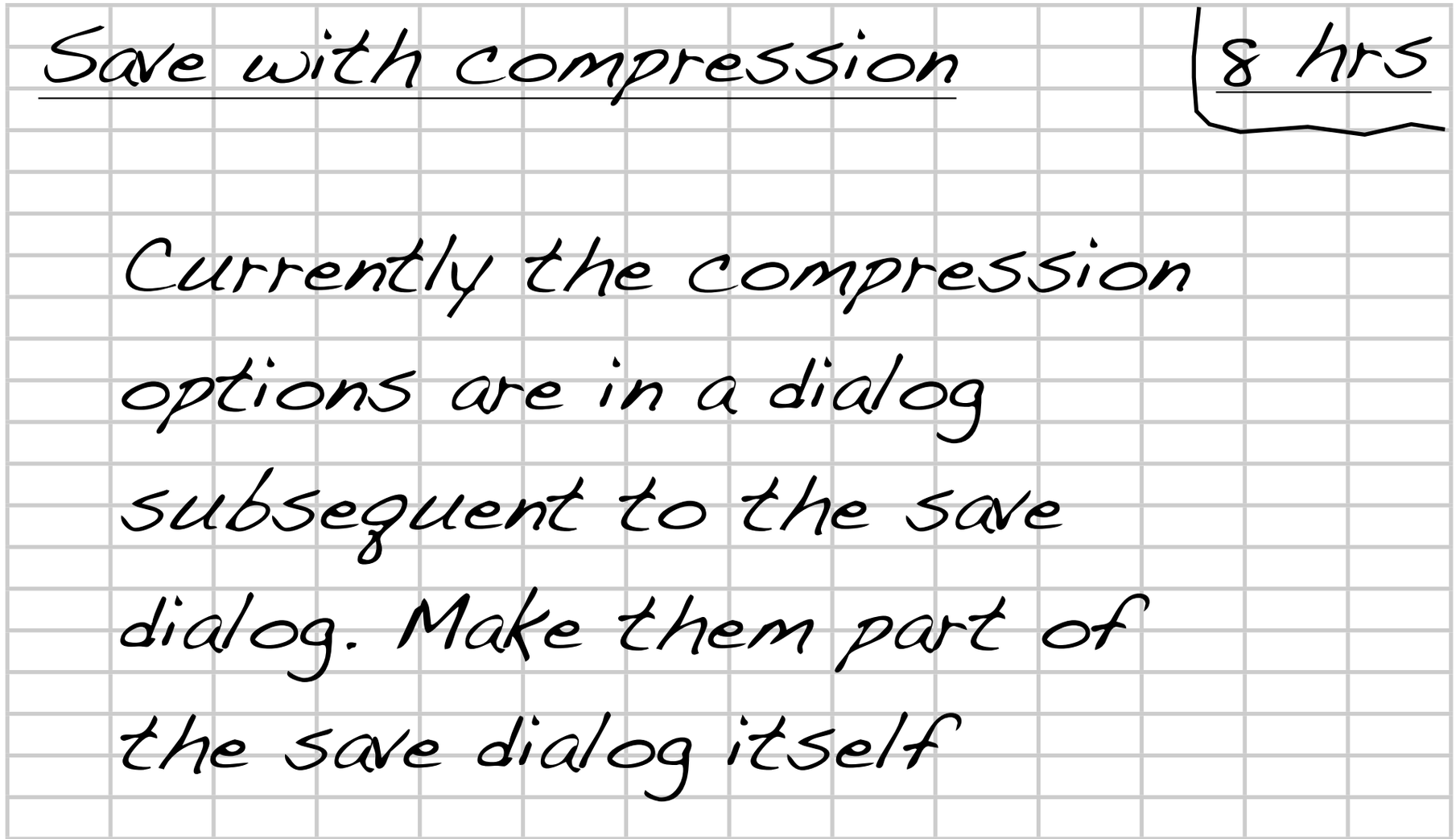
- Sometimes you might need some privacy
  - then go work alone
    - come back with the idea (NOT the code)
      - quickly reimplemented with two
      - benefits the whole team, not you alone
- Rotate pairs frequently
  - every couple of hours, at natural breaks in development
  - with a timer, every 60 minutes (or 30 minutes for difficult problems)

Stories

plan using units of customer-visible functionality

name

estimate



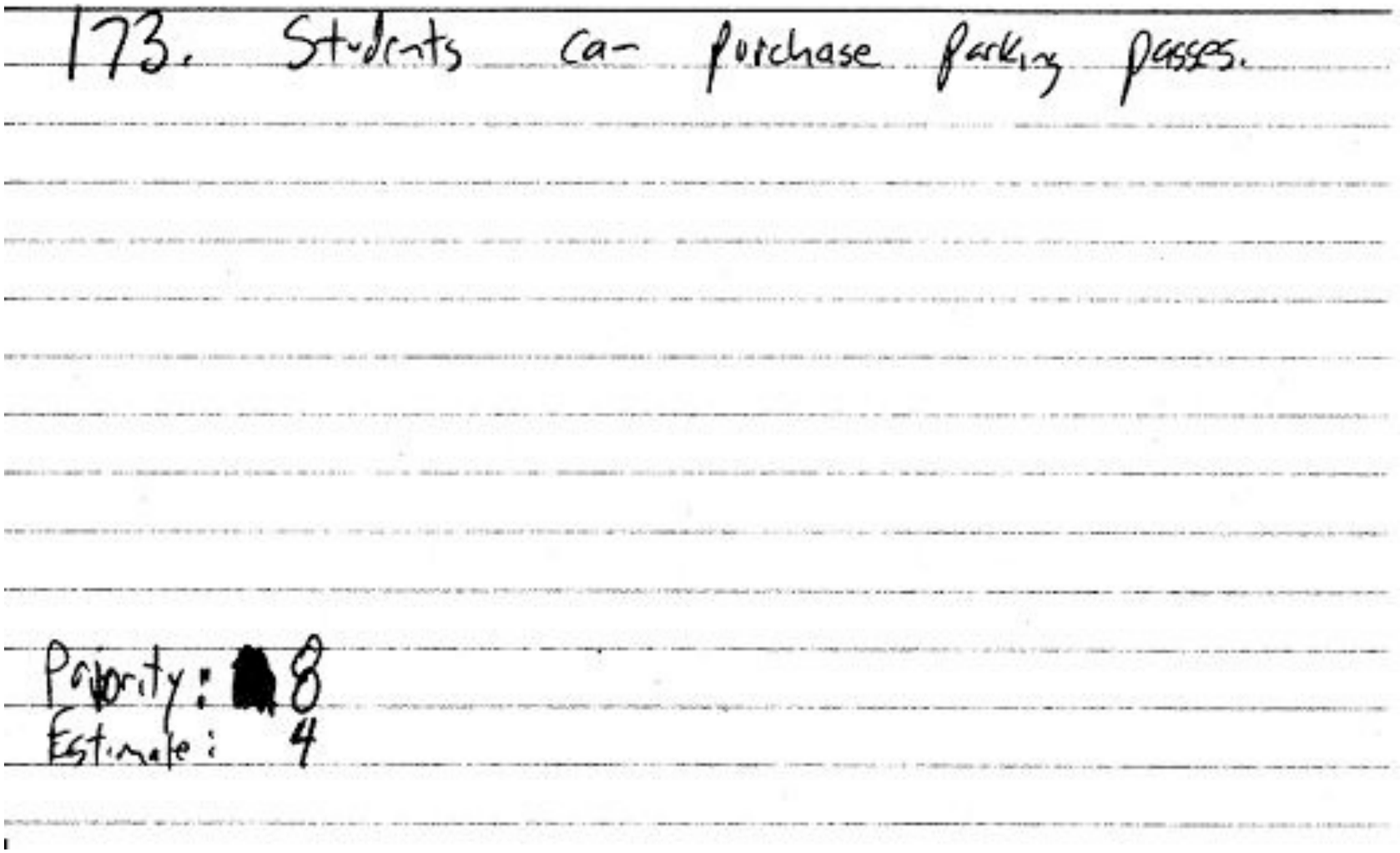
index card

short description

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.

Another example



User Stories vs. Use Case

- Not the same artifact!

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, ...

- See next theme!!!

Conclusion XP

- It is a **process** that is
  - **incremental**: growing software instead of designing
  - **iterative**: learning while doing.