Software Engineering I (02161)

Week 7

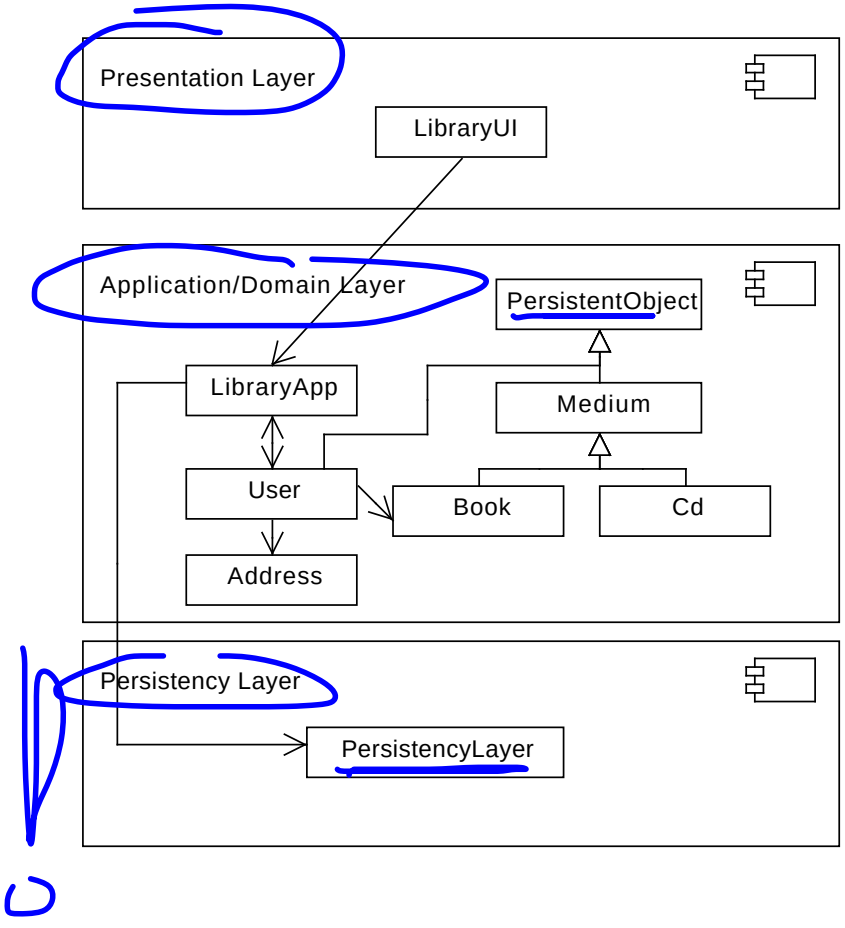
Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2013

Contents

Layered Architecture: Persistency Layer for the library application



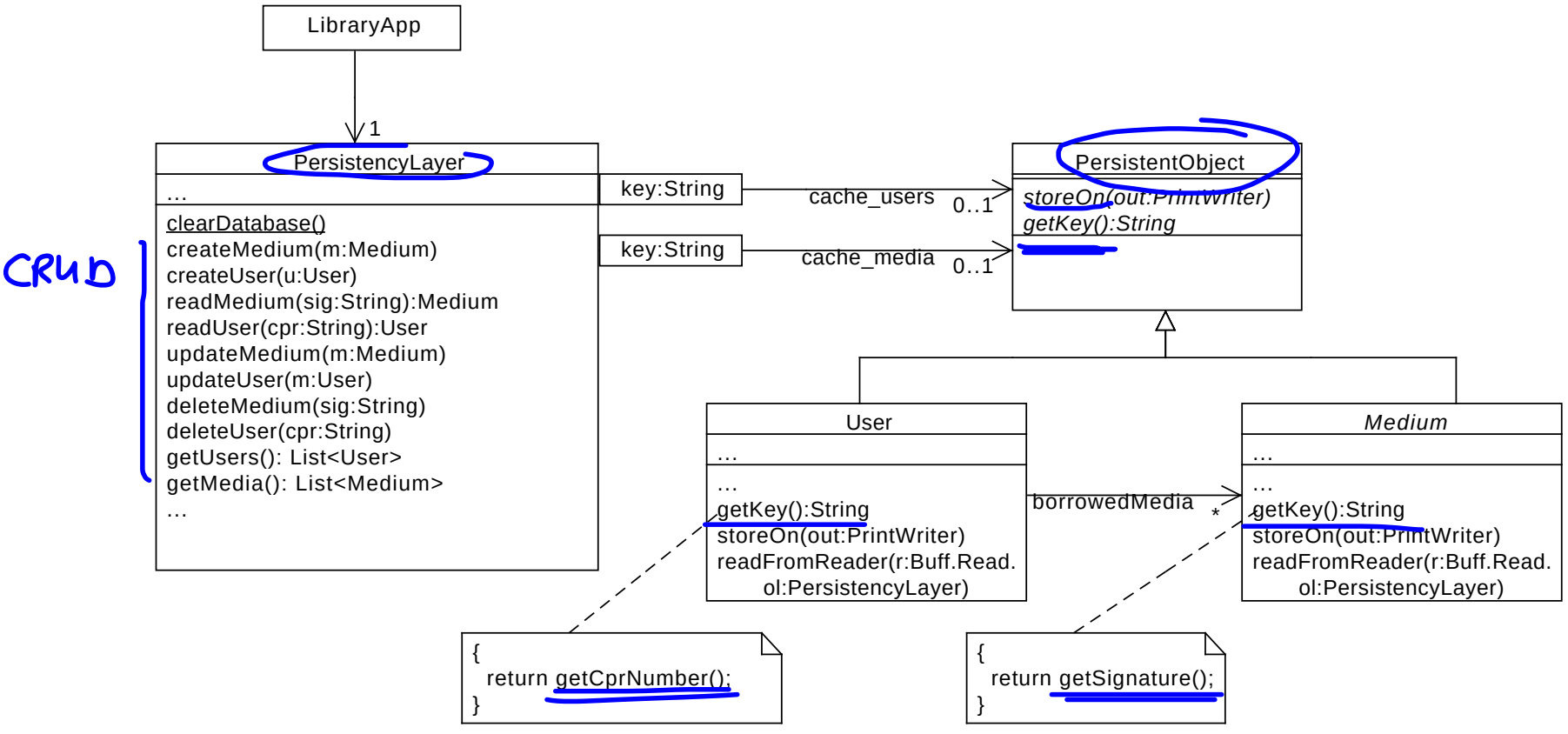
- Data stored in two files `users.txt` & `media.txt`; address has no file
- A book

```
dtu.library.app.Book
b01
some book author
some book title
Mar 13, 2011
<empty line>
```

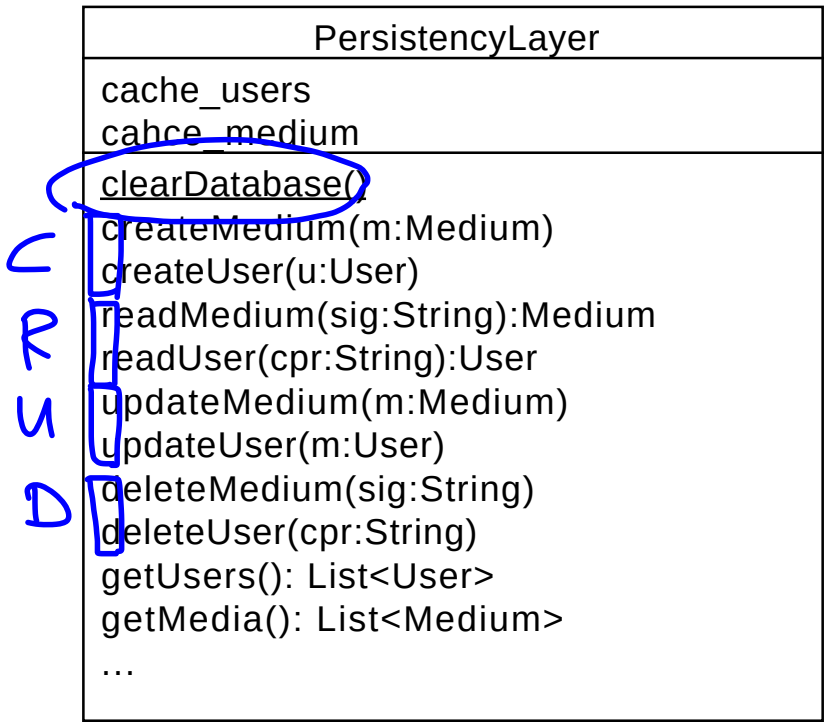
- A user

```
dtu.library.app.User
cpr-number
Some Name
a@b.dk
Kongevejen
2120
Hellerup
b01
c01
<empty line>
```

Persistency Layer



Layered Architecture: Persistency Layer for the library application



- CRUD: Create, Read, Update, Delete
- clearDatabase
  - removes the text files to create an empty database
  - Used with tests in @Before methods: Independent tests

Issues: Object identity

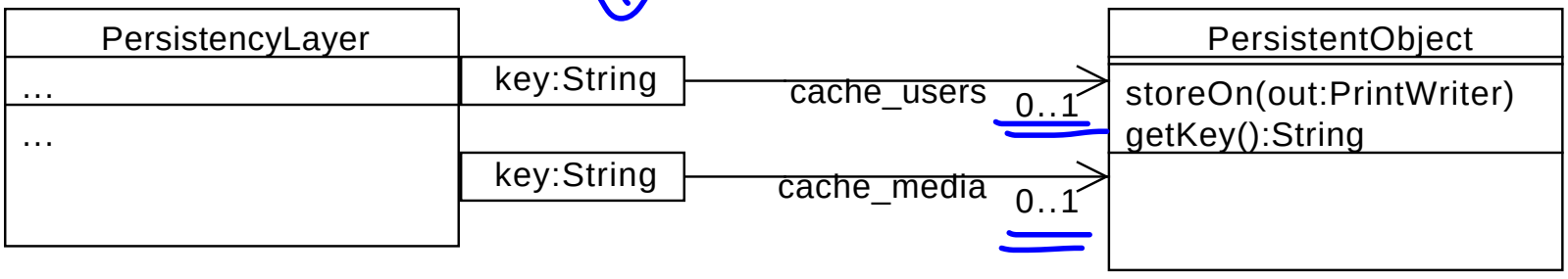
```
PersistencyLayer pl = new PersistencyLayer();
User user1 = pl.readUser("12345");
User user2 = pl.readUser("12345");
assertNotSame(user1,user2) // ?
assertSame(user1,user2)    // ?
```

Solution: Qualified Associations / Maps

Interface

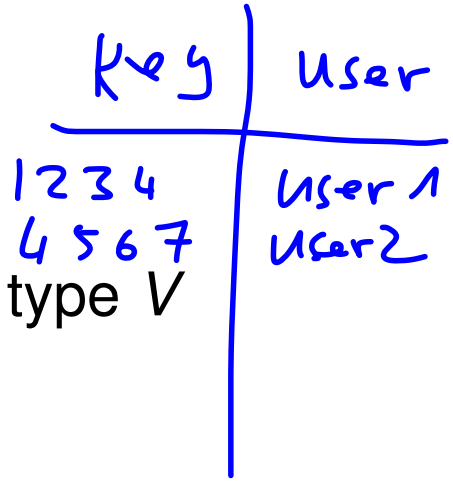
```
Map<String,PersitentObject> cacheUsers =
                            new HashMap()<String,PersistentObject>
Map<String,PersitentObject> cacheMedia =
                            new HashMap()<String,PersistentObject>
```

UML Notation




```
public User readUser(String key) {
   if (cacheUsers.contains(key)) { return cacheUsers.get(key); }
   User user = readObjectFromFile(String key);
   if (user != null) { cacheUsers.put(key,user); }
   return user;
}
```

Map<K,V> Interface



- Dictionary (table): keys of type *K*, values of type *V*
- Implementation class: HashMap<K,V>
- Operations
  - `m.put(aK,aV)`
  - `m.get(aK)`
  - `m.containsKey(aK)`
- Properties

Course Alg + Datastructures

Test 1

```
m.put(aK,aV);
assertTrue(m.containsKey(aK));
assertSame(aV,m.get(aK));
```

Test 2

```
assertFalse(m.containsKey(aK));
assertNull(m.get(aK));
```

Exercise tasks:

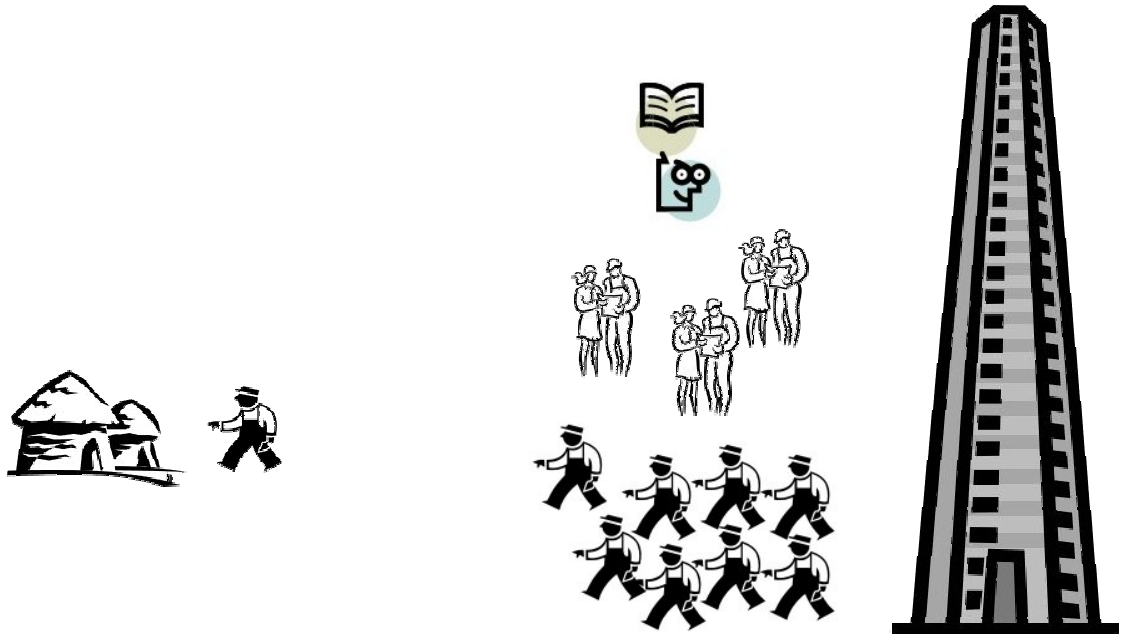
Advanced exercise

1) Implement the persistency layer (tests provided)
2) Intergrate persistentcy layer in the library application (tests have to be written)

- Additional information
  `http://www2.imm.dtu.dk/courses/02161/2013/slides/pe_persistency.pdf`

Contents

Software Development Challenges



- Challenges of Software Development
  - On **time**
  - In **budget**
  - No **defects**
  - Customer **satisfaction**

Software Development Process

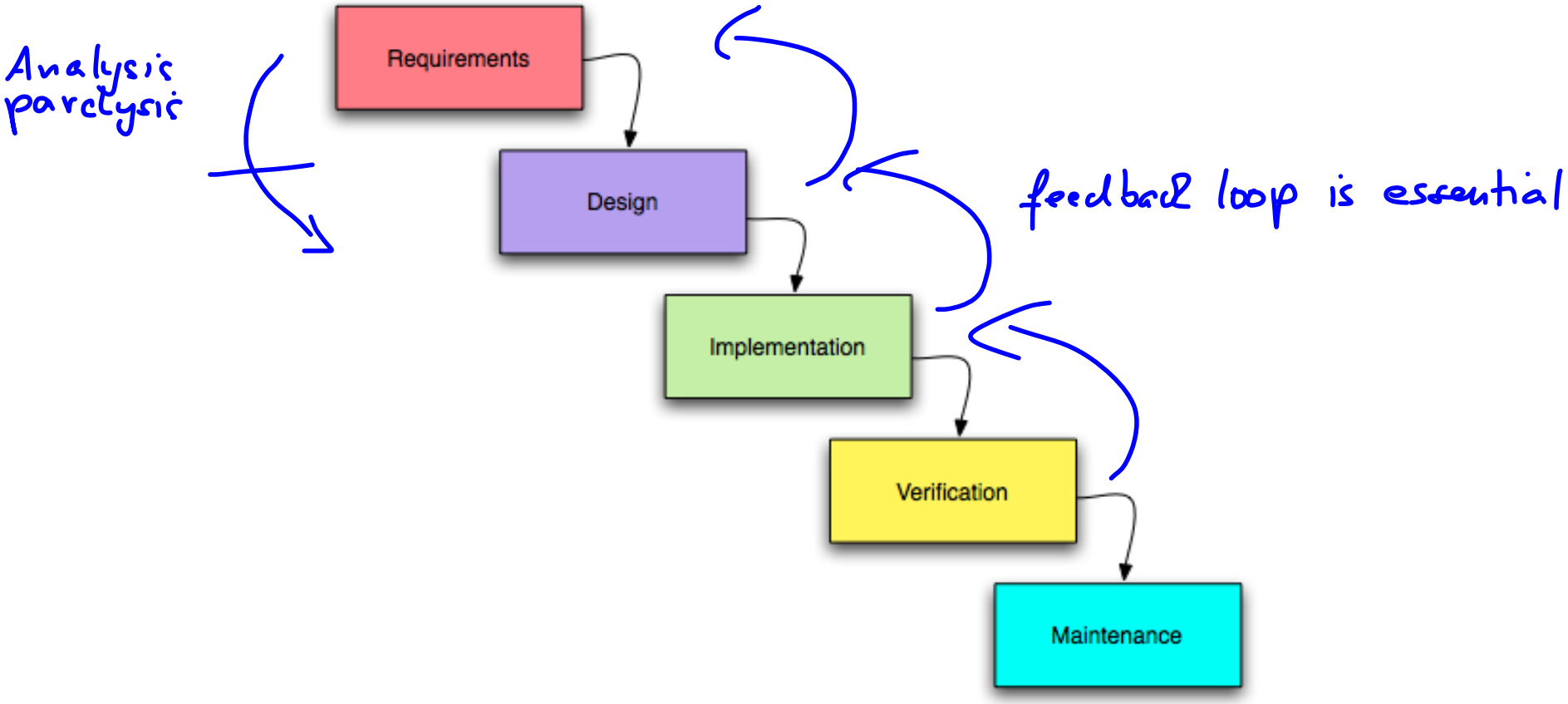
- Activities in Software Development
  - *Understand* and *document* what the customer wants: **Requirements Engineering**
  - *How* to build the software: **Design**
  - *Build* the software: **Implementation**
  - *Validate*: **Testing**, **Verification**, **Evaluation**

→ Set of techniques: Use cases, CRC cards, refactoring, test-driven development, ...

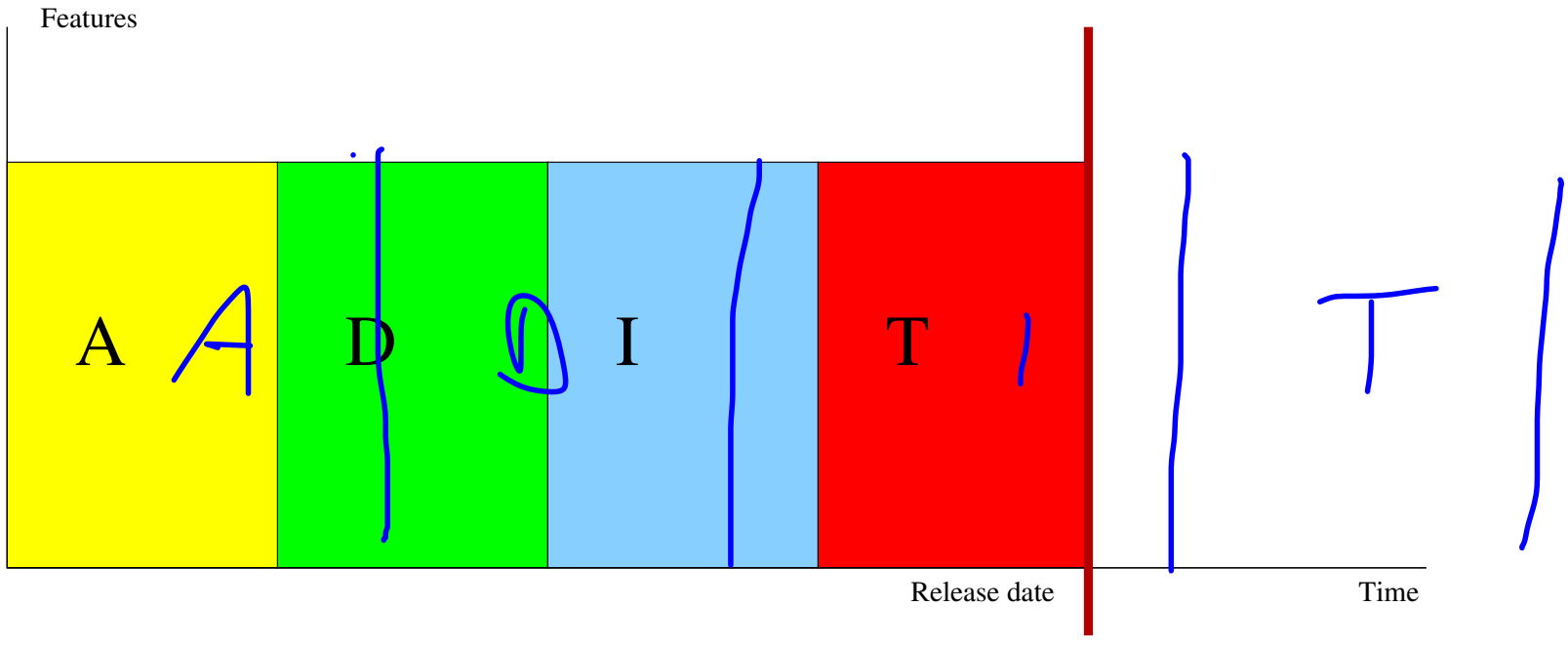
- How to apply the techniques: in which order?

→ Different **software development processes**: Waterfall, Iterative processes, agile, lean, ...

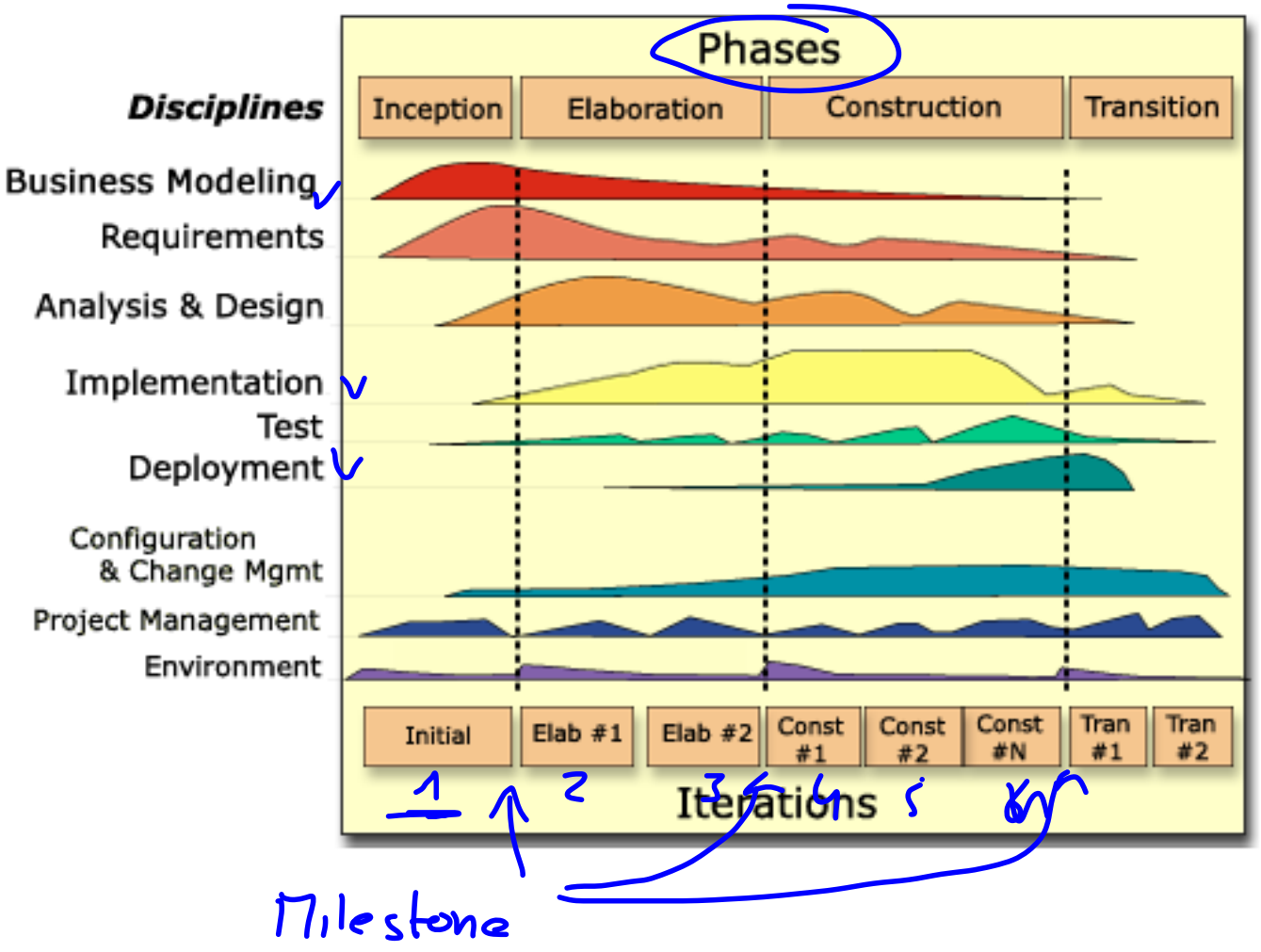
Waterfall process



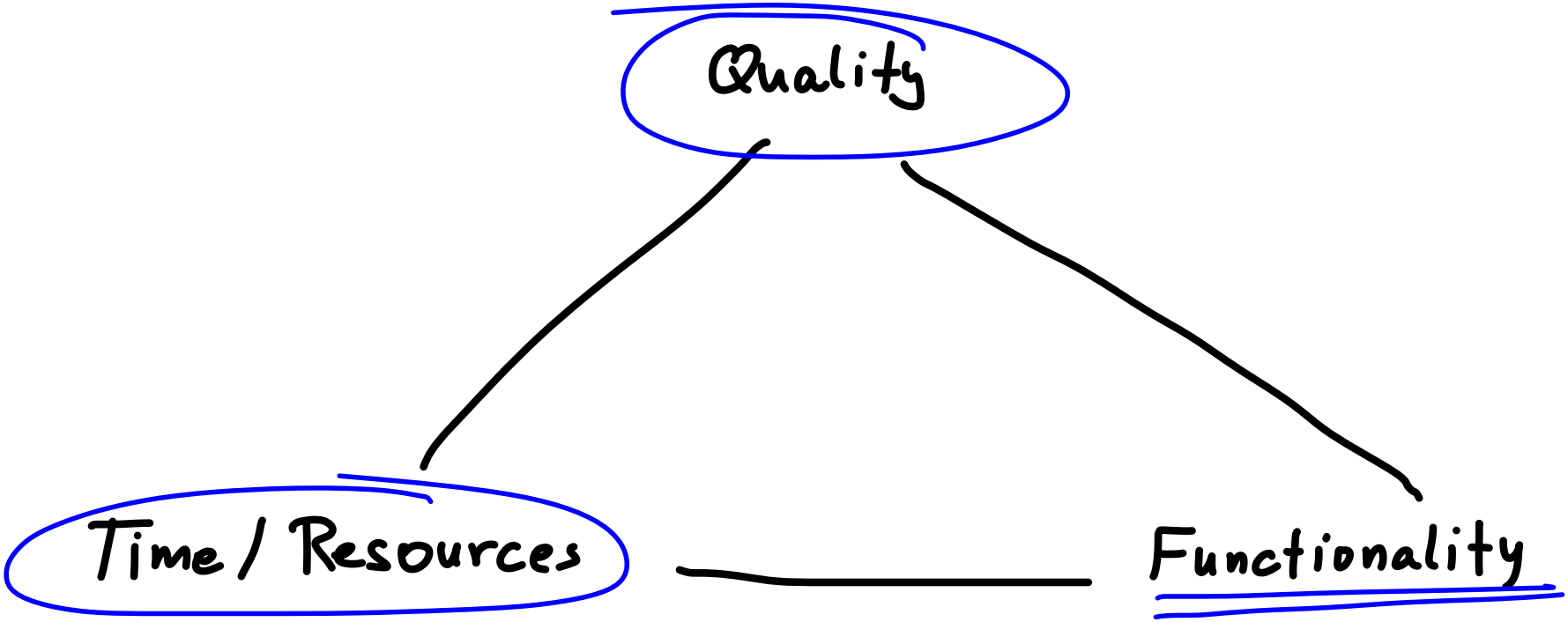
Delays in waterfall processes



Iterative Processes: E.g. (Rational) Unified Process



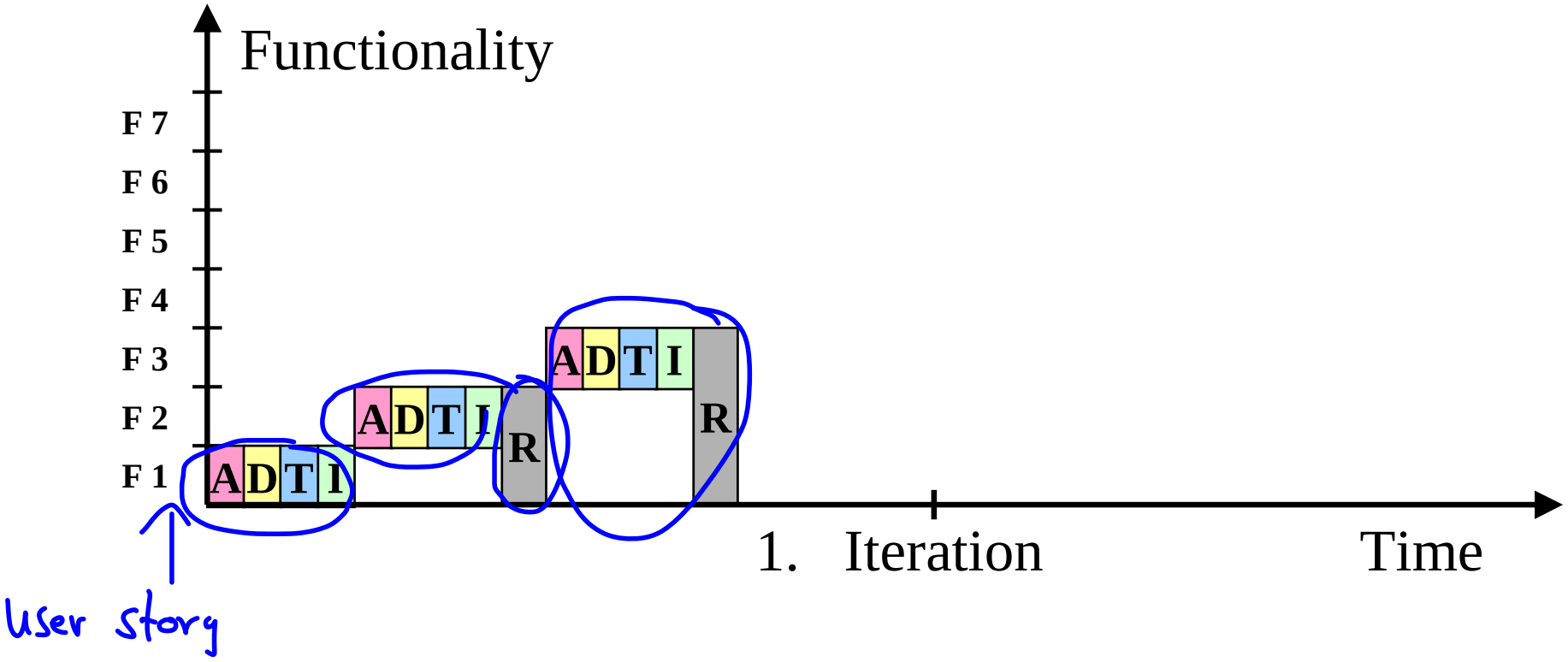
Resource Triangle



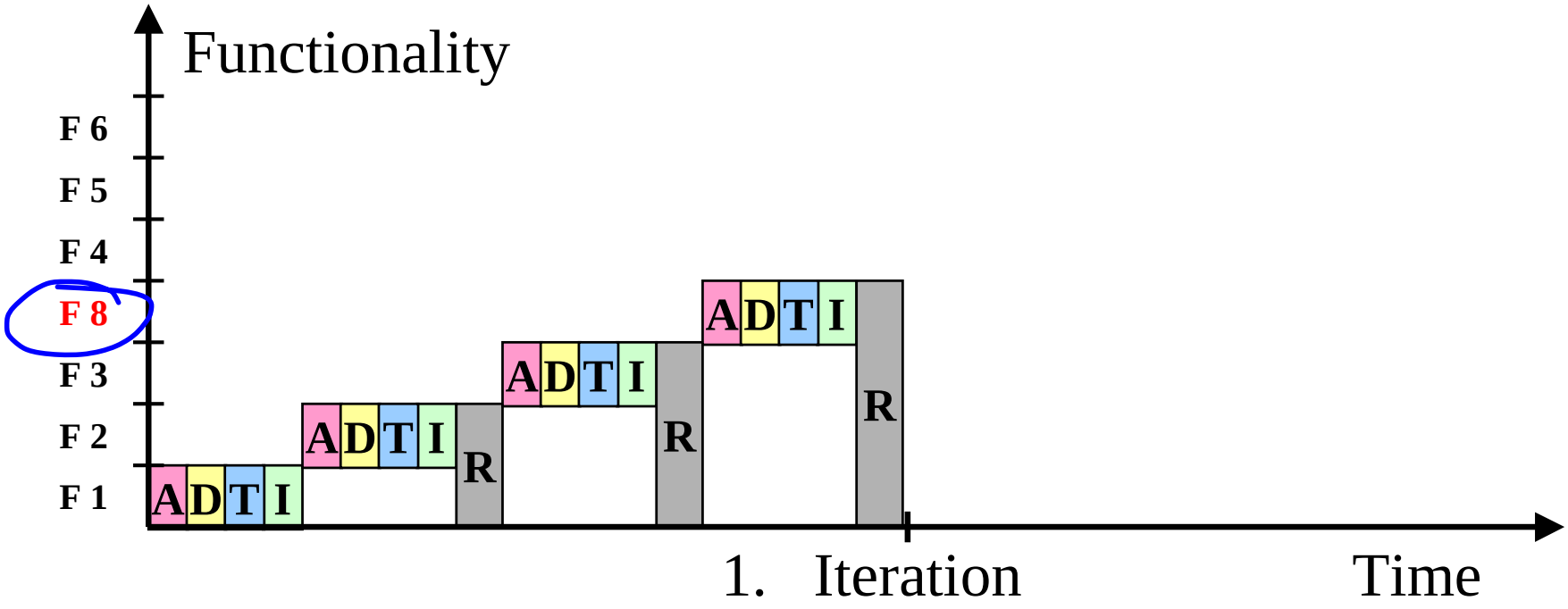
- Can only fix two of them at the same time

variable
make a priority list
highest priorities first

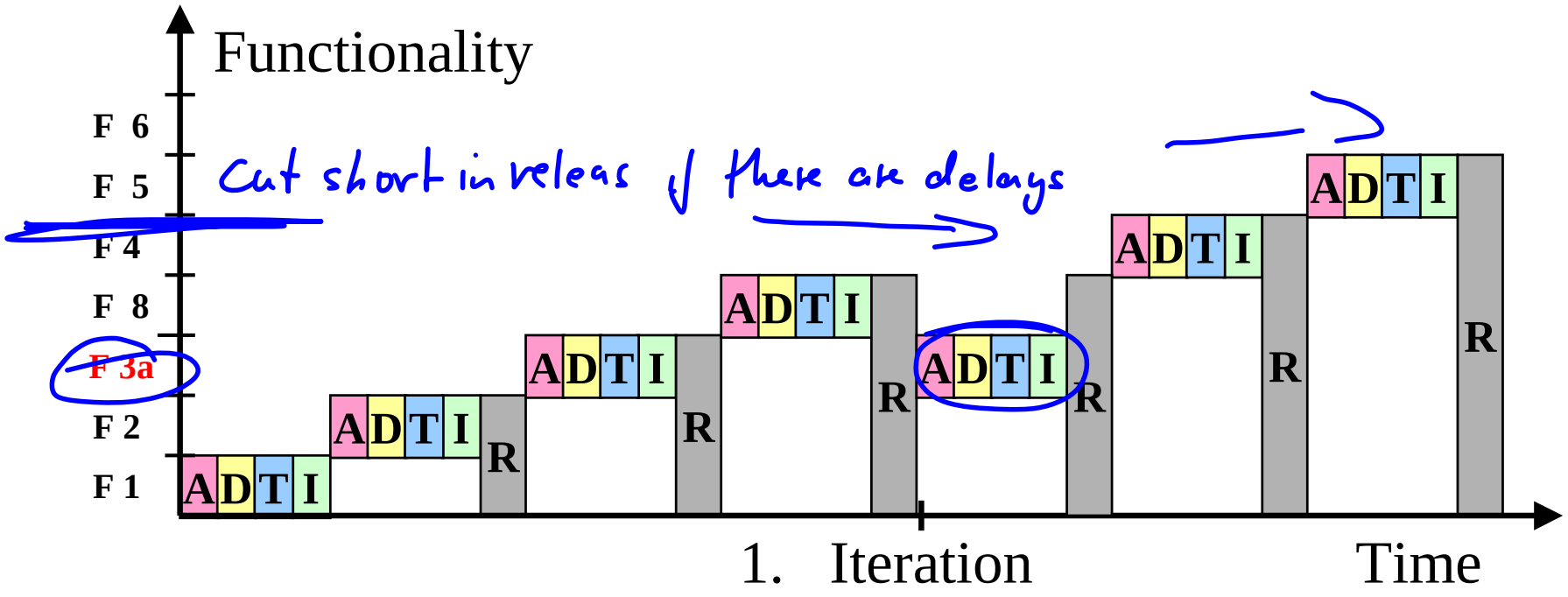
Agile processes and Lean Software Development



Agile processes and Lean Software Development



Agile processes and Lean Software Development



Agile Processes and Lean Software Development

agilemanifesto.org

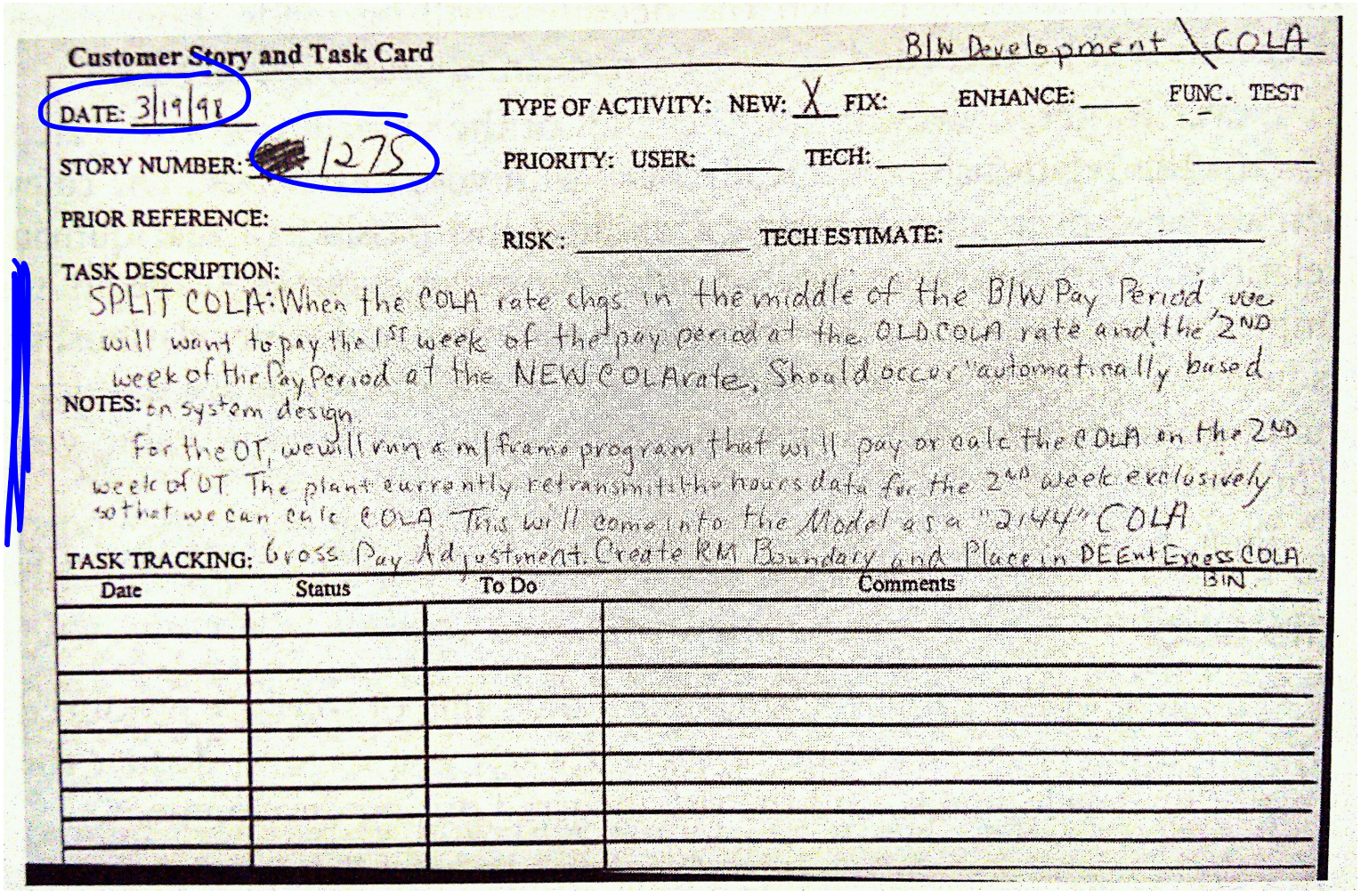
2009

- Agile processes: eXtreme Programming (XP), Scrum, Feature Driven Development (FDD), *Lean Software Development*
- Common characteristics
  - Short iterations
  - Focus on *marketable features* (Lean/Kanban) / user stories (XP) / product backlog items (Scrum)
  - New, **extreme** practices
  - Applying **values** and **principles** from *Lean Production*

User stories

- Introduced with Extreme Programming
- Focus on features
  - ”As a customer, I want to book and plan a single flight from Copenhagen to Paris”.
  - Recommended, but not exclusive: ”As a <role>, I want <goal/desire> so that <benefit>”
- Difference to Use Cases: User stories can be defined for non-functional requirements
  - ”The search for a flight from Copenhagen to Paris shall take less than 5 seconds”
- Documented by user story cards, i.e. index cards

Example of a User story card



Customer Story and Task Card — B/W Development \ COLA

DATE: 3/19/98

TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST ___

STORY NUMBER: [redacted] 1275

PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___

RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:

SPLIT COLA: When the COLA rate chgs. in the middle of the B/W Pay Period we will want to pay the 1ST week of the pay period at the OLD COLA rate and the 2ND week of the Pay Period at the NEW COLA rate. Should occur automatically based on system design.

NOTES: For the OT, we will run a m/frame program that will pay or calc the COLA on the 2ND week of OT. The plant currently retransmits the hours data for the 2ND week exclusively so that we can calc COLA. This will come into the Model as a "2144" COLA

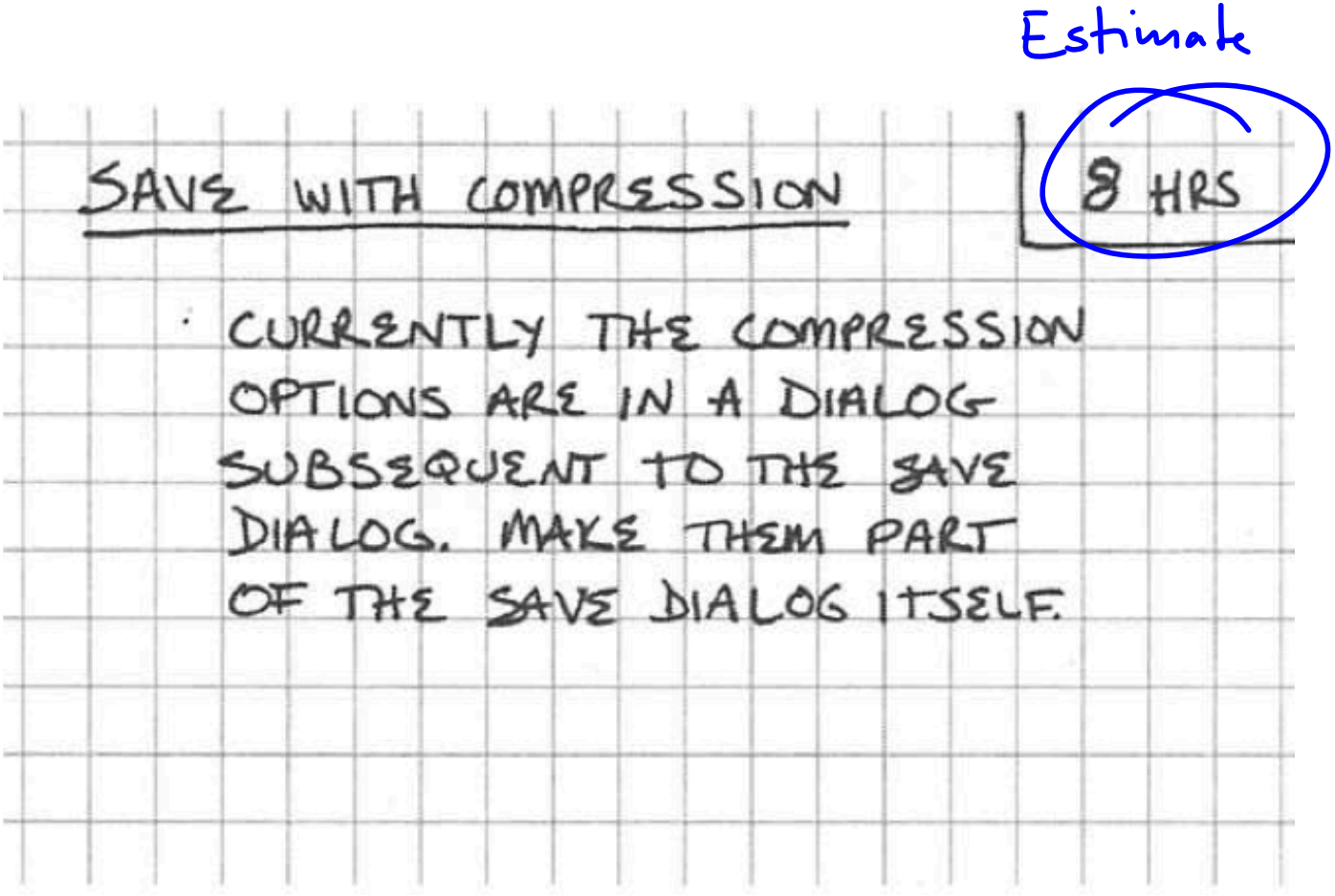
TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in DEEntExpress COLA BIN

Date	Status	To Do	Comments

Kent Beck, Extreme Programming, 1st ed.

- User story card: A contract between the customer and the devloper to talk about the user story

Example of a User story card



Kent Beck, Extreme Programming 2nd ed.

User stories and requirements engineering

- Requirements engineering is done *in parallel* with the development of the system
- Requirements engineering
  - Epics
    - → coarse level user stories
  - User story cards
    - → not too much details
- User stories are assigned to iterations
  - → priority for the customer
- In an iteration
  - refine user stories
  - → provide detail
- → Compare with waterfall
  - Requirement phase: as detailed as possible

acceptance tests: formal descr. of use story

Comparision: User Stories / Use Cases

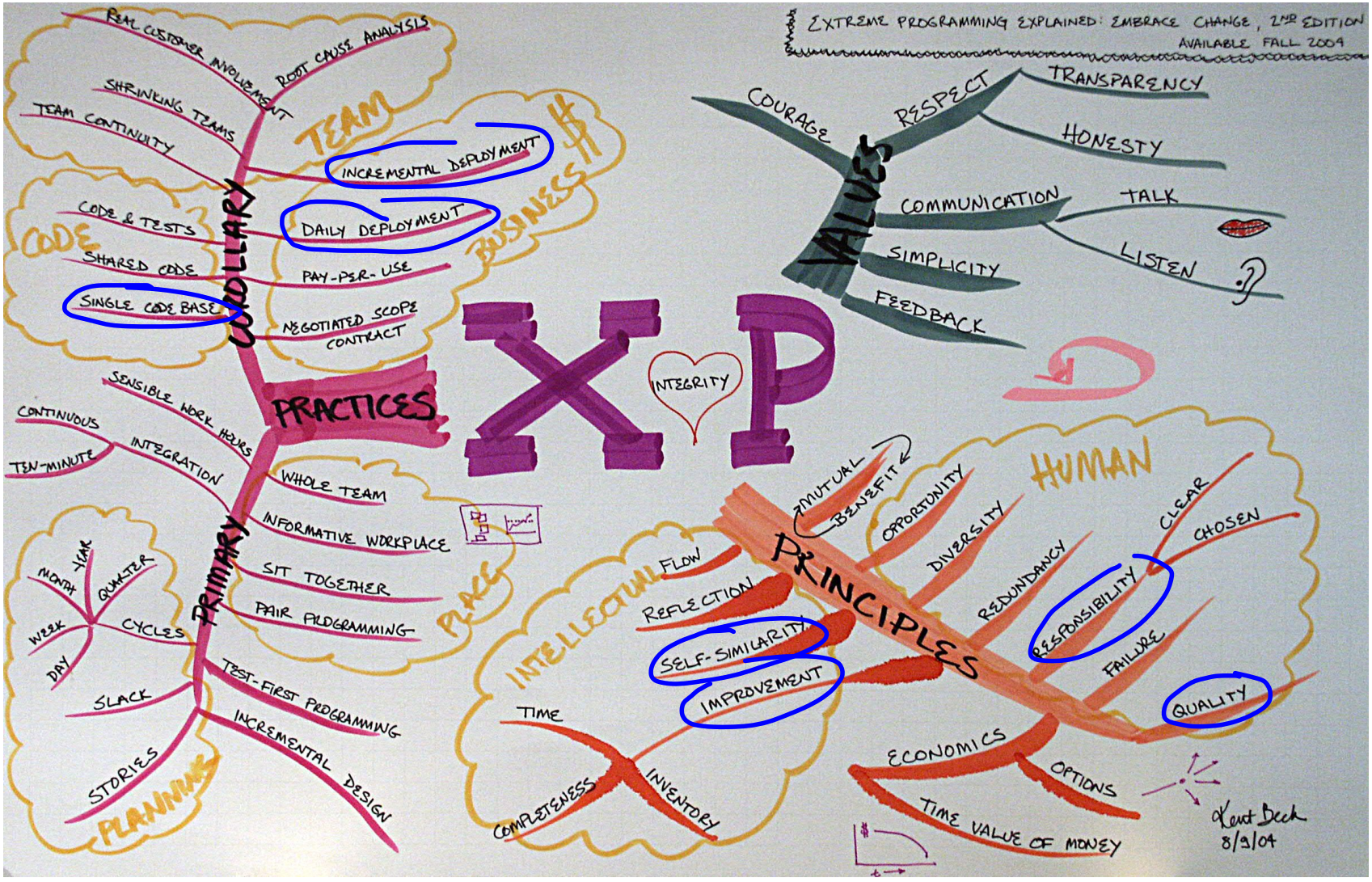
Use Story

- one *concrete* scenario/feature
  - concrete data
- requirements of relevance for the user
  - functiional: e.g. use case scenario
  - non-functional

Use Cases

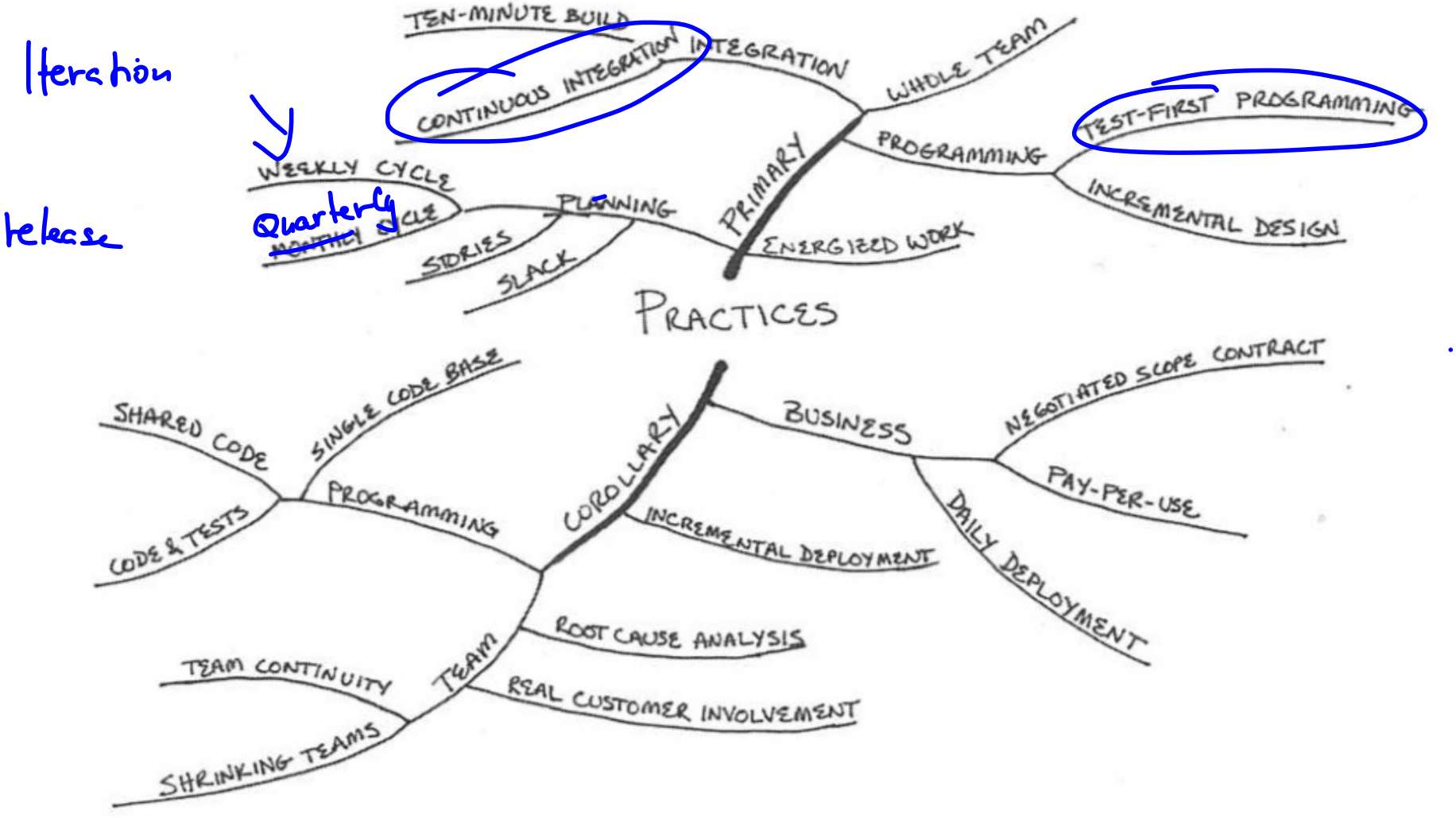
- several *abstract* scenarios with one goal
- only functional requirements

eXtreme Programming (XP)



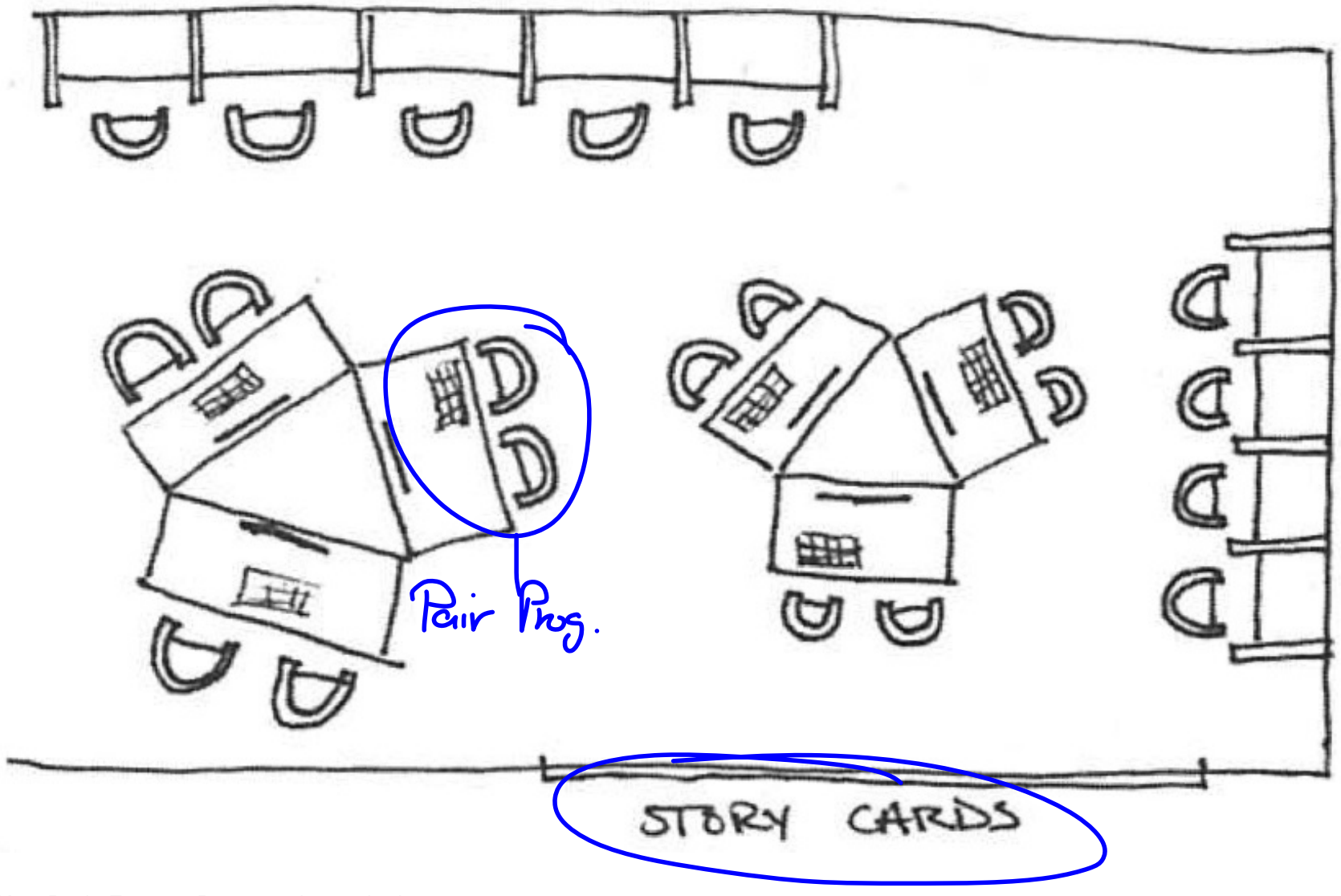
Kent Beck, Extreme Programming 2nd ed.

eXtreme Programming practices



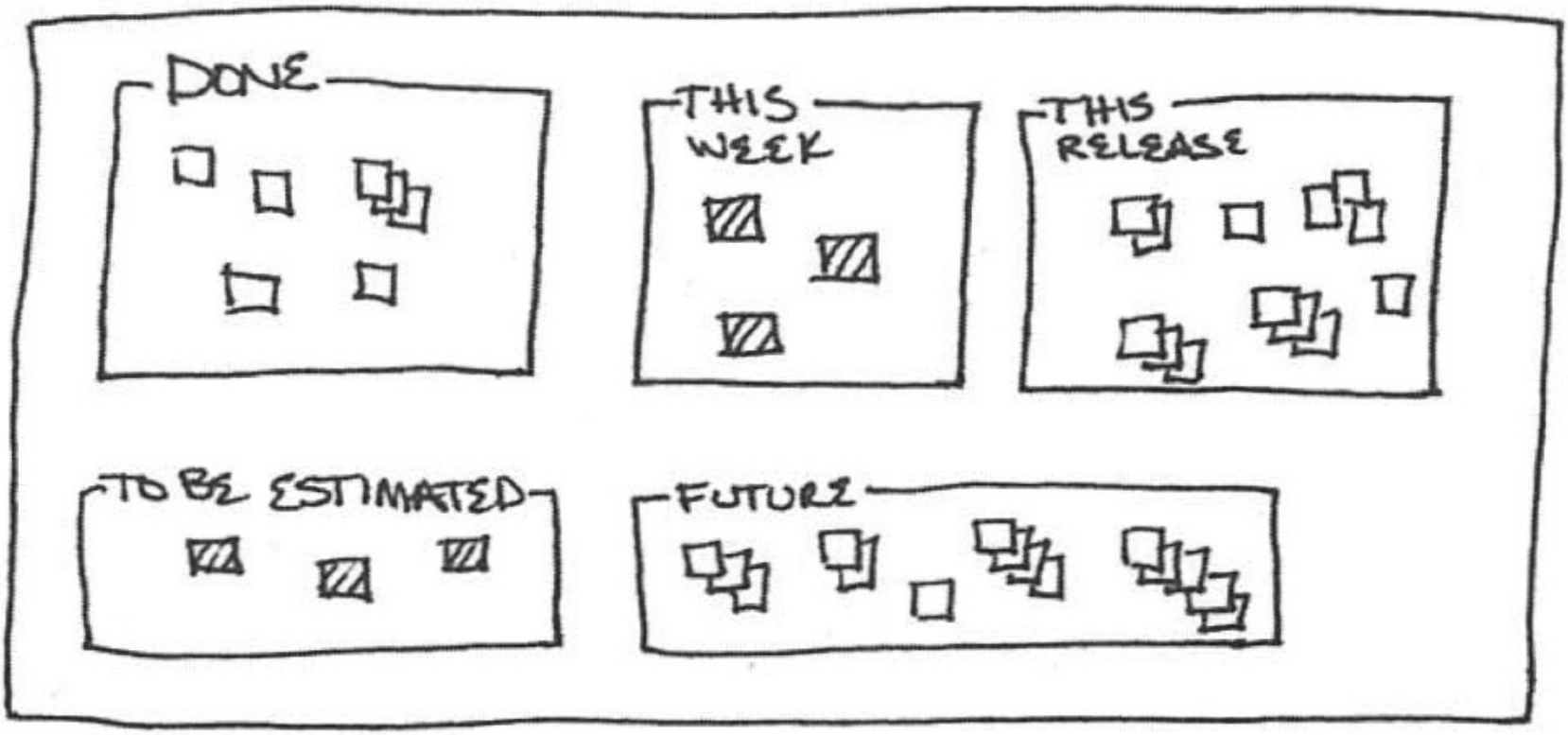
Kent Beck, eXtreme Programming, 2nd edition

Sit-together



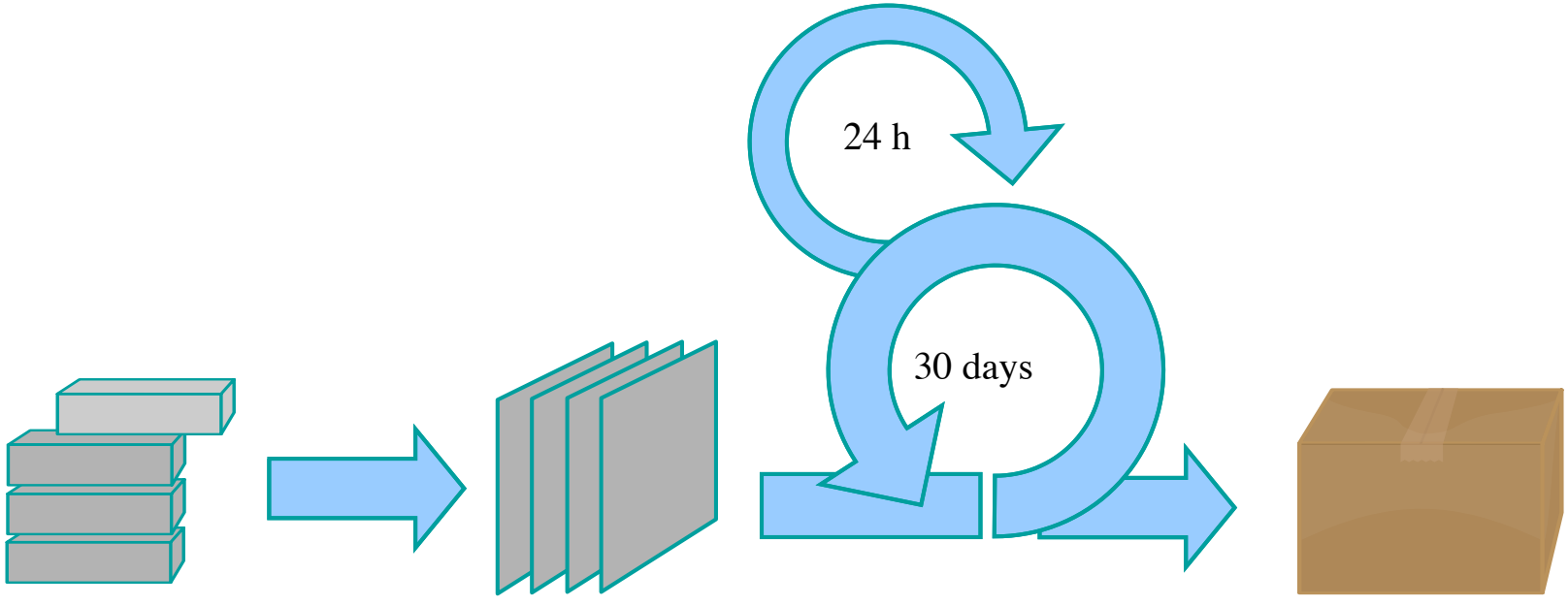
Kent Beck, Extreme Programming 2nd ed.

Visual wall



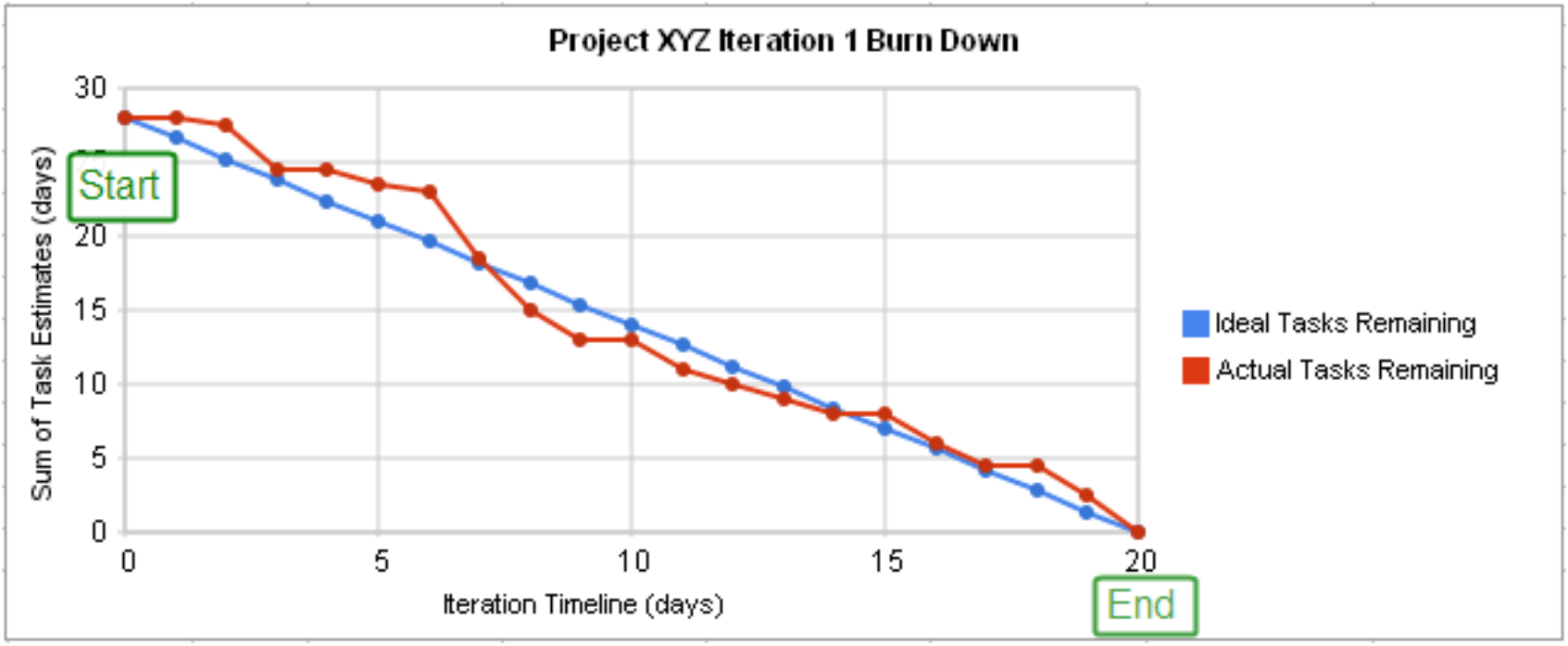
Kent Beck, Extreme Programming 2nd ed.

Scrum



Wikipedia

Burn Down Charts



Wikipedia

Lean Software Development

- Lean Production:
  - Reduce the amount of *waste*
  - Generate *flow*
- Waste: resources used with does not produce value for the customer
  - time needed to fix bugs
  - time to change the system because it does not fit the customers requirements
  - time waiting for approval
  - ...

Cycle time

- Increase feedback: Reduce time it takes to go through the process: *cycle time*

Cycle time

$$cycle\_time = \frac{number\_of\_features}{feature\_implemantion\_rate}$$

Software: 250 features, 50 weeks, feature_implementation_rate = 5 features/week

- Waterfall: cycle_time = 250 / 5 = 50 weeks

$\rightarrow$ 1 cycle

Cycle time

- Increase feedback: Reduce time it takes to go through the process: *cycle time*

Cycle time

$$cycle\_time = \frac{number\_of\_features}{feature\_implemantion\_rate}$$

Software: 250 features, 50 weeks, feature_implementation_rate = 5 features/week

- Waterfall: cycle_time = 250 / 5 = 50 weeks

$\rightarrow$ 1 cycle

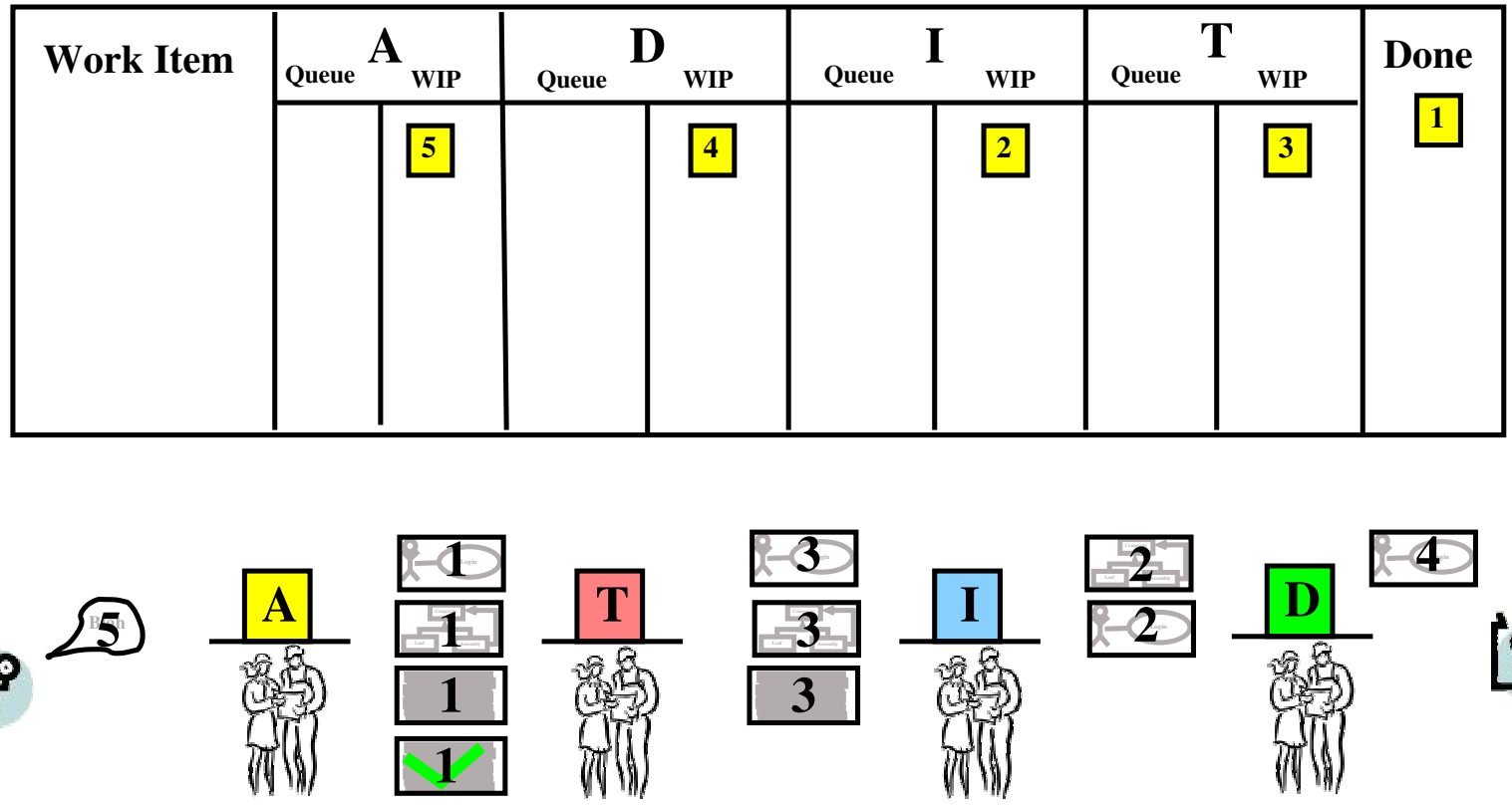
Reducing the cycle time

- Agile: cycle_time = 1 / 5 = 8 hours

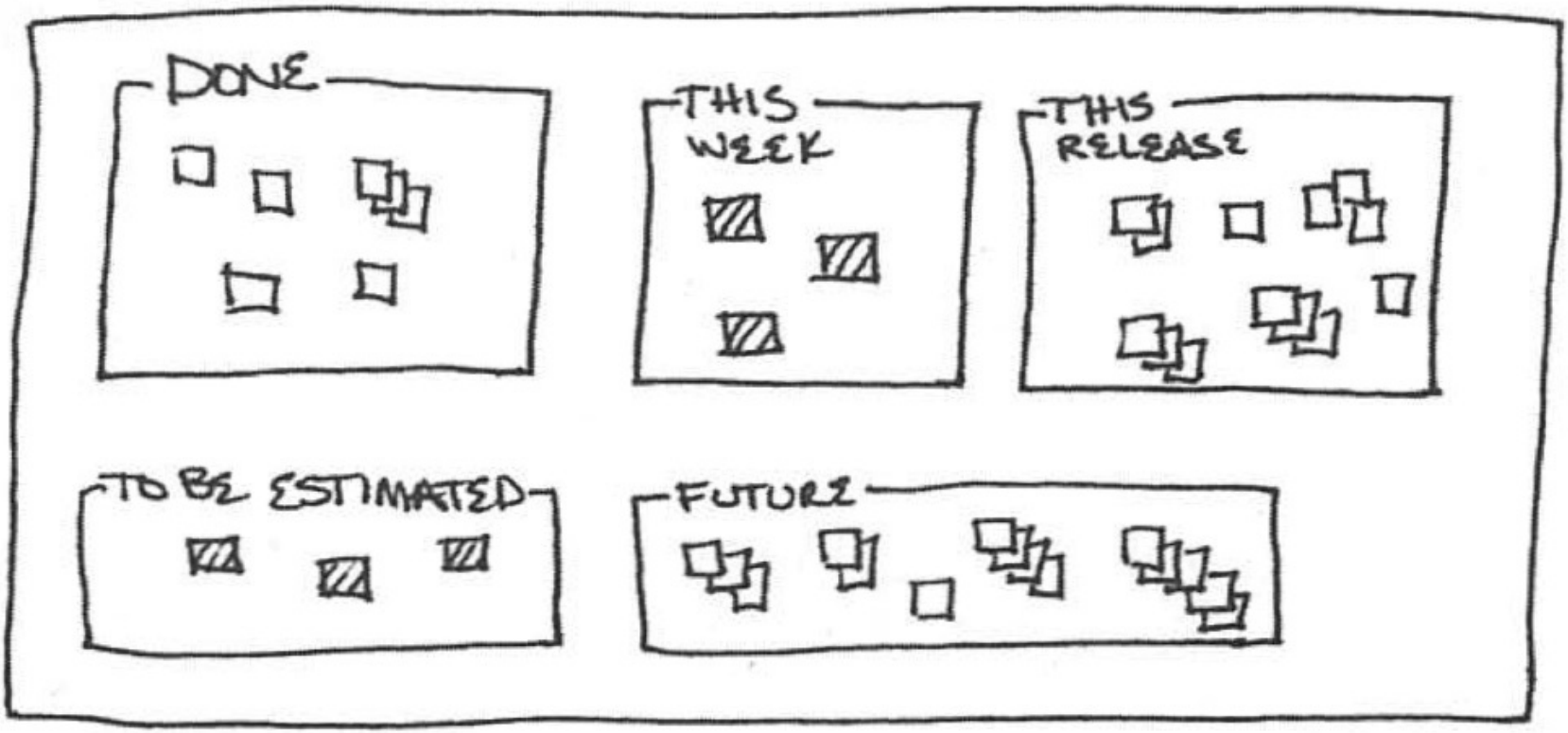
$\rightarrow$ 250 cycles

$\rightarrow$ Process improvement: incease in features / week

Generating flow using Pull and Kanban



Visual wall / Kanban



Software Engineering: Flow through Pull with Kanban

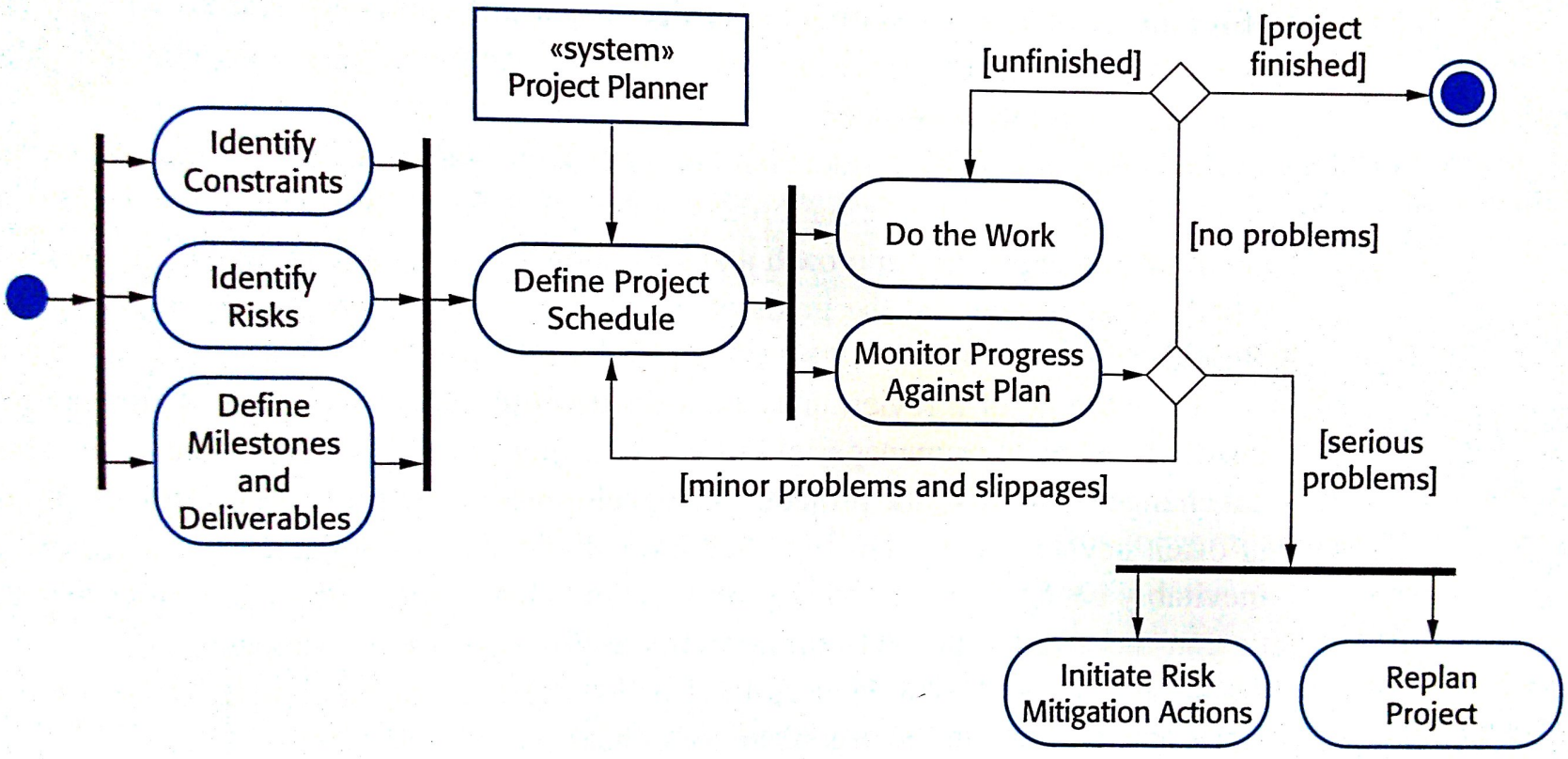
- Process controlling: local rules
- Load balancing: Kanban cards and *Work in Progress (WIP) limits*
- Integration in other processes: e.g. Scrum + Kanban = Scrumban
- `www.targetprocess.com`: Electronic kanban board usefull for your project

Contents

Project Planning

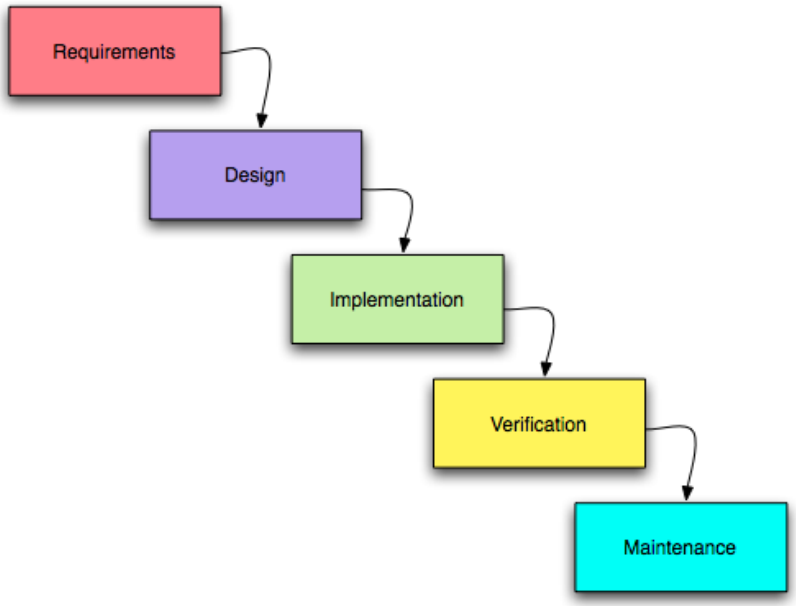
- Project plan
  - Defines:
    - How work is done
  - Estimate
    - Duration of work
    - Needed resources
    - → Price
- Project planning
  - Proposal stage
    - → Price
    - → Time to finish
  - Project start-up
  - During the project
    - Progress (tracking)
    - Adapt to changes

Process planning and executing



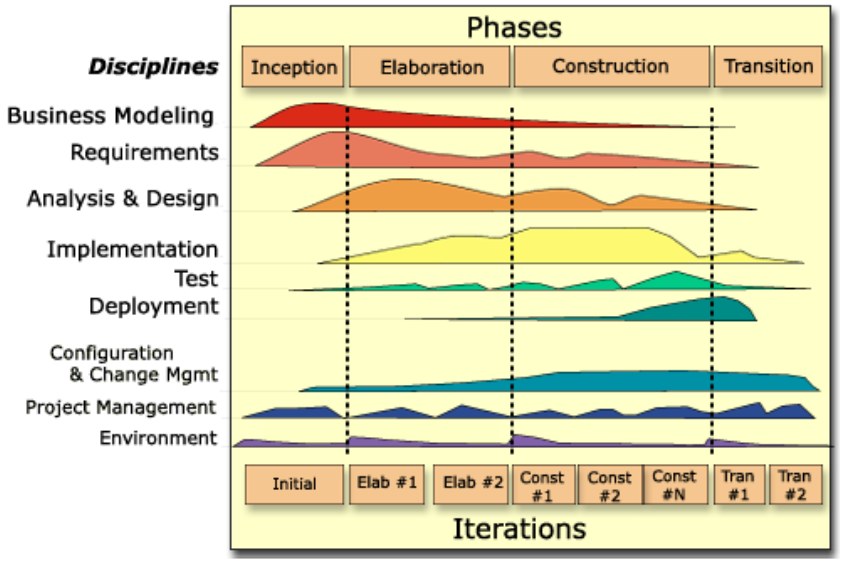
Processes

- Waterfall



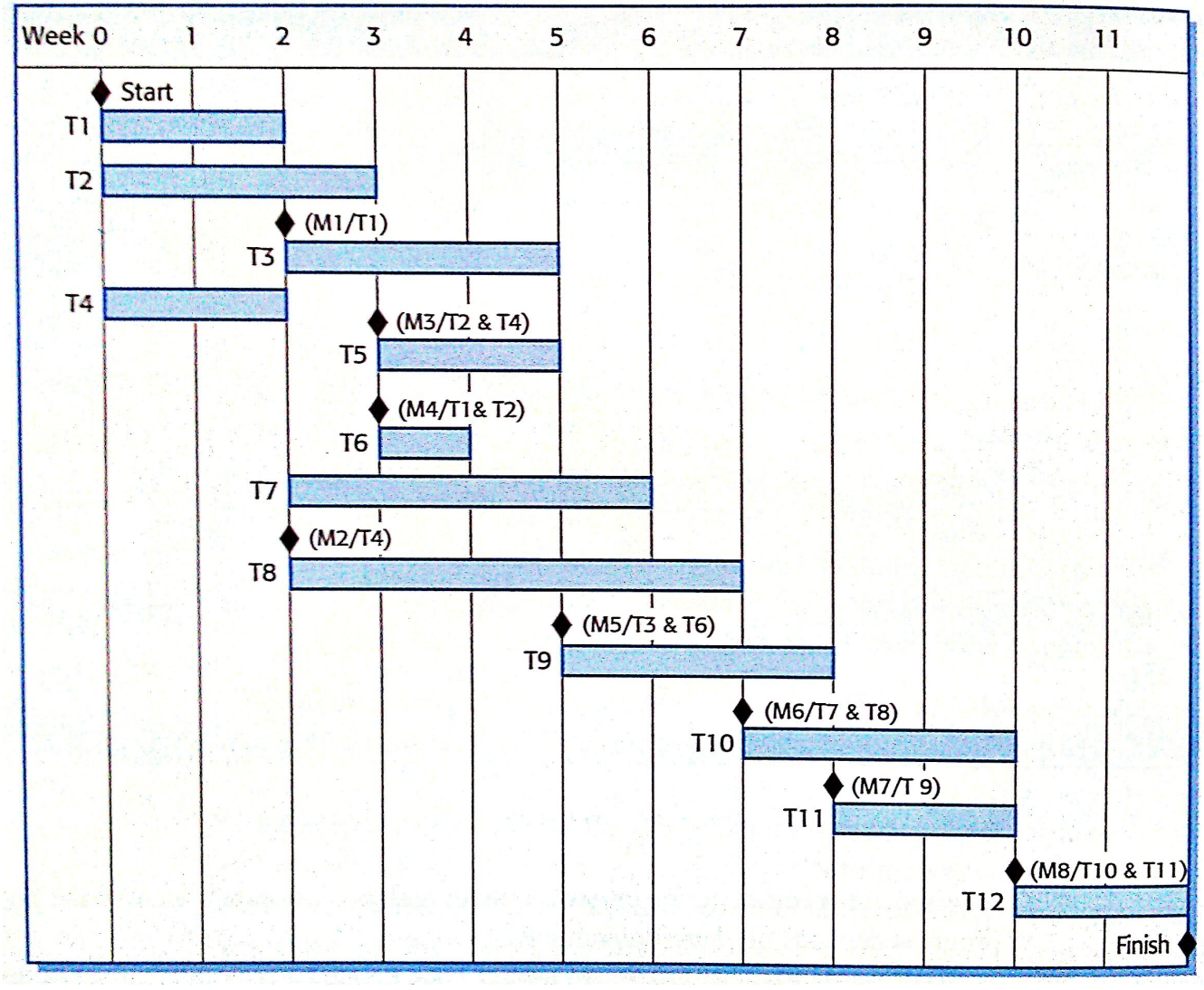
- milestones/deliverables: system specification, design specification, . . .
- Typical tasks: Work focused on system components

- Iterative Development (e.g. RUP)



- Milestones/deliverables: Each phase: go ahead to next phase
- Typical tasks: Work focused on system components

Schedule Representation: Gantt Chart / Bar chart



Project estimation techniques

- Experienced based
  - XP: story points
  - Comparision with similar tasks
- Algorithmic based
  - e.g. COCOMO, COCOMO II, ...

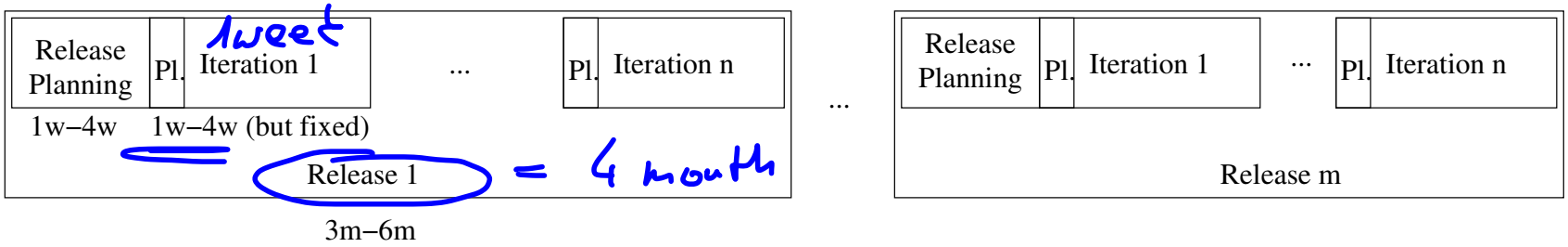
Algorithmic cost modeling: COCOMO

- Constructive Cost Model (COCOMO) by Bary Boehm et al., 1981
- based on empirical studies
- Effort: in person months: $PM = a * LOC^b$
  - Lines of code (LOC)
  - $2.4 \leq a \leq 3.6$: type of software
  - $1 \leq b \leq 1.5$: cost drivers: platform difficulty, team experience, ...
- Project duration: $TDEV = 3 * PM^{0.33+0.2*(b-1.01)}$
- Staffing: $STAFF = PM / TDEV$

Planning Agile Projects

- *fixed* general structure

→ quarterly cycle / weekly cycle practices in XP



- Releases (quarterly cycle)
  - make (business) sense
  - *user stories* / *themes*
- Iterations with releasees (weekly cycle)
  - *user stories*
- *time boxing*
  - fixed: release dates and iterations
  - adjustable: scope

Planning game

- Customer defines:
  - user stories
  - priorities
- Developer define:
  - costs, risks
  - suggest user stories
- Customer decides: is the user story worth its costs?
  - → split a user story
  - → change a user story

Project estimation and monitoring

- Estimation: two possibilities
  1) Estimate *ideal time* (e.g. person days / week) * load_factor
  2) Estimate *relative* to other user stories: *story points*
- Monitoring

  ad 1) New *load factor*: total_iteration_time / user_story_time finished

  ad 2) *velocity*: Number of points per iteration

→ What can be done in the next iteration (yesterdays weather)

- Important: If in trouble focus on *few* stories and finish them

Contents

Process to be used for the exam project

1. Create a use case diagram
2. Select the most important use case scenarios and define user stories for them
3. Determine a basic architecture
5. Create a project plan
5. Repeat:
   - 2 people take a user story/task due (highest priority first)
     a) detail use case scenario
     b) acceptance tests
     c) (contribution to systematic tests)
     d) CRC cards for the design
     e) report any design issues in the report
     f) test-driven implementation of the scenario
     g) (create sequence diagram for the scenario)
   - Meet often to coordinate the design
   - Don't forget to update your plan as you learn
6. Collect the material you have written and finish the report

Example Plan

- Remark: report structure ≠ project structure
  - Report structure: waterfall (by technique)
  - Project structure: agile (by user story)

Project plan for a MUD game

number of persons	hours a week per person	hours a week	no. of weeks	hours for the whole project
4	8	32	5	160

	User Story/Tasks	Ideal man hour	man hour with load factor	Total hours in week	Total hours
Week 1	Creating the base structure of the report	1	2	2	**2**
	Player starts game	2	4	6	**6**
	Player looks into a room	2	4	10	**10**
	Player moves to adjacent room	2	4	14	**14**
	Writing about the use cases	2	4	18	**18**
	Player advances to next level	2	4	22	**22**
	Player finishes game	2	4	26	**26**
	Player takes up object	2	4	30	**30**
	Syst. tests for the use cases in this iteration	2	4	34	**34**
Week 2	Players lays down object	2	4	6	**38**
	Player registers successful for the game	2	4	10	**42**
	Player registers, but name is already used	2	4	14	**46**
	Writing the introduction	2	4	18	**50**
	Writing about the design	2	4	22	**54**
	Syst. tests for the use cases in this iteration	2	4	26	**58**
	…	…	…	…	**…**

User stories

Replan often? Add new stories, Change stories