Software Engineering I (02161)

Week 8

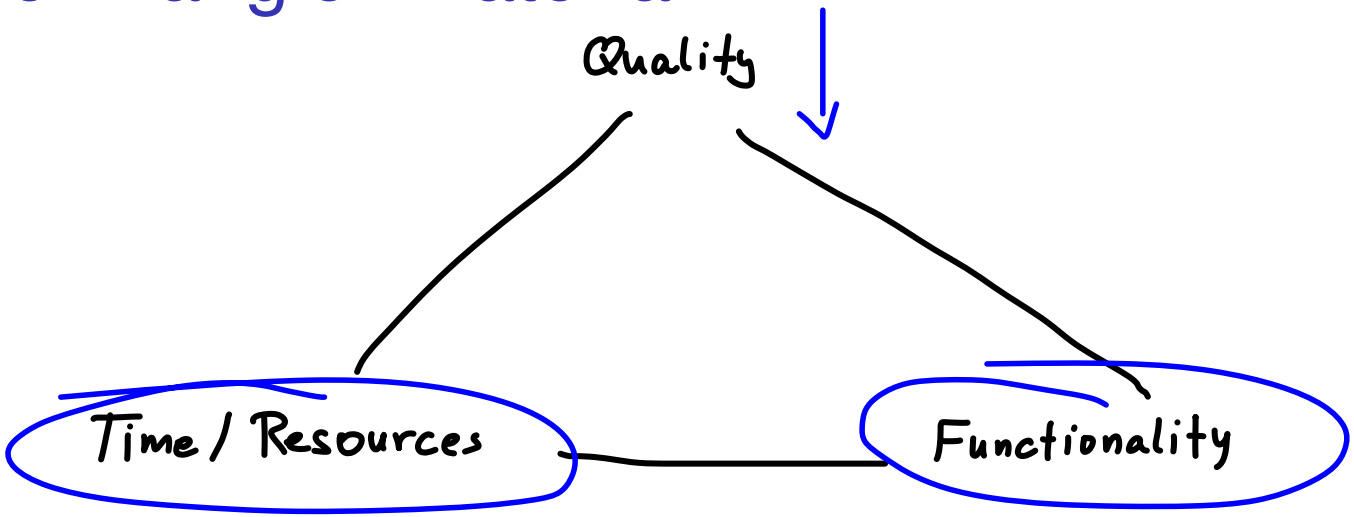
Assoc. Prof. Hubert Baumeister

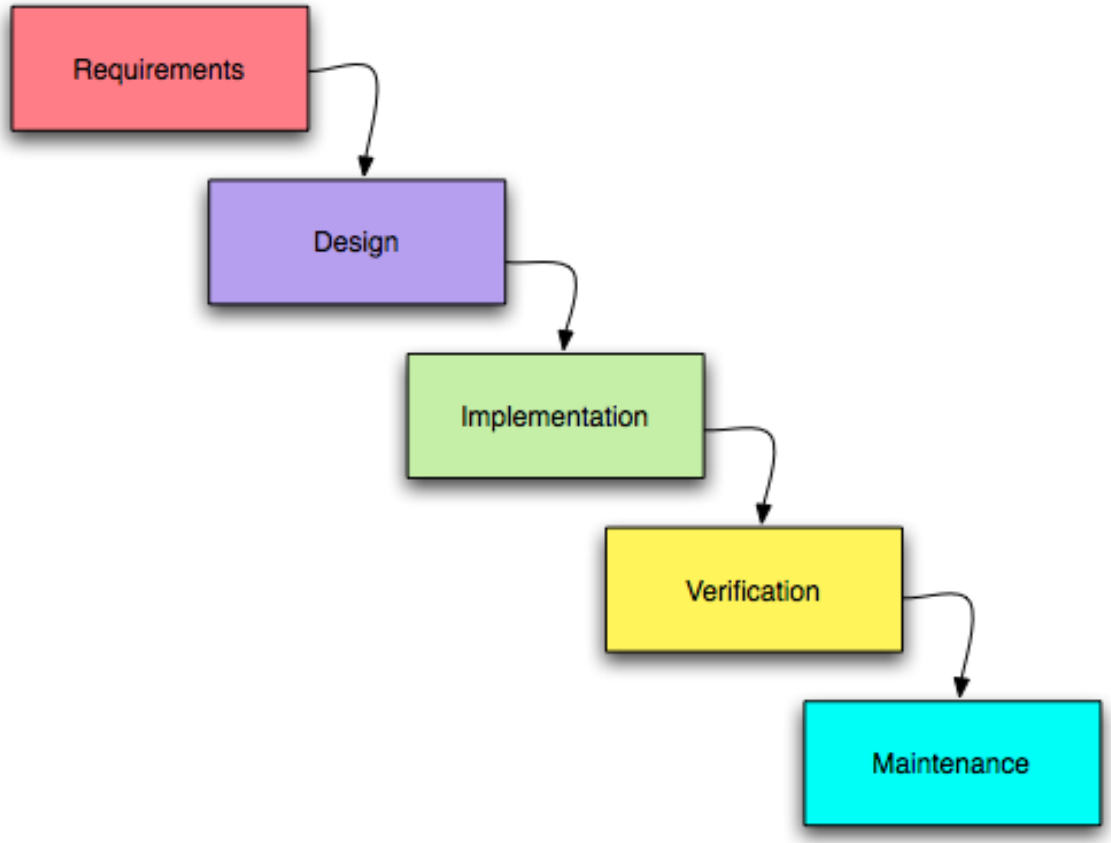
DTU Compute
Technical University of Denmark

Spring 2013

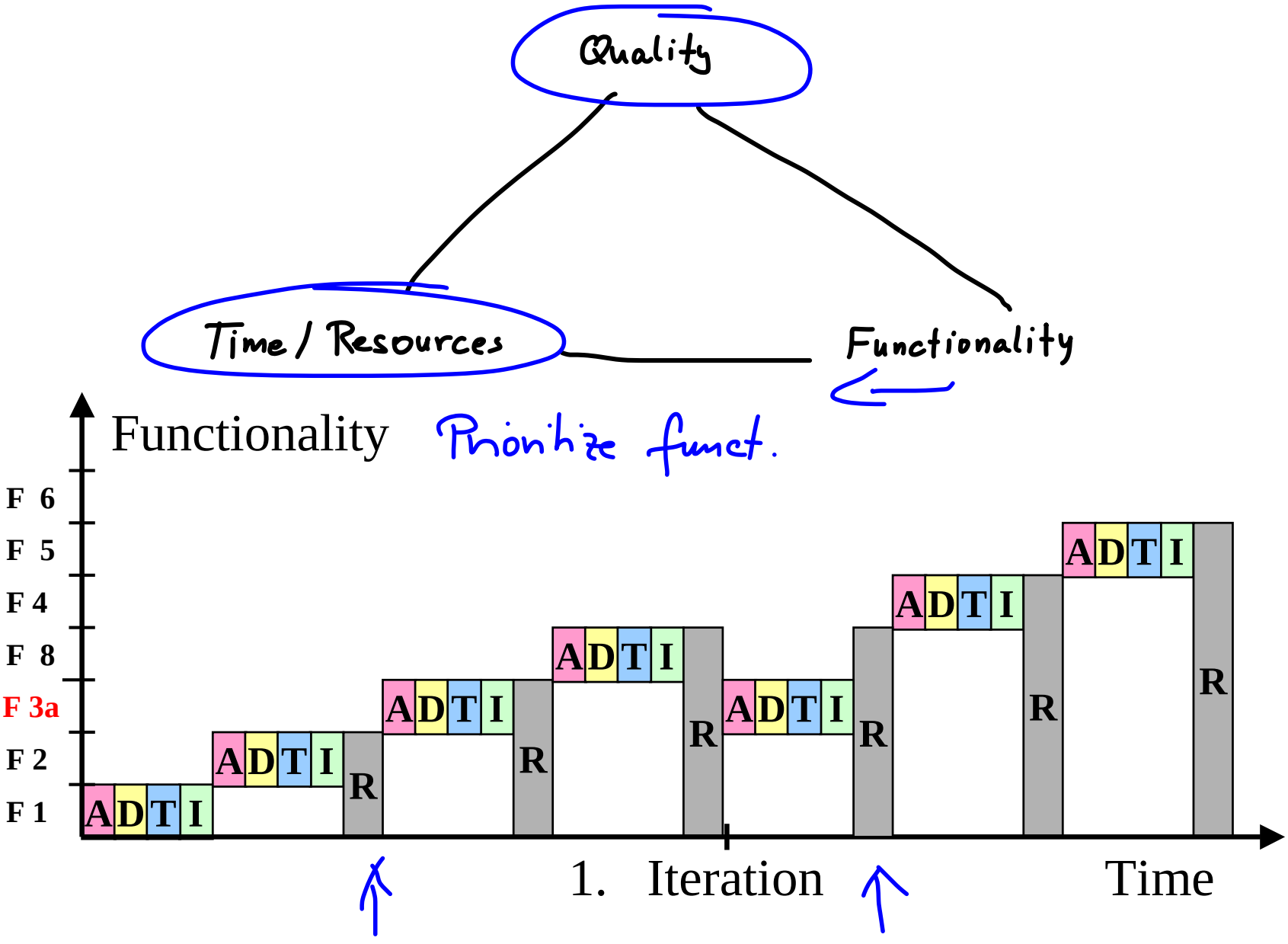
Contents

Resource Triangle: Waterfall





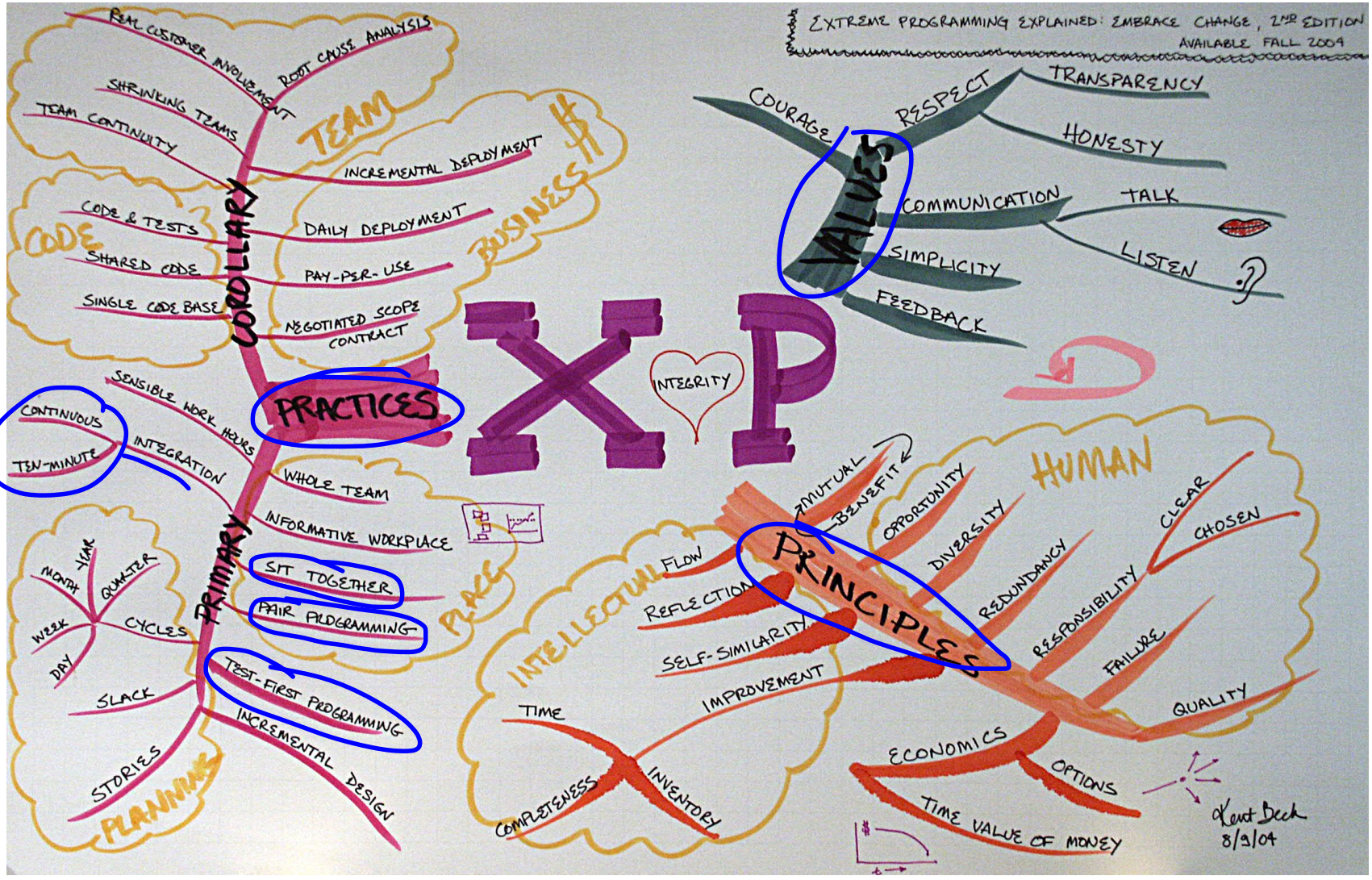
Resource Triangle: Agile



Agile processes

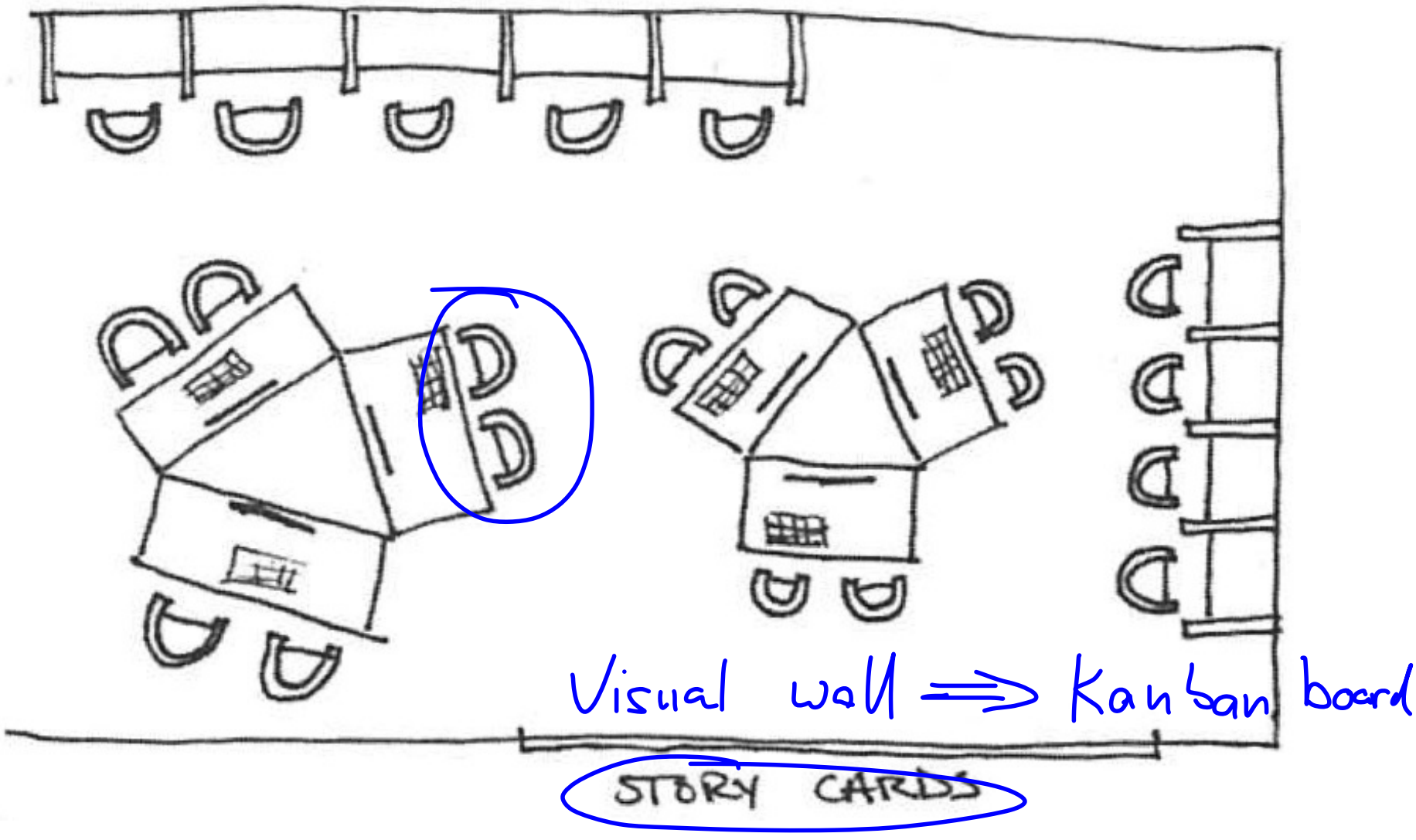
- Agile software development methods
  - Extreme Programming
  - Scrum
  - Lean Software Development
  - Kanban
- Common characteristic
  - Short iterations: Each iteration produces a software increment
    - = Small batch sizes
      Ideal batch size: one (single piece flow)
  - Driven by user stories/Backlog items/smallest marketable feature/...

eXtreme Programming (XP)



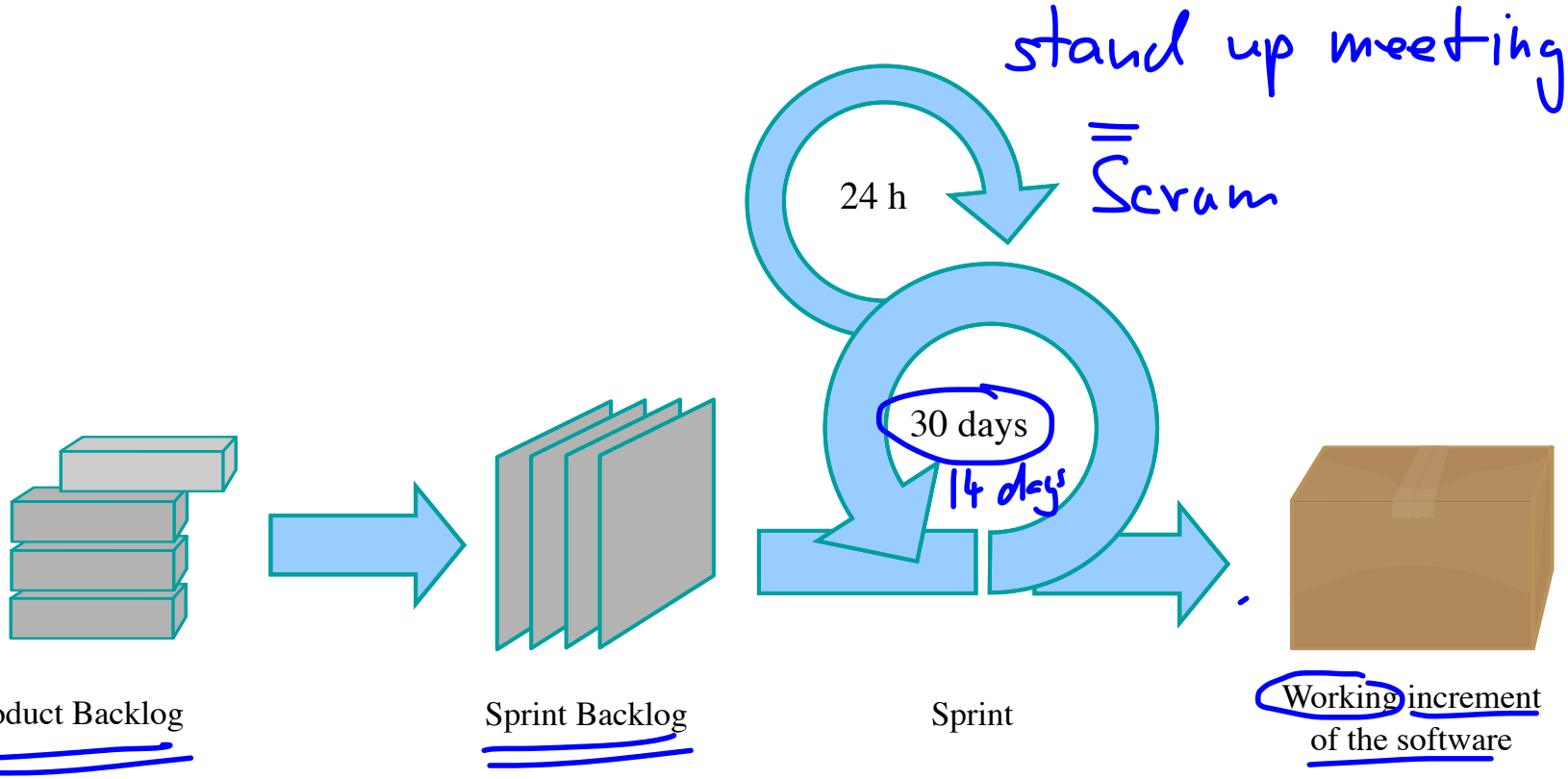
Kent Beck, Extreme Programming 2nd ed.

Sit-together



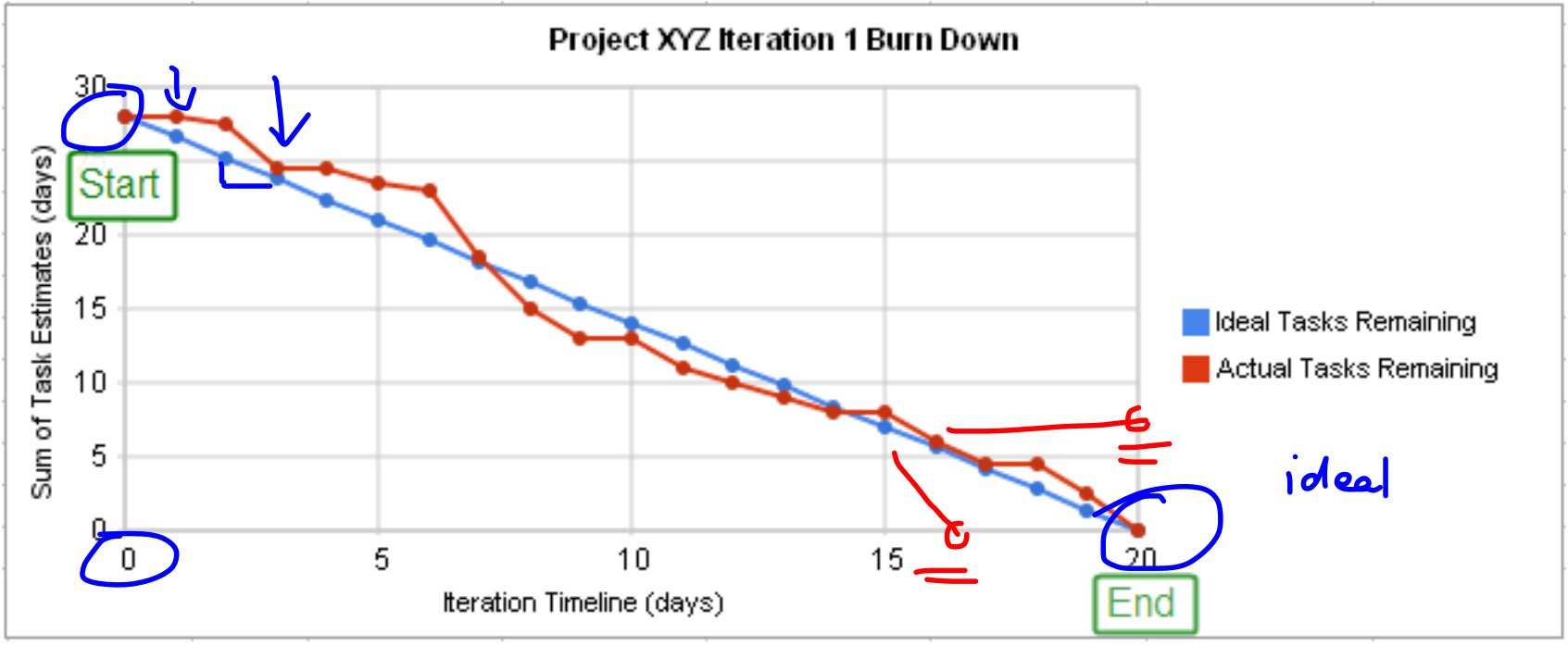
Kent Beck, Extreme Programming 2nd ed.

Scrum



Usually practices are from XP

Burn Down Charts



Wikipedia

Lean Software Development

- Lean Production:
  - Reduce the amount of *waste*
  - Generate *flow* → streamlined processes
- 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 → cannot be avoided always
  - …

Cycle time

Cycle time

Time it takes to go through the process one time

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

Batch size = number_of_features in an iteration

Cycle time: Waterfall

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

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

- Waterfall: cycle_time = 250 / 5 = 50 weeks

→ 1 cycle

- Question: How to reduce the cycle time?
  - → Get feedback from the process

Reducing the cycle time

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

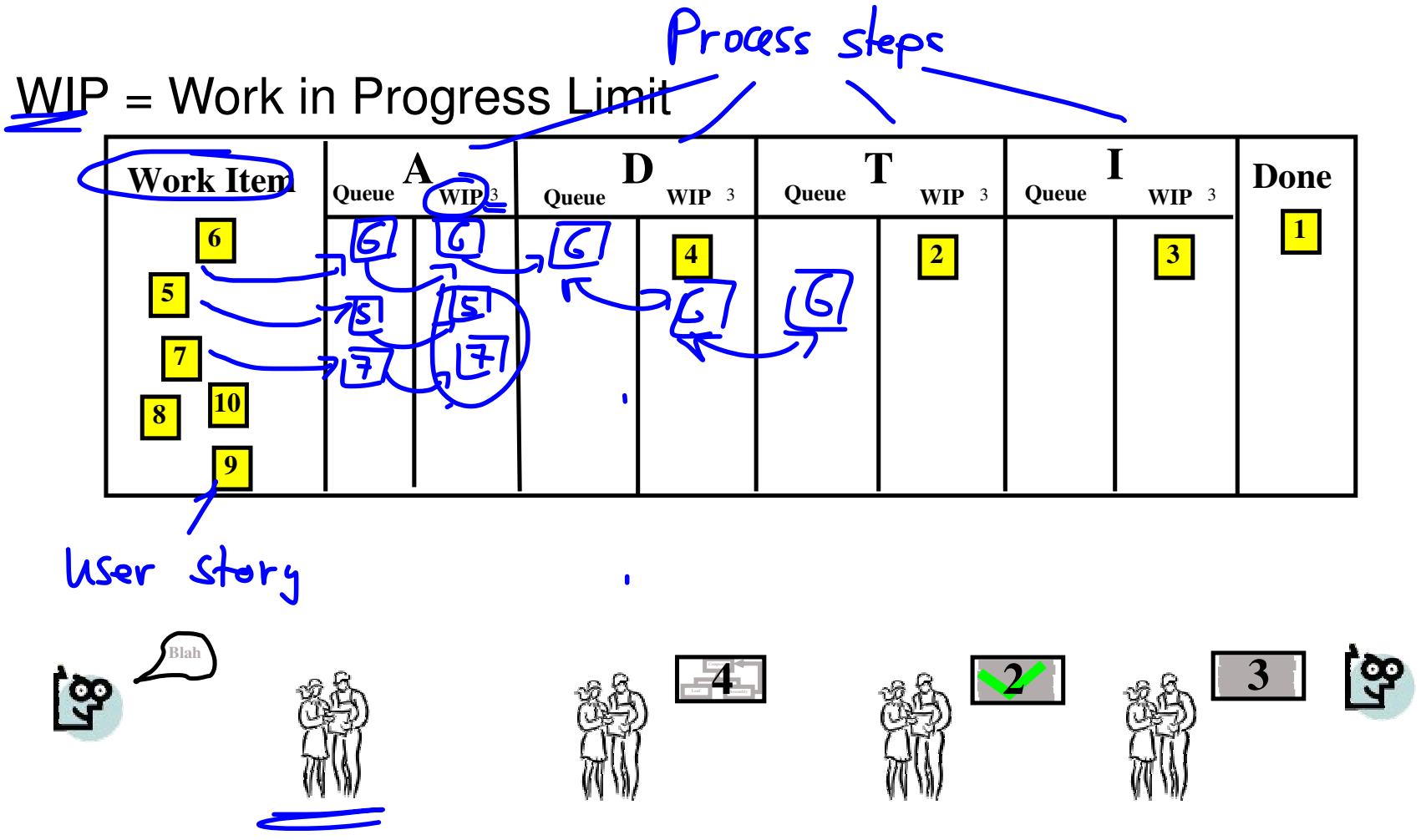
$$cycle\_time = \frac{number\_of\_features}{feature\_implemantion\_rate}$$

- Agile: cycle_time = 1 / 5 = 8 hours

→ 250 cycles

→ Process improvement: incease in features / week

Generating flow using Pull and Kanban



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

- Visual rep. of the process: Discover problems

Figure from David Anderson www.agilemanagement.net

Online Tool

- `www.targetprocess.com`: Electronic Kanban board useful for your project

Contents

From Requirements to Design

Design process

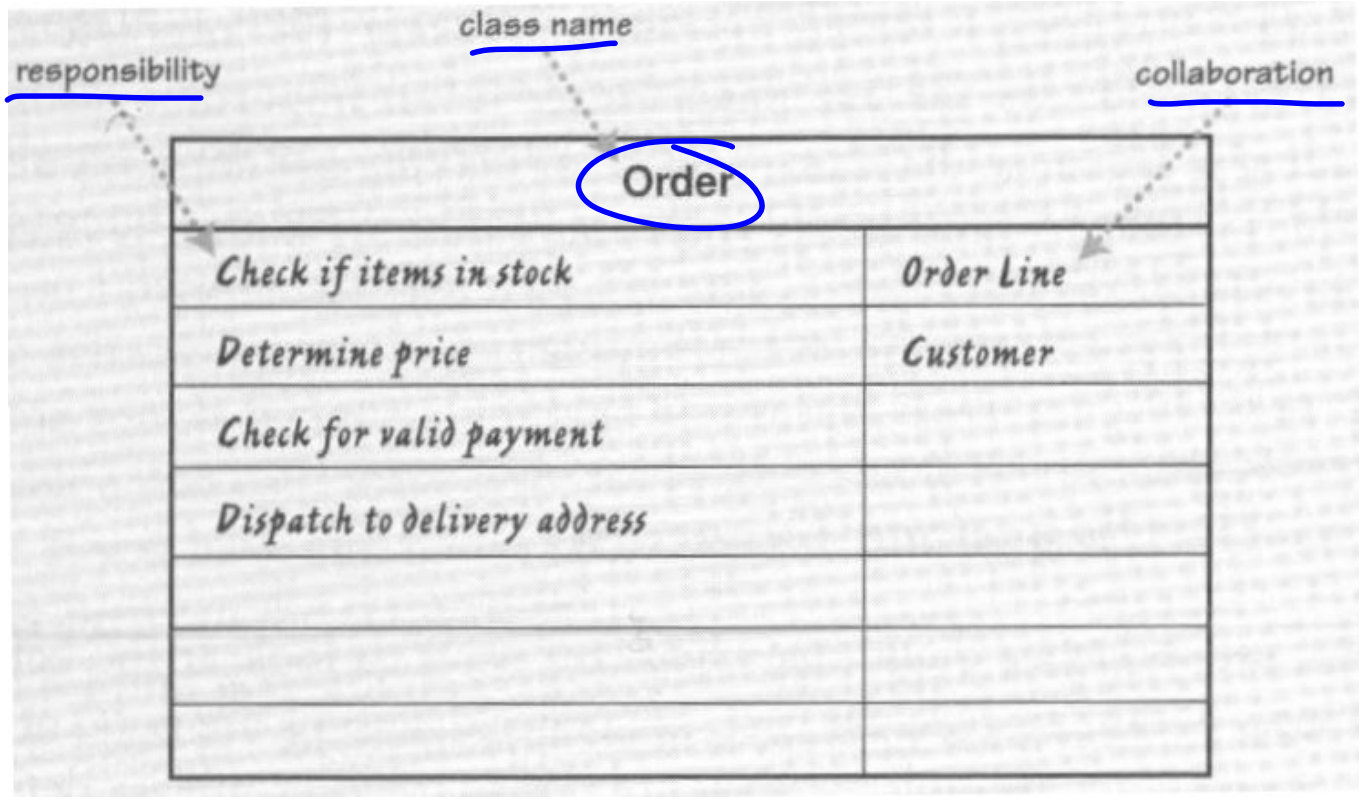
1. Glossary/architecture: possible classes, attributes, and operations
2. Take one use case scenario / user story
   a) Devise *a test* for the scenario
   b) *Realize* that scenario by adding new classes, attributes, associations, and operations so that you **design** can execute that scenario
   c) **implement**
3. Repeat step 2 with the other use case scenarios / user stories

UML Class diagram

Introduction CRC Cards

- Class Responsibility Collaboration
- Developed in the 80's by Kent Beck and Ward Cunningham
- Used to
  - Analyse a problem domain
  - Discover object-oriented design
  - Teach object-oriented design
- Object-oriented design:
  - Objects have state and behaviour
  - Objects delegate responsibilities
  - "Think objects"

CRC Card Template



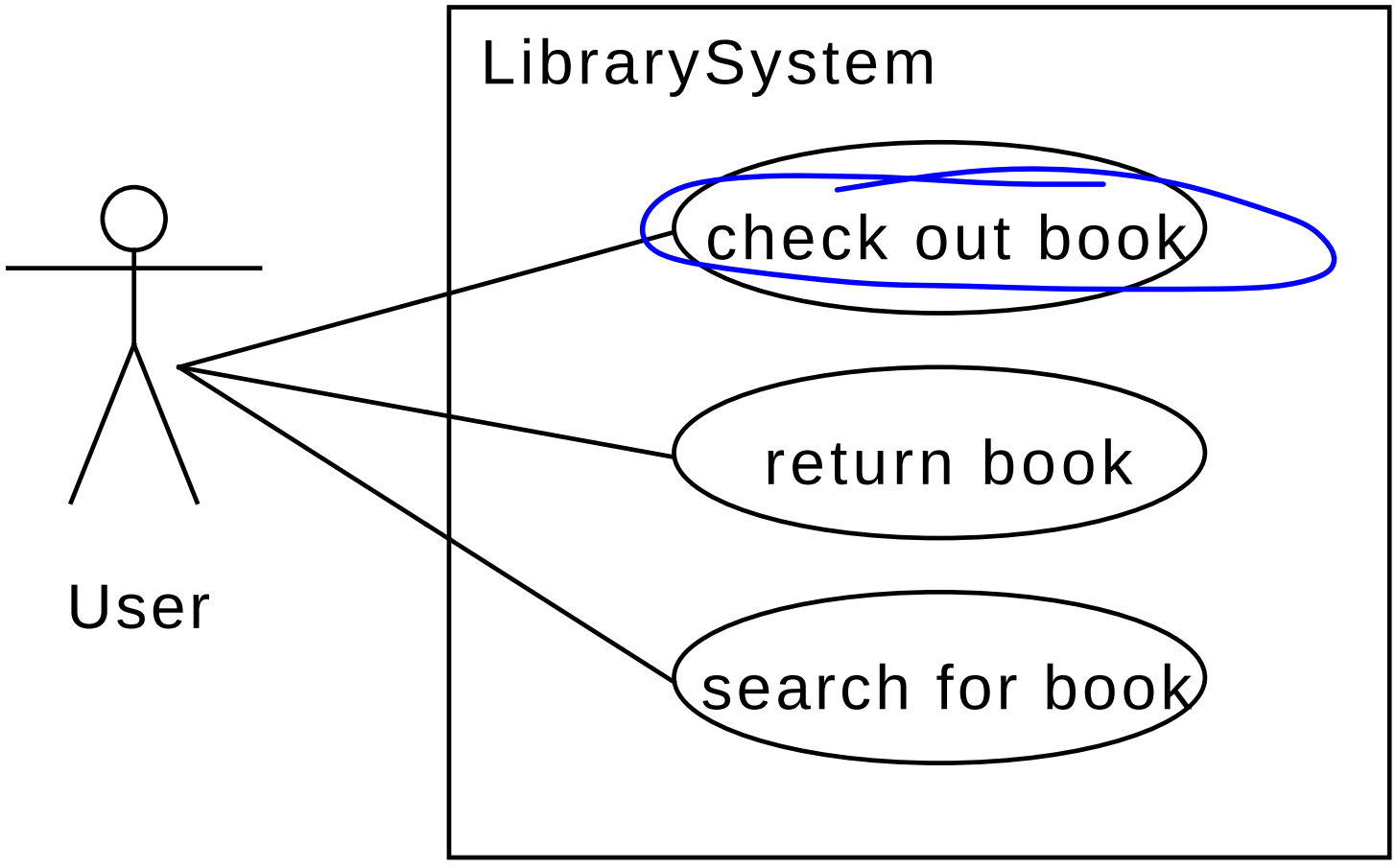
A larger example

- `http://c2.com/doc/crc/draw.html`

Process

- Basic: Simulate the execution of use case scenarios / user stories
- Steps
  1. Brainstorm classes/objects/components
  2. Assign classes/objects/components to persons (group up to 6 people)
  4. Execute the scenarios one by one
     a) add new classes/objects/components as needed
     b) add new responsibilities
     c) delegate to other classes / persons

Library Example: Use Case Diagram



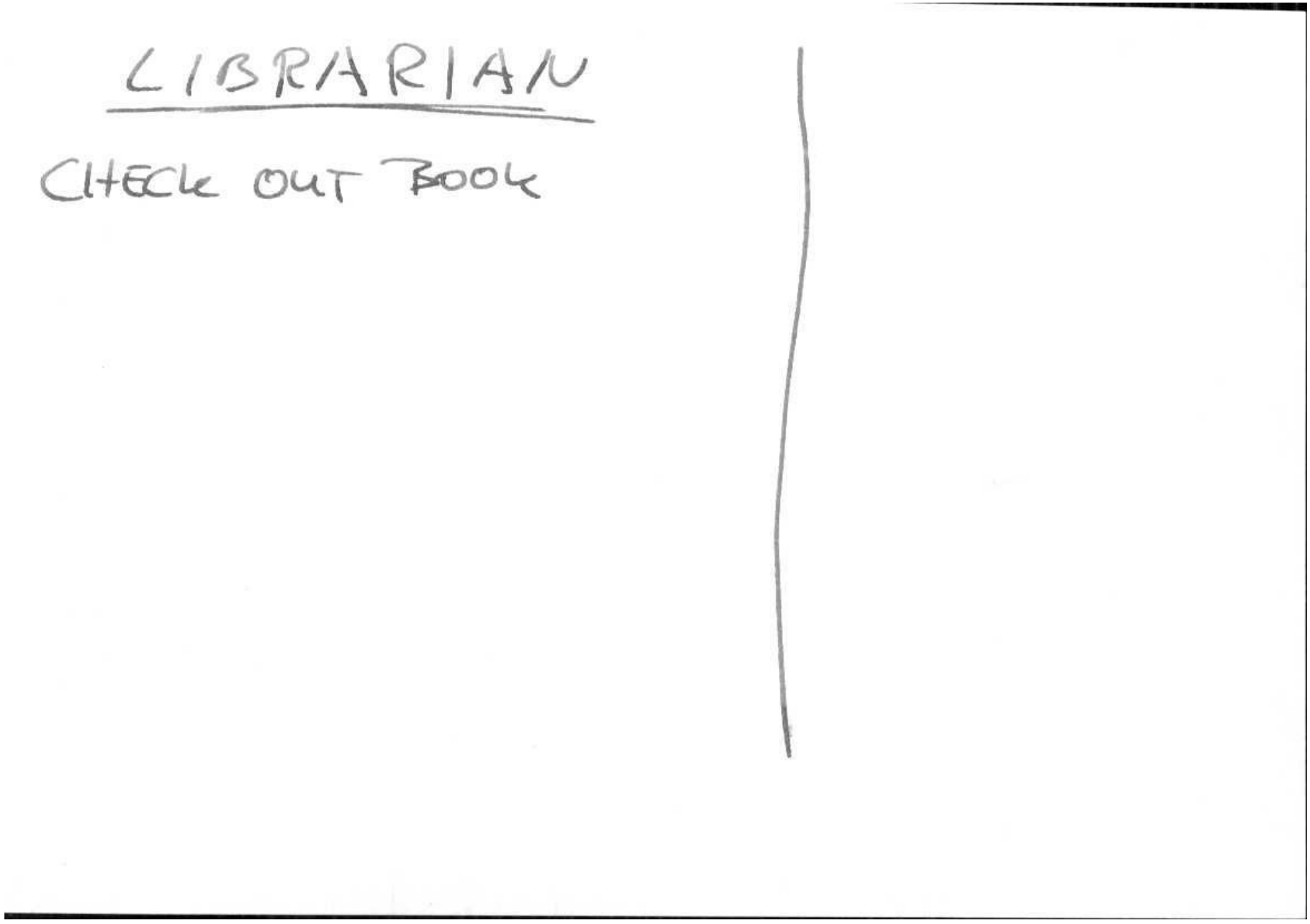
Library Example: Detailed Use Case *Check Out Book*

- **Name:** Check Out Book
- **Description:** The user checks out a book from the library
- **Actor:** User
- **Main scenario:**
  1. A user presents a book for check-out at the check-out counter
  2. The system registers the loan
- **Alternative scenarios:**
  - The user already has 5 books borrowed
    - 2a The system denies the loan
  - The user has one overdue book
    - 2b The system denies the loan

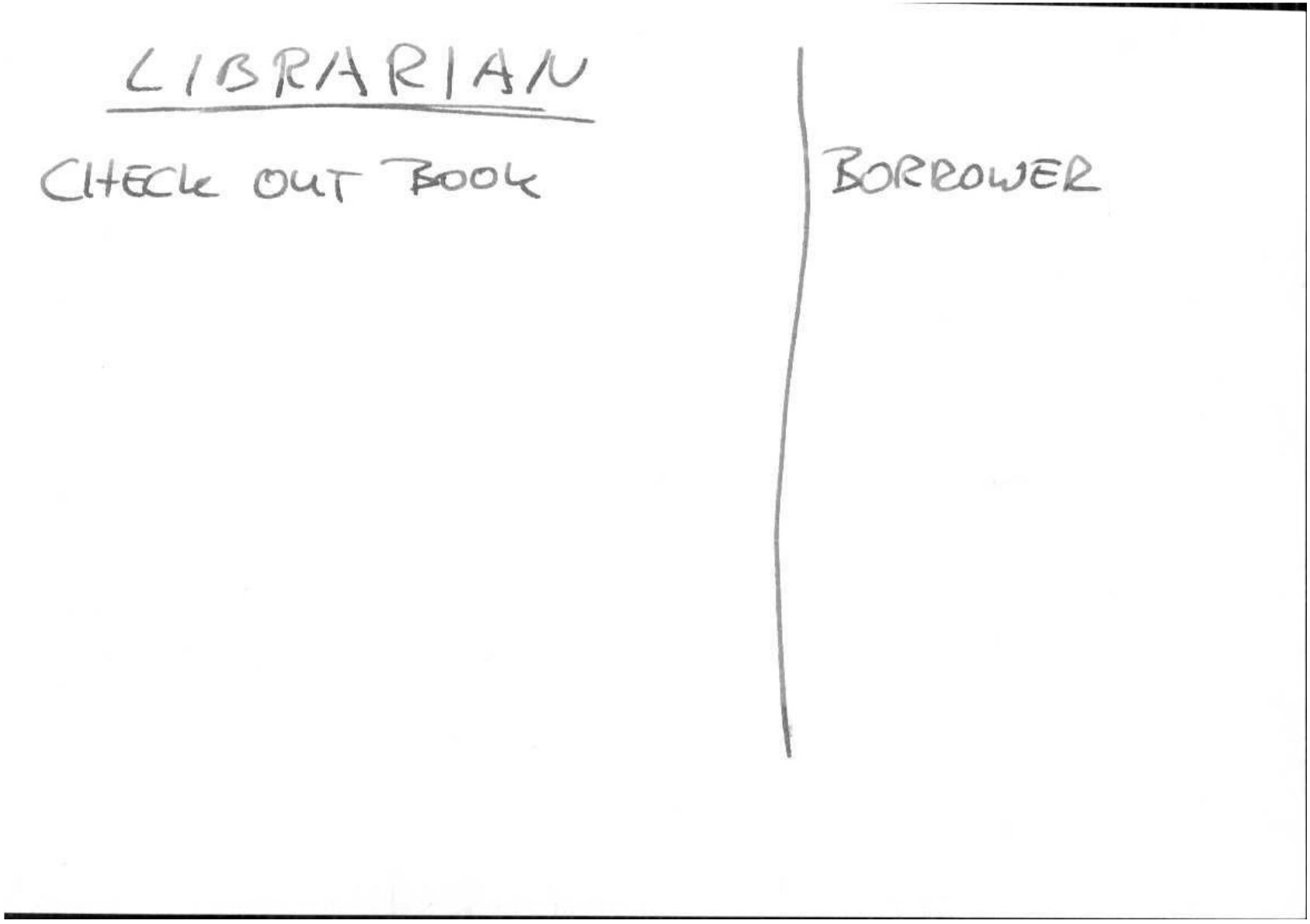
Example II

- Set of initial CRC cards: Librarien, Borrower, Book
- Use case **Check out book** main scenario (user story)
  - ”What happens when Barbara Stewart, who has no accrued fines and one outstanding book, not overdue, checks out a book entitled Effective C++ Strategies+?”

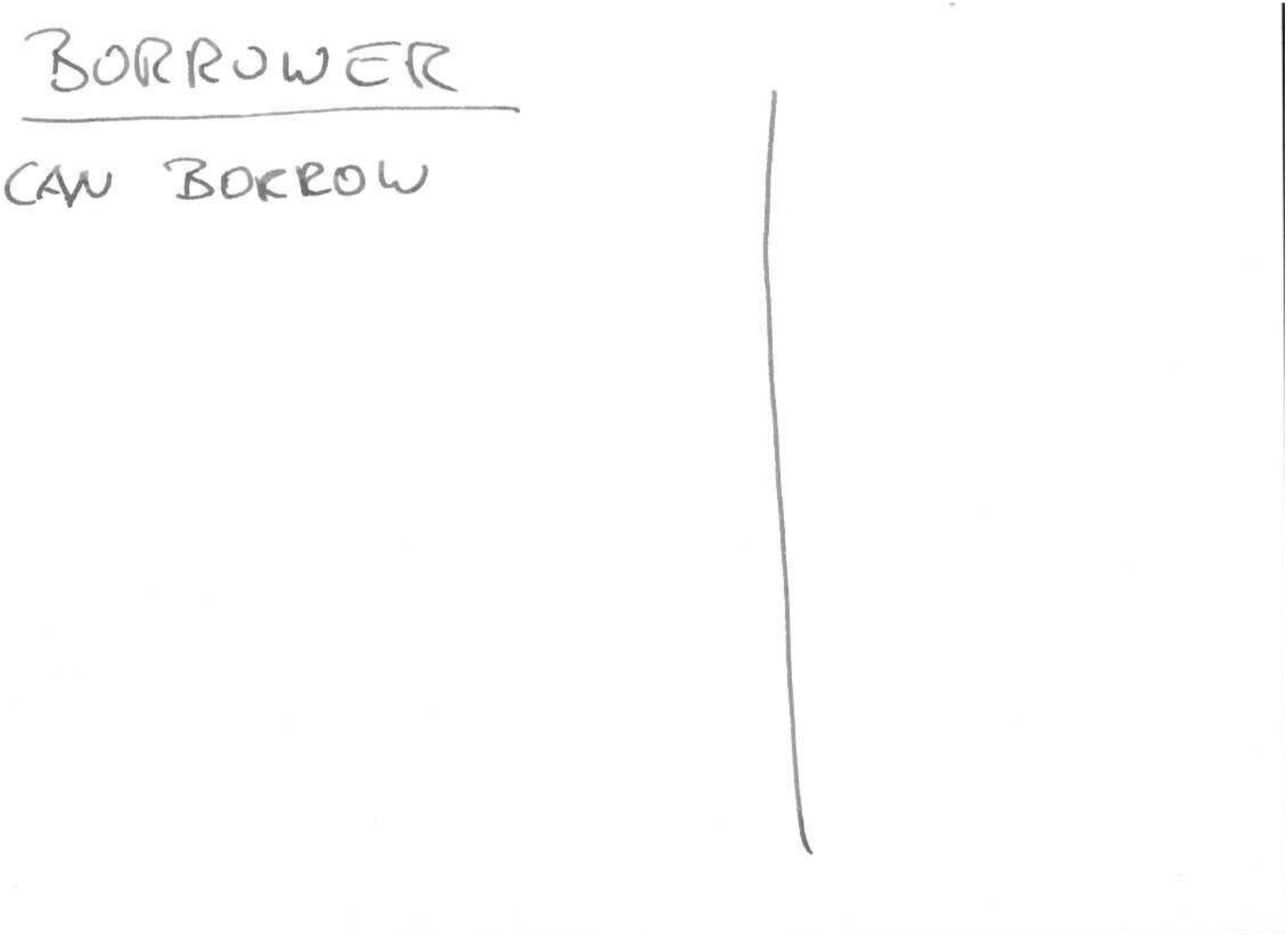
Library Example: CRC cards



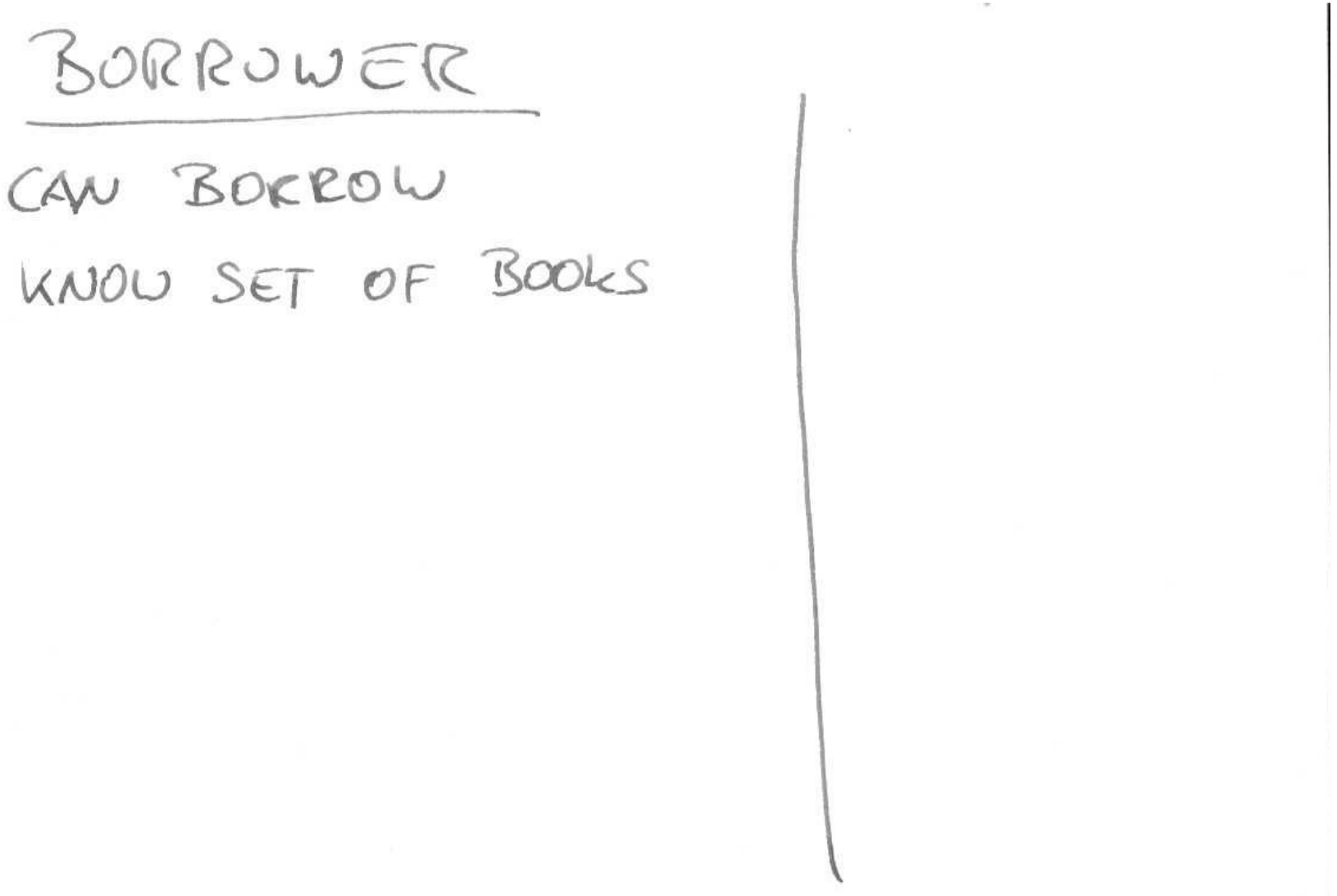
Library Example: CRC cards



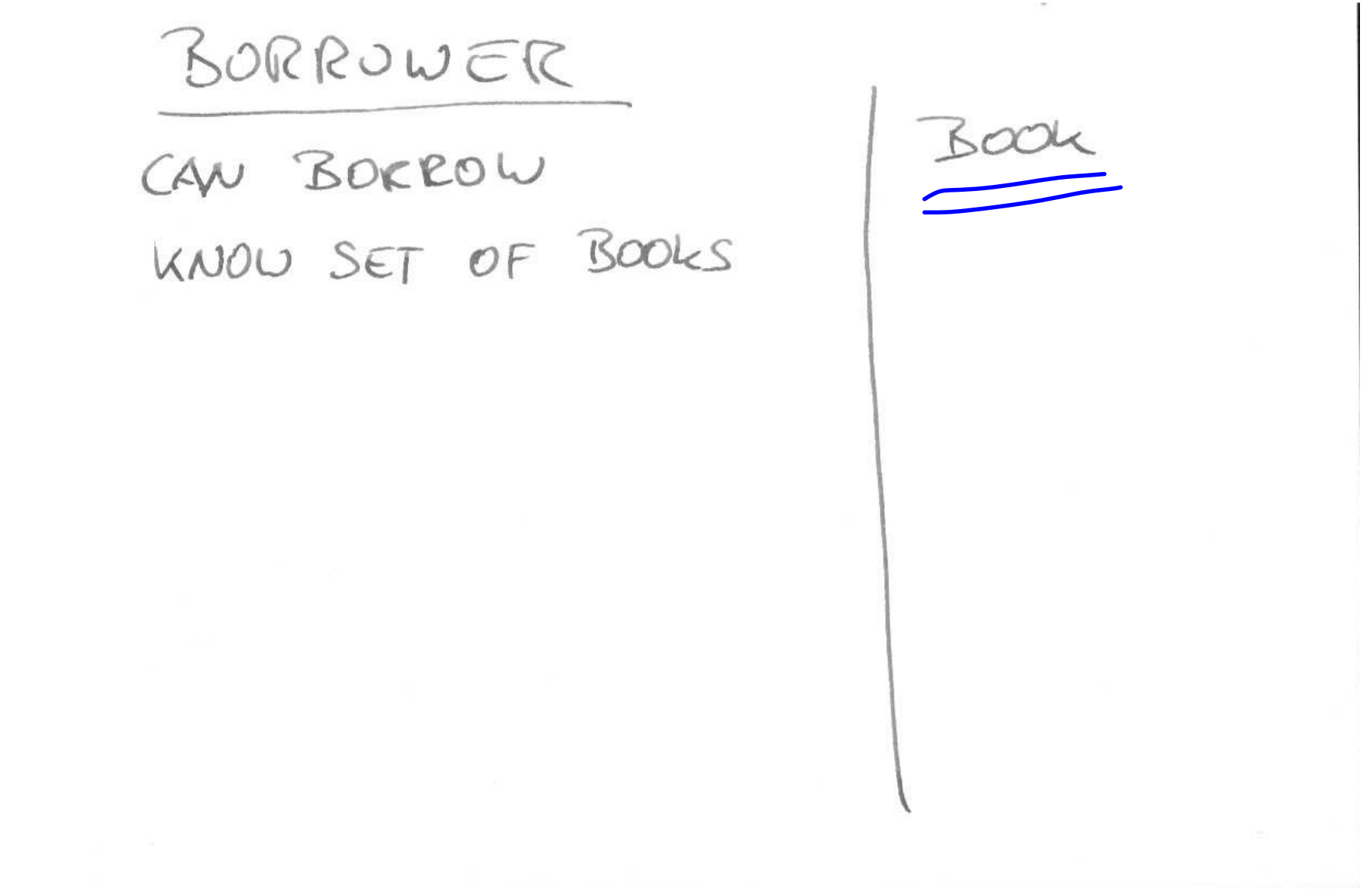
Library Example: CRC cards



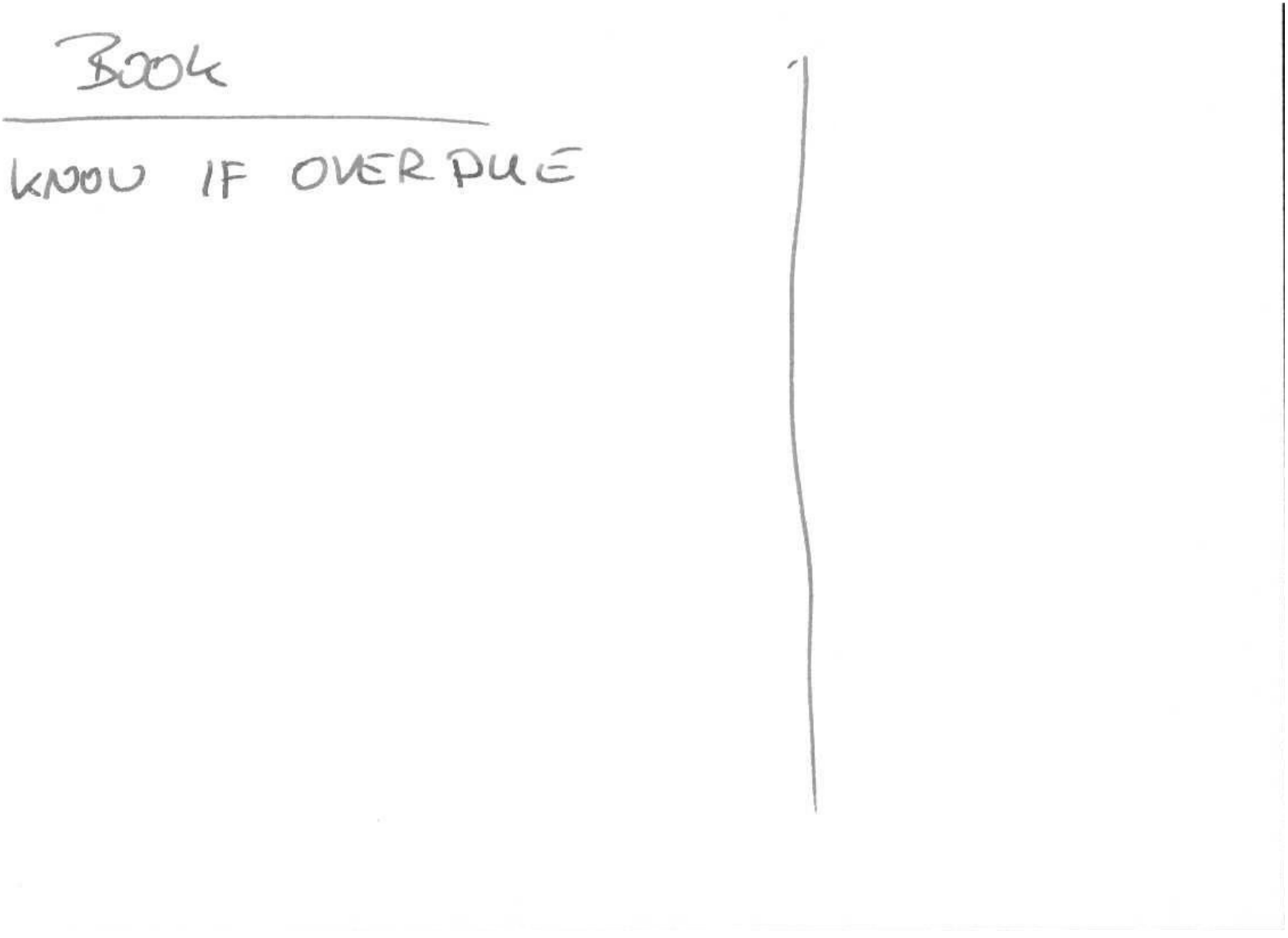
Library Example: CRC cards



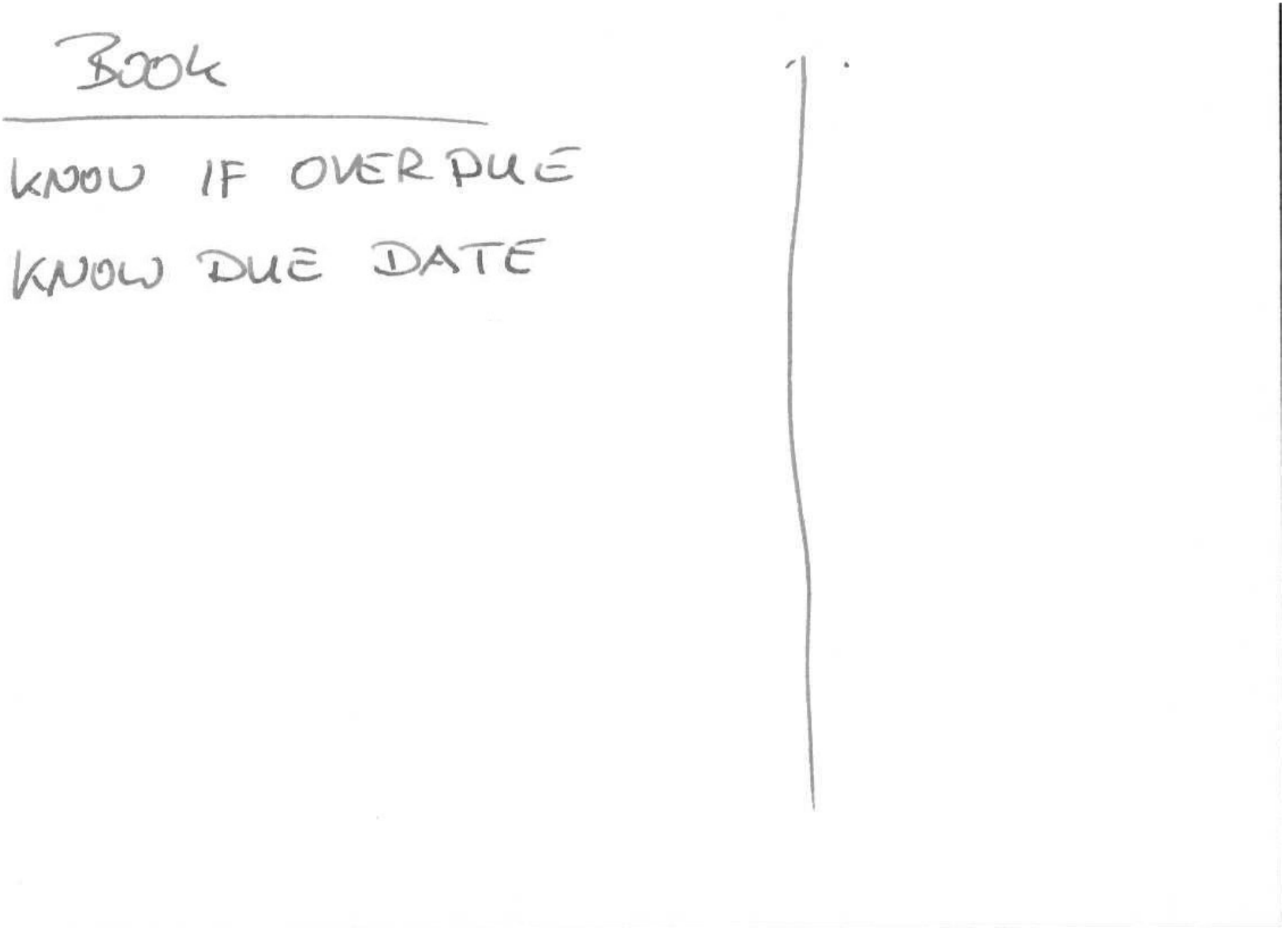
Library Example: CRC cards



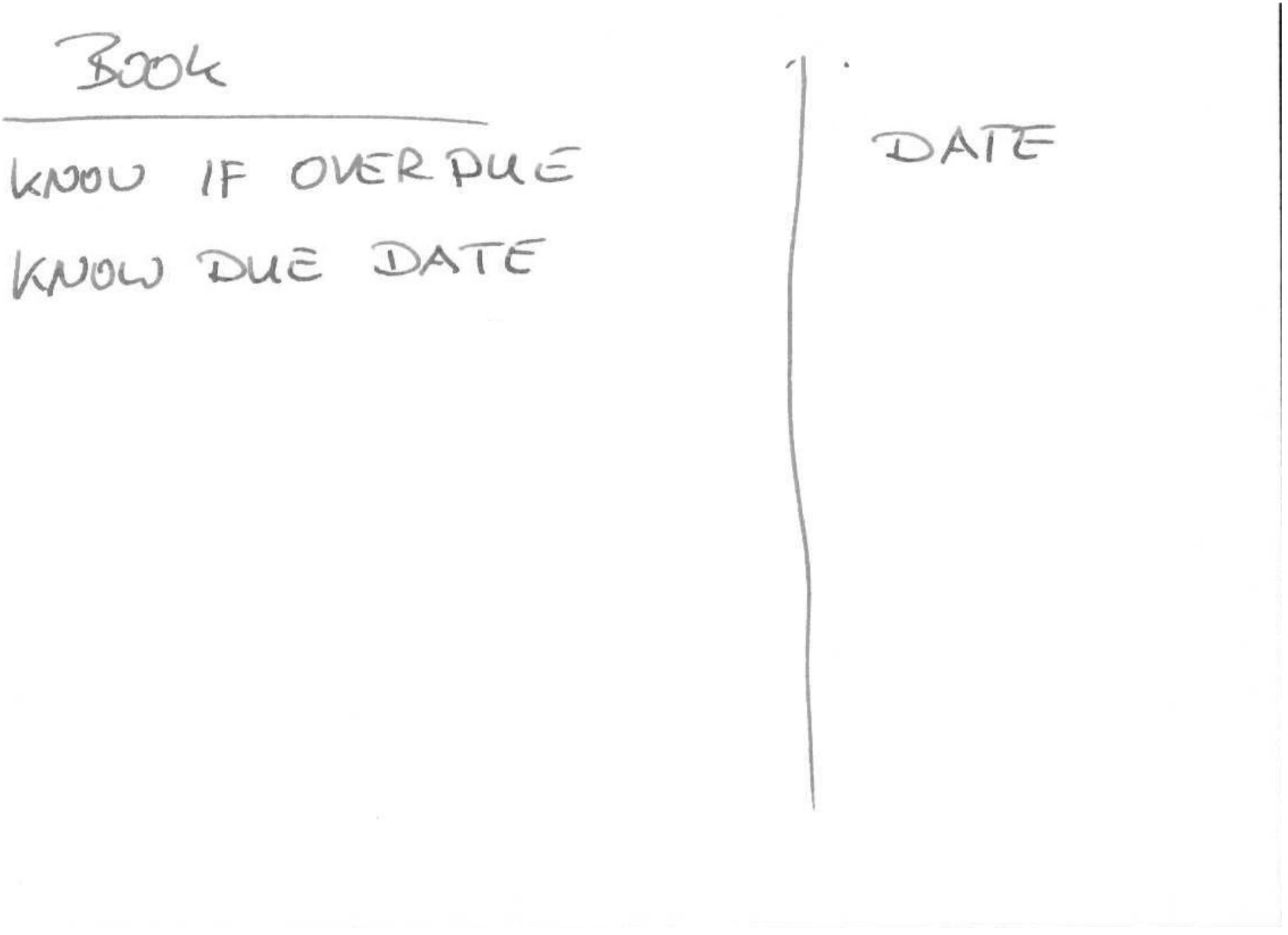
Library Example: CRC cards



Library Example: CRC cards

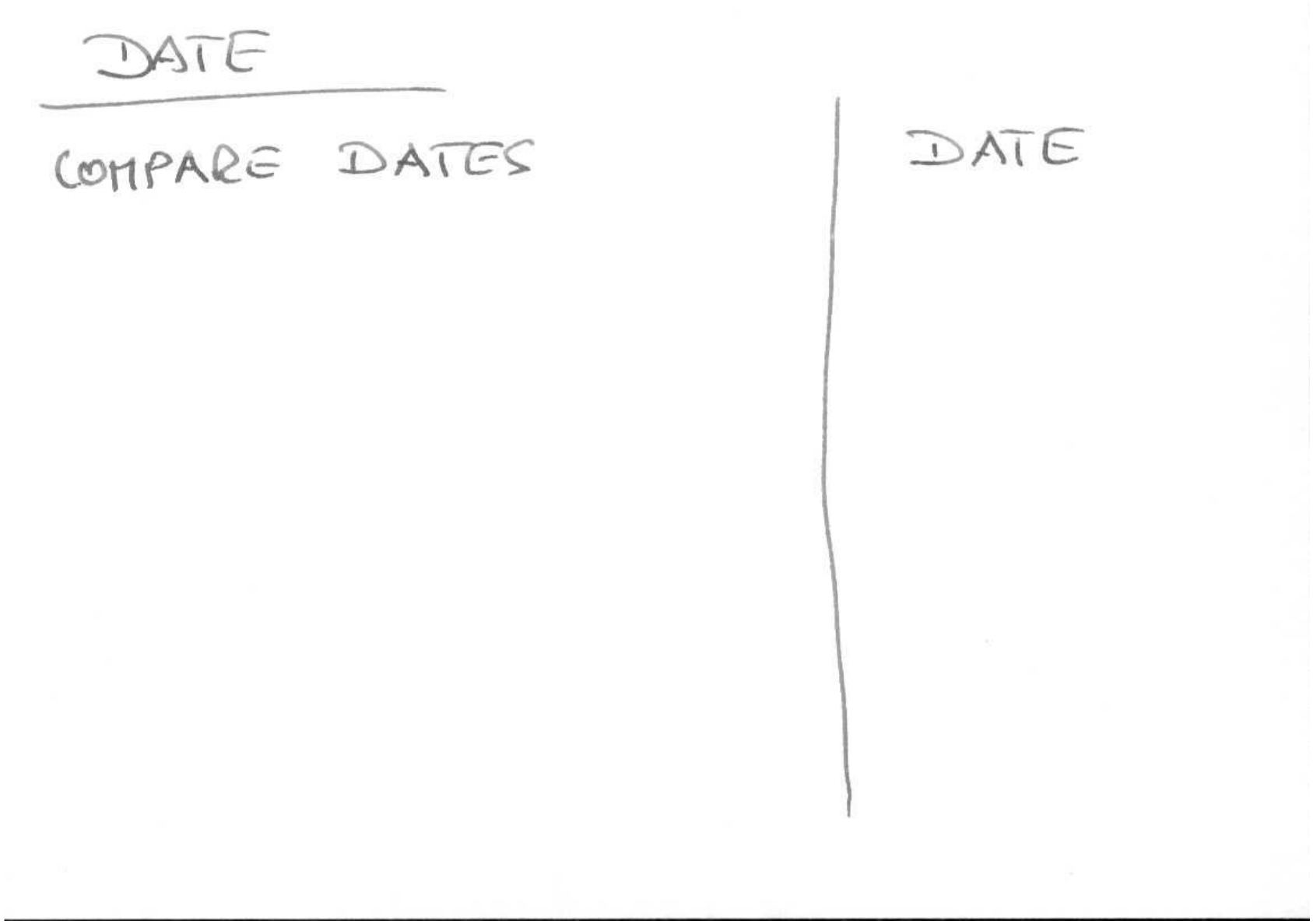


Library Example: CRC cards

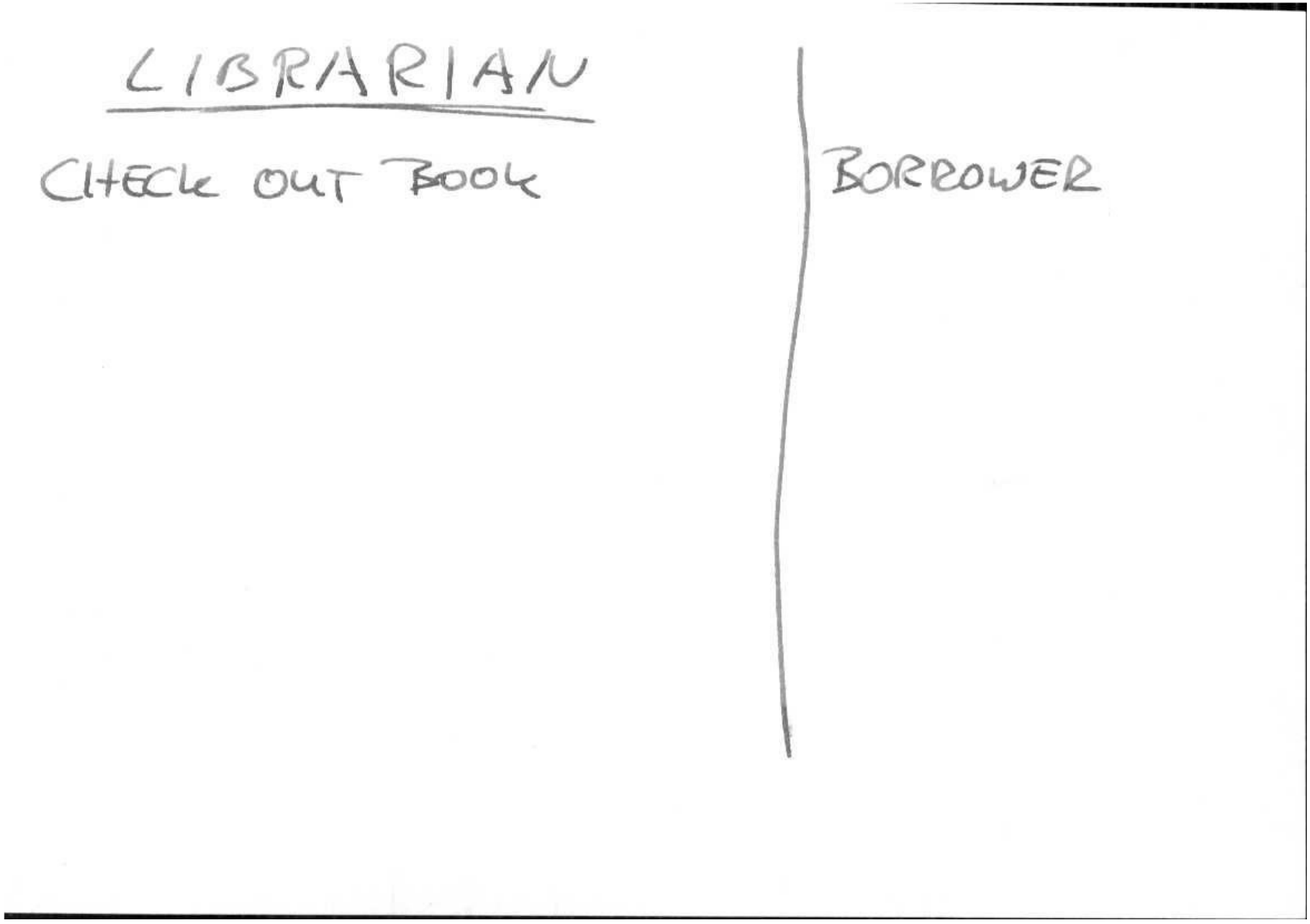


Library Example: CRC cards

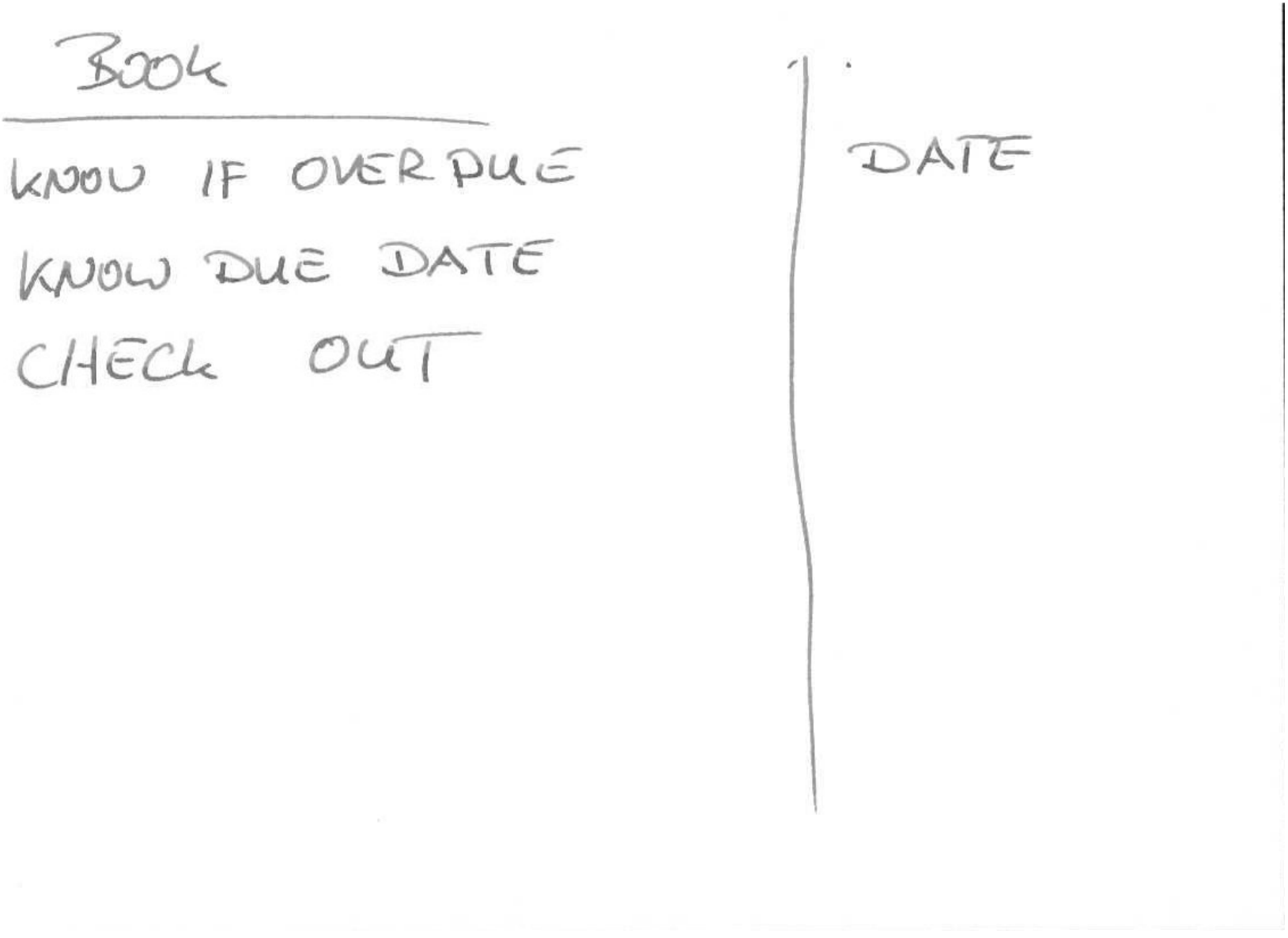
Library Example: CRC cards



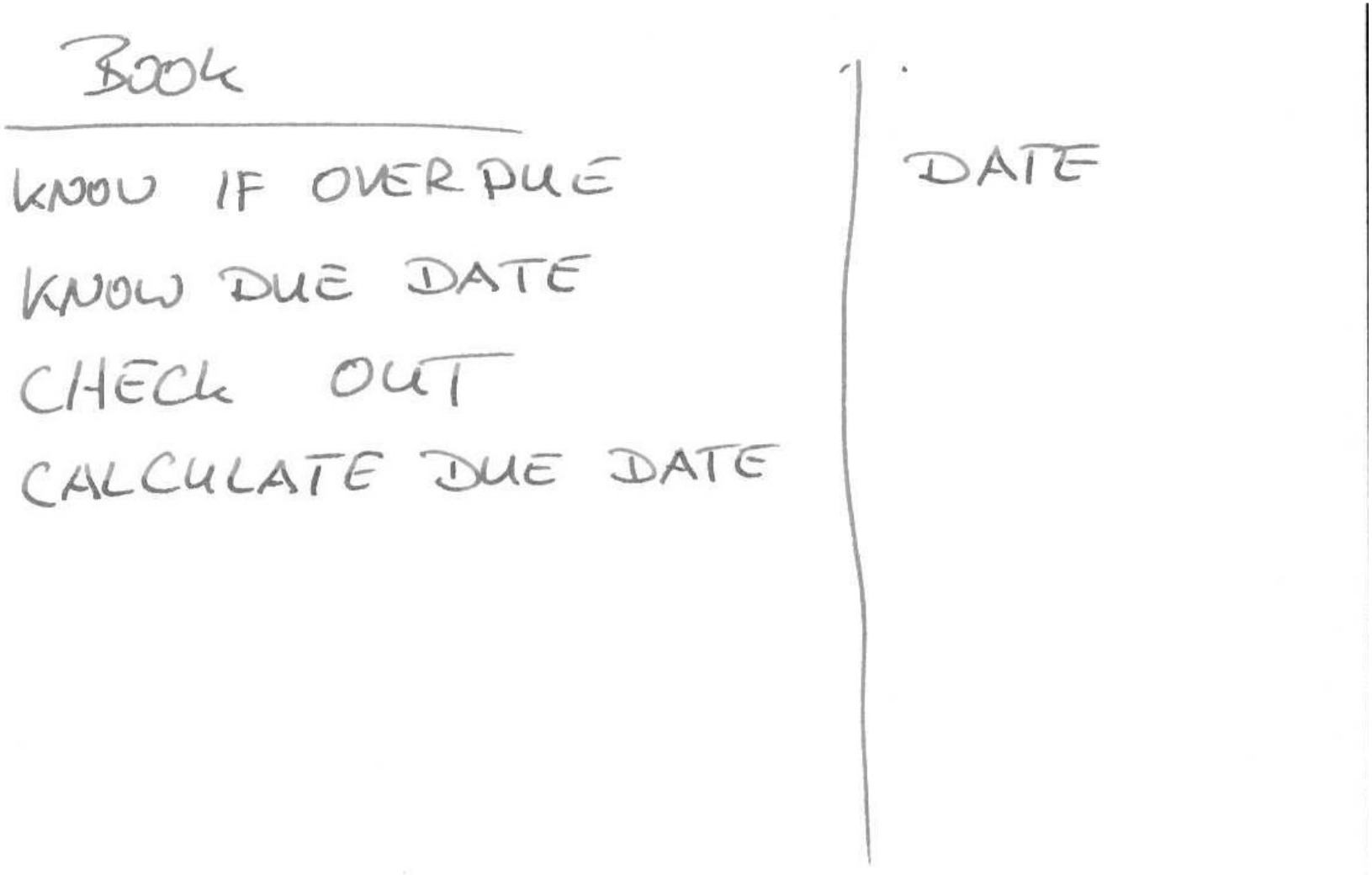
Library Example: CRC cards



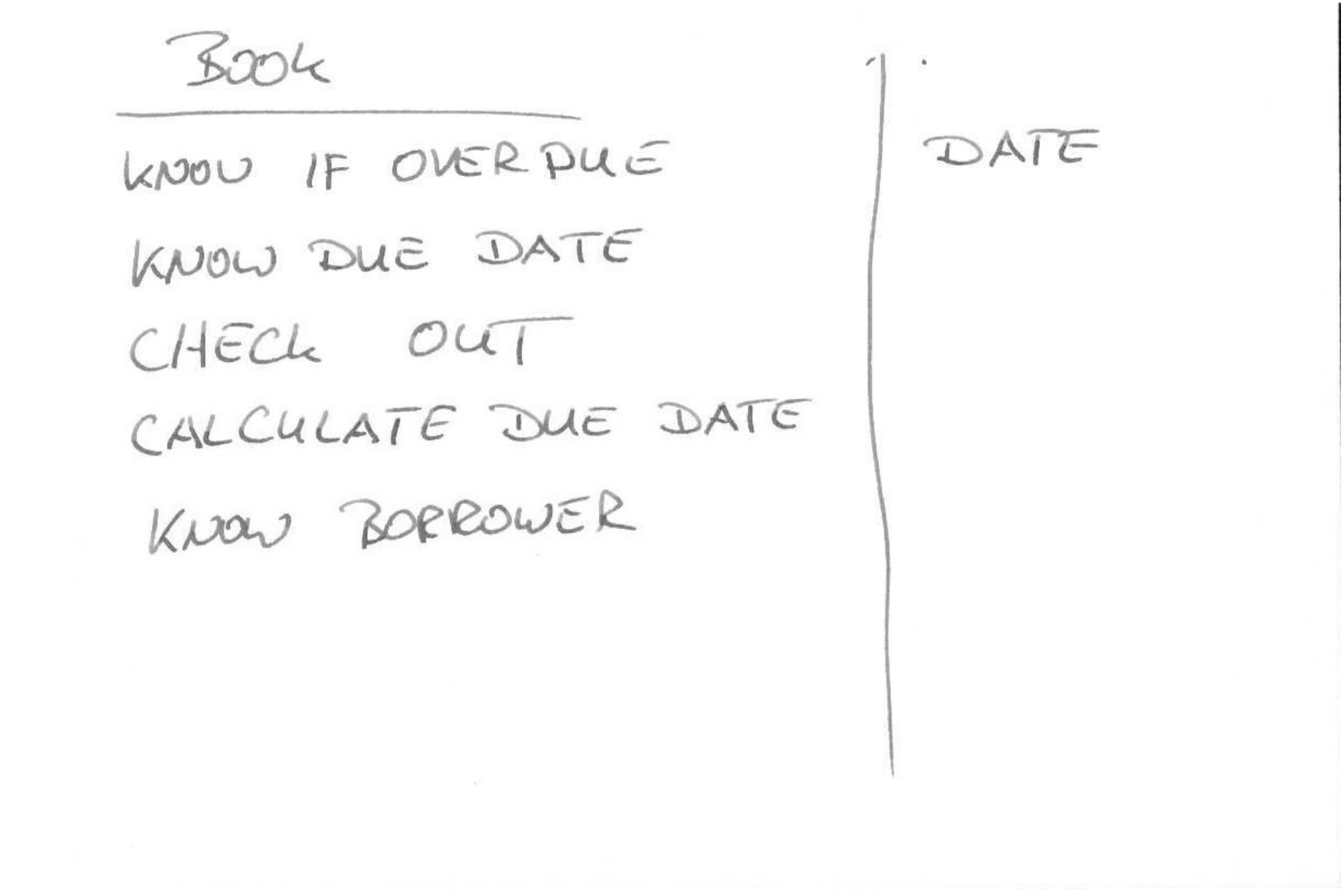
Library Example: CRC cards



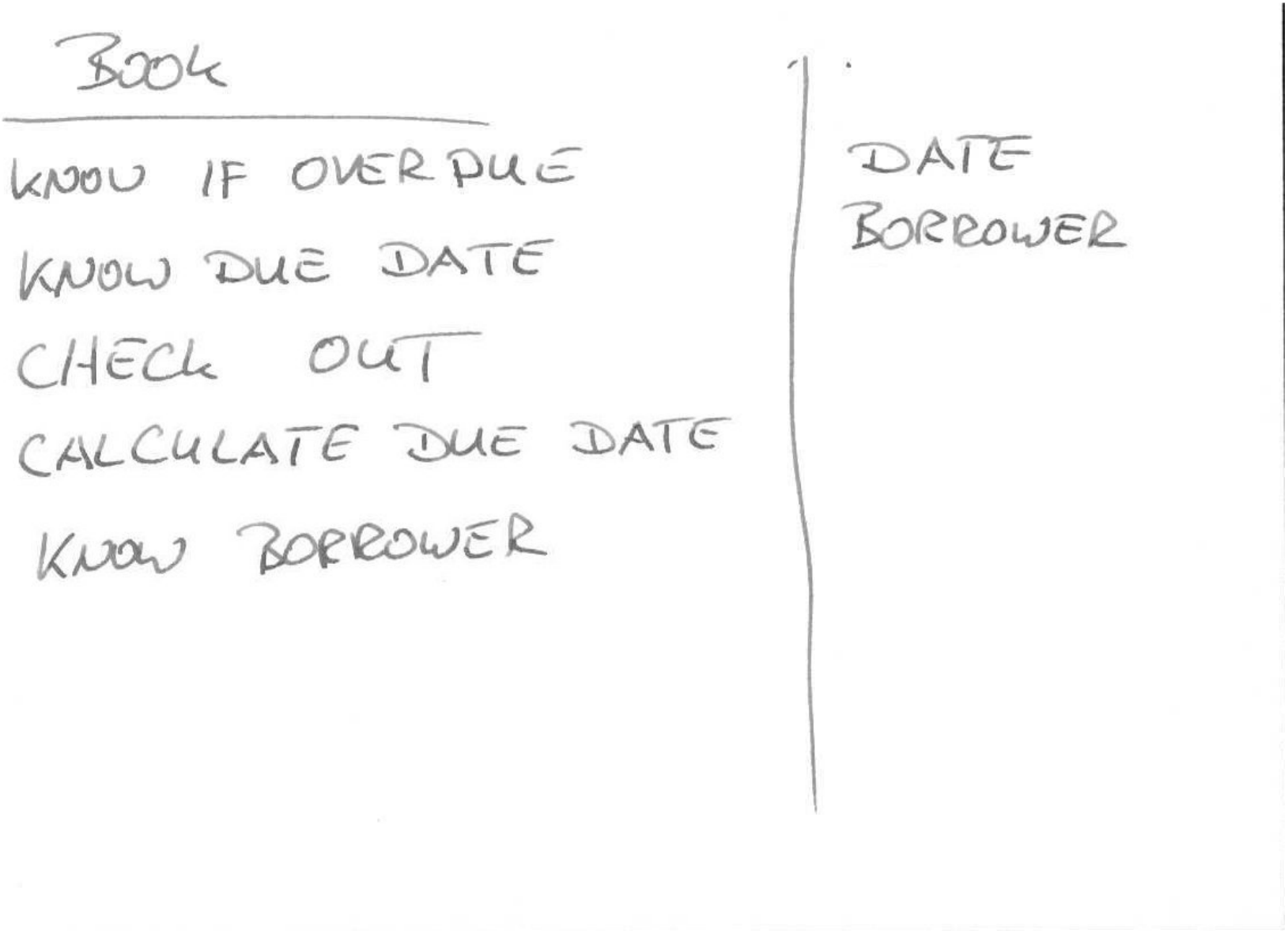
Library Example: CRC cards



Library Example: CRC cards



Library Example: CRC cards



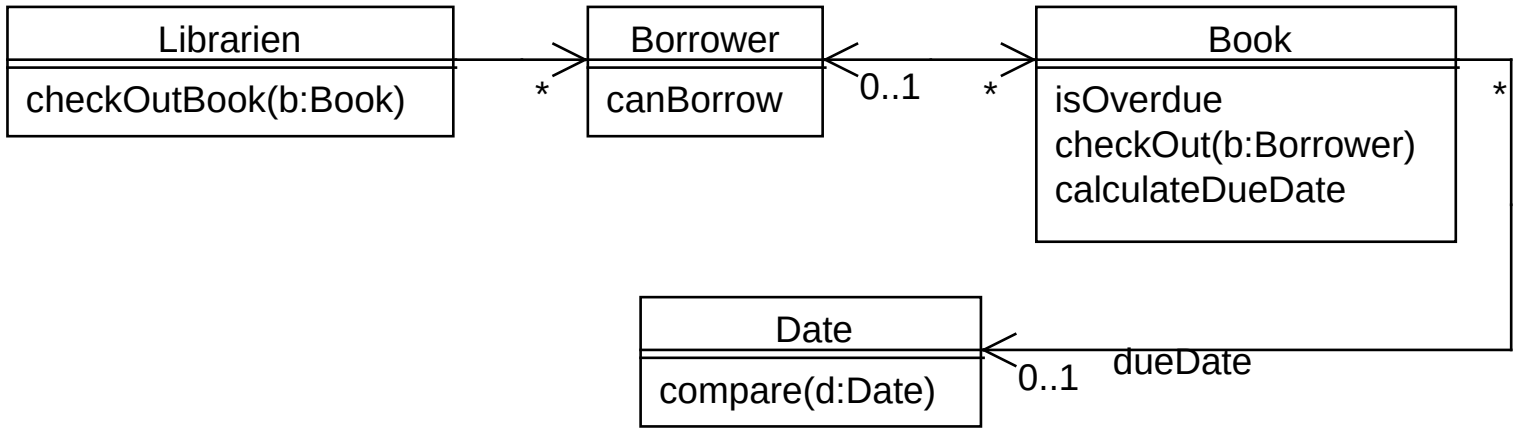
Library Example: All CRC cards

Process: Next Steps

- Review the result
  - Group cards
  - Check cards
  - Refactor
- Transfer the result
  - Implement the design test-driven
  - UML model

→ Class diagram + Sequence diagram

Example: Class Diagram (so far)



- Process: Agile software development
  - Take a user story (according to plan)
  - Create an automatic acceptance test
  - Design the behaviour of the user story using CRC cards
  - Implement the ~~the~~ design test-driven

Contents

What is version control

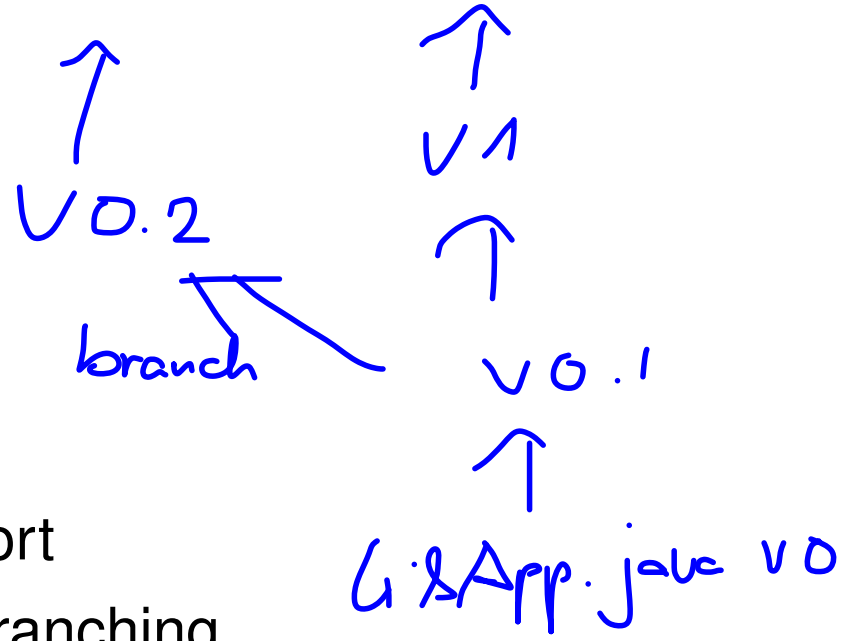
Version Control

- Stores and mangages versions documents (e.g. .java files)
- Manages concurrent work on documents
- Manages different software release versions
- Various systems: CVS, svn, git, ...

Very similar

Supported by DTU

CVS

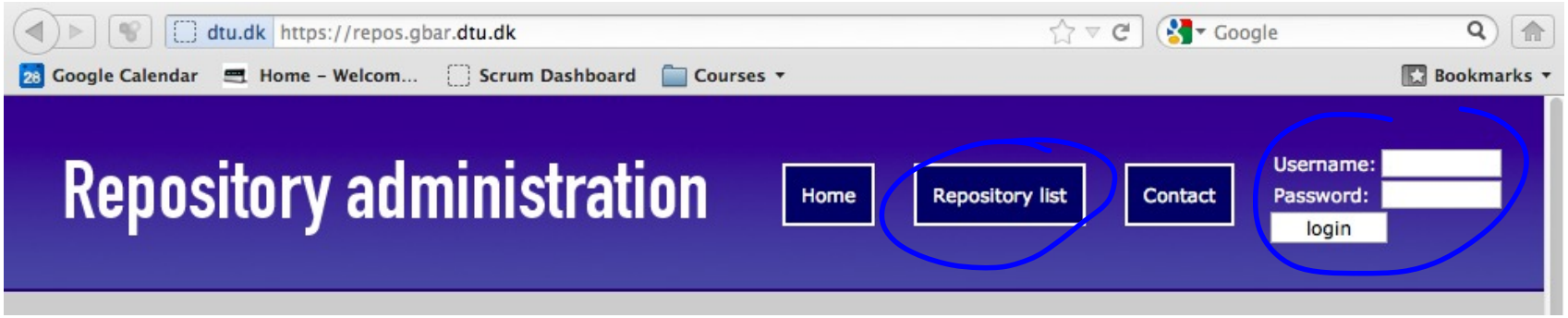


- Concurrent Versions System
- One central repository
- Command line tools, IDE support
- Files have a tree of versions: branching
- Release: File versions having same *tag*
- Versions: **diffs** (differences) to previous versions

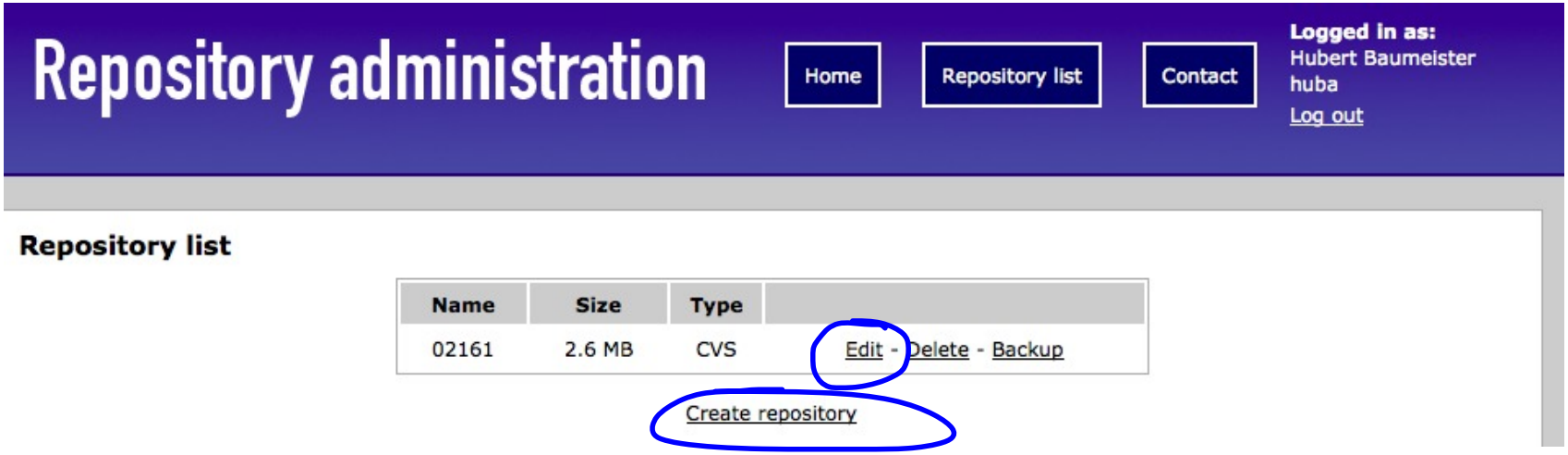
Unix: diff & patch commands

Creating a repository

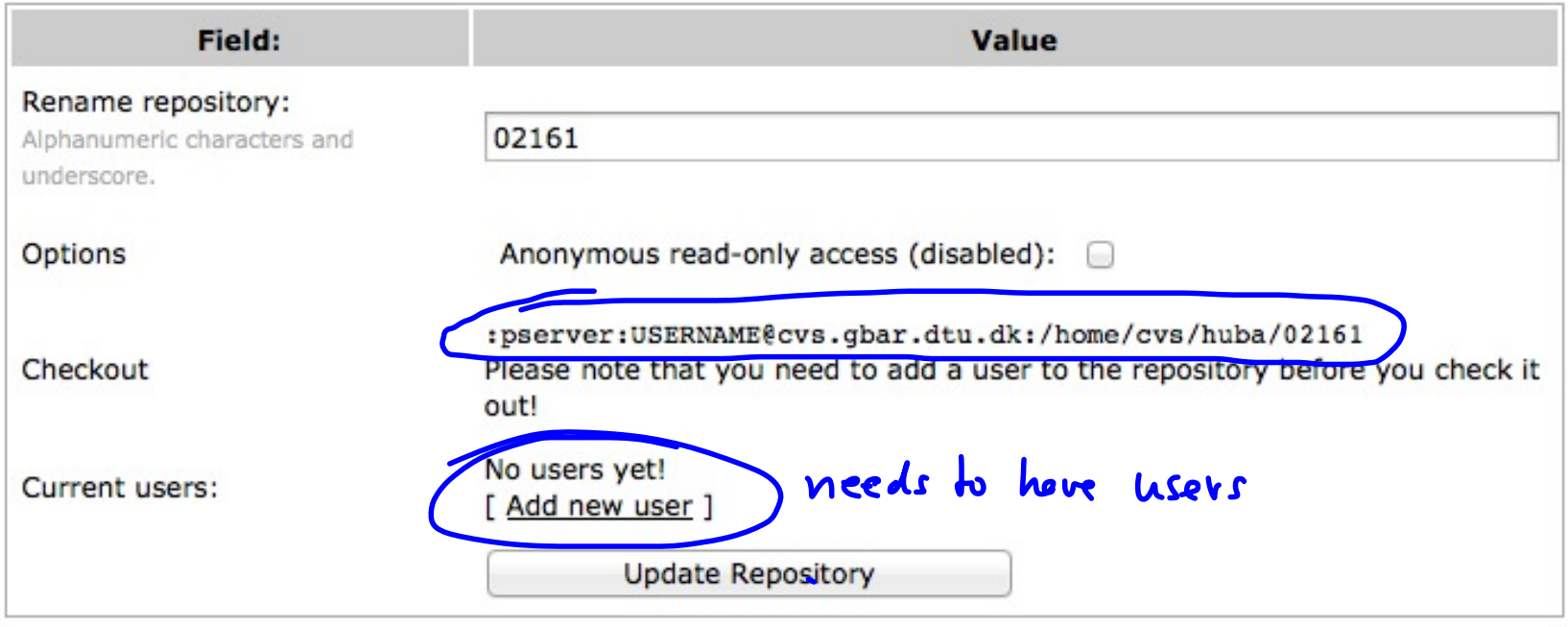
`http://repos.gbar.dtu.dk`



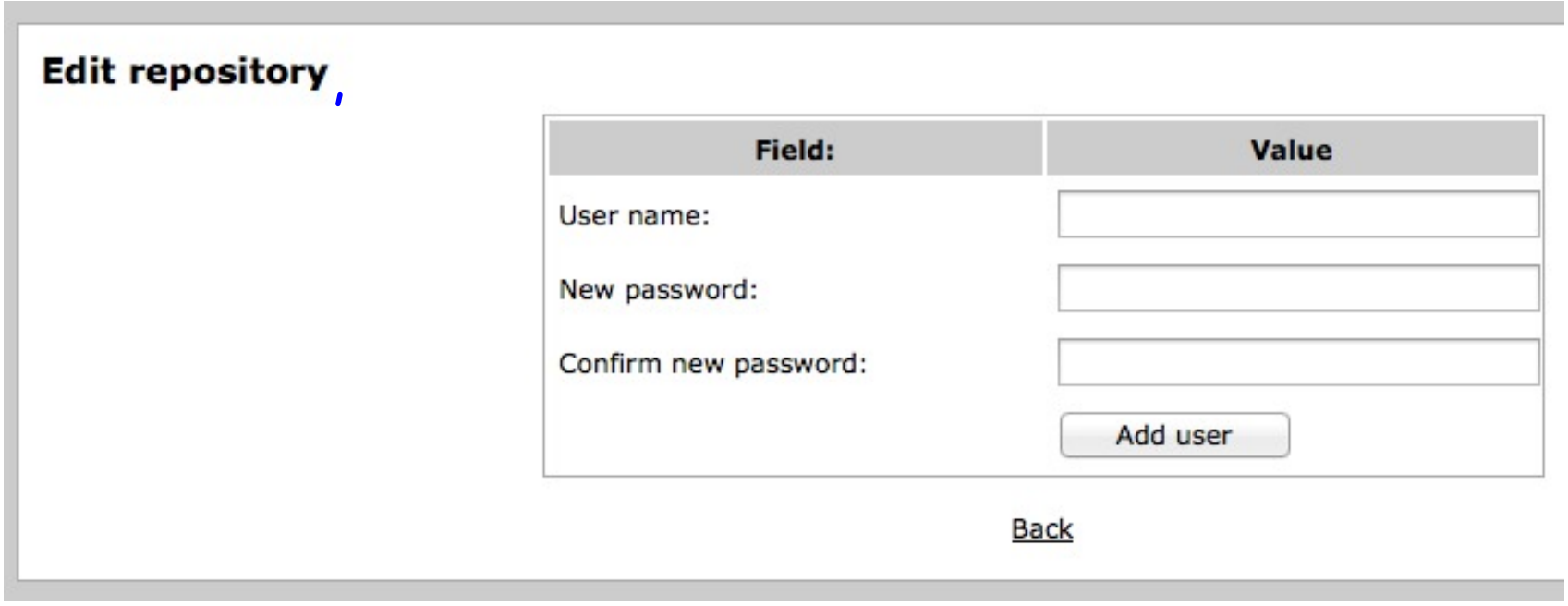
Creating a repository



Creating a repository

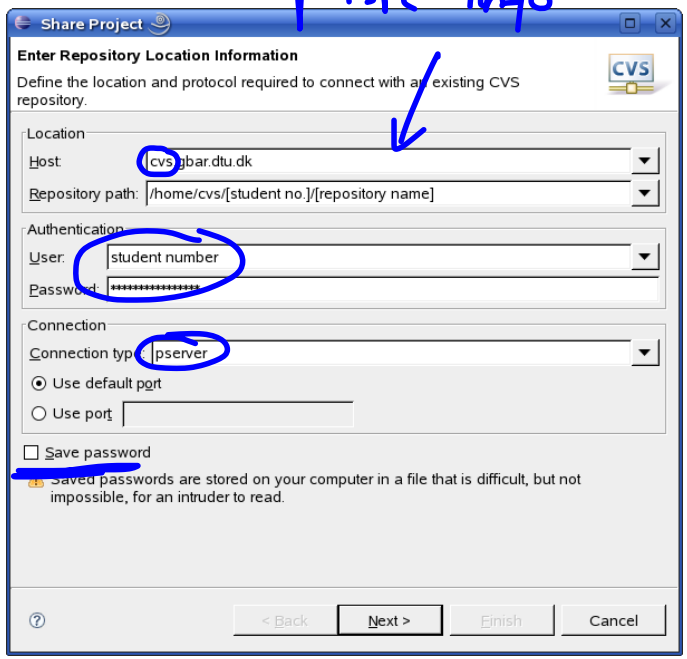


Creating a repository



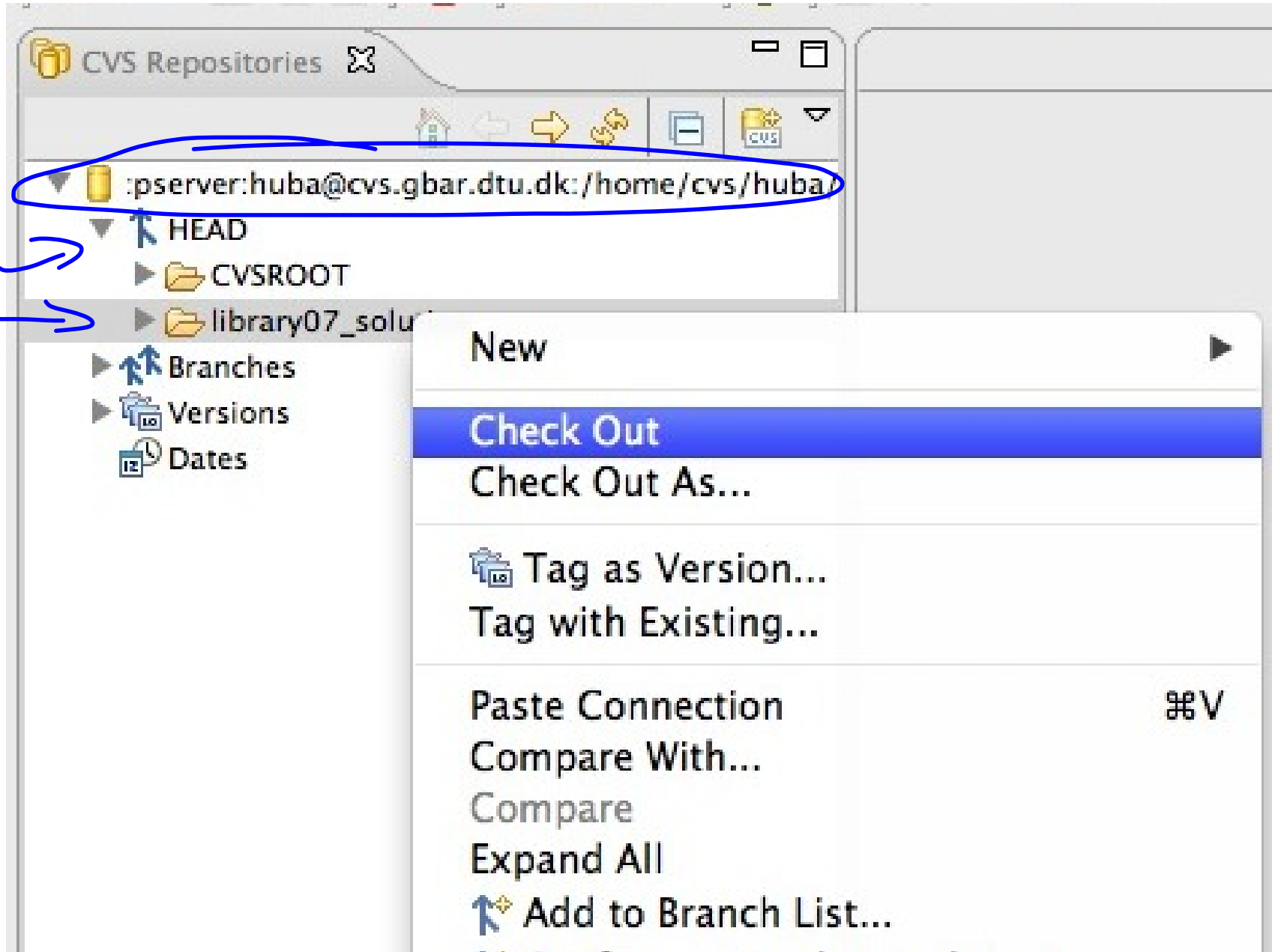
Create a project and *share* it

- Menu: Team→share project and `create a new repository location`

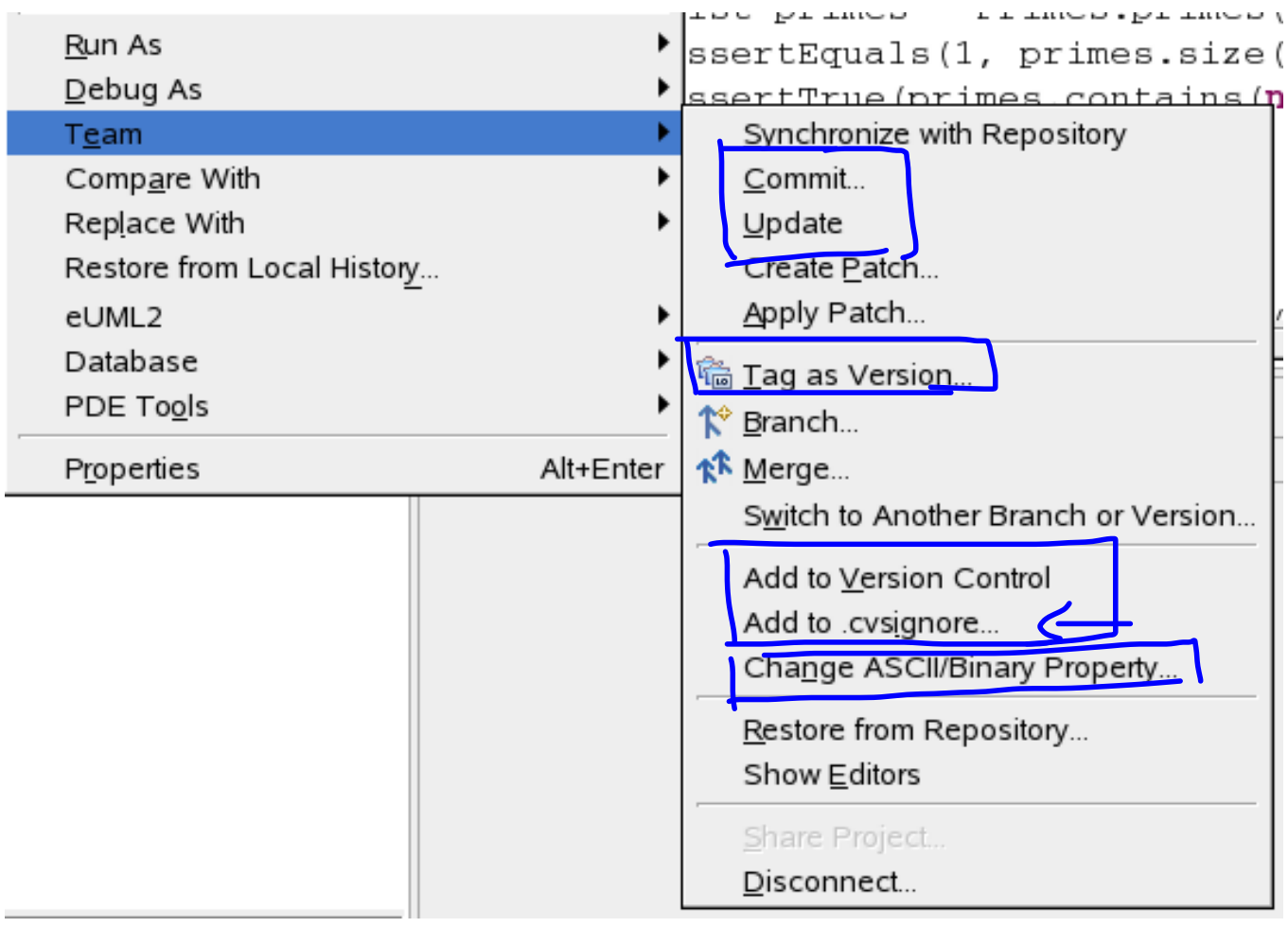


Checking out a project

- CVS Repository Exploring perspective



Package Explorer Team Menu Project



Steps in Developing a Program using CVS

1. Create Repository
2. Create a project and *share* the project
3. For all the programming tasks in an **iteration**
   - 3.1 Run tests; Update project; run tests; fix tests
   - 3.2 Work on the implementation so that all tests run
   - 3.3 Commit your changes
     - 3.3.1 *Update* the project; run tests
     - 3.3.2 *Fix* all compile time errors and all **broken** tests;
     - 3.3.3 *Commit* your changes
4. *Tag* you files for major **project milestones**

Important: Commit only if all tests pass

Committing changes

- Fails if someone else committed the file before
- If fail $\rightarrow$ update, merge, commit

Update a project

- Gets newest version of the file
- If conflicts
  - → *text* files are **merged**
  - → other files are **overwritten**
    - based on **lines**
    - successful merge
    - unsuccessful merge

Unsuccessful merge

- **Same** lines have been changed

```
    public Address() {
        // TODO Auto-generated constructor stub
    }
<<<<<<< Address.java
    public String getStrasse() {  // Local change
=======
    public String getGade() { // Committed change
>>>>>>> 1.2
        return street;
    }
```

Package Explorer Compare With Menu

Compare result: Compare with latest from HEAD