Validation: CRC Cards

Massimo Felici

Room 1402, JCMB, KB

0131 650 5899

mfelici@inf.ed.ac.uk

What are CRC Cards?

- CRC: "**Class**-**Responsibility**-**Collaborator**"
- CRC cards provide the means to validate the class model with the use case model.
  - It is a useful early check that the anticipated uses of the system can be supported by the proposed classes.
  - It is a brainstorming technique that works with scenario walkthroughs to stress-test a design
- **Responsibilities** are a way to state the **rationale** of the system design

Responsibility-based Modeling

- Responsibility-based modeling is appropriate for designing software classes as well as for **partitioning** a system into subsystems
- The underlying **assumptions** are:
  - People can intuitively make meaningful value judgments about the allocation of responsibilities
  - the central issues surrounding how a system is partitioned can be captured by asking what the responsibility of each part has toward the whole
    - Is it really the responsibility of this object to handle this request?
    - Is it its responsibility to keep track of all that information?

Design by Responsibilities

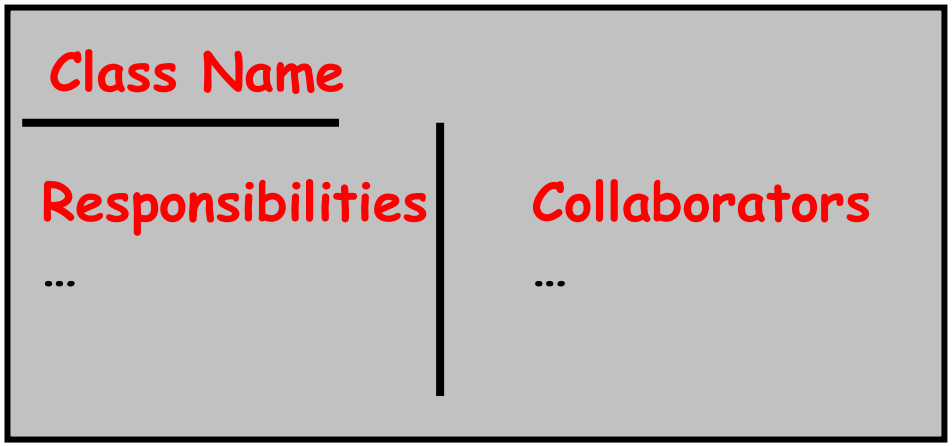
- Responsibility-based Modeling allows
  - The identification of the **components** from which the system is constructed
  - The allocation of **responsibilities** to system components
  - The identification of the **services** provided by them
  - The assessment how components satisfy the requirements as stated by the **use cases**
- Five activities:
  1. **Preparation**: collection and selection of use cases
  2. **Invention**: (incremental) identification of components and responsibilities
  3. **Evaluation**: questions and scenarios stress test the design
  4. **Consolidation**: further assessment of the tested components
  5. **Documentation**: recording identified reasons and scenarios
- Types of Responsibilities
  - To do something (active responsibilities)
  - To provide information (acting as a contact point)

Steps in Responsibility-based Design

1. Identify **scenarios** of use; bound the scope of design
2. Role play the scenarios, evaluating **responsibilities**
3. Name the required responsibilities to carry a scenario toward
4. Make sure that each component has sufficient **information** and **ability** to carry out its responsibility
5. Consider **variations** of the scenario; check the **stability** of the responsibility
6. Evaluate the components
7. Ask the **volatility**/**stability** of the component
8. Create variations
9. Run through the variant scenarios to investigate the stability of the components and responsibilities
10. Simulate if possible
11. Consolidate the components by level
12. Identify **subsystems**
13. Identify the different levels
14. Document the design **rationale** and key **scenarios**
15. Decide which scenarios to document
16. List the components being used that already exist
17. Specify each new component

CRC Cards: How do they look like?

- CRC Cards explicitly represent multiple objects simultaneously
  - The **Name** if the class it refers to.
  - The **Responsibilities** of the class. These should be high level, not at the level of individual methods.
  - The **Collaborators** that help discharge a responsibility.



CRC Cards and Quality

- **Too many responsibilities**
  - This indicates **low cohesion** in the system
  - Each class should have at most three or four responsibilities
  - Classes with more responsibilities should be split if possible
- **Too many collaborators**
  - This indicates **high coupling**
  - It may be the division of the responsibilities amongst the classes is wrong
- **CRC Cards**
  - provide a good, early, measure of the quality of the system (design). Solving problems now is better that later.
  - are flexible – use them to record changes during validation

CRC Cards in Design Development

1. Work using **role play**. Different individuals are different objects
2. Pick a **use case** to building a **scenario** to hand simulate
3. Start with the person who has the card with the responsibility to initiate the use case
4. In discharging a **responsibility** a card owner may only talk to collaborators for that responsibility
5. Gaps must be repaid and re-tested against the use case

Using CRC Cards

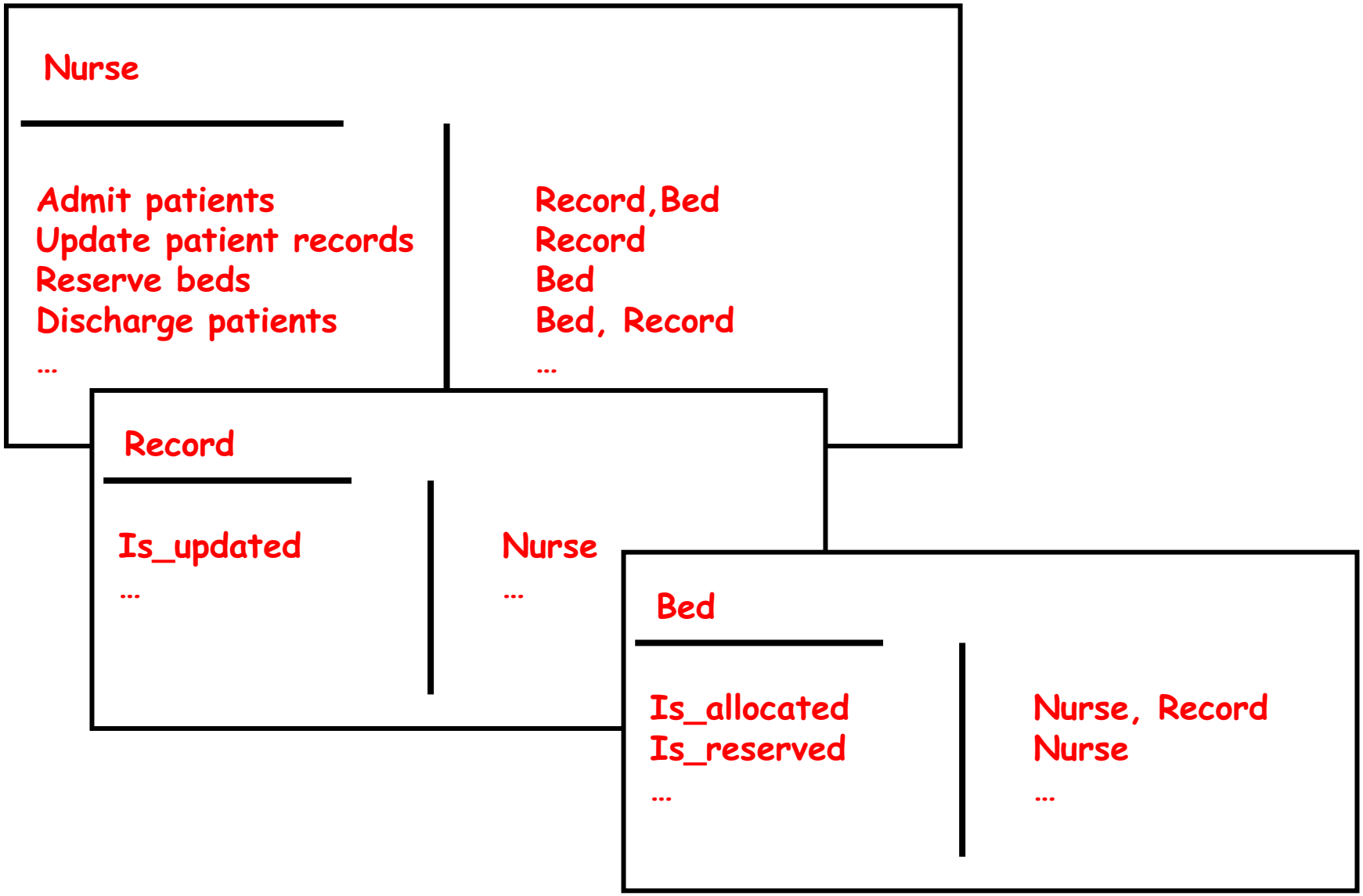
1. Choose a coherent set of **use cases**
2. Put a **card** on the table
3. Walk through the scenario, naming **cards** and **responsibilities**
4. Vary the **situations** (i.e., assumptions on the use case), to stress test the cards
5. Add cards, push cards to the side, to let the design evolve (that is, evaluate different **design alternatives**)
6. Write down the key **responsibility decisions** and **interactions**

Using CRC Cards: An Example

Specimen Use Cases

1. **Patient admitted to ward.** When a patient arrives on a ward, a duty nurse must create a new record for this patient and allocate them to a bed.
2. **Nurse handover.** The senior duty nurse at the end of their shift must inform the new staff of any changes during the previous shift (i.e., new patients, patients discharged, changes in patient health, changes to bed status or allocations).

Using CRC Cards: An Example continued



What CRC Card help with

- Check use case can be achieved
- Check associations are correct
- Check generalizations are correct
- Detect omitted classes
- Detect opportunities to **refactor** the class model. That is: to move responsibilities about (and operations in the class model) without altering the overall responsibility of the system

Principles for Refactoring

- **Do not** do both **refactoring** and **adding functionality** at the same time
  - Put a clear separation between the two when you are working
  - You might swap between them in short steps, e.g., half an hour refactoring, an hour adding new function, half an hour refactoring what you just added
- Make sure you have good **tests** before you begin refactoring
  - Run the tests as often as possible; that way you will know quickly if your changes have broken anything
- Take **short deliberate steps**
  - Moving a field from one class to another, fusing two similar methods into a super class
  - Refactoring often involves many localized changes that result in a large scale change
  - If you keep your steps small, and test after each step, you will avoid prolonged debugging

When to Refactor?

- When you are adding a function to your design (program) and you find the **old design** (code) getting in the way
  - When that starts becoming a problem, stop adding the new function and instead refactor the old design (code)
- When you are looking at design (code) and having difficulty **understanding** it
  - Refactoring is a good way of helping you understand the design (code) and preserving that understanding for the future

OO Design using CRC Cards

Use a team of (ideally) 5-6 people, including developers, 2 or 3 domain experts, and an "object-oriented technology facilitator"

1. Review quality of **class model**
2. Identify opportunities for **refactoring**
3. Identify (new) classes that support system **implementation**
4. Further detail: sub-responsibilities of class responsibilities, attributes, object creation, destruction and lifetimes, data passed, etc.

OO Analysis using CRC Cards

Similar team, but replace some domain experts with developers. However, always include at least one domain expert

1. Session focuses on a part of **requirements**
2. Identify **classes** (e.g., noun-phrase analysis)
3. Construct **CRC cards** for these and assign to members
4. Add **responsibilities** to classes
5. Role-play **scenarios** to identify **collaborators**

Common Domain Modeling Mistakes

- Overlay specific noun-phrase analysis
- Counter-intuitive or incomprehensible class and association names
- Assigning multiplicities to associations too soon
- Addressing implementation issues too early
  - Presuming a specific implementation strategy
  - Committing to implementation constructs
  - Tackling implementation issues (e.g., integrating legacy systems)
- Optimizing for reuse before checking use cases achieved

Reading/Activity

- Kent Beck and Ward Cunningham. A Laboratory for Teaching Object-Oriented Thinking. In Proceedings of OOPSLA '89.
- Other CRC-related resources by Cunningham
  - A CRC Description of HotDraw
  - How Do Teams Shape Objects? How Do Objects Shape Teams?
  - CRC-Card Experience Connects Developers and Customers to Essence of the Problem
- Alistair Cockburn's papers
  - Using CRC Cards
  - Responsibility-based Modeling

Summary

- We should try to check the completeness of the class model (early assurance the model is correct)
- CRC Cards are a simple way of doing this
- CRC Cards support responsibility-based modeling and design
- CRC Cards identify errors and omissions
- They also give an early indication of quality
- Use the experience of simulating the system to refactor if this necessary