Validation (1): CRC Cards

CS3: SEOC1

Note 5

CRC Cards

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.

Introduced in:
"A Laboratory For Teaching Object-Oriented Thinking", K. Beck & W. Cunningham – OOPSLA'89

CRC Cards Document

The name of 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. Maybe the division of responsibilities amongst the classes is wrong.

CRC cards: provide a good, early, measure of the quality of the system. Solving problems now is better than later.

Cards 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 build 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 repaired and re-tested against the use case

Using CRC Cards

Skeleton Card:

Class Name	
Responsibilities	**Collaborators**
Responsibility 1	**Collaborators 1**
Responsibility 2	**Collaborators 2**
...	

Example:

Manager	
Responsibilities	**Collaborators**
Initialise system	
Check free beds	**Hospital**
...	

A Larger Example

Nurse	
Responsibilities	**Collaborators**
Admit patients	**Bed, Record**
Update patient records	**Record**
Reserve beds	**Bed**
Discharge patients	**Bed, Record**
...	

Record	
Responsibilities	**Collaborators**
Is_Updated	**Nurse**
...	

Bed	
Responsibilities	**Collaborators**
Is_Allocated	**Nurse Record**
Is_Reserved	**Nurse**
...	

Specimen Use Cases

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.

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).

What CRC Cards help with

- Check use cases can be achieved.
- Check associations are correct
- Check generalisations 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: half an hour refactoring, an hour adding new function, half an hour refactoring the code 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 superclass. Refactoring often involves making many localized changes that result in a larger 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 function to your program and you find the old code getting in the way. When that starts becoming a problem, stop adding the new function and instead refactor the old code
- When you are looking at code and having difficulty understanding it. Refactoring is a good way of helping you understand the code and preserving that understanding for the future.

OO Analysis 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. 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

OO Design using CRC Cards

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

1. review quality of class model
2. identify opportunities for refactoring
3. identify (new) classes that support system implementation
4. more detail: sub-responsibilities of class responsibilities; attributes; object creation, destruction and lifetimes; data passed

Common Domain Modelling mistakes *(from Note 4)*

- Overly 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 (eg, integrating legacy systems)
- Optimising for reuse before checking use cases achieved
- "Premature pattern-isation"

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 identify errors and omissions.
- They also give an early indication of *quality*.
- Use the experience of simulating the system to refactor if this is necessary.