Agile Software Development

Lecturer: Raman Ramsin

Lecture 17

Practices: Design and Kanban
Design Practices: CRC Cards

- CRC – Class, Responsibilities, and Collaborators
  
  Potential classes are written on CRC Cards. Each Card has three compartments:
  
  - **Class**: The name of the class.
  - **Responsibilities**: A list of the responsibilities of the class (the functions it performs and even the information it is responsible to keep and provide).
  - **Collaborators**: A list of other classes with which this class collaborates in order to fulfill the responsibilities.

<table>
<thead>
<tr>
<th>Class name: BankAccount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities:</td>
</tr>
<tr>
<td>Maintain balance</td>
</tr>
<tr>
<td>Collaborators:</td>
</tr>
<tr>
<td>Bank</td>
</tr>
</tbody>
</table>
CRC Cards: Design Process

Steps:

1. Two or more team members write down on index cards the names of the most important classes involved in the feature.

2. The cards are fleshed out with lists of the responsibilities of each class and the names of collaborators.

3. Each design idea is validated by playing out a plausible scenario of the computation, each developer taking on the role of one or more classes.

For instance, the dialogue may go as follows (through which classes are anthropomorphized):

1. "Hello, Authentication Controller! I am a Web Request and I would like the contents of this resource."

2. "Very well, let me have your Credentials so I can give them, together with the name of your requested operation, to our Access Control List; I will then redirect you to a View component depending on the result."
Design Practices: Rules of Simplicity

A set of criteria, in priority order, proposed by Kent Beck to judge whether some source code is "simple enough":

1. The code is verified by automated tests, and all such tests pass.
2. The code contains no duplication.
3. The code expresses separately each distinct idea or responsibility.
4. The code is composed of the minimum number of components (classes, methods, lines) compatible with the first three criteria.
Design Practices: Simple Design

- A team adopting the "simple design" practice bases its software design strategy on the following principles:
  - Design is an ongoing activity, which includes refactoring and heuristics such as YAGNI ("You Aren't Gonna Need It").
  - Design quality is evaluated based on the rules of code simplicity.
  - All design elements such as "design patterns", etc. are seen as having costs as well as benefits, and design costs must be justified.
  - Design decisions should be deferred until the "last responsible moment", so as to collect as much information as possible on the benefits of the chosen option before incurring its costs.

- Expected benefits:
  - Mitigates the common risk of overdesign ("gold plating").
  - Keeps the software easy to change.
Design Practices: Quick Design Session

- In "simple design", developers handle local design decisions moment-to-moment, but for design choices that may have far-reaching consequences:
  - Two or more developers meet for a **quick design session** at the whiteboard, possibly using design aids such as CRC cards.

- Important guidelines for an effective design session:
  - Considering several credible alternatives, ideally three or more, so that the final choice is based on considerations such as simplicity or conceptual integrity.
  - Assessing each alternative on the basis of a concrete, specific scenario.
    - For instance, envisioning how the acceptance test associated with a given user story would unfold under each possible design.

- Benefits:
  - The design activity is spread out throughout the effort's duration.
  - Quick design sessions address the need for more strategic decisions, while refactoring takes care of local design issues.
Lean Practices: Kanban

- In the Kanban Method:
  - The use of iterations, roles, and effort estimates is deemphasized.
  - **Lead Time** (cycle time) is used instead of velocity:
    - Lead Time is the average time elapsed between the definition of a user story and that story being used by actual users under normal conditions.
  - The task board is replaced with a “Kanban board”; unlike a task board, the kanban board is not "reset" at the beginning of each iteration.

- In the Kanban board, which is the most important element of Kanban:
  - Columns represent the different processing states of a "unit of value", which is generally (but not necessarily) equated with a user story.
  - Each column typically has a limit on the WIP (Work In Process/Progress).
    - If a given state, for instance "in manual testing", has a WIP limit of 2, then the team may not start testing a third user story.
  - If a state is blocking the flow, the priority is to clear current work-in-process; team members will swarm to help clear the blockage.
Kanban: Process

1. **Visualize the workflow:**
   1. Split the work into pieces, write each item on a card and put on the wall.
   2. Use named columns to illustrate where each item is in the workflow.

2. **Limit WIP** – assign explicit limits to how many items may be in progress at each workflow state.

3. **Measure the lead time,** and optimize the process to make lead time as small and predictable as possible.

![Kanban Board Diagram]

[Kniberg et al. 2009]
Kanban: Pitfalls and Benefits

**Pitfalls**
- It is not advisable that the Kanban board should serve as a pretext to reintroduce a "waterfall"-like, linear sequence of activities.
- Teams should be wary of Kanban boards not accompanied by WIP limits.

**Benefits**
- In some contexts, measuring lead time rather than velocity, and dispensing with iterations, may be the more appropriate choice.
  - for instance, when there is little concern with achieving a specific release date, or when the team's work is by nature continuous and ongoing, such as enhancement or maintenance.
### Kanban Board: Typical Example

**Definition of Done:**
- Goal is clear
- First tasks defined
- Story split (if necessary)

**Definition of Done:**
- Code clean & checked in on trunk
- Integrated & regression tested
- Running on UAT environment

**Definition of Done:**
- Customer accepted
- Ready for production

### Feature / story
- Date when added to board: 2009-08-20, 2009-09-30
- (description)
- ★ = priority
- ★★ = panic
- Who is analyzing / testing right now

### Task / defect
- =task
- =completed
- =blocked
- =who is doing this right now

### What to pull first
1. Panic features ★★ (should be swarmed and kept moving. Interrupt other work and break WIP limits as necessary)
2. Priority features ★
3. Hard deadline features (only if deadline is at risk)
4. Oldest features
Kanban Process Enactment: Example (1)

[Kniberg et al. 2009]
Kanban Process Enactment: Example (2)

[Kniberg et al. 2009]
Kanban Process Enactment: Example (3)

[Kniberg et al. 2009]
Kanban Process Enactment: Example (4)

We’ll do D!
No, wait… that would break the Kanban limit of 2!

K is pretty urgent. I’ll schedule it next.

Get coffee. Then tell us what this #&@ stack trace means.

[Kniberg et al. 2009]
Kanban Process Enactment: Example (5)

[Kniberg et al. 2009]
Kanban Process Enactment: Example (6)

[Kniberg et al. 2009]
References


