5.2 Essential Characteristics of the Unified Process

Use-case-driven

- What is a use case?
  - A prose representation of a sequence of actions.
  - Actions are performed by one or more *actors* (human or non-human) and the system itself.
  - These actions lead to valuable results for one or more of the actors – helping the actors achieve their goals.
- Use cases are expressed from the perspective of the users, in natural language, and should be understandable by all project stakeholders.
- *Use-case-driven* means the use cases are employed from requirements gathering through code and test.

Architecture-centric

- *Software architecture* captures decisions about:
  - The overall structure of the software system.
  - The structural elements of the system and their corresponding interfaces.
  - The collaborations among these structural elements and their expected behavior.
- *Architecture-centric* means the software architecture provides the center point around which all other development evolves.
  - Provides a ‘big picture’ of the system.
  - Provides an organizational framework for development.
  - Facilitates reuse.
  - Provides a framework for evolving the system by attending to modifiability qualities of the system.
  - Guides the prioritization and choice of use cases for development.

Iterative and incremental

- An iterative and incremental approach allows start of development with incomplete, imperfect knowledge.
- An iterative and incremental process is like solving a jigsaw puzzle: neither top-down nor bottom-up but *accretionary* and *convergent*.
- Being *iterative* and *incremental* provides the UP with the following advantages:
  - Logical progress toward a robust architecture.
  - Effective management of changing requirements.
  - Effective means to address changes in planning.
  - Continuous integration.
  - Early understanding of the system (‘Hello world!’ effect).
  - Ongoing risk assessment.

5.3 Unified Process Requirements and Analysis

Use cases

- What is a use case?
  - A prose representation of a sequence of actions.
  - Actions are performed by one or more *actors* (human or non-human) and the system itself.
  - These actions lead to valuable results for one or more of the actors—helping the actors achieve their goals.
- Can be viewed as a necessary (but not sufficient) collection of requirements for that interaction.
- Additional requirements are identified in design to round out sufficiency.

Use cases (cont'd)

- Use cases:
  - Help focus on achieving well-defined ***user goals***.
  - May vary widely in their level of detail.
- One captures use cases by:
  - Interviewing users and discussing their use of the system.
  - Naming and describing each use of system.
  - Decomposing complex, interwoven tasks.
- *Hint:* Any processes in which the system must participate and any events to which the system must respond are good starting points for identifying candidate use cases.

Use case format

- *Name.* Short and descriptive.
- *Short Description*. A few lines acting as a summary or abstract of the use case.
- *Actors*. List the actors that interact with this use case.
- *Trigger*. The goal that motivates the use case interaction.
- *Preconditions*. Specify what conditions must be true before the use case starts.
- *Incoming information*. Information that is needed in the event flow of the use case.
- *Results*. Information that results from the event flow of the use case.
- *Postconditions*. Specify what conditions must be true when the use case ends.
- *Event Flow (Process)*. Use a sequentially-numbered list of brief statements describing the steps of the use case.

Use case example

Name: Create new task in PIM

Short description: A new task and all related information are created in the PIM.

Actors: TaskCreator

Trigger: The TaskCreator wishes to create a a new task.

Preconditions:

1. TaskCreator must have all needed task information available.
2. Task management application must be selected.

Incoming information: Task description, task priority, task due date, task category

Results: Task creation confirmation

Postconditions:

1. A new task is created from the incoming information.

Use case example (cont'd)

Event flow:

1. Use case begins when TaskCreator selects to 'Create a new task.'
2. System prompts for task category.
3. TaskCreator enters task category.
4. System prompts for task description.
5. TaskCreator enters task description.
6. System prompts for task priority.
7. TaskCreator enters task priority.
8. System prompts for task due date.
9. TaskCreator enters task due date.
10. System displays task category, name, priority, and due date for review.
11. System prompts TaskCreator to 'Accept' or 'Cancel'.
12. Use case ends when TaskCreator selects 'Accept'.

Note use of 'use case begins when…' and 'use case ends when…' constructs.

Structuring the use-case

- Follow a consistent structure for all detailed use cases.
- *Always* use "The use case begins when…" and "The use case ends when…" stylistic 'bookends' in your event flows.
- The primary scenario should be the one most readily traced through the use case.
- Defer exceptional conditions and branches to alternative scenarios.
- Alternate scenarios (a.k.a. extensions) should follow the primary scenario in the text.
- Alternative scenarios are identified by the step number at which they occur in the primary scenario plus a letter.
  - *Example*: Three possible alternate scenarios that begin at step 6 in a primary scenario would be labeled 6a, 6b, and 6c.

Example event flow with alternate scenario

Event Flow:

1. The use case begins when the user selects ‘Artifact→Physical Care’ from the CMS menus.
2. System displays a blank entry form for the artifact’s climate requirements.
3. User enters the artifact’s climate requirements.
4. User selects ‘Continue’ to end climate requirements entry.
5. System displays a blank entry form for the artifact’s fire requirements.
6. …

5a. Museum facilities cannot support climate requirements of artifact:
    1. System informs user that Museum cannot support climate requirements of artifact.
    2. System specifies requirement(s) that cannot be met.
    3. System displays climate control system modification request form.
    4. User enters the climate requirements for the artifact.
    5. User selects ‘Submit’ to submit modification request to facilities staff.
    6. *Continue event flow with step 5.*

Sidebar: identifying the system boundary

- Like most analysis and design concepts, *system* and *system boundary* may vary with the level of abstraction.
- The *system* represents the software and hardware elements which are the current focus of attention.
  - *Example*: In the *Museum Automation Problem*, each individual subsystem—climate, security, on-site education—can be considered '*the system*' during its analysis and design.
- The *system boundary* represents the division between the system itself, including software and hardware, and the actors: primary, supporting, and off-stage.
- Effectively defining the scope of the system during the requirements effort can help delineate the system boundary.

Prioritize use cases

- Planning in an iterative and incremental process is risk-driven.
- Schedule analysis, design, and implementation of use cases representing core system (required *vs*. desirable or optional) functionality and high-risk use cases for early iterations.
- Conversely, schedule development of non-core-functionality and low-risk use cases for later iterations.
- Need to balance functionality and risk factors.
- At this stage, establish priority based on:
  - Proportion of contribution to overall required system functionality.
  - Technological, skills, and dependency risks.

Risk factors

- *Requirements risks*
  - Have you properly identified requirements for system?
  - Have you reviewed project scope?
  - Can you effectively set functional requirements priorities?
- *Technological risks*
  - Are you using a new, untested technology?
- *Dependency risks*
  - Are you depending on a third-party product?
- *Skills risks*
  - Are you planning to do something with which you are completely unfamiliar?
  - Do you have the right staff with the necessary skills?

Detail use cases

- Highest-priority use cases require early attention to detail.
- Identify the use cases scheduled for the first iteration of development, generally about 10% of total.
- For each first-iteration use case, write a detailed (fully-dressed) use case.
- Because these use cases are being detailed at an early stage, they may require refinement in later iterations.
- Avoid getting involved in detailed UI design issues at this stage: write use cases in an *essential* (mostly UI-free) style.

Sidebar: essential style

- I agree with Larman about keeping the UI specifics out of the use cases and focusing on the actor's *intent* during the interaction.
- The UI is a part of the system that is expected to change substantially over the course of development.
- Early involvement with UI specifics can solidify issues that are best left flexible at this point.
- So:
  - Postpone *specific* UI descriptions and interactions until later in design and even then, confine them to a 'look & feel' artifact.
  - It's OK to use *general* UI terminology and user actions: displays, menus, buttons; selects, enters, confirms.
  - Avoid specific UI "look-and-feel" (*e.g.* radio buttons, pop-up menu, drop-down menu) or user action descriptions (scrolls, highlights, drags).