Lectures II–III:

Patterns for Effective Use Case Driven Functional Requirements Engineering

Tommi Kärkkäinen

UNIVERSITY OF JYVÄSKYLÄ
Faculty of Information Technology
Department of Math. Information Technology
Autumn 2005

Preliminaries on Patterns for Effective Use Cases

The following material is based on the Book by Adolph and Bramble: "Patterns for Effective Use Cases", The Agile Software Development Series, Cockburn and Highsmith (Eds.), Addison-Wesley, 2003. In the book, the use case patterns were presented in the form:

- The Pattern Name
- A Picture
- The Context
- The Problem Statement
- A Metaphoric Story
- The Forces Affecting the Problem
- The Solution
- Examples

An example of a use case pattern description (without the picture) is given in Appendix A in Figure 1.1.

Notice that pattern languages, as originated by Alexander et al. (1977) have been applied to give a structured descriptions of many best practices related to software development: Design Patterns (Gamma et al., 1994), Object-oriented Reengineering Patterns (Demayer et al., 2003) etc. (Search the web and Amazon for more, e.g. http://hillside.net/patterns/.)

A pattern sample in different domain

Many of the techniques for software development have their counterparts in other parts of life, or, many of the techniques can be applied in other parts of life. As an example (not deadly serious ;-), let us present a discussion on the difficulty of organizing a course *Introduction to Software Engineering* in the similar pattern form that will be applied to describe effective use case production.

Problem Defining agenda for an introductory course in software engineering is difficult, and the resulting course will easily overlap some other courses or end up too far from both the context of students, practice in the field, and academic theory.

Forces

- In a university, it is difficult to know what the students know when entering a course.
- There is no rigorous, general theory of software development.
- It is impossible to maximize the happiness of all key players - users, managers, and developers - in software development, because we are dealing with a multicriteria optimization problem.
- The field of software engineering has not a visible boundary nor a clear definition (cf. “computer science” and “information system development method”).

- Without (someone) implementing the system there is no actual software project, but the skills of participants of the course and the practitioners in the field with this respect vary a lot.
- The real success of organizing a course (or writing a thesis, or carrying out a project etc.) is to find the leading thought (in Finnish “punainen lanka”) and to follow it without loosing it. Concerning SE this happens along with a software development project (and it’s subprojects).

Solution Focus the contents of the course mainly on the activities and tasks that underlie quality work and are close to producing and evaluating the actual software.

- Development and technological skills in “Ohjelmointi I, II” etc.
- Life-cycle prerequisities and notations in "Oliopohjainen tietojärjestelmien kehittäminen".
- More abstract and general project issues (e.g. management, risks, maturity, evaluation etc.) in "Ohjelmistotuotanto".
- Capstone projects in “Sovellusprojekti” or “Projektin hallinta”.

Examples There are simply too much badly organized SE related stuff and course material in the web, so be critical…

Okey, let’s go back to the actual topic. We shorten here the original presentation of use case patterns to compact review of the form
Name x *Problem* x *Forces* x *Solution* (+Comments) x *Examples*
Notice that a use case may have up to four general elements (and levels of detail - a four bit presentation as Cockburn puts it)

- actors and goals
- main success scenario
- failure conditions
- failure handling

The purpose, however, is not to produce all the details (i.e. final deliverables) in one run, but as a result of balanced and managed development efforts. Appendix A starts with a few summaries from Cockburn’s book (and Adolp’s and Bramble’s note on the relation between use cases and business process modelling, BPM).

In general, the patterns to be presented are divided into *development* and *structural* patterns, whose interactions are illustrated in Figures A and B in Appendix A. This kind of division is an example of a more general division of the world of software delopment into static (class implemenation or phase product; what is their appropriate structure) and dynamic (message passing or interaction between the user and system or activities to test a piece of code; what are the appropriate guidelines for actions to produce the static structure). Notice that such a division can also be applied to classify different UML diagrams according to these two basic categories.

Here we represent the different patterns in the order (which differs from the original one in the reference book) that tries to capture the importance of different patterns from the overall project point of view. Hence, the presentation is focused around the following questions:

1. How does a high-quality deliverable (e.g. a set of use cases describing the functional requirements of a System-under-Consideration (SuD)) look like?
2. How to organize the development work in order to end up with the high-quality deliverable?

Answers to these questions are to be discussed through the following subsets of use case patterns:

1. Structure: The Use Case Set: Most general and important (critical to success of the whole project) patterns in the summary level (cf. Figure 4.12 in Appendix A) describing the phase product as a whole (here the functional requirements in form of use cases).
2. Development: The process: Most important patterns in the summary level describing the activities to produce the phase product.
3. Structure: The Use Case: How does a use case should look like?
4. Structure: The Scenario: How does a basic flow of steps should look like?
5. Structure: The Step: How does an individual step should look like?
6. Structure: Use Case Relationships: What kind of refactoring can be applied for a set of use cases?
7. Development: Edition: How to organize the refactoring?
8. Development: The team: How to organize the work in general?

Description of Patterns for Effective Use Cases

- Separately, in Finnish

Afterthougts on Patterns for Effective Use Cases

- Many of the things here are management issues
- Quantization of functional requirements for estimating the required resources (mainly time and peopleware) to realize the functionality
  - FPMs (Function Point Methods)
  - Using O-O metrics like quantization of use case set as input for a suitable functional estimation technique (statistical methods, predictive data mining methods (e.g. neural networks), AI(/rule)-based methodologies etc.)
- Trace requirements to design artifacts proactively (V&V)
- Version and configuration management (for all artifacts in all phases of the project)
- *Hence, all things that follow in the course have a bijective (one-to-one, in both ways) relation to (use case driven) requirements management.*