XIV. From Requirements to Design in the UP

Outline

- Elaboration phase
  - Characteristics and principles leading this phase; artifacts to be derived
- Describe System Sequence Diagram
- Defining conceptual model
  - Conceptual classes and "techniques" to identify them

Elaboration activities

- What is developed during the elaboration phase?
  - **Majority of requirements are discovered and stabilized**
  - **Major risks are mitigated or retired**
  - **The core architectural elements are implemented and proven**
- It consists of around 2-4 iterations
  - Each iteration should not last longer than **six weeks** and should be timeboxed
- The code developed constitutes a prototype

  (this is **not a trow-away prototype development process**)

- Elaboration in one sentence:
  - Build the core architecture, resolve the high risk elements, define most requirements, and estimate the overall schedule and resources (C.Larman)

Elaboration best practices

- Do short **timeboxed risk-driven** iterations
- Start **programming early**
- Adaptively design, implement, and test the core and risky parts of the architecture
- **Test early, often, realistically**
- Adapt based on feedback from tests, users, developers
- **Write most of the use cases and other requirements in detail**, through a series of workshops, once per elaboration iteration

Which functionalities we should implement first?

- Organize requirements and iterations by risk, coverage, and criticality:
  - **Risk**: includes both technical complexity and other factors, such as uncertainty of effort or usability
  - **Coverage**: implies that all major parts of the system are at least touched on in early iterations
  - **Criticality**: refers to functions of high business value
- Before the first iteration rank each UC
- Revise ranking before each iteration
- Risk is the main factor to consider planning the next iteration

What artifacts may start in elaboration?

- **Domain model**: visualization of the domain concepts; it is similar to a static information model of the domain entities
- **Design model**: set of diagrams that describes the logical design.
- **Software Architecture Document**: a learning aid that summarizes the key architectural issues and their resolution in the design
- **Data Model**: this includes the database schemas
- **Test Model**: what will be tested and how
- **Implementation Model**: source code, executables, database, and so on
- **UI prototypes**: user interface, usability models

Common mistakes in Elaboration

- Planning more than few months for the phase
- Planning a single iteration (possible for stable, well-understood problems)
- No production of code
- Consider elaboration a requirement phase carried on before construction
- Trying to derive a full and careful design before programming
- There is no early and realistic testing
- ...

Elaboration – SSD

- Define System Sequence Diagram (SSD)
  - A SSD is a picture that shows, for a particular scenario or UC, the events that external actors generate, their order, and inter-system events.
    All systems are treated as **black-box**. Interest on events that **cross the system boundary** from actors to systems.

- Example deriving an SSD from a UC

- In general SSDs can be used to show only the main success scenario, nevertheless relevant alternative scenarios should be represented

Domain model

- A domain mode illustrates meaningful (to the modelers) conceptual classes in a problem domain; it is the most important artifact to create during OO analysis
- A domain model is a representation of **real-world conceptual classes**, not of software components. It is **not** a set of diagrams describing software classes, or software objects with responsibilities

Domain model and UML

- Using UML notation, a domain model is illustrated with a set of class diagram in which no operations are defined (sort of E/R diagram). It may show:
  - Conceptual classes
  - Association between conceptual classes
  - Attributes of conceptual classes

Domain model and UML

- Lets draw an example together
  - flight booking system

Conceptual classes

- The domain model illustrates conceptual classes or vocabulary in the domain. Formally, a conceptual class may be considered in terms of its symbol, intension, and extension:
  - **Symbol** – words or images representing a conceptual class
  - **Intension** – the definition of a conceptual class
  - **Extension** – the set of examples to which the conceptual class applies

:Flight1
AZ142 9.00 13.00

:Flight2
AZ342 7.00 8.00

:Flight3
AZ754 12.00 14.30

Flight
Code taking-off time landing time

"A flight represents a connection operated by an airline company between two airport. It ha a code a departing time and a landing time"

Domain analysis vs. Structured analysis

- Software problems can be complex...divide and conques principle is a common strategy to deal with complexity
- Structured analysis decomposes the problems in terms of **functions and processes**
- OO analysis the decomposition is by **things or entities in the domain**

How can we identify conceptual classes?

- It is useful to have "proven" techniques making easier the identification of conceptual classes...**We would like to include everything is necessary**!!!
- Rule of thumb: it is better to **overspecify** a domain model with lots of fine-grained conceptual classes that to underspecify it
  - It is common to forget conceptual classes at the begin...you should it as soon as you discover it
  - It is possible to have conceptual classes with **no attributes**!! Nevertheless in that case they should have a **behavioral role**

How can we identify conceptual classes?

- We discuss two main strategies to identify conceptual classes have shown their potential:
  - Conceptual Class Category list
  - Identify noun phrases
- Analysis pattern another possible solution
  - Partial domain models defined by experts for the particular domain

Conceptual Class Category List Strategy

- Start the creation of a domain model by making a list of conceptual class reading a cetegory list. Example with the flight reservation system

Conceptual Class Category	Examples
Physical or tangible objects	Registers, Airplane
Specifications, design, or descriptions of things	Product Specification, Flight Description
Places	Store, Airport
Transactions	Sale, Payment, Reservation
Transactions line items	Sales Line Item
Roles of people	Cashier, Pilot
Containers of other things	Store, Bin, Airplane
Things in a container	Item, Passenger
Other computer or electro-mechanical systems external to the system	Credit Payment System, Air Traffic Control
Abstract noun concepts	Hunger, Acrophobia
Organisations	Sales Department, Object Airline

Conceptual Class Category List Strategy

Conceptual Class Category	Examples
Events	Sale, Payment, Meeting Flight, Crash, Landing
Processes (may be)	Selling a Product, Booking a Seat
Rules and policies	Refund Policy, Cancellation Policy
Catalogs	Product Catalog, Parts Catalog
Record of finance, work, contracts, legal matters	Receipt, Employment Contract, Maintenance Log
Financial instruments and services	Line of Credit, Stock
Manuals, documents, reference papers, books	Daily Price Change List, Repair Manual

Noun Phrase Identification Strategy

- Linguistic Analysis on the requirements (Use Cases here)
  - Identify noun and noun phrases in textual descriptions of a domain
  - Mechanical mapping noun-to-class mapping is not possible, words are also ambiguous
- Weakness of this approach is the imprecision of natural language; **different noun phrase may represent the same conceptual class or attribute**
- Lets try with the flight reservation system

Domain models the case of report objects

- A report object provide information on other object in the domain model (as for instance a receipt) should we include it in our conceptual model?
- In general including in the domain model such kind of concept only duplicates information found elsewhere
- In some case however a report object has a business value (for instance a receipt gives to the client the right to return bought items)

The case of description conceptual classes

- A description conceptual class only provides information on instances of other conceptual classes
- Should we include such kind of classes in the domain model
- The "item problem" all item instances include information on themselves
- The item problem can be solved including in the conceptual model a ProductSpecification conceptual class (there will obviously be a relation with the described objects class)
- When including a description conceptual class:
  - Description is independent from its existence
  - Deleting instances results in a loss of information
  - Reduce redundant or duplicated information

Steps in deriving a domain model

1. Apply the discussed strategies to identify relevant conceptual classes
2. Draw them in a domain model
3. Add the associations necessary to record relationship for which there is a need to preserve some memory
4. Add the attributes necessary to fulfill the information requirements