D50: Advances in Software Engineering

Requirements Model

Wolfgang Emmerich

Lecture Overview

- ***Object-Oriented Software Engineering (OOSE) from Jacobson et al.***
- ***The basics of 'a use case driven approach'***
- ***Development of a Requirements Model:***
  - ***actors***
  - ***use cases***
  - ***interface descriptions***
  - ***problem domain objects***
- ***Relevant notations from the UML (Unified Modeling Language)***

2

- We have selected Ivar Jacobsons OOSE method for this course because we do not think that it is feasible to teach all relevant methods.
- The selection of object-oriented software engineering derives partially from the fact that it supports use case modelling, which we believe is important for the elicitation of requirements from stakeholders. It is also due to the fact that we believe OOSE gives more detailed process guidance than the other two methods.
- The sub-title of the book 'a use case driven approach' summarises the originality of the approach, its value and much of its structure. A 'use case model', in essence a representation of how the system is used, and will be used, has a formative relationship with all the models used in OOSE.
- The focus of this lecture will be on a subset of this method that identifies the notations and the development procedures and heuristics in order to derive a requirements model.
- The lecture will also introduce the first elements of the UML as the form of notation. This is a new notation language, only just launched in a complete form by the previously competing authors of object-oriented methods: Grady Booch, James Rumbaugh and Ivar Jacobson.
- The next slide gives a little bit of background information of where OOSE originated from...

OOSE Background

- Originated in Sweden
- "Object-Oriented Software Engineering - A Use Case Driven Approach" by Ivar Jacobson, Magnus Christerson, Patrik Jonsson & Gunnar Overgaard, Addison-Wesley , 1992
- Pragmatic method based on experience
- Popular and successful
- Complete method

3

- The OOSE emphasis on the user and use cases is not surprising given Scandinavian concern with participative design.
- The method originates from work in the electronics firm Ericsson, and the book contains a lengthy and very detailed telecommunication example. Telecommunication is a domain where the application of object-orientated methods is particularly successful. This is partly due to the fact that telecommunication software is so complex and changes so drastically that structure-oriented method have failed. The ESS5 switching system produced by AT&T, for instance, comprises 5 million lines of C++ code and has been delivered in different configurations to at least 20 different telecom networks in different countries.
- Besides having, or perhaps because of having practical origin, the object-oriented software engineering method is successful both commercially and as a teaching method.
- The method is also complete, in the sense of covering all stages of system development, procedures (and notations), and supported by a case tool called Objectory, which is available here at City University.
- Indications from the documentation for the UML suggest that it includes definitions of all the essential concepts that are required for OOSE.
- In order to start the introduction of OOSE and UML, we are going to look at what the constituents of a method are...

What comprises a Method?

- ***Method described via***
  - ***syntax (how it looks)***
  - ***semantics (what it means)***
  - ***pragmatics (heuristics, rules of thumb for use)***

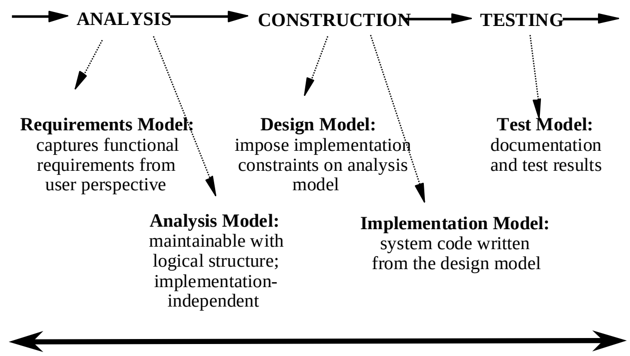
4

- A method that is to be used during in a certain stage of a software engineering project has to be defined in terms of the notation that is used for producing the products expected during that stage and the development heuristics and procedures used for producing the products.
- The notation is defined in terms of syntax and semantics. The syntactic part of the notation can be determined in terms of a grammar, be it for a textual or a graphical language. Hence the syntax determines the grammatical correctness of the grammar. Rules such as dataflows must start from or lead to a process of a Dataflow diagram would be defined as part of the grammar for a graphical definition. For the grammar of C++ we would include rules that each statement must be finished with a semicolon.
- The semantics of a language can be distinguished in static and dynamic respects. The static part typically identifies scoping and typing rules, such as that declaring occurrences of identifiers must be unique within a certain scope and that applied occurrences must match with particular declarations. The dynamic semantics defines the meaning for different concepts of the notation (that usually have to be correct with respect to syntax and static semantics).
- The pragmatics of a method identifies suggestions and heuristics of how a notation should be used. It identifies which concept of the notation should be used to express different concerns (such as operations should express behaviour and attributes should express states) and may suggests orders for the development (such as identify the different classes, then establish the state they capture in terms of attributes and finally determine the operations).
- We now look at the coarse-grained pragmatics of OOSE and reveal what different development steps Jacobson suggests...

UCL

System Dev. as Building Models

■ 3 stages and 5 models



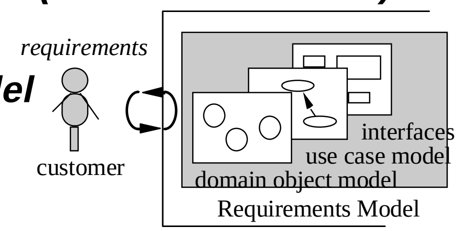
Seamless, incremental transition between stages and models, iterations possible

5

- Jacobson views system development as a process consisting of stages that produce model descriptions. Each partial model is an abstraction of the system enabling the developer to make decisions necessary to move closer to the final (complete) model, the tested executeable code of the system. Each modelling step adds more structure; each model is more formal. In the diagram a sequence of specific objectives is shown under the model titles.
- One of the advantages of object-orientation is that it favours *incremental development*. Incremental development denotes a software development process, where it is not necessary to complete all the requirements before the design can start. In these processes it is, therefore, not uncommon to define the requirements, design implementation and test components even though the requirements of other components have not yet been fully defined. Incremental processes are supported by object-orientation because the same concepts, i.e. objects, classes, inheritance, attributes and operations are used througout the different models. Hence, analysis objects integrate seamlessly with design objects; from these again there is a seamless integration with implementation objects. The reverse direction is equally well supported.
- This lecture is concerned only with the first OOSE model, requirements; the next is about analysis and the following two about design.
- The next thing to do is to show the relationship between processes and models in OOSE i.e. between the processes and the methods and techniques for modelling ...

Analysis Stage

- **_Primary objectives_**
  - **_to determine what the system must do_**
  - **_to embed the software system in its environment_**
- **_Two concerns_**
  - **_to get the right thing_**
  - **_to get the thing right (now and for future)_**
- **_Products_**
  - **_Requirements Model_**
  - **_Analysis Model_**



6

- The requirements stage is concerned with *what* users expect a system to achieve, it is not concerned with *how* the system is going to achieve its objectives. It is sometimes difficult for a software engineer to keep these two different perspectives apart; hence care should be taken in order to not confuse them.
- In order to identify the objects that are involved in performing certain functions for a system, it is not sufficient to focus on the system in isolation. The requirements stage, therefore, takes a more wholistic view and identifies the embedding of the future system into its environment. In requirements definitions, it is therefore not uncommon to have objects, such as the librarian of a library support system, that will have no representation in the system itself. Also existing systems that need to be integrated with a new system are often regarded as objects.
- The analysis stage produces two modules to address different concerns. The requirements model defines how to get the right thing and the analysis model defines how to get the thing right.
- The requirements model has three elements, two models and a set of more generally defined interface descriptions.
- The analysis model, the second component of this phase, is a description in terms of interface, entity and control objects, considered in detail in the next lecture.
- From the practical point of view taken in this course, it is necessary to balance the theoretical view with a detailed, sequential account of the process that we look at on the next slide ...

Producing A Requirements Model

- ***Inputs***
  - ***1 Derive possible use cases***
  - ***2 Discriminate between possible use cases***
  - ***3 Generate use case desciptions***
  - ***4 Identify associations between use cases***
  - ***5 Refine & complete use cases & use case model***
  - ***6 Describe and test user interfaces***
  - ***7 Describe system interfaces***
  - ***8 Identification of problem domain objects***
  - ***9 Check incorporation of requirements***
- ***Outputs***
- ***Notations***

7

- This slide presents an overview of the procedures that have to be followed when developing the requirements model. The requirements model consists of use cases. These are used to identify different scenarios, for instance for different user roles, as to how the system will be used.
- The first step is very much a brainstorming activity where different scenarios are captured in as many use cases as possible. The second step tries to identify, order the different scenarios and get rid of duplicates. The third step refines each of the use cases with a text describing each use case in more detail. The fourth step identifies extension so and usage relationships between the so far isolated use cases. The fifth step refines and completes the use case model. Then the user interface of the system is defined and tested as the sixth step. These first prototypes of the system often generates requirements that were unidentified previously. After that the seventh step defines the system interfaces to systems that previously existed and need to be integrated. The eighth step is, in fact, already a preparation of the analysis phase and it identifies the domain objects. The final and optional step validates that the informal requirements definitions that served as an input have been captured in the use cases, user interface and system interface definitions.
- The lectures will concentrate on the concepts and models used, leaving detailed procedures to tutorial work. For your convenience the last two pages of these notes include a complete breakdown of the above steps.
- A definition of the inputs and outputs for the steps follows on the next slide...

Requirements Model: Input & Output

- ***Inputs :***
  - ***User requirements specifications [multi media]***
  - ***Documentation of existing systems, practices etc. that are to be followed [text, graphic]***
  - ***Exchanges between developers and users and specifiers [m m]***
- ***Outputs :***
  - ***use case model [graphic]***
  - ***concise descriptions of use cases [text]***
  - ***user interface descriptions [text ... prototypes]***
  - ***system interfaces [protocols]***
  - ***problem domain object list (names, attrib.) [text]***

8

- The inputs to requirements modelling are the diverse sources from which system requirements may be derived, and the variety of media in which they may be carried. Usually there are textual descriptions that stake holders produce in order to outline the goals that the system should meet. Also (especially bigger organisations) have business process descriptions outlining the workflow that a system might have to support. Finally, meetings with future stake holders where requirements are explicitly elicitated are often recorded on audio or video tapes and these tapes might be transcribed in order obtain the relevant textual description from which requirements can be extracted.
- The output will formalise these inputs both in terms of their content, as use cases and the use case model, and in term of their representation via UML diagrams and texts.

Requirements Model: Notations

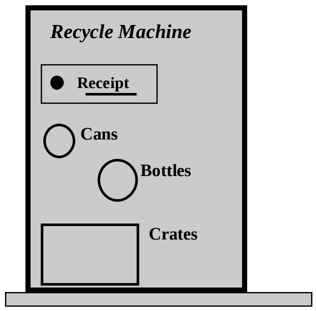
- ***Notations introduced :***
  - ***use case diagram (system box, ellipses, names, actor icons, actor/case links, <uses> and <extends> associations)***
  - ***association (<extends>, <uses>)***
  - ***use case descriptions***

9

- The notations introduced include all those included in the UML for use case model, plus the first *associations*, the generic term used for relationships in UML.
- Let us now begin to consider requirements with an example taken from Jacobson's book...

Requirements Example

Multi-purpose recycling machine



Machine must:
- receive & check items for customers,
- print out receipt for items received,
- print total received items for operator,
- change system information,
- signal alarm when problems arise.

10

- For the details of this example, consider pages 155-156 of [JCJÖ92].

Actors

- ***An actor is:***
  - ***anything external to the system, human or otherwise***
  - ***a user type or category***
- ***A user doing something is an occurrence of such a type***
- ***A single user can instantiate several different actor types***
- ***Actors come in two kinds:***
  - ***primary actors, using system regularly***
  - ***secondary actors, enabling primary actors***

Analysis begins with the identification of actors external to the system; they are a generic way of describing the potential users of the system. In identifying actors we will need to consider scenes or situations typical to the system and its use. Please note that the users might not necessarily be humans but they might also be other systems that use the system through its system interfaces.

The important distinction here is between the *actor* and the *user*. Actors are a type perspective while users denote particular instances of these types. A particular system user, e.g. Jane a warehouse manager, may at different times take on the roles of many different actors, e.g. supervisor, driver or operator. Actors only relate to the system in specified ways in particular *use cases*.

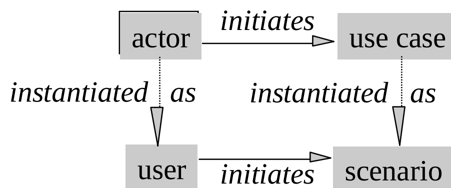
A distinction between primary and secondary actors is made in OOSE but not UML. Examples:

- recycling machine:
  - customer (primary)
  - operator (secondary)
- warehouse:
  - supervisor, worker, truck driver, forklift operator (all primary)
- air traffic control:
  - controller, supervisor, pilot (primary),
  - maintenance team (secondary).

The interaction between the system and the actor is a sequence known as a ‘use case’ which we will detail on the next slide...

USE CASES

- **_A use case_**
  - **_constitutes complete course of events initiated by actor_**
  - **_defines interaction between actor and system_**
  - **_is a member of the set of all use cases which_**
- **_Use cases together define all existing ways of using the system_**



12

- A use case is a generic description of an entire transaction of events involving the system and objects external to it. A use case can therefore be seen as a description of different states and the events that make the system transit from one state to another.
- Together the uses cases represent all the defined ways of using the system and the behaviour it exhibits whilst doing so.
- Again we separate types and instances for use cases. Each use case is a specific type of using the system. A scenario (in UML) denotes an instance of a use case. When a user (an actor instance) inputs a stimulus, the use case instance (a UML scenario) executes and starts a transaction belonging to the use case, consisting of actions to perform.
- In OOSE the sytem model, as a whole, is use case driven. So if you want to change the system's behaviour, you should remodel the appropriate actor(s) and use case(s).
- On the next slide, we revisit the recycling machine example and look at examples of its use cases...

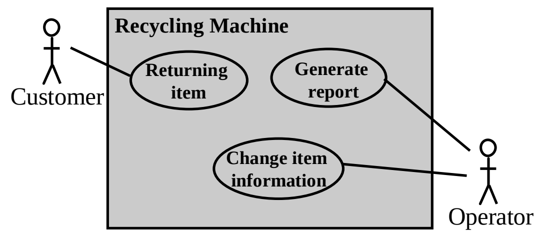
Example Use Case

- *Returning items is started by Customer when she wants to return cans, bottles or crates. With each item that the Customer places in the recycling machine, the system will increase the received number of items from Customer as well as the daily total of this particular type. When Customer has deposited all her items, she will press a receipt button to get a receipt on which returned items have been printed, as well as the total return sum.*

- The top of this slide includes a description of an example use case for the recycling machine, plus one scenario that instantiates the use case for a particular user.
- Another example of a use case for a familiar London Underground machine is given below:
- Destination_Ticket (alternative to Zone_Ticket) begins when a potential Traveller approaches the ticket machine. The machine displays an introductory message inviting choice of destination. Traveller picks destination. Machine dispays message inviting choice of ticket. Traveller picks a ticket type and the machine responds with the price. After the traveller has inserted enough money, the machine dispenses the ticket and any change. Machine then prepares for its next customer.
- The set of all use cases is represented in the 'use case model', for which there is a special diagram in OOSE, adapted almost exactly in the UML. We look at these diagrammatic notation on the next slide...

Use Case Model

- ***A use case model***
  - ***presents a collection of use cases***
  - ***characterise behaviour of whole system, plus external actors***



14

- A ‘use case model’ combines all the use cases of a system and at the top level helps to visualise the context of the system and its boundary.
- The diagram notation used for expressing the use case model is defined in the UML. Actors are classes, notated in their simplest form as stick figures with an instance name (or class box). Ellipses represent the different use cases and have an identifier naming them. Also the whole name is given a name. Lines identify the associations between actors and use cases.
- In this model an actor, for example a ‘clerk’ in a model of a bank system, can be associated with an number of different cases, e.g. ‘counter transaction’, ‘cheque clearing’, ‘audit’ and more than one actor with one use case e.g. ‘customer’ and ‘operator’ in a ‘stuck_item’ use case in the recycling machine example.
- The identification of each use case requires a detailed consideration of the system’s requirements. A systematic approach representing the different use cases will be presented on the next slide.

Identifying Use Cases

- Consider situation,
- Identify actors,
- Read specification,
- Identify main tasks,
- Identify system information,
- Identify outside changes,
- Check information for actors,
- Draft initial use cases [text]
- Identify system boundary,
- Draft initial use case model [graphic]

15

- In order to kick off the use case modelling different scenes and situations should be identified from the problem domain that is to be addressed by the system to be developed.
- The next step should aim at identifying the different actors that are involved in each scene. Remember that not only the human actors should be identified but also actors that are other system should be considered.
- Keeping this information in mind the specifications, transcripts of recorded information that form the input to the requirements modelling stage should be revisited for each actor in order to identify the main tasks that the actor would need to perform with the system.
- Then the information objects that each actor would need to access (read), create (write) or change would need to be identified.
- Actors would usually use the system in response to outside events/changes. A clerk in a bank, for instance would use the system in order to input a transaction that is required to bank a customer`s cheque. Hence, the fact that a customer has handed in a cheque would be considered as an outside event.
- Next, the events/changes that actors need to be informed about should be identified.
- Then the gathered information should be used to draft use cases, essentially detailed text descriptions of the interactions between actors and the system.
- Then the system boundary should be drawn, clearly separating what parts of the processes/procedures are going to be embedded into the system and which are not.
- Finally, the initial use case model should be drafted using the graphic UML notation. Note that at any of these steps it might become necessary to interact with stakeholders in order to resolve incompleteness and inconsistencies.

When is a Use Case .... ?

- **_Discrimination between possible use cases_**
  - **_Estimate frequency of use,_**
  - **_Examine degree of difference between cases_**
  - **_Distinguish between 'basic' and 'alternative' courses of events_**
  - **_Create new use cases where necessary_**

16

- Discrimination between cases is difficult because there may be so many levels of difference. OOSE provides only weak rules for discriminating between separate use cases.
- The simplest discrimination are frequency, variation and alternation.
- A fairly useful suggestion of Jacobson is to distinguish between *basic* and *alternative* courses of events. A basic sequence of events would identify the normal situations in which a system would be used. A sequence of events would be denominated as alternative if the events represent exceptional conditions that would not be considered as normal.
- This distinction, for instance, allows developers later to tune the system to perform efficiently for those cases that are rather usual and trade in performance of those cases that occur less frequently.
- On the next slide we will revisit the recycling machine example and elaborate the 'Returning item' use case...

Elaborated Example

BASIC - When the Customer returns a deposit item, it is measured by the system. The measurements are used to determine what kind of can, bottle or crate has be deposited. If accepted, the Customer total is incremented, as is the daily total for that specific item type.

ALTERNATIVE - If the item is not accepted, 'NOT VALID' is highlighted on the panel.

BASIC - When Customer presses the receipt button, the printer prints the date. The customer total is calculated and the following information printed on the receipt for each item type: name, number returned, deposit value, total for this type. Finally the sum that the Customer should receive is printed on the receipt.

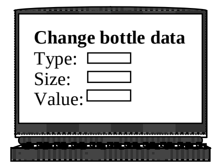
17

- In this returning item use case we now distinguish basic, i.e. different normal sequences of events from alternative flows, i.e when error conditions appear.
- We would have to elaborate the use case we had earlier for the London Underground ticket machine to include at least one altenative for a Traveller who makes choices in the 'wrong' order.
- In considering such problems you can often find that a use case, while being independent, may have a clear association within another use case because it somehow represents a special case that extends an existing use case.
- We will now look at different associations between use cases...

User Interface Descriptions

- ***Describe user interfaces***
- ***Test on potential users,***
- ***if necessary using***
- ***simulations or prototypes***

Operator's interface



- Use cases are used to formalise requirements from informal requirements specifications, transcripts of recorded requirements elicitation meetings and other discussions with stakeholders.
- An orthogonal (and constructive) way of obtaining further requirements is to use the information that was accumulated during the use case modelling for the description or even the prototyping of *user interfaces* for the later system. A user interface is human machine interface through which human actors interact with the system.
- There are mechanisms by means of which just the graphical user interface with its windows, menus and forms can be effectively generated. After exposing users to these prototypes they will be able to tell what they like and what they do not like and even more importantly what is missing.
- However, it is important to note that these user interface prototypes are included solely as a means of requirements capture and building use cases, not for the purposes of detailed design. They are regularly discarded after they have served to identify the requirements.
- The user interfaces in the recycling machine example include:
  - customer panel (holes, buttons etc.),
  - receipt panel,
  - operator interface.
- System interfaces for non-human actors are defined in terms of the protocols necessary for the communication between the different systems involved.

Problem Domain Objects

- **_Object in specification have direct counterpart in the application environment_**
- **_System knowledge obligatory_**
- **_Refinement in stages :_**
  - **_Object noun ->_**
  - **_Logical attributes ->_**
  - **_Static associations_**
  - **_Inheritance ->_**
  - **_Dynamic associations ->_**
  - **_Operations_**

19

- In the longer term the identification of problem domain objects is an essential prerequisite for preparing a class diagram. At the requirements model stage its importance lies in the necessity for:-
  - definition of objects in use cases, and,
  - communication between the developers and those who have commissioned or use will use the system.
- In OOSE objects are refined progressively in stages. The *later stages of refinement* are not really possible within context of the requirements model because they must cope with dynamic characteristics. In the view of Jacobson, other methods (presumably like OMT), rely completely on object models, which can result in a fixed and inflexible structure.
- The next slide revisits the recycling machine example to illustrate how to find basic domain objects...

Object Examples

OBJECT	ATTRIBUTES
name	***characteristic / information : type***
Deposit item	***name: string, total: integer, value: ECU***
Can	***width: cm, height: cm***
Bottle	***width: cm, height: cm, bottom: cm***
Crate	***width: cm, height: cm, lenght: cm***
Receipt	***total cans: int, total bottles: int, ...***
Customer panel	***receipt button: button***
Operator panel	***bottle data: cm, ...***

20

- The left hand side identifies the various objects. They were found by searching the use case descriptions for relevant nouns. Candidate attributes for these objects are identified by looking at properties of these nouns in the use case descriptions.
- At this stage 'attributes' are particular characteristics associated with each object in the problem domain. At next stage we introduce the notation in which every object contains name, attributes (containing information derived from 'static' associations of object) and operations (defined via its dynamic asociations)
- On completion of such a list we have the essential outputs from the requirements stage. The next slide displays where we are...

Summary

- ***System development as model building***
- ***Requirements model "to get the right thing"***
- ***System use in context via the use case model***
- ***User interface descriptions***
- ***Problem domain objects as prelude to class diagram***

21

In OOSE almost everything is a model of some kind and building systems means building such models. The first one, for requirements, is the means for ensuring that, from the point of view of the users and those commissioning the system, it will be clearly related to documented expectations of use. The use case model provides the first stage of formalising the requirements in a way that they can be used throughout the subsequent stages of development. Specific user interface descriptions, a subject in itself not covered here, provide an additional dimension. Finally the problem domain objects comprise the first stage in the primary sequence of formalisation, the development of a class diagram in UML.

For your background reading, we would suggest the following references:

[JCJÖ92] The Requirements Model. Section 7.2. pp. 153-174. 1992.

[Boeh88] B.W. Boehm: A Spiral Model of Software Development and Enhancement. IEEE Computer. pp 61-72. May 1988.