MTAT.03.094
Software Engineering

Lecture 04: Analysis

Kristiina Rahkema
On behalf of Dietmar Pfahl
email: rahkema@ut.ee

Fall 2016
Labs This Week

Lab 2 Assignment:

1. Project Planning
   - Refinement of requirements into Tasks
   - Prioritization of Tasks (P1, P2, P3, P4)
   - Responsibility assignment
   - Effort estimation

2. Five Use Case Descriptions

3. (Initial) Domain Model
Labs Next Week

- Part of the lab time will be used for
  - Assessment (evaluation) of your Lab 1 solutions

- Each project team must attend with all its members present

- If not all team members present, penalty applies
Schedule of Lectures

Week 01: Introduction to SE
Week 02: Requirements Engineering I
Week 03: Requirements Engineering II

**Week 04: Domain Analysis**
Week 05: Development Infrastructure I
Week 06: Development Infrastructure II
Week 07: Architecture and Design
Week 08: Refactoring

Week 09: Verification and Validation I
Week 10: Verification and Validation II

Week 11: No lecture
Week 12: Agile/Lean Methods
Week 13: Industry Lecture (Testing)

**Week 14: TBD**
Week 15: Measurement / Course wrap-up, review and exam preparation

Week 16: no lecture
Acknowledgements

Textbooks/Slides:

• Ian Sommerville: Software Engineering, 9th edition, 2010
  (http://www.softwareengineering-9.com/)

• Ivan Marsic: Software Engineering, 2012
  (http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf)
Structure of Lecture 04

• Preliminaries
  • UML Class Diagrams
• Domain Analysis and Modelling Example
  • Identifying Concepts (Responsibilities)
  • Attributes
  • Associations
UML Classes

• A class describes a group of objects with
  • similar properties (attributes),
  • common behaviour (operations),
  • common relationships to other objects,
  • and common meaning (“semantics”).

• Example

  employee:
  • has a name, employee# and department;
  • an employee is hired, and fired; an employee works in one or more projects

We won’t use this today
Objects vs. Classes

- The instances of a class are called objects

<table>
<thead>
<tr>
<th>Fred_Bloggs:employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: Fred Bloggs</td>
</tr>
<tr>
<td>Employee #: 234609234</td>
</tr>
<tr>
<td>Department: Marketing</td>
</tr>
</tbody>
</table>

- Two different objects may have identical attribute values (like two people with identical name and address)

- Objects have associations with other objects
  - E.g. Fred_Bloggs:employee is associated with the KillerApp:project object
  - But we will capture these relationships at the class level (why?)
Associations

- Objects do not exist in isolation from one another
  - A relationship represents a connection between things
- Types of relationships
  - Association
  - Aggregation and Composition
  - Generalization
- Class diagrams show classes and their relationships
Associations

- **StaffMember**
  - staffName
  - staff#
  - staffStartDate

- **Client**
  - companyAddress
  - companyEmail
  - companyFax
  - companyName
  - companyTelephone

- **liaises with**

  **Multiplicity**
  - A client has exactly one staffmember as a contact person
  - A staff member has zero or more clients on His/her clientList

  **Name of the association**

  **Direction**
  - The “liaises with” association should be read in this direction

  **Role**
  - The staffmember’s role in this association is as a contact person
  - The clients’ role in this association is as a clientList
Multiplicity

- **Optional** (0 or 1) 0..1
- **Exactly one** 1 (alternative: 1..1)
- **Zero or more** 0..* (alternative: *)
- **One or more** 1..*
Aggregation and Composition

• **Aggregation**
  - This is the “Has-a” or “Whole/part” relationship

• **Composition**
  - Strong form of aggregation that implies ownership:
    - if the whole is removed from the model, so is the part
    - the whole is responsible for the disposition of its parts
Generalisation

- Sub-classes **inherit** attributes, associations, & operations from the superclass.
- A sub-class may override an inherited aspect.
- Super-classes may be declared **abstract**, meaning they have no instances.
  - Implies that the sub-classes cover all possibilities.

Superclass associations are inherited by subclasses.

- Two subclasses
  - A superclass
Class Diagram

Aggregation
This is the “Has-a” or “Whole/part” relationship

Composition
Strong form of aggregation that implies ownership:
- If the whole is removed from the model, so is the part
- The whole is responsible for the disposition of its parts

Can be used to model the domain structure, i.e. concepts and their properties and relationships
Structure of Lecture 04

• Preliminaries
  • UML Class Diagrams
• Domain Analysis and Modelling Example
  • Identifying Concepts (Responsibilities)
  • Concept Attributes
  • Concept Associations
Domain Models

• Captures the most important concepts of the domain and their associations
  • The domain is the background knowledge of the users of the system, e.g. the domain of the librarian includes books, loans, returns, patrons, …
• Helps us to understand and think about the concepts we will use in the project
• Provides a useful “glossary” for the project
Elements of a Domain Model

- The following elements enable us to express time invariant static business rules for a domain:
  - **Domain classes** – each domain class denotes a concept (type of object).
  - **Attributes** – an attribute is the description of a named slot of a specified type in a domain class; each instance of the class separately holds a value.
  - **Associations** – an association is a relationship between two (or more) domain classes that describes links between their object instances. Associations can have roles, describing the multiplicity and participation of a class in the relationship.
Simple Domain Model Example

- **Domain class**
  - **Person**
    - Attribute: first name : String, last name : String, salary
    - Association: +employer (0..1), +owner (1..*), +employee (1..*)

- **Car**
  - Attribute: type, model name
  - Association: +owner (1..*)

- **Company**
  - Attribute: name
  - Association: +employee (0..1)

<<Rule>>
If a person is not employed by a company then they do not have a car.

Domain class
What are Domain Classes?

• Each domain class denotes a concept.
  • Concept = Descriptor for a set of things that share common properties.

• Domain Classes can be:
  • Business objects - things that are manipulated in the business e.g. Order.
  • Real world objects - things that the business keeps track of e.g. Contract, Site.
  • Actors/Workers/Persons - e.g. Controller and Customer.
  • Events that transpire - e.g. Sale and Payment.

• A domain class has attributes and associations with other classes.
How to create a Domain Model?

Perform the following in very short iterations:

• Make a list of candidate domain classes.
• Draw these classes in a UML class diagram.
• If possible, add brief descriptions for the classes.
• Identify any associations that are necessary.
• Decide whether some domain classes are really just attributes.
• Where helpful, identify role names and multiplicity for associations.
• Add any additional static rules as UML notes that cannot be conveyed with UML symbols.
• Group diagrams/domain classes by category into packages.
• Concentrate more on just identifying domain classes in early iterations!
How to identify Domain Classes?

• An obvious way to identify domain classes is to identify nouns and phrases in textual descriptions of a domain.

Consider a use case description as follows:

1. **Customer** arrives at a **checkout** with **goods** and/or **services** to purchase.

2. **Cashier** starts a new **sale**.

3. **Cashier** enters **item identifier**.

4. System records the **sale line item** and presents the **item description, price** and running **total**.
Example: ATM Machine

(a) System

- Actor (Bank customer)
- (ATM machine)
- Deposit
- Withdraw
- Transfer

(b) Domain Model

- Actor (Remote datacenter)
- Concept 1
- Concept 2
- Concept 3
- Concept n
Consider the following problem description, analyzed for Subjects, Verbs, Objects:

The ATM verifies whether the customer's card number and PIN are correct.

\[ \text{SC} \quad \text{V} \quad \text{RO} \quad \text{OA} \quad \text{OA} \]

If it is, then the customer can check the account balance, deposit cash, and withdraw cash.

\[ \text{SR} \quad \text{V} \quad \text{OA} \quad \text{V} \quad \text{OA} \quad \text{OA} \]

Checking the balance simply displays the account balance.

\[ \text{SM} \quad \text{OA} \quad \text{V} \quad \text{OA} \]

Depositing asks the customer to enter the amount, then updates the account balance.

\[ \text{SM} \quad \text{V} \quad \text{OR} \quad \text{OA} \quad \text{V} \quad \text{OA} \]

Withdraw cash asks the customer for the amount to withdraw; if the account has enough cash,

\[ \text{SM} \quad \text{OA} \quad \text{V} \quad \text{OR} \quad \text{OA} \quad \text{V} \quad \text{SC} \quad \text{V} \quad \text{OA} \]

the account balance is updated. The ATM prints the customer's account balance on a receipt.

\[ \text{OA} \quad \text{V} \quad \text{SC} \quad \text{V} \quad \text{OA} \quad \text{C} \]

Analyse each **subject** and **object** as follows:

- Does it represent a person performing an action? Then it’s an actor, ‘R’.
- Is it also a verb (such as ‘deposit’)? Then it may be a method, ‘M’.
- Is it a simple value, such as ‘color’ (string) or ‘money’ (number)? Then it is probably an attribute, ‘A’.
- Which NPs are unmarked? Make it ‘C’ for class.
- Verbs can also be classes, for example:

  Deposit is a class if it retains state information
Identifying Concepts (Domain Classes) from noun phrases

• Vision and Scope, Glossary and Use Cases are good for this type of linguistic analysis

• However:
• Words may be ambiguous or synonymous
• Noun phrases may be attributes or parameters rather than classes:
  • If it stores state information or it has multiple behaviors, then it’s a class
  • If it’s just a number or a string, then it’s probably an attribute
How to identify Attributes?

A domain class sounds like an attribute if …

• It relies on an associated class for its identity – e.g. ‘order number’ class associated to an ‘order’ class.
  • The ‘order number’ sounds suspiciously like an attribute of ‘order’.
• It is a simple data type – e.g. ‘order number’ is a simple integer.
  • Now it really sounds like an attribute!
Building a Domain Model

Step 1: Identifying the boundary concepts

Step 2: Identifying the internal concepts
Domain Modelling Strategies for Concept Identification

- 'Outside-In’ Approach: First identify boundary concepts, then internal concepts.
  - Internal concepts might be further classified into control and entity concepts.

- 'Setting-up-an-enterprise’ Approach: What workers need to be hired and what things acquired?
  - Start with 'worker' concepts and their responsibilities.
    - Usually (at least) one 'Controller'.
  - Distinguish between 'doing' (D) and 'knowing' (K) responsibilities.
    - Usually: D=worker and K=thing; but not always clear.
Use Cases vs. Domain Model

In use case analysis, we consider the system as a “black box”

In domain analysis, we consider the system as a “transparent box”

(a) Use Case 1
(a) Use Case 2
(a) Use Case N

(b) Domain Model
(b) Actors
Use Case Diagrams

First tier use cases
- UC1: Unlock
- UC2: Lock
- UC3: AddUser
- UC4: RemoveUser
- UC5: InspectAccessHistory
- UC6: SetDevicePrefs
- UC7: AuthenticateUser
- UC8: Login

Second tier use cases
- UC1: Unlock
- UC2: Lock
- UC3: AddUser
- UC4: RemoveUser
- UC5: InspectAccessHistory
- UC6: SetDevicePrefs
- UC7: AuthenticateUser
- UC8: Login

Subsystem 1: Device Control
- Tenant
- Landlord
- LockDevice
- LightSwitch
- Timer

Subsystem 2: Account Management
- Tenant
- Landlord
- Account Management Subsystem
- UC3: AddUser
- UC4: RemoveUser
- UC5: InspectAccessHistory
- UC6: SetDevicePrefs
- UC7: AuthenticateUser
- UC8: Login
Use Case 1: Unlock

**Use Case UC-1:** Unlock

<table>
<thead>
<tr>
<th>Related Requirements:</th>
<th>REQ1, REQ3, REQ4, and REQ5 stated in Table 2-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiating Actor:</td>
<td>Any of: Tenant, Landlord</td>
</tr>
<tr>
<td>Actor's Goal:</td>
<td>To disarm the lock and enter, and get space lighted up automatically.</td>
</tr>
<tr>
<td>Participating Actors:</td>
<td>LockDevice, LightSwitch, Timer</td>
</tr>
</tbody>
</table>
| Preconditions:        | • The set of valid keys stored in the system database is non-empty.  
                        • The system displays the menu of available functions; at the door keypad the menu choices are “Lock” and “Unlock.” |
| Postconditions:       | The auto-lock timer has started countdown from autoLockInterval. |

**Flow of Events for Main Success Scenario:**

→ 1. Tenant/Landlord arrives at the door and selects the menu item “Unlock”

2. include::*AuthenticateUser*(UC-7)

← 3. System (a) signals to the Tenant/Landlord the lock status, e.g., “disarmed,” (b) signals to LockDevice to disarm the lock, and (c) signals to LightSwitch to turn the light on

← 4. System signals to the Timer to start the auto-lock timer countdown

→ 5. Tenant/Landlord opens the door, enters the home [and shuts the door and locks]
# Example NL Requirements

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>5</td>
<td>The system shall keep the door locked at all times, unless commanded otherwise by authorized user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed).</td>
</tr>
<tr>
<td>REQ2</td>
<td>2</td>
<td>The system shall lock the door when commanded by pressing a dedicated button.</td>
</tr>
<tr>
<td>REQ3</td>
<td>5</td>
<td>The system shall, given a valid key code, unlock the door and activate other devices.</td>
</tr>
<tr>
<td>REQ4</td>
<td>4</td>
<td>The system should allow mistakes while entering the key code. However, to resist “dictionary attacks,” the number of allowed failed attempts shall be small, say three, after which the system will block and the alarm bell shall be sounded.</td>
</tr>
<tr>
<td>REQ5</td>
<td>2</td>
<td>The system shall maintain a history log of all attempted accesses for later review.</td>
</tr>
<tr>
<td>REQ6</td>
<td>2</td>
<td>The system should allow adding new authorized persons at runtime or removing existing ones.</td>
</tr>
<tr>
<td>REQ7</td>
<td>2</td>
<td>The system shall allow configuring the preferences for device activation when the user provides a valid key code, as well as when a burglary attempt is detected.</td>
</tr>
<tr>
<td>REQ8</td>
<td>1</td>
<td>The system should allow searching the history log by specifying one or more of these parameters: the time frame, the actor role, the door location, or the event type (unlock, lock, power failure, etc.). This function shall be available over the Web by pointing a browser to a specified URL.</td>
</tr>
<tr>
<td>REQ9</td>
<td>1</td>
<td>The system should allow filing inquiries about “suspicious” accesses. This function shall be available over the Web.</td>
</tr>
</tbody>
</table>
## Extracting the Responsibilities

<table>
<thead>
<tr>
<th>Responsibility Description</th>
<th>Type</th>
<th>Concept Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinate actions of all concepts associated with a use case, a logical grouping of use cases, or the entire system and delegate the work to other concepts.</td>
<td>D</td>
<td>Controller</td>
</tr>
<tr>
<td>Container for user’s authentication data, such as pass-code, timestamp, door identification, etc.</td>
<td>K</td>
<td>Key</td>
</tr>
<tr>
<td>Verify whether or not the key-code entered by the user is valid.</td>
<td>D</td>
<td>KeyChecker</td>
</tr>
<tr>
<td>Container for the collection of valid keys associated with doors and users.</td>
<td>K</td>
<td>KeyStorage</td>
</tr>
<tr>
<td>Operate the lock device to armed/disarmed positions.</td>
<td>D</td>
<td>LockOperator</td>
</tr>
<tr>
<td>Operate the light switch to turn the light on/off.</td>
<td>D</td>
<td>LightOperator</td>
</tr>
<tr>
<td>Operate the alarm bell to signal possible break-ins.</td>
<td>D</td>
<td>AlarmOperator</td>
</tr>
<tr>
<td>Block the input to deny more attempts if too many unsuccessful attempts.</td>
<td>D</td>
<td>Controller</td>
</tr>
<tr>
<td>Log all interactions with the system in persistent storage.</td>
<td>D</td>
<td>Logger</td>
</tr>
</tbody>
</table>

[Note: Incomplete, e.g., 'Timer' missing ...]
Domain Model (1a)

Domain concepts for subsystem #1 of safe home access

[Note: Incomplete, e.g., Logger, Timer missing ...]
Domain Model (1b)

```
HouseholdDeviceOperator
```

```
LockOperator
```

```
LightOperator
```

```
MusicPlayerOperator
```

```
AlarmOperator
```

...
## Extracting Associations

<table>
<thead>
<tr>
<th>Concept pair</th>
<th>Association description</th>
<th>Association name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller ↔ StatusDisplay</td>
<td>Controller sends current status of locks etc. to StatusDisplay for display</td>
<td>...</td>
</tr>
<tr>
<td>Controller ↔ KeycodeEntry</td>
<td>KeycodeEntry device reads Input and sends it to Controller for further processing</td>
<td>...</td>
</tr>
<tr>
<td>Key ↔ Controller</td>
<td>Controller obtains Key based on Input and metadata (door, time)</td>
<td>obtains</td>
</tr>
<tr>
<td>KeyChecker ↔ Controller</td>
<td>Controller sends Key for checking to KeyChecker and receives check result for further action</td>
<td>conveys requests</td>
</tr>
<tr>
<td>KeyStorage ↔ KeyChecker</td>
<td>KeyChecker retrieves valid keys from KeyStorage for comparison with Key</td>
<td>retrieves valid keys</td>
</tr>
<tr>
<td>Key ↔ KeyChecker</td>
<td>KeyChecker verifies Key</td>
<td>verifies</td>
</tr>
<tr>
<td>HouseholdDeviceOperator ↔ Controller</td>
<td>Controller activates Devices through requests to HouseholdDeviceOperator</td>
<td>conveys requests</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Domain Model (2a)

Domain model for UC-1: Unlock

Associations: who needs to work together, not how they work together

Concept pair | Association description | Association name

Values, e.g.: activated, stopped
### Extracting Attributes

<table>
<thead>
<tr>
<th>Concept</th>
<th>Attributes</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller</td>
<td>numOfAttempts</td>
<td>Used to determine whether another attempt is allowed</td>
</tr>
<tr>
<td></td>
<td>maxNumOfAttempts</td>
<td>Used to limit the retry attempts</td>
</tr>
<tr>
<td>HouseholdDeviceOperator</td>
<td>deviceStatuses</td>
<td>On, off, etc.</td>
</tr>
<tr>
<td>Key</td>
<td>userIdentityCode</td>
<td>Personal code of the resident</td>
</tr>
<tr>
<td></td>
<td>timestamp</td>
<td>Allows for tracking history</td>
</tr>
<tr>
<td></td>
<td>doorLocation</td>
<td>Allows possibility of different codes for doors and also relevant for tracking history</td>
</tr>
<tr>
<td>KeyChecker</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
More examples with explanation can be found in the Textbook
Simple examples of domain modeling

• Course management and POS

• Bank accounts
  - http://documentation.genesez.org/javaee/de.genesez.uml.modeling.domain.html
POS system

ProcessSale: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates the inventory. The customer receives a receipt from the system and then leaves the store with the items.
Next Lecture

- Date/Time:
  - Friday, 30-Sep, 10:15-12:00
- Topic:
  - Development Infrastructure I (by Sander Soo, Nortal)
- For you to do:
  - Remember that all team members must be present in next week’s lab sessions (assessment of Lab task 1 solutions)!
  - Work on Lab 2 homework assignment