Software Specification and Design - Week3

By Keeratipong Ukachoke

Today Topics

- Use case & re-engineering example (IBM)
- User story
- Elaboration - First iteration
- Domain model
- In class assignment

Use case & re-engineering (IBM)

Use case & Re-engineering

- The problem
  - Many companies have legacy systems that are hard to difficult and maintain
  - This is normal situation
- Solution
  - Redevelop them from scratch?
  - Re-engineering

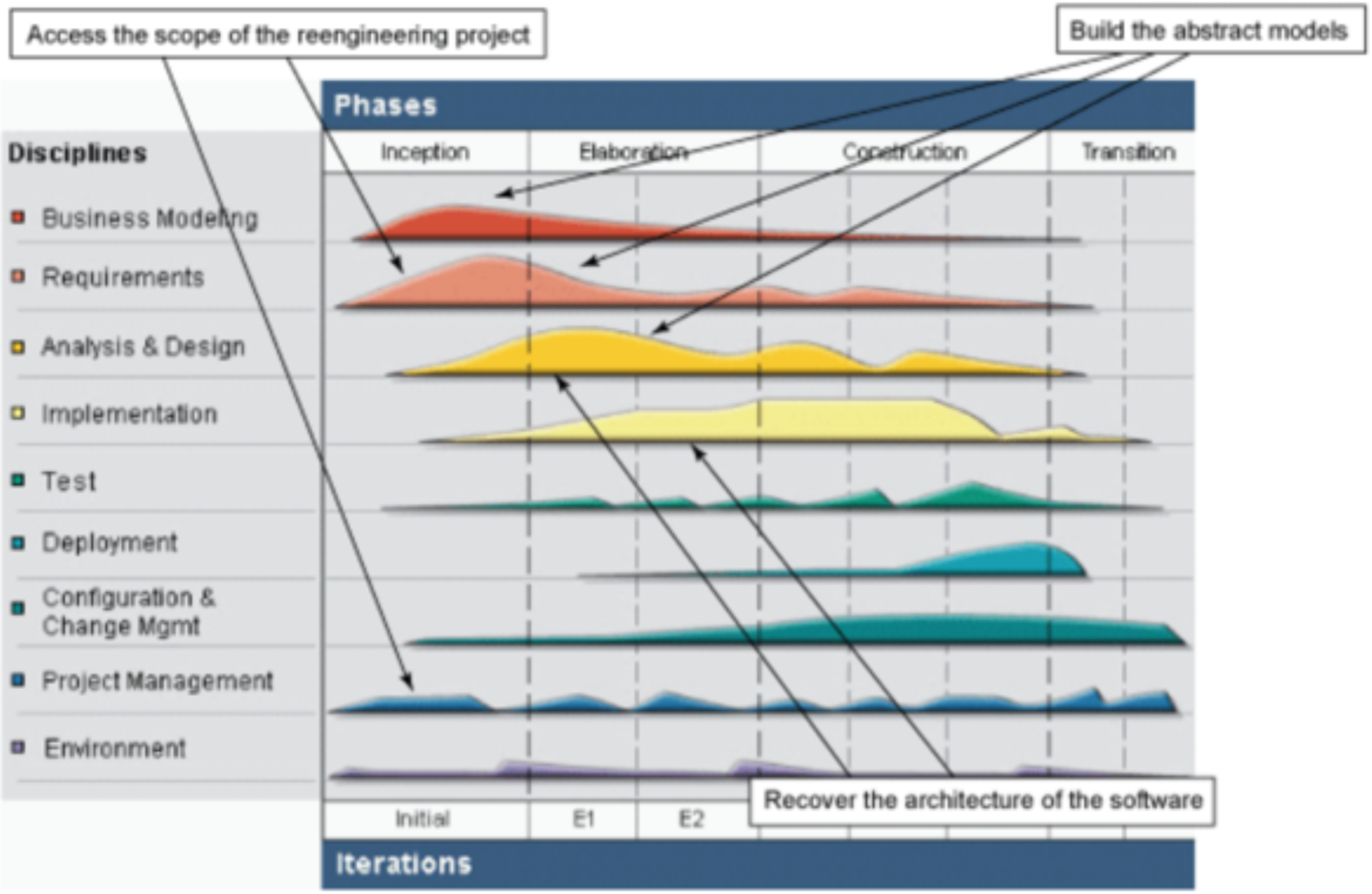
Use case & Re-engineering

- Re-engineering composes of two sub projects
    - Reverse-Engineer
        - What it is?
    - Forward-Engineer
        - What it is?

Use case & Re-engineering

- Reverse engineer
  - Our goal is to make some senses out of the current code so that we can develop something more
  - We can use use cases as the central of the process

IBM case study - RUP



A RUP diagram showing re-engineering process

IBM case study

- Assess the scope of the reengineering project
    - Look at the quality of the code
    - Judge how much we should re-engineer it
    - If it’s really bad, we can just extract the know how

IBM case study

- Some documents at this stage
  - Vision document
  - Supplementary specifications
  - Risk lists

IBM case study

- Build the abstract models
  - Many legacy systems are very complex
  - To get the model, we need to understand the architecture itself
  - How we turn the code into some higher representations?

IBM case study

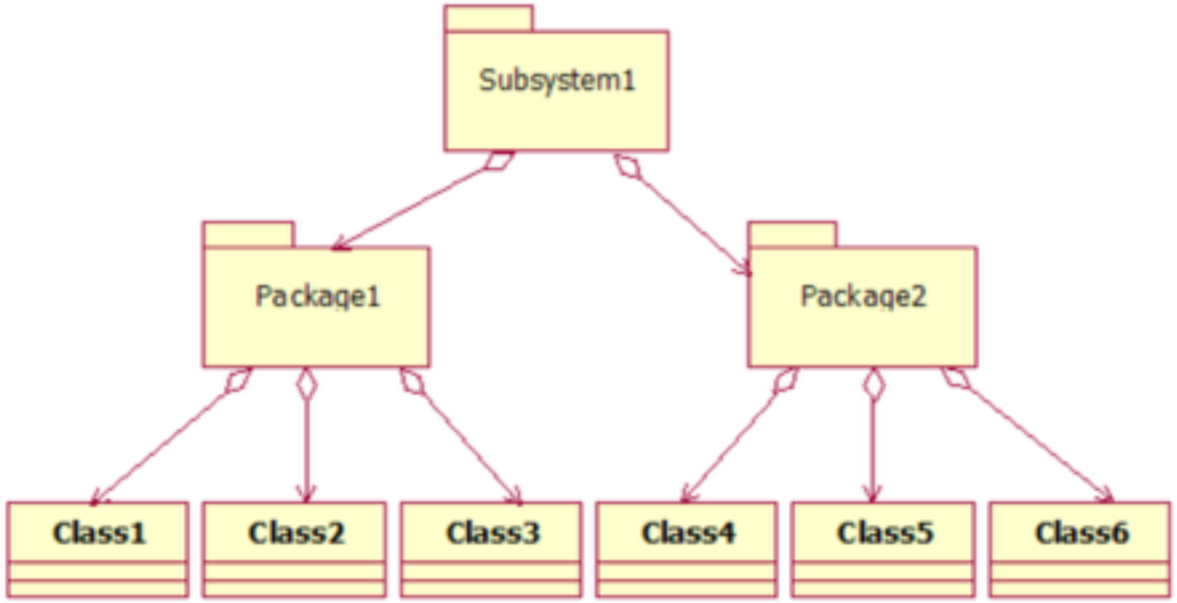
- How to understand the current system
  - Talk to the users that use it
  - Bad thing - They don’t know how the inner stuff works
  - Good thing - They should have a good perspective on legacy system.
  - They know steps of what happen when you interact with the system.
  - Wow, what did we just get? Use cases!!!

IBM case study

Now we have use cases, what next?

IBM case study

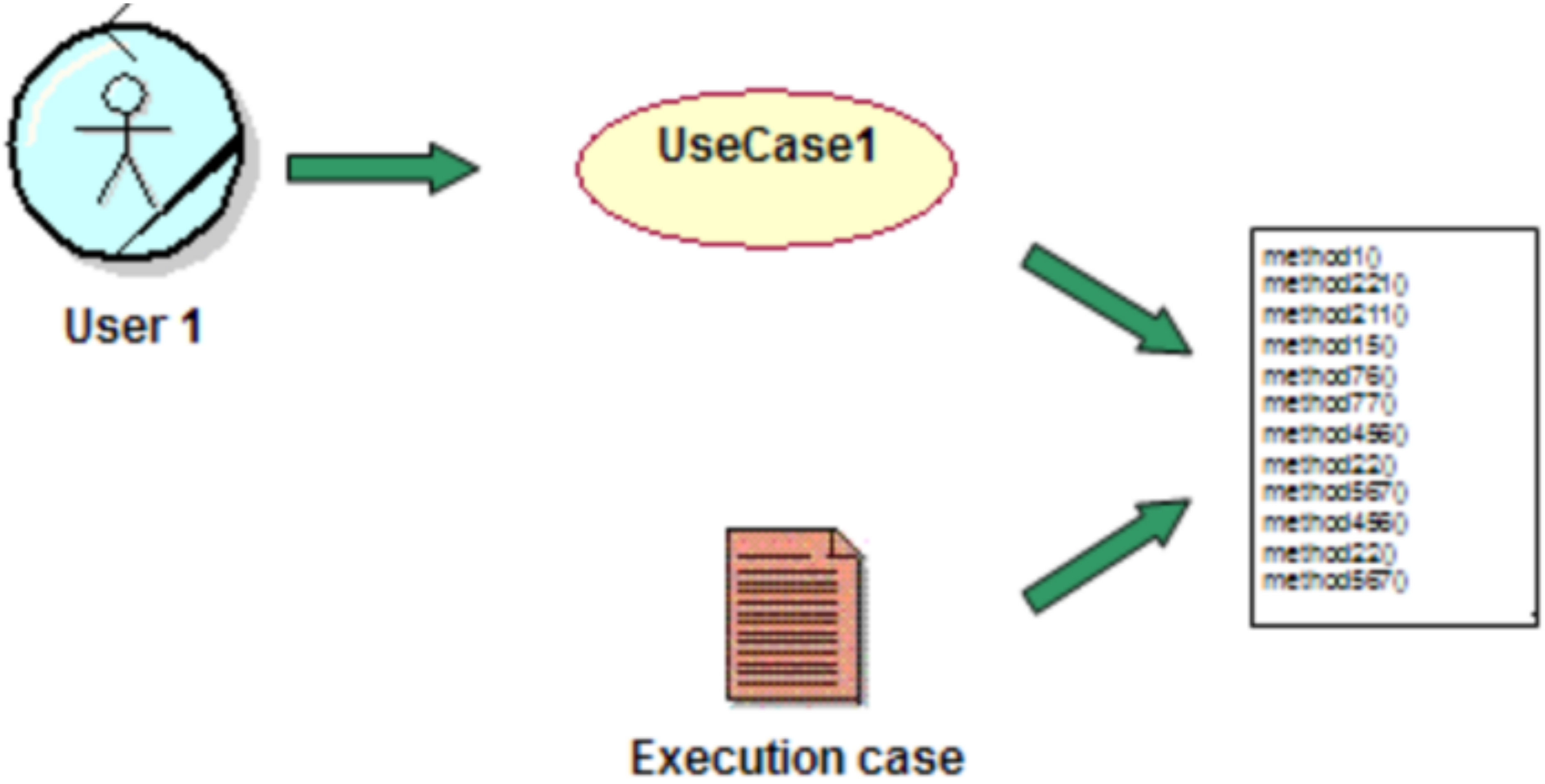
- Recover the architecture of the software
  - Step 1: Analyse the implementation model
  - Folders, libraries, packages, classes



IBM case study

- Recover the architecture of the software
    - Step 2: Run the use case
    - We can’t run on all possible input
    - For now, follow what the user say
    - Record the trace

IBM case study

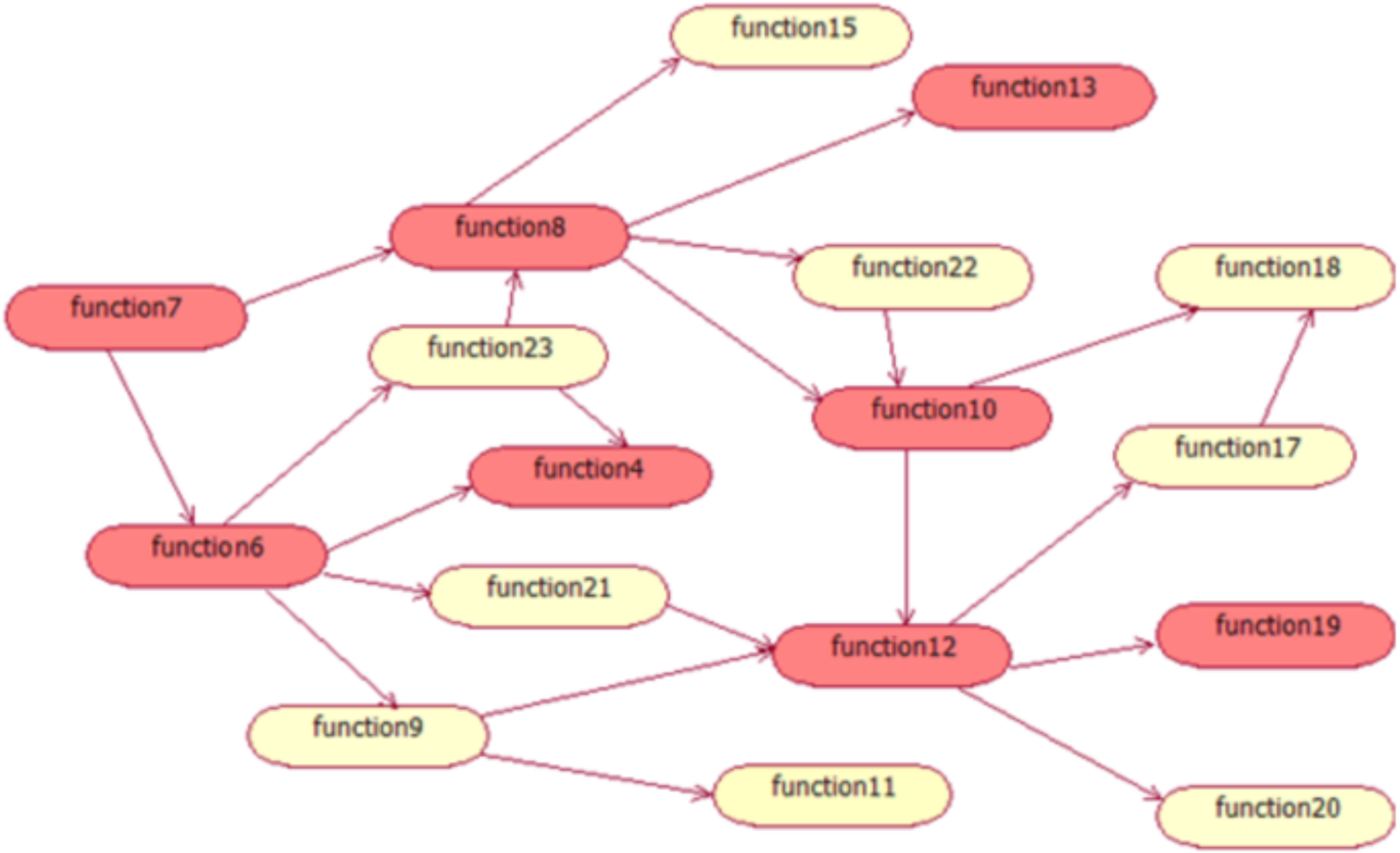


```
method1()
method221()
method211()
method15()
method76()
method77()
method456()
method22()
method567()
method456()
method22()
method567()
```

IBM case study

- Recover the architecture of the software
    - Step 3: Analyse the call graph
    - Find the functions that are called directly
    - Is that complete?

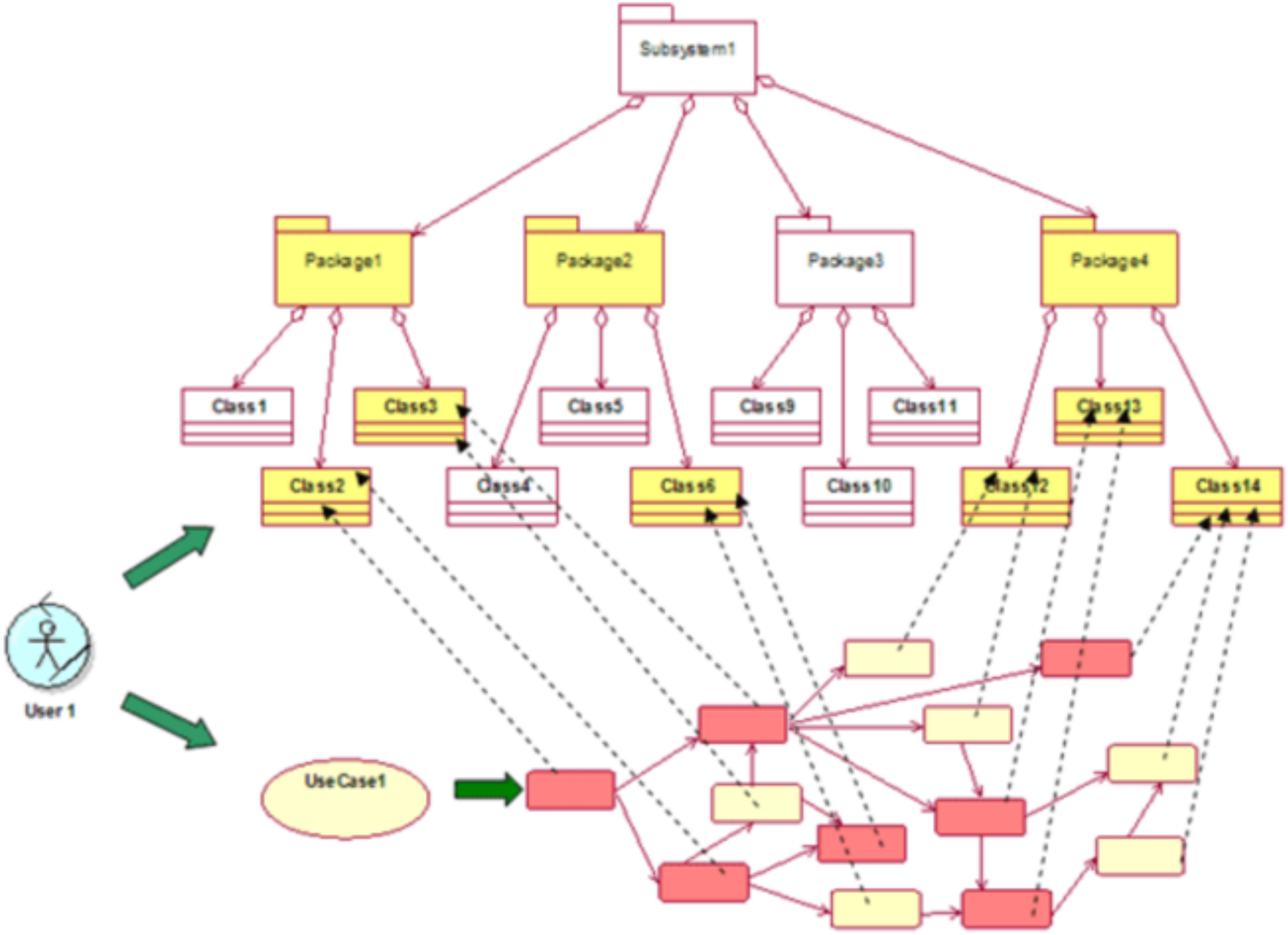
IBM case study



IBM case study

- Recover the architecture of the software
    - Step 4: Map functions to the implementation model
    - All function must be part of some classes right (Java)?
    - This can make us visually what classes are part of use case

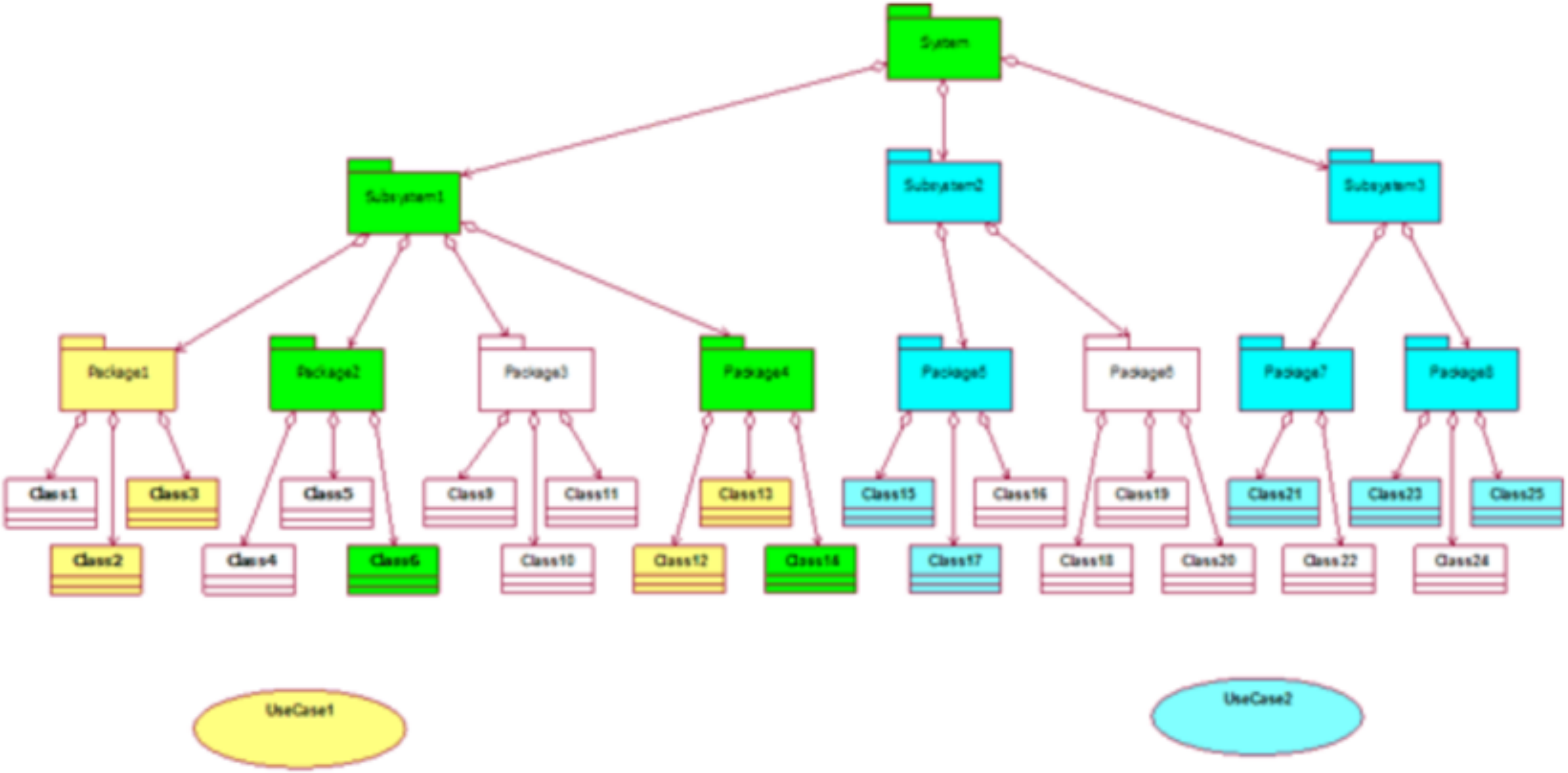
IBM case study



IBM case study

- Recover the architecture of the software
    - Step 5: Validate and rebuild the high level architecture
    - Find specific elements for each use case
    - Find common elements

IBM case study



IBM case study

- Summarise
  - Now that we retrieve the mapping between use cases and implementation elements we can start working on forward engineering
  - For details please see

    http://www.ibm.com/developerworks/rational/library/sep06/dugerdil/

User Story

User story

- Describe of what a user does/need
- Usually in a few sentences
- Who/What/Why
- Use a lot in Agile (often with a Post-it note)

Create a user story

- Who write user stories?
- We usually do in a meeting
- Can be changed later

INVEST guideline

- Independent
- Negotiable
- Valuable
- Estimable
- Scalable
- Testable

User story format

- There are many formats available
  - As a <role>, I want <goal> so that <benefit>
  - As a <role>, I want <goal>
  - In order to <benefit> as a <role> I want <goal>
  - As a <role>, I can <action>so that <benefit>

User story examples

- POS example
  - As a cashier, I want to search for a product by a sku
  - As a manager, I want to override the any operation at POS
  - As a system admin, I want to monitor a connection status between POS and warehouses

User story benefits

- Very brief
- Emphasize more on discussion
- Little maintenance
- Break projects into small parts
- Easier to estimate

User story drawbacks

- Can’t scale very well
- Not very informative
  (In XP, it suggests that customers are always in a team)
- Non-functional requirements?

User story vs Use case

- Different structures.
- Different delivered format
- Scope size

Use case - Process sale

Preconditions: Cashier is identified and authenticated

Postconditions: Sale is saved. Tax is correctly calculated. Account and inventory updated. Commission recorded. Receipt is generated. Payment authorization approvals are record

Main Success Scenario:

1. Customer arrives at POS with items

2. Cashier starts a new sale

3. Cashier enters item id

4. System records sale line item and present item description, price, total

- - - Cashier repeat steps 3-4 until done

5. System presents total with taxes calculated

6. Cashier tells customer the total, and asks for payment

7. Customer pays and system handles payment

8. System logs completed sale and sends sale and payment information to the external Accounting and inventory system

9. System presents receipt

10. Customer leaves with items

Use case and user stories

Can you think of what user stories can come up with that one use case?

Elaboration

POS - First iteration

Our goals

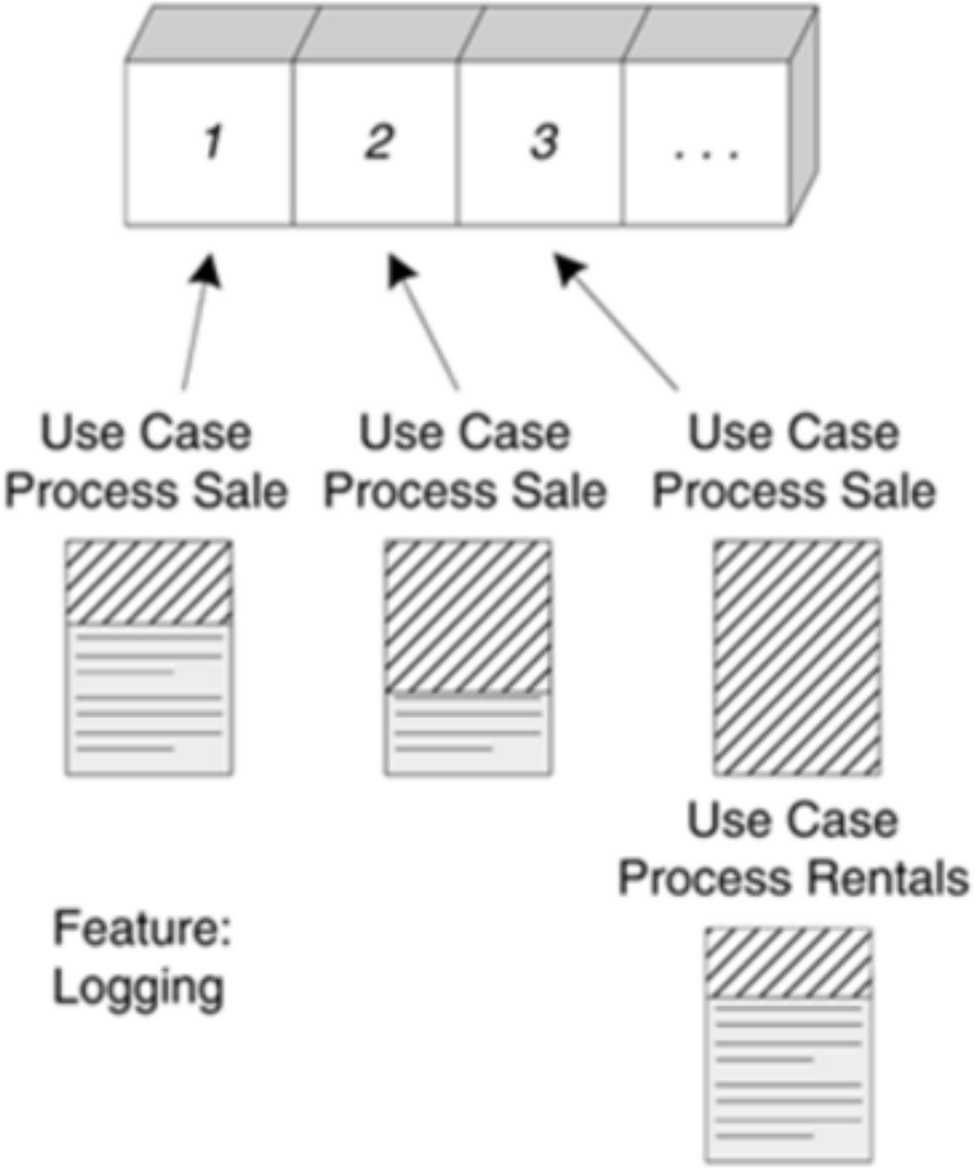
- Implement a basic, key scenario of Process Sale use case: entering items and receiving a cash payment
- Only the success flow
- No 3rd party software
- No complex pricing rules

POS - First iteration

Pitfall : Don’t implement all the requirements at once

- We don’t have all requirements & use cases
- Just a subset of use cases is enough
- Just start!

POS - First iteration



A use case or feature is often too complex to complete in one short iteration.

Therefore, different parts or scenarios must be allocated to different iterations.

Inception vs Elaboration

What we did in Inception (again)

- actors, goals, use cases named
- most use cases are in brief format, ~10% are in fully dressed
- some ui prototyping
- identify what to buy/build?
- Brief architecture designed

Inception vs Elaboration

What we will do in elaboration

- develop and test core, risky architecture
- discover majority of requirements
- mitigate risks
- usually between two to six weeks

Domain model

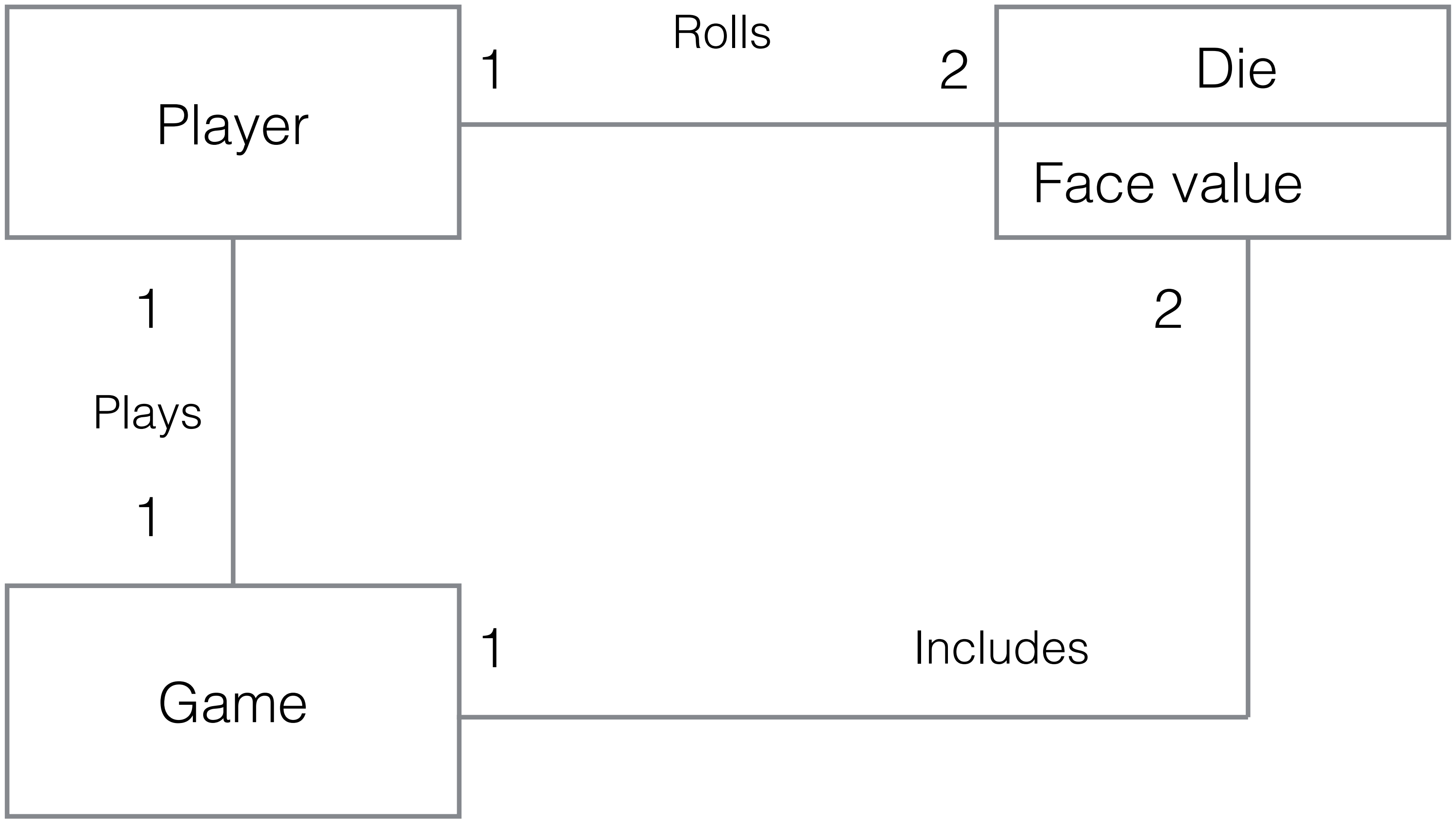
- A visual representation of conceptual classes or real-situation objects in a domain

- Sometimes called conceptual models, domain object models, analysis object model

- Are not software objects or software classes

- [domain objects] [associations] [attributes]

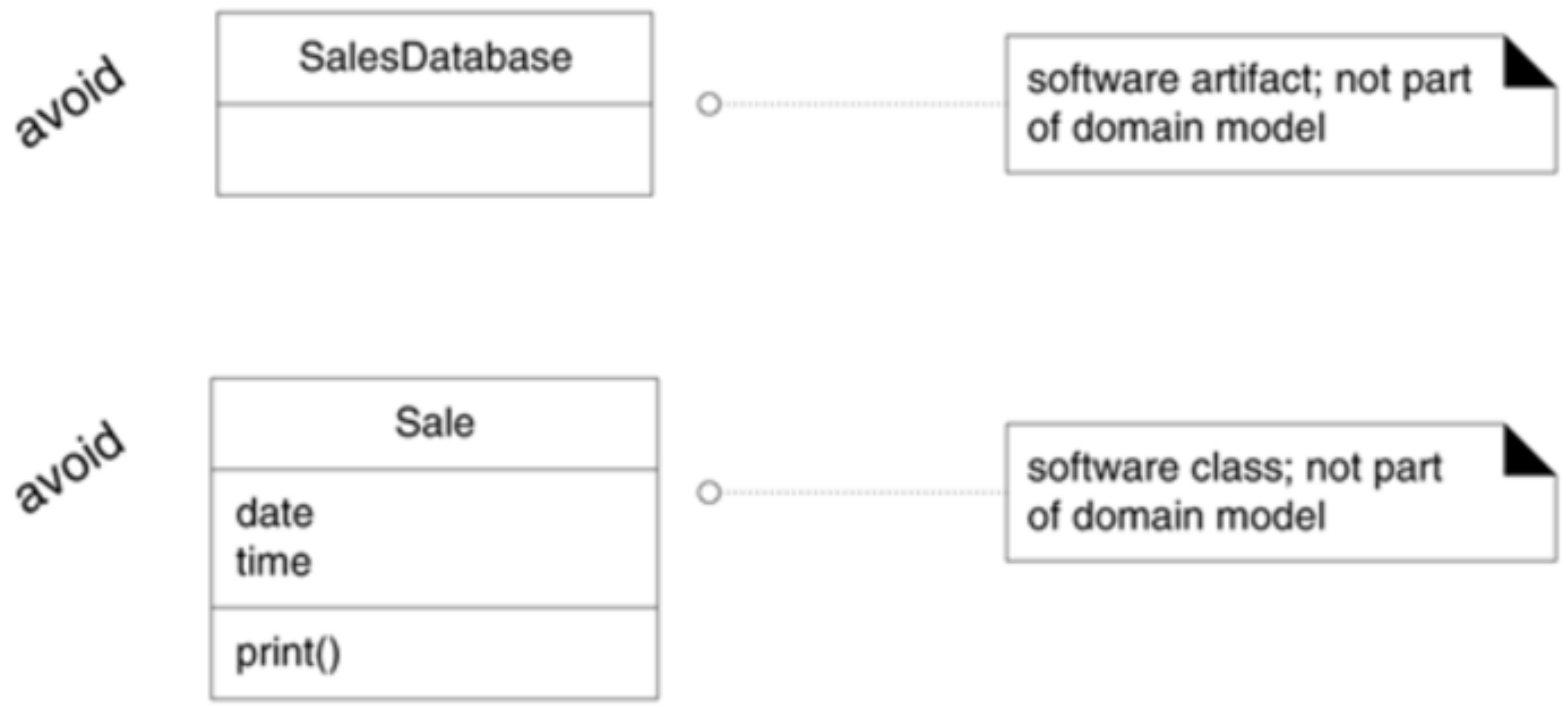
Domain model



Domain model - Things to avoid

- Describe software artifacts like Window, Database
- Specify method to a model
- See examples

Domain model - Things to avoid



avoid
SalesDatabase
software artifact; not part of domain model
avoid
Sale
date
time
print()
software class; not part of domain model

Domain model - conceptual class

- Symbol - words or images representing the model
- Intension - the definition of the model
- Extension - the set of examples to of the model

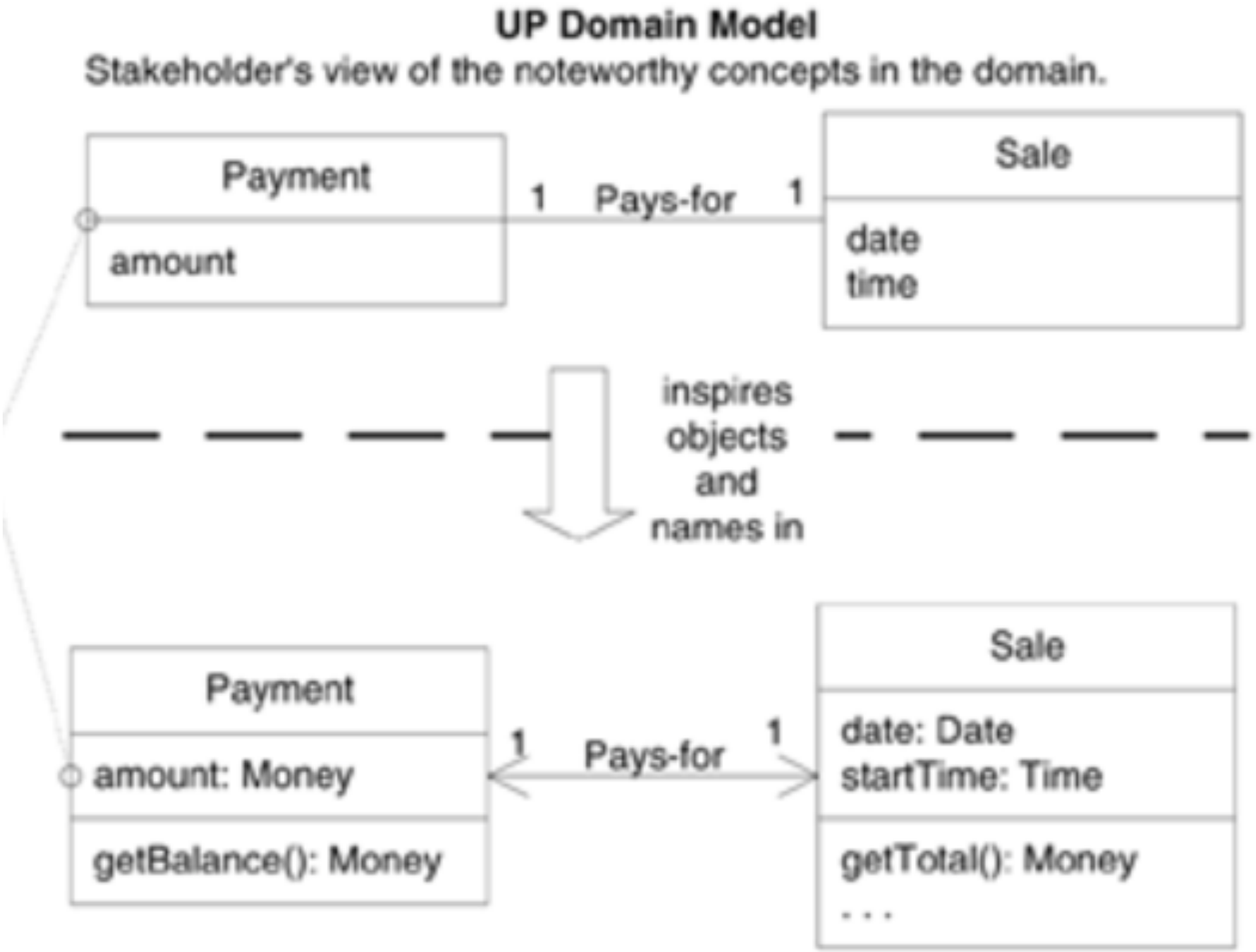
Domain model - conceptual class

- Symbol - Sale
- Intension - A sale represent the event of a purchase transaction. It has a date and time
- Extension - { sale-1, sale-2, sale-3 }

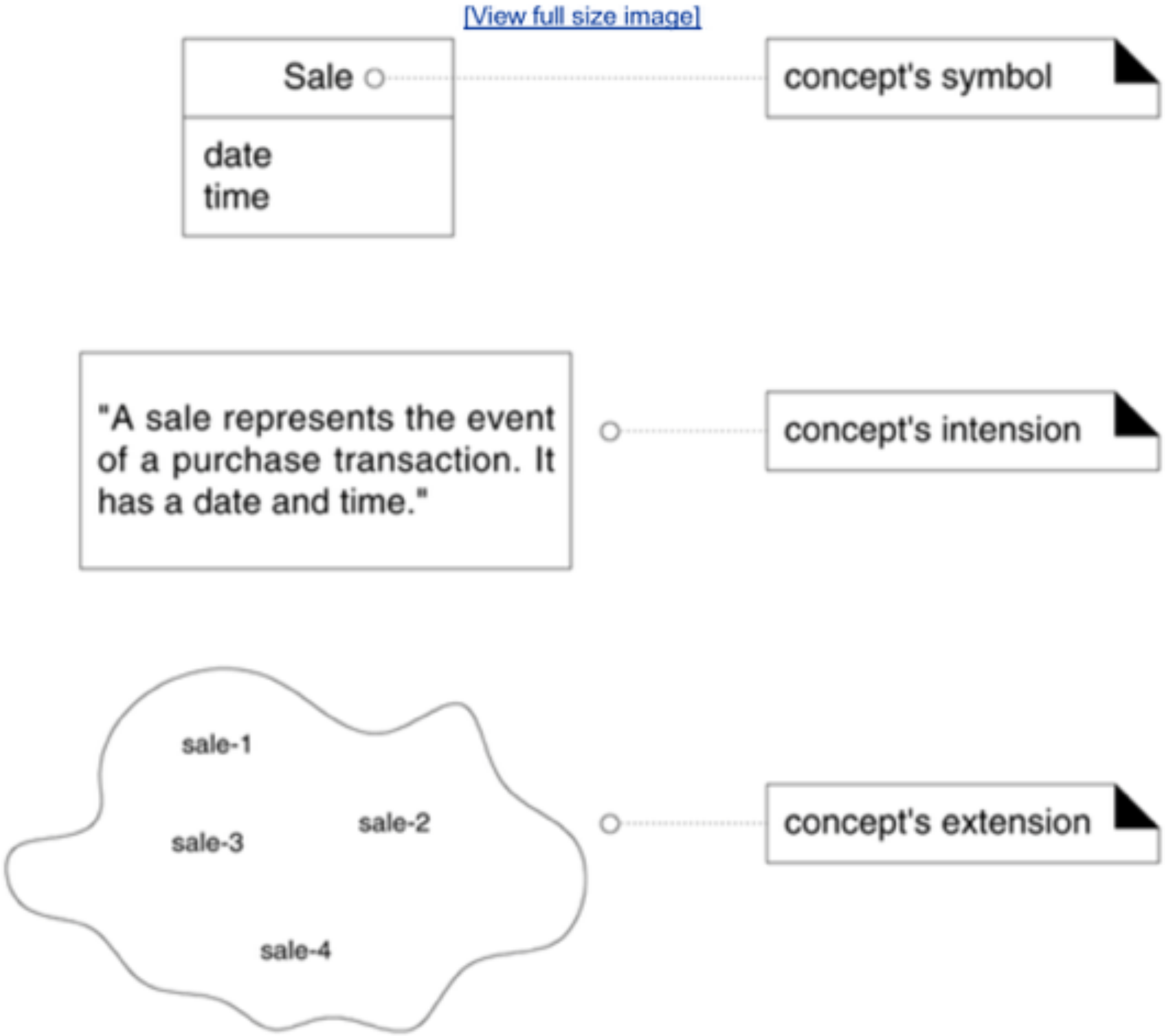
Why create a domain model?

- To understand key concepts of the business
- Get the big picture without worrying about the software details
- Domain model acts as inspirational to create software classes

Why create a domain model?



Domain model - conceptual class



"A sale represents the event of a purchase transaction. It has a date and time."

Domain model - From use case

Preconditions: Cashier is identified and authenticated

Postconditions: Sale is saved. Tax is correctly calculated. Account and inventory updated. Commission recorded. Receipt is generated. Payment authorization approvals are record

Main Success Scenario:

1. Customer arrives at POS with items

2. Cashier starts a new sale

3. Cashier enters item id

4. System records sale line item and present item description, price, total

- - - Cashier repeat steps 3-4 until done

5. System presents total with taxes calculated

6. Cashier tells customer the total, and asks for payment

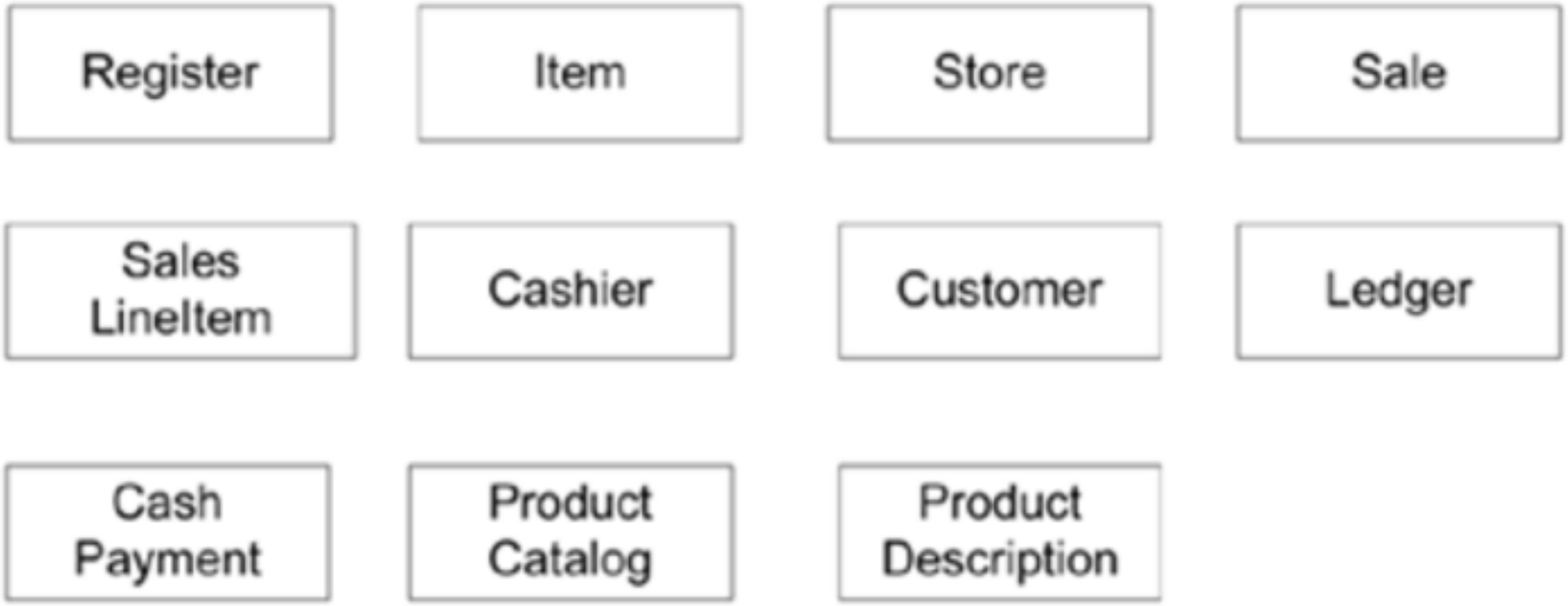
7. Customer pays and system handles payment

8. System logs completed sale and sends sale and payment information to the external Accounting and inventory system

9. System presents receipt

10. Customer leaves with items

Domain model - candidates



Register

Item

Store

Sale

Sales LineItem

Cashier

Customer

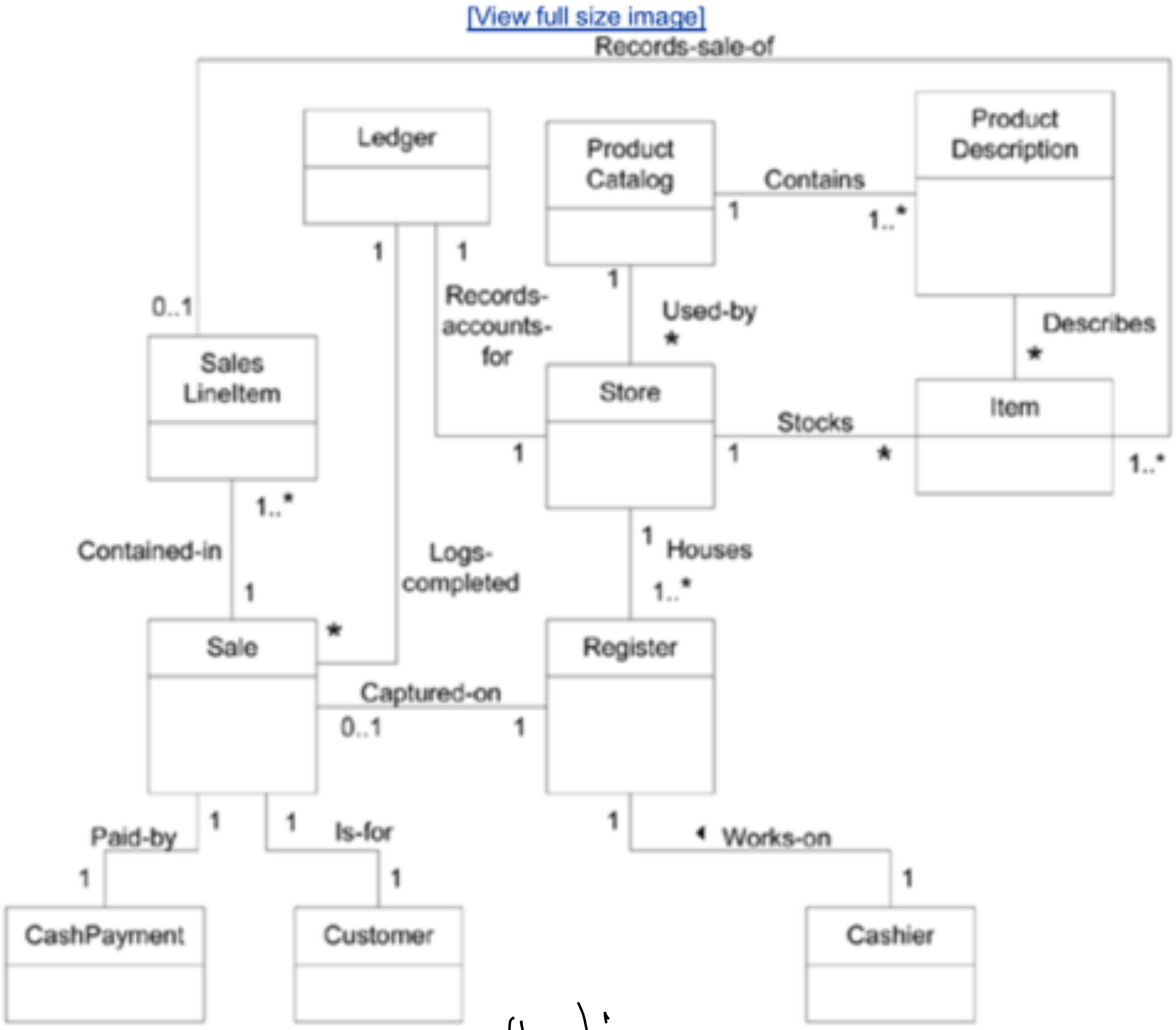
Ledger

Cash Payment

Product Catalog

Product Description

Domain model - POS



have relationship
in real life but not
for programming
don't need to
write

write relationship
do need to show features

Domain model - Attributes vs classes

- If that thing is raw number or text in the real world it might be an attribute
- In the previous model, What is store?
- Flight and airport. What is the relationship?

Description class

- Contains a information that describe something else

- If we don’t have description class, what happen when items are sold out?

- Reduce redundancy

- See example of Airline domain

- See example of mobile packages domain

Association

- When to show association?
- Why too many association is bad?
- Will the association be implemented in software?
- See examples

Association

- How should we name association?
- Has and Use are not very good.
- Sale ‘Use’ CashPayment => Bad
- Sale ‘Paid-by’ CashPayment => Better

Association

- Multiplicity, see examples
- Multiple associations are also possible

Attributes

- When to show attributes?
- No foreign keys

Assignment

- To be explained in class