CHAPTER 4 – Domain Modeling

Introduction
- When, Why, How, What

CRC-Cards
- Problem Decomposition
  + Functional vs. Object-Oriented
  + Classes, Responsibilities & Collaborations, Hierarchies
- Group work
  + Creative thinking
  + Brainstorming & Role-playing

Families of Systems
- Commonalities and Variations
- Feature Diagrams

Conclusion
- Correctness & Traceability

Literature (1/2)

Books
- [Ghez02a], [Somm04a], [Pres01a]
  - ➡ Chapters on Specification / (OO)Analysis/ Requirements + Validation

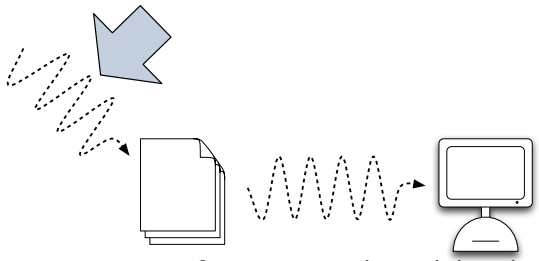
- [Booc94a] Object-oriented analysis and design: with applications, Grady Booch, Addison-Wesley, 1994
  - ➡ A landmark book on what object-oriented decomposition is about.
- [Bell97a] The CRC Card Book, David Bellin and Susan Suchman Simone, Addison-Wesley, 1997.
  - ➡ An easy to read and practical guide on how apply CRC cards in brainstorm sessions with end users.
- [Wirf90a] Designing Object-Oriented Software, R. Wirfs-Brock, B. Wilkerson, L. Wiener, Prentice Hall, 1990.
  - ➡ A book explaining through some good examples how to identify the objects in a requirements specification.
- [Czar00] Generative Programming, K. Czarnecki, U. W. Eisenacker, Addison-Wesley, 2000.
  - ➡ A book explaining "domain engineering", in particular how feature models can be used to specify a family of systems.

Literature (2/2)

Following articles are available on the web
- [Nuse00] Requirements Engineering: A Roadmap", B. Nuseibeh and S. Easterbrook, Proceedings of International Conference on Software Engineering (ICSE-2000), 4-11 June 2000, Limerick, Ireland, ACM Press. [See http://mcs.open.ac.uk/ban25/papers/sotar.re.pdf]
  - An overview of the state-of-the-art in Requirements Engineering. Part of the Future of Software Engineering where a number of prominent authors gave their year 2000 vision on how their software engineering subfield would evolve.
- [Beck89a] "A Laboratory For Teaching Object-Oriented Thinking", Kent Beck, Ward Cunningham, OOPSLA'89 Conference Proceedings. [See http://c2.com/doc/oopsla89/paper.html]
  - The initial paper introducing CRC cards
- [Rubi] "Introduction to CRC Cards", David M. Rubin, White Paper by Softstar Research. [See http://www.softstar-inc.com/Download/Intro%20to%20CRC.pdf]
  - An introduction; talks also about brainstorming and similar techniques.
- [Czar04] "Overview of Generative Software Development" by K. Czarnecki. In J.-P. Banatre et al. (Eds.): Unconventional Programming Paradigms (UPP) 2004, Mont Saint-Michel, France, LNCS 3566, pp. 313Ð328, 2005 [See www.swen.uwaterloo.ca/~kczarnec/gsdoverview.pdf]
  + An overview of generative programming which includes feature modeling.

"Product Line Hall of Fame"
- http://splc.net/fame.html

When Domain Modeling ?



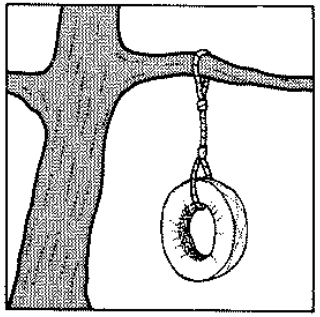
A requirements specification must be *validated*
- Are we building the right system?

A requirements specification must be *analyzed*
- Did we understand the problem correctly ?
  = Are we modeling the problem domain adequately?

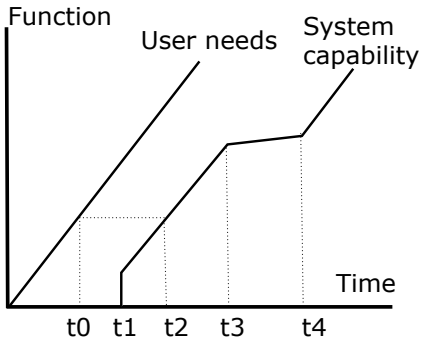
Why Domain Modeling ?

The 30++ years of software development taught us one fundamental lesson...

- The customer doesn't know what he wants!
- And if he does, he will certainly change his mind.



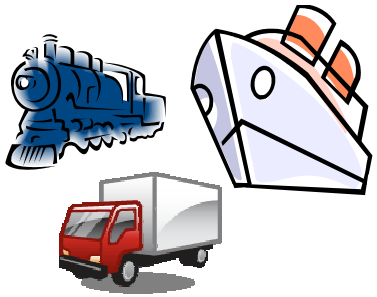
What the customer actually wanted



Why Use Cases are not Sufficient ?

Develop an information system for a transportation company in 1860.

- "Pony Express" Use Cases
  - + refresh horse
  - + replace whip
  - + clean pistol



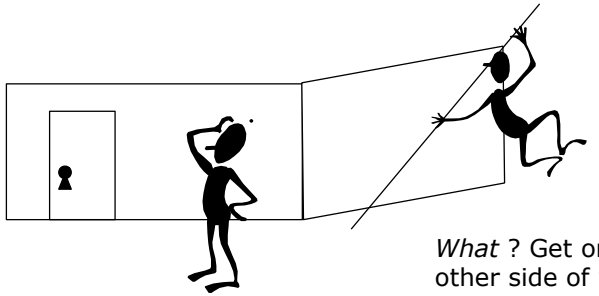
100 years later, "Pony Express" is still operating in the transportation business ...

- How about the Use Cases?
  - + refresh horse
    ⇒ add fuel
  - + replace whip
    ⇒ perform repair
  - + clean pistol
    ⇒ include protection

How Domain Modeling ?

Domain Models help to anticipate changes, are more robust.

- Focus on the what (goal), not on the how (procedure)!



What ? Get on the other side of wall.

How ? Open door, break lock, …
… jump over the wall

What is Domain Modeling ?

Model of Problem Domain

- *Requirements* Model
  - + Focus on WHAT

vs. Model of Solution Domain

- *Design* Model
  - + Focus on HOW

Examples

- CRC Cards
  - + Model the concepts in the problem domain in object-oriented terms.
    - Classes and Inheritance
- Feature Diagrams
  - + Model the requirements of a *family* of systems
    - Commonalities and variations.

Domain Engineering = (definition from [Czar00])

- Domain Engineering is the activity of collecting, organizing, and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets (i.e. reusable workproducts), as well as providing an adequate means for reusing these assets (i.e. retrieval, qualification, dissemination, adaptation, assembly, etc.) when building new systems.

What are CRC Cards ?

CRC = Class-Responsibility-Collaborations

Class: Name	
superclass: list of superclasses subclass: list of subclasses	
responsibility 1 responsibility 2 …	collaborations required to achieve responsibility1 collaborations required to achieve responsibility2 …

+ a short description of the purpose of the class on the back of the card

CRC Cards
- compact, easy to manipulate, easy to modify or discard!
- easy to arrange, reorganize
- easy to retrieve discarded classes

Usually CRC cards are not maintained electronically
- May be used by computer illiterates

Problem Decomposition (1/2)

Object-Oriented Decomposition	Functional Decomposition
Decompose according to the objects a system must manipulate. ⇒ several coupled "is-a" hierarchies	Decompose according to the functions a system must perform. ⇒ single "subfunction-of" hierarchy

Example: Order-processing software for mail-order company	
Order - place - price - cancel Customer - name - address LoyalCustomer - reduction	OrderProcessing - OrderMangement • placeOrder • computePrice • cancelOrder - CustomerMangement • add/delete/update

Problem Decomposition (2/2)

Object-Oriented Decomposition	Functional Decomposition
⇒ distributed responsibilities	⇒ centralized responsibilities

Example: Order-processing software for mail-order company	

`Order::price(): Amount`
`{sum := 0`
`FORALL this.items do`
`{sum := sum + item. price}`
`sum:=sum-(sum*customer.reduction)`
`RETURN sum`
`}`
	`computeprice(): Amount`
	`{sum := 0`
	`FORALL this.items do`
	`sum := sum + item. price`
	`IF customer isLoyalCustomer THEN`
	`sum := sum - (sum * 5%)`
	`RETURN sum`
	`}`

`Customer::reduction(): Amount`
`{ RETURN 0%}`
`LoyalCustomer::reduction(): Amount`
`{ RETURN 5%}`	

Functional vs. Object-Oriented

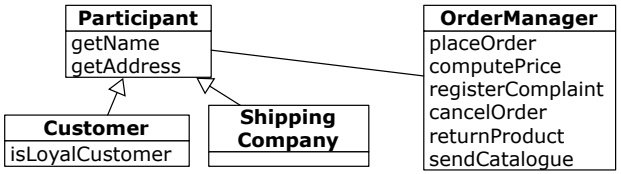
Functional Decomposition
- Good with stable requirements or single function (i.e., "waterfall")
- Clear problem decomposition strategy
- However
  - + Naive: Modern systems perform more than one function
    - What about "produceQuarterlyTaxForm"?
  - + Maintainability: system functions evolve => cross-cuts whole system
    - How to transform telephone ordering into web order-processing?
  - + Interoperability: interfacing with other system is difficult
    - How to merge two systems maintaining customer addresses?

How to find the objects ?

Object-Oriented Decomposition
- Better for complex and evolving systems
- Encapsulation provides robustness against typical changes

God Classes

... or how to do functional decomposition with an object-oriented syntax



Symptoms

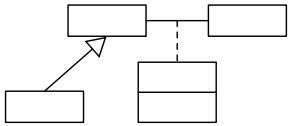
- Lots of tiny "provider" classes, mainly providing accessor operations
  - + most of operations have prefix "get", "set"
- Inheritance hierarchy is geared towards data and code-reuse
  - + "Top-heavy" inheritance hierarchies
- Few large "god" classes doing the bulk of the work
  - + suffix "System", "Subsystem", "Manager", "Driver", "Controller"

Responsibility - driven Design in a Nutshell

Responsibility-driven design is the analysis method using CRC Cards.

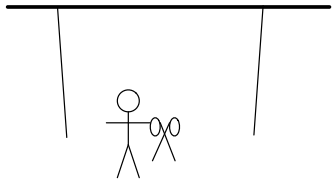
How do you find objects and their responsibilities?

- Use nouns & verbs in requirements as clues.
  - + Noun phrases lead to objects
  - + Verb phrases lead to responsibilities
- Determine how objects collaborate to fulfill their responsibilities.
  - + To collaborate objects will play certain roles
- Why is this important?
  - + Objects lead to classes
  - + Responsibilities lead to operations
  - + Collaborations & Roles lead to associations
- Is it that simple?
  - + No requires creative thinking!



Creative Thinking

Good problem decomposition requires creative thinking.



2 string puzzle

Within one large empty room, there are two long ropes are hanging from the ceiling. The ropes are too far away to reach the one while holding the other. A woman comes in holding a pair of scissors and she ties the ropes together.

How did she achieve this?

Solution

See Communications of the ACM, Vol. 43(7), July 2000, p. 113

Identifying Objects

- Start with requirements specification/scope description/....
- 1. Look for noun phrases:
  - + separate into obvious classes, uncertain candidates, and nonsense
- 2. Refine to a list of candidate classes. Some guidelines are:
  - + Model physical objects — e.g. disks, printers
  - + Model conceptual entities — e.g. windows, files
  - + Choose one word for one concept — what does it mean within the domain ?
  - + Be wary of adjectives — does it really signal a separate class ?
  - + Be wary of missing or misleading subjects — rephrase in active voice
  - + Model categories of classes — delay modeling of inheritance
  - + Model interfaces to the system — e.g., user interface, program interfaces
  - + Model attribute values, not attributes — e.g., Customer vs. Customer Address

Identifying Objects: Example (1/2)

"We are developing order-processing **software** for a **mail-order company** called National Widgets, which is a **reseller** of **products** purchased from various **suppliers**.

- Twice a year the **company** publishes a **catalogue of products**, which is mailed to **customers** and other **interested people**.
- **Customers** purchase **products** by submitting a **list of products** with **payment** to National Widgets. National Widgets fills the **order** and ships the **products** to the **customer's address**.
- The order-processing **software** will track the **order** from the **time it is received** until the **product is shipped**.
- National Widgets will provide **quick service**. They should be able to ship a **customer's order** by the fastest, most efficient means possible."

Identifying Objects: Candidate Classes (2/2)

Nouns & Synonyms	Candidate Class Name
software	-: don't model the system
mail-order company, company, reseller	Company (?: model ourselves)
products	Product (+: core concept)
suppliers	Supplier (+: core concept)
catalogue of products	Catalogue (+: core concept)
customers, interested people	Customer (+: core concept)
list of products, order, customer's order	Order (+: core concept)
payment	Payment (+: core concept)
customer's address	Address (?: customer's attribute)
time it is received	-: attribute of Order
time product is shipped	-: attribute of Order
quick service	-: attribute of Company

*** Expect the list to evolve as analysis proceeds.

- Record why you decided to include/reject candidates
- Candidate Class list follows configuration management & version control

Responsibilities/Collaborations

What are responsibilities?

- The public services an object may provide to other objects,
  - + the knowledge an object maintains and provides
  - + the actions it can perform
- ... not the way in which those services may be implemented
  - + specify what an object does, not how it does it
  - + don't describe the interface yet, only conceptual responsibilities

What are collaborations?

- other objects necessary to fulfill a responsibility
  - + when collaborating these other objects play a role
  - + to play this role, other objects must have certain responsibilities
- empty collaborations are possible
  - + can you argue this responsibility in terms of the class description?

Identifying Responsibilities

To identify responsibilities (and the associated collaborations):

- Scenarios and Role Play.
  - + Perform scenario walk-throughs of the system where different persons "play" the classes, thinking aloud about how they will delegate to other objects.
- Verb phrase identification.
  - + Similar to noun phrase identification, except verb phrases are candidate responsibilities.
- Class Enumeration.
  - + Enumerate all candidate classes and come up with an initial set of responsibilities.
- Hierarchy Enumeration.
  - + Enumerate all classes in a hierarchy and compare how they fulfill responsibilities.

Design guideline(s)

- *** Distribute responsibilities uniformly over classes
  (Classes with more than 12 responsibilities are suspicious)
- *** A class should have few collaborators
  (Classes with more than 8 collaborators are suspicious)

Hierarchies

- Look for "kind-of" relationships
  - + Liskov Substitution principle:
    You may substitute an instance of a subclass for any of its superclasses.
  - + Does the statement "every subclass is a superclass" make sense
    - "Every Rectangle is a Square" vs. "Every Square is a Rectangle"
- Factor out common responsibilities
  - + Classes with similar responsibilities may have a common superclass
- "kind-of" hierarchies are different from "part-of" relationships
  - + Often, the whole will share responsibilities with its part
    (suggesting "kind-of" instead of "part-of")
- Name key abstractions
  - + Not finding a proper name for the root is a symptom for an improper "kind-of" hierarchy Design guideline(s)

Design guideline(s)

- *** Avoid deep and narrow hierarchies
  Classes with more than 6 superclasses are suspicious

Brainstorming

Team

- Keep small: five to six persons
- Heterogeneous:
  - + 2 domain-experts (involved in day-to-day work; not management)
  - + 2 analysts (build connections, abstractions and metaphors)
  - + 1 experienced OO-designer (programmer ⇒ involvement)
  - + 1 facilitator (chairs the meeting)

Tips

- All ideas are potentially good (i.e., may trigger the creative thinking)
- No censorship (even on yourself), no rejection
- Think fast, ponder later
- Produce as many ideas as possible
- Give every voice a turn
- round-robin (with an optional "pass" policy)

Design Guideline(s)

- *** Use white-boards and paper CRC Cards for smooth communication.

Role-playing

Role-playing is a way to achieve common understanding between all parties involved (domain experts, analysts, ...)

Basic Steps

- 1. Create list of scenarios
- 2. Assign Roles
- 3. Each member receives a number of CRC Cards
- 4. Repeat
- 4.1 Rehearse Scenarios
  - + Script = Responsibilities on CRC Cards
- 4.2 Correct CRC Cards and revise scenarios
  - + Rehearsals will make clear which parts are confusing
- 4.3 Until scenarios are clear
- 5. Perform final scenario

Guideline(s)

- *** For tips and techniques concerning role-play, see [Bell97a]

Role-playing: Example

USE CASE 5	Place Order
Goal in Context	Customer issues request by phone to National Widgets; expects goods shipped and to be billed.

Class: Customer	
provideInfo	CustomerRep

Class: CustomerRep	
acceptCall	Customer

Class: Catalogue	

Class: Product	

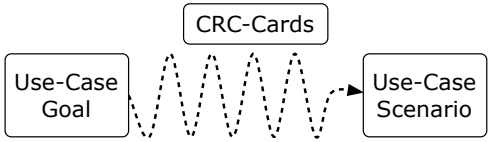
Class: ……	

Class: ……	

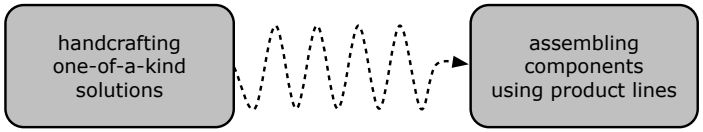
Use cases versus CRC-Cards

- Use cases are a requirements *specification* technique.
- CRC Cards are a requirements *validation* technique.

Use cases & CRC cards complement each other!



Families of Systems



Single System — System Families

Examples from "Product Line Hall of Fame" (http://splc.net/fame.html)
- Mobile phones (Nokia) [1 new phone every day !]
- Television sets, medical systems (Philips)
- Gasoline Systems Engine Control (Bosch)
- Telephone switches (Philips, Lucent)

Feature Models

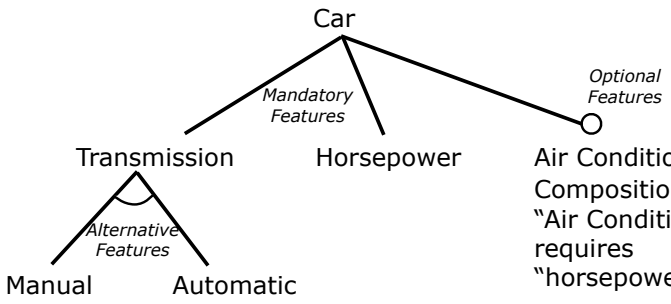
Feature
- a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or system

Feature Models:
- define a set of reusable and configurable requirements for specifying the systems in a domain
- = a model that defines features and their dependencies, typically in the form of a *feature diagram*
- = defines the commonalities and variations between the members of a *software product line*

Feature diagram
- = a visual notation of a feature model, which is basically an and-or tree. (Other extensions exist: cardinalities, feature cloning, feature attributes, …)

Product Line
- = a family of products designed to take advantage of their common aspects and predicted variabilities

Feature Diagram: Example



Composition Rule:
"Air Conditioning" requires "horsepower" > 100

Rationale:
"Manual" more fuel efficient.

From [Czar00], p. 39

Correctness & Traceability

Correctness

- Are we building the system right?
  - + Good maintainability via a robust model of the problem domain.
    - - Specifying the "what" not the "how"

Are we building the right system?

- Model the problem domain from the customer perspective
- Role-playing scenarios helps to *validate* use cases
  - + Paper CRC Cards are easy to reorganize
- Feature Diagrams focus on commonalities/variations
  - + Makes product differences (and choices) explicit

Traceability

- Requirements <-> System?
  - + Via proper naming conventions
  - + Especially names of classes and operations

Summary (i)

You should know the answers to these questions

- Why is it necessary to validate and analyze the requirements?
- What's the decomposition principle for functional and object-oriented decomposition?
- Can you give the advantages and disadvantages for functional decomposition? What about object-oriented decomposition?
- How can you recognize "god classes"?
- Why do you want to identify objects, responsibilities and collaborations?
- What is a responsibility? What is a collaboration?
- Can you supply 3 issues to take care of when chairing a brainstorm meeting?
- What do feature models define ?
- How does domain modeling help to achieve correctness? Traceability?

You should be able to complete the following tasks

- Apply noun identification & verb identification to (a part of) a requirements specification.

Summary (ii)

Can you answer the following questions?

- How does domain modeling help to validate and analyze the requirements?
- What's the problem with "god classes"?
- Why are many responsibilities, many collaborators and deep inheritance hierarchies suspicious?
- What is the solution for the "2 string puzzle" (see Creative Thinking on page 19)? Why is it a good example of creative thinking?
- Can you explain how role-playing works? Do you think it helps in creative thinking?
- Can you compare Use Cases and CRC Cards in terms of the requirements specification process?
- Do CRC cards yield the best possible class design? Why not?
- Why are CRC cards maintained with paper and pencil instead of electronically?
- What would be the main benefits for thinking in terms of "system families" instead of "one-of-a-kind development ? What would be the main disadvantages ?