Lecture 8: “Use Case”-Driven Design

- ➜ The Role of UML in the Software Process
  - E.g. ICONIX
- ➜ Domain Models
- ➜ Use Cases

Where UML fits in

Analysing Requirements

- Use cases - functionality from users’ perspective
- Class diagrams - key domain concepts & terminology
- Activity diagrams - workflow of the organisation
- State diagrams - for domain objects with interesting lifecycles

Design

- Class diagrams - Map of the software structure
- Sequence diagrams - explain common scenarios
- Package diagrams - show the overall architecture
- State diagrams - for object with complex lifecycles
- Deployment diagrams - physical layout of the software

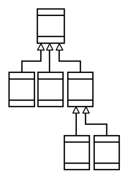
Documentation

- Any sketches that explain key design decisions
- E.g. patterns used, conceptual architecture, unused design alternatives (!)

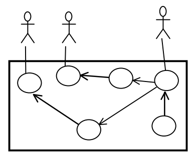
Understanding Legacy Code

- Any sketches that drill down into key parts

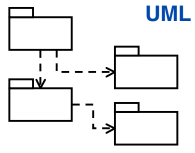
Refresher: UML Notations



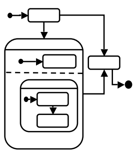
UML Class Diagrams
- information structure
- relationships between data items
- modular structure for the system



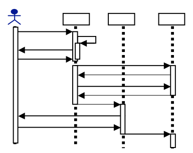
Use Cases
- user's view
- Lists functions
- visual overview of the main requirements



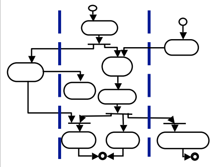
UML Package Diagrams
- Overall architecture
- Dependencies between components



(UML) Statecharts
- responses to events
- dynamic behavior
- event ordering, reachability, deadlock, etc

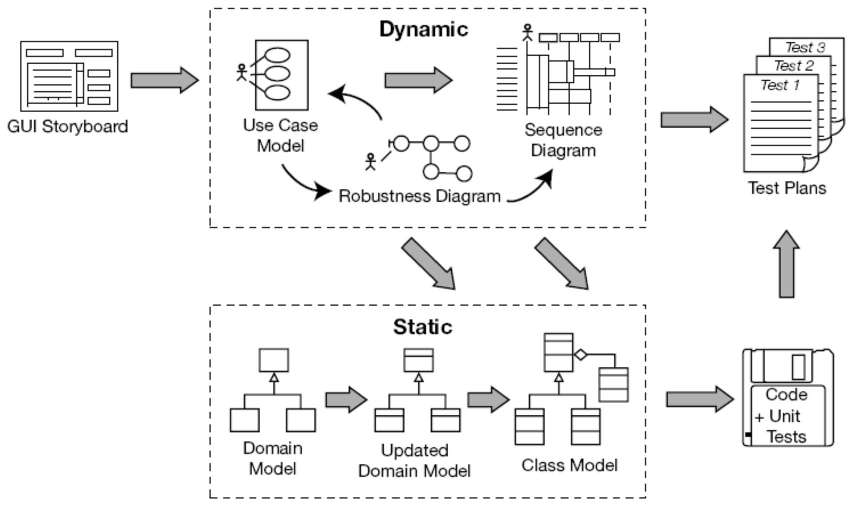


UML Sequence Diagrams
- individual scenario
- interactions between users and system
- Sequence of messages

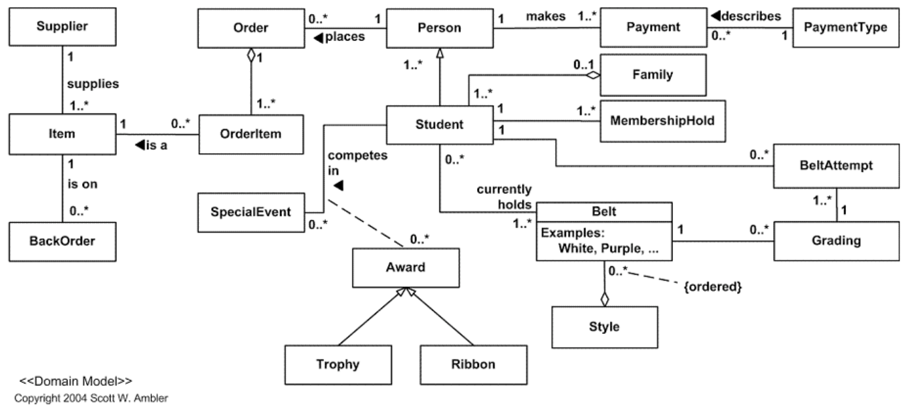


Activity diagrams
- business processes;
- concurrency and synchronization;
- dependencies between tasks;

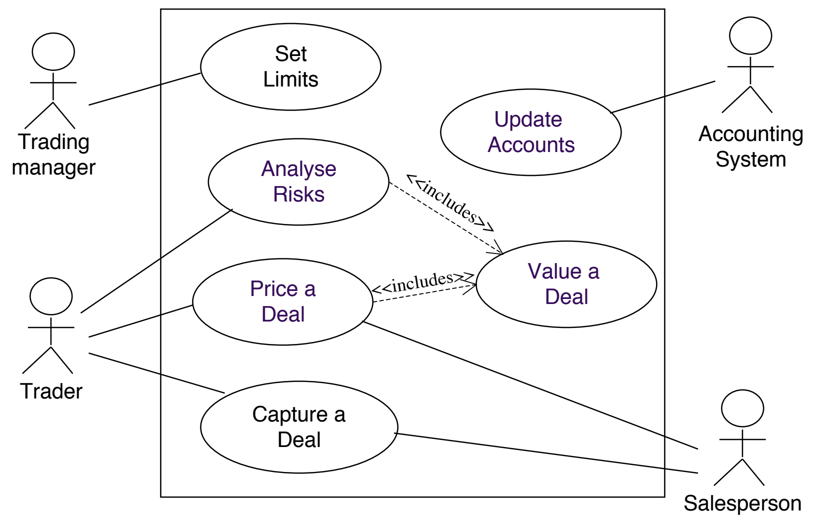
ICONIX process



Domain Model



Use Case Diagram



Documenting Use Cases

For each use case:

- prepare a "flow of events" document, written from an actor's point of view.
- describe what the system must provide to the actor when the use case is executed.

Typical contents

- How the use case starts and ends;
- Normal flow of events;
- Alternate flow of events;
- Exceptional flow of events;

Documentation style:

- Choice of how to elaborate the use case:
  - English language description
  - Activity Diagrams - good for business process
  - Collaboration Diagrams - good for high level design
  - Sequence Diagrams - good for detailed design

Detailed Use Case

Buy a Product

Main Success Scenario:

1. Customer browses catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address, next-day or 3-day delivery)
4. System presents full pricing information
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

Extensions:

3a: Customer is Regular Customer
  .1 System displays current shipping, pricing and billing information
  .2 Customer may accept or override these defaults, returns to MSS at step 6

6a: System fails to authorize credit card
  .1 Customer may reenter credit card information or may cancel

Finding Use Cases

Browse through existing documents

- noun phrases may be domain classes
- verb phrases may be operations and associations
- possessive phrases may indicate attributes

For each actor, ask the following questions:

- Which functions does the actor require from the system?
- What does the actor need to do ?
- Does the actor need to read, create, destroy, modify, or store some kinds of information in the system ?
- Does the actor have to be notified about events in the system?
- Does the actor need to notify the system about something?
- What do those events require in terms of system functionality?
- Could the actor's daily work be simplified or made more efficient through new functions provided by the system?

Good Advice (from RUP)

Adapt the Process

- Rightsize your process
- Continuously reevaluate what you do

Balance Stakeholder Priorities

- Understand the problem domain
- Describe requirements from the user's perspective
- Prioritize requirements for implementation
- Leverage legacy systems

Collaborate across Teams

- Build high-performance teams
- Organise around the architecture
- Manage versions

Demonstrate Value Iteratively

- Manage risk
- Do the project in iterations
- Embrace and manage change
- Measure progress objectively

Elevate the level of abstraction

- Use patterns
- Architect with components and services
- Actively promote reuse
- Model key perspectives

Focus continuously on quality

- Test your Own Code
- Use test automation where appropriate
- Everyone owns the product