COMP3004 Notes

William Findlay

January 15, 2019

Contents

1 Introduction

1.1 Development Life Cycle

1.1.1 Requirements Analysis

- requirements
  - ➢ functional
  - ➢ non-functional
- functional model
- dynamic model
- analysis object model

1.1.2 High Level System Design

- subsystem decomposition
- system architecture strategies

1.1.3 Detailed Object Design

- detailed object model
  - ➢ class diagrams
- design patterns and contracts

1.1.4 Implementation

- map associations to
  - ➢ collections (easy)
  - ➢ storage (hard)

1.1.5 Testing

- unit testing
- integration testing
- system testing

1.2 Team Work

- we can't each do a part and put it together
- we have to do it all together

1.3 Tools

- VirtualBox
- VM
  - ➢ Qt Framework comes with it
  - ➢ Dia comes with it
- C++

1.4 Textbook

- textbook is a good indication of how much detail you need for deliverables
  - ➢ follow the arena case study
  - ➢ perfect level of detail

2 Software Engineering Overview

2.1 Definitions

- software engineering
  - software
    - code
    - application
  - engineering
    - technical process for achieving a task
    - building something
  - what **is** software engineering
    - requirements analysis
    - building software
  - what is **not** software engineering
    - building tiny little program
- system
  - what is a system in software engineering?
    - a very **large** piece of software
    - so big, we don't call it
      - a program
      - an application
- we need a **reliable process**
  - a *recipe*
  - why?
  - wanted:
    - **reliable** systems
    - **modifiable** systems
      - we don't want to throw away code to add a new feature
  - we need a **plan**

2.1.1 The Plan

- two ingredients
  - technical
  - management

Technical Aspects

- **understand** the problem
  - *how do we do this?*
  - **ask the client**
- figure out an **optimal solution**

Management Aspects

- keep things **on track**
- plan for change
  - *anything can change at any time*

2.2 Technical Aspects

2.2.1 Application Domain

- **relevant to the problem**

- the *client's world*
- airport example
  - planes
  - runways
  - gates
  - passengers
  - luggage
- we are **not** experts here
  - the *client* is

2.2.2 Solution Domain

- the **fix** for the problem
- *our* world
- GUI
- design patterns

2.2.3 Building Models

- what is a model?
- why do we need a model?
- what can go wrong?
- types
  - functional
  - dynamic
  - object

The Point of Models

- look at a small scale version
  - don't necessarily build a small scale version
  - look at some different *views* of it
- figure out
  - *how will it work?*

Modeling the Application Domain

- requirements analysis
  - **describe** problem to be solved
  - **describe** system requirements
  - **identify** objects required

2.3 Management Aspects

- communication tools
- configuration management
- rationale management
- software development process

2.3.1 Dealing With Change

- the earlier the better

2.3.2 The Stakeholders

- client
  - ➢ users
  - ➢ interacts with
    - ■ project managers
    - ■ requirements team manager
      - ▹ QA
- development team
  - ➢ project manager
  - ➢ architect
  - ➢ analyst
  - ➢ designers
  - ➢ programmers
  - ➢ testers
  - ➢ operations

2.4 Software Development Phases and Products

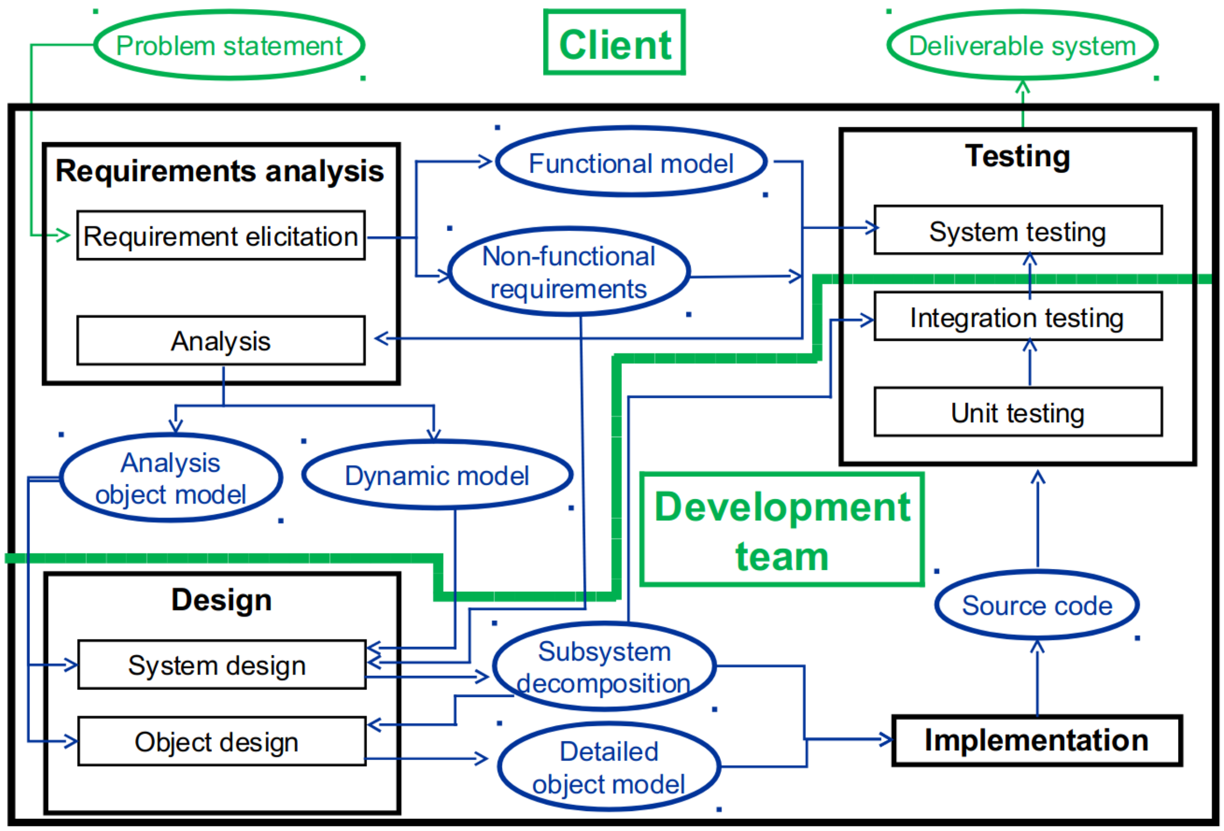

Figure 1: Phases of software development.

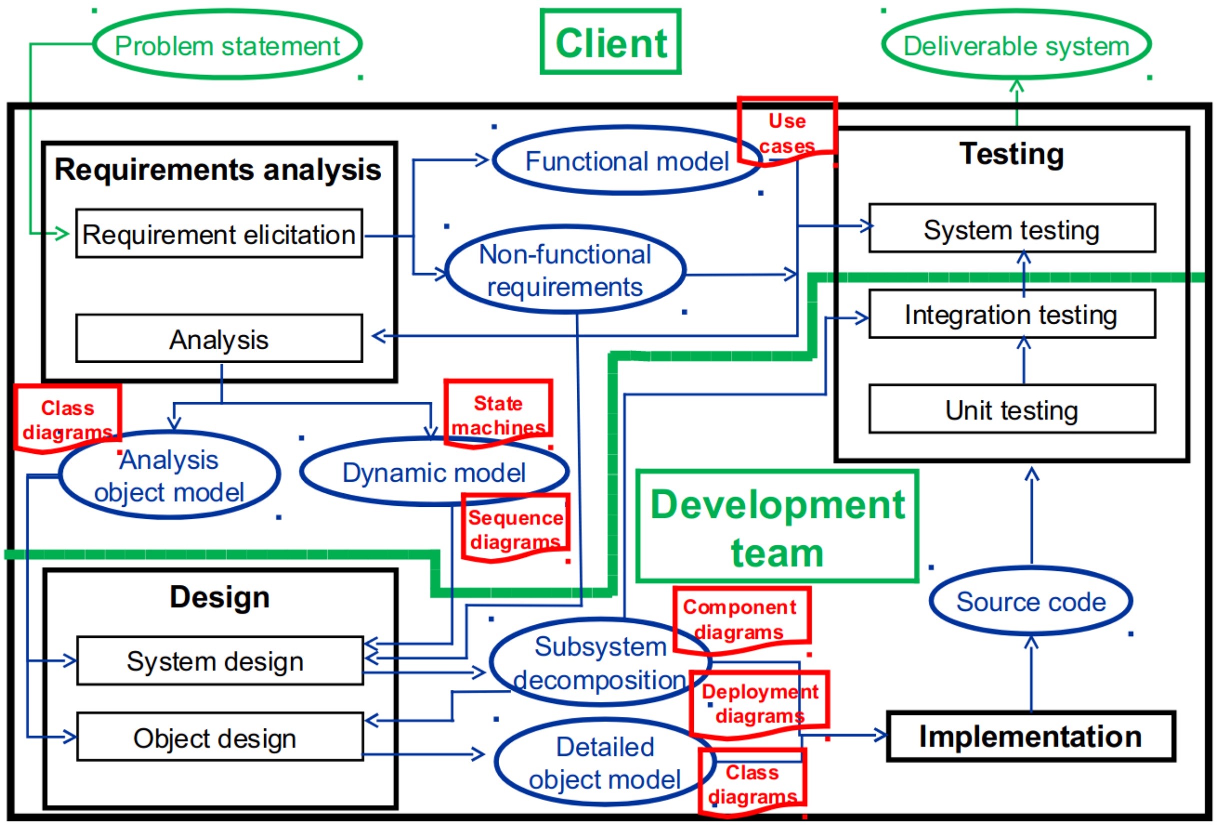

Figure 2: Phases with their products.

3 Team Organization

3.1 People Management

- we **all** manage people
  - ➢ what others expect of us
  - ➢ what we expect of others
- **communication**
  - ➢ speak up about issues
- everyone has bad days
  - ➢ your own
    - ■ don't be a diva
  - ➢ other people's
    - ■ humor and empathy
- **celebrate successes**

3.1.1 Four Factors in Managing People

- consistency
  - ➢ treat others equally
  - ➢ equally ≠ identically
- respect
  - ➢ appreciate different skills
- inclusion
  - ➢ listen to all ideas

- honesty
  - about work
  - about skills

3.1.2 Recipe for Success

- **team meetings** are **essential**
  - Discord
  - in person
- assign people roles that
  - they are **good at**
  - they **enjoy**
- leader works **for** the team
  - encourage
  - motivate
  - listen

3.2 Team Structure

- team leader
- primes (all four people have one or two of these)
  - documentation (ONE or TWO people)
    - documents have consistent formatting
  - requirements (ONE or TWO people)
    - ensure all requirements are documented and traceable
  - architecture/design (ONE or TWO people)
    - ensure design is complete **and optimal**
  - testing (TWO people, MUST pick ANOTHER ROLE)
    - ensure all features match requirements
  - configuration (ONE person ONLY, MUST pick ANOTHER ROLE)
    - ensure deliverable is packed correctly
- coding (all four people are assigned here
  - back end
  - front end

4 The Project

4.1 Deliverable 1

- requirements analysis document
  - Christine says she might scale this slightly back
- implementation of selected features
  - demo

4.2 Deliverable 2

- algorithm design document
  - and **slides** for presentation
  - we **cannot** modify the slides we submit
- in-class presentation
  - on the document/slides we submitted
- implementation of selected features
  - demo

4.3 Deliverable 3

- system design document
- implementation of selected features
    - demo

4.4 Deliverable 4

- document revisions
    - algorithm design document specifically, Christine thinks
- implementation of selected features
    - demo

4.5 Expectations

- **everyone** has to work, **no exceptions**
    - 25% each
- **follow** the **formats discussed in class**
- **end results matter**
    - **not** effort
    - **only** results
- submissions **must** be accompanied by **peer evaluations**
    - grades will be adjusted based on contribution

4.6 System

- Carleton University Animal Care System
    - cuACS

5 UML Notation

5.1 UML Overview

- **unified modeling language**
- what is it?
    - a tool for **expressing system models**
        - functional
        - dynamic
        - object

5.1.1 The UML Family

- each notation is for a **specific model**
- models and notations
    - functional
        - use case diagrams
    - dynamic
        - state machine diagrams
        - sequence diagrams
        - activity diagrams
    - object
        - class diagrams

5.2 Use Case Diagrams

- what is a use case?
  - ➢ behavior observed by **external entities**
  - ➢ entities called **actors**
    - ■ end users
      - ▷ different roles
    - ■ external systems
      - ▷ systems that our system will **interact with**
  - ➢ can also be represented *textually*
    - ■ table-based
- what are use case diagrams?
  - ➢ graphical representation of use cases
- purpose
  - ➢ system boundaries
  - ➢ **always use a box in the drawing**

5.2.1 Some Rules

- the box is important
- ovals for use cases
  - ➢ use cases are labeled with verb phrases
- actors
  - ➢ draw as stick figures
  - ➢ an actor is a **role**
  - ➢ not necessarily a person
  - ➢ a person can have more than one role
- in our project
  - ➢ SQL and Qt are **not external roles**
  - ➢ they are part of the system

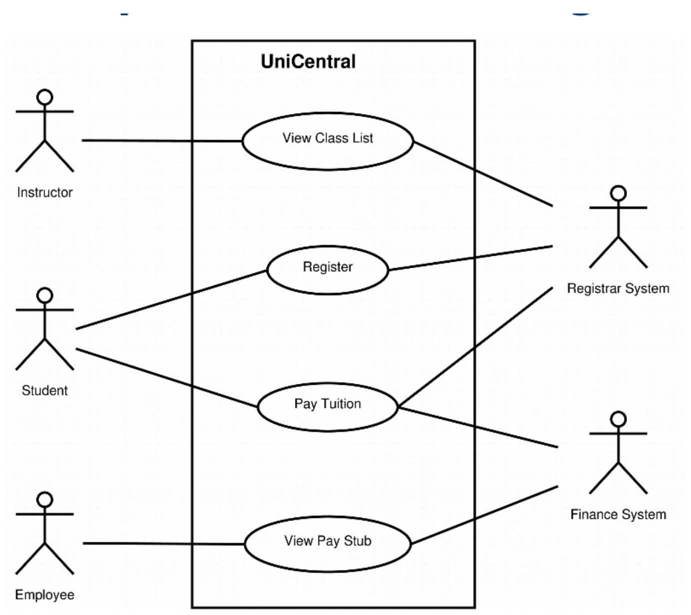

Figure 3: An example of a use case diagram. The stick figures are actors. The bubbles inside the box are use cases. A use case is always labeled with a verb phrase.

5.3 Class Diagrams

- graphical representation of classes and **objects**
- purpose
  - ➢ describe a system
  - ➢ in terms of **classes**
  - ➢ include
    - ■ attributes
    - ■ operations
    - ■ associations

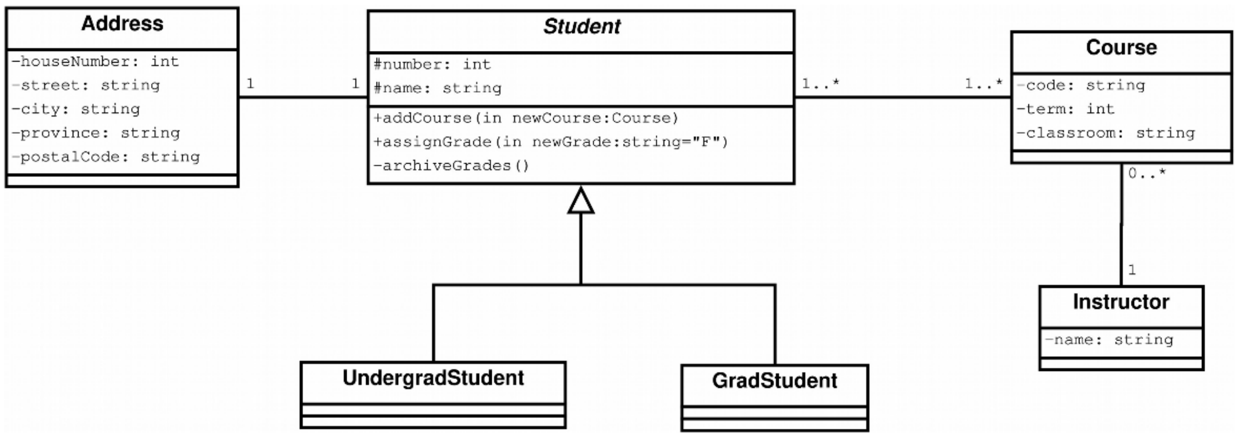

Figure 4: An example of a class diagram. Each class is represented by a box with a name, attributes, and operations and is connected to other classes by associations.

5.3.1 Some Rules

- three sections
  - ➢ class name
  - ➢ attributes
  - ➢ operations
- attributes
  - ➢ access specifier
    - ■ + public
    - ■ # protected
    - ■ − private
  - ➢ name
  - ➢ : followed by data type
- operations
  - ➢ access specifier
    - ■ + public
    - ■ # protected
    - ■ − private
  - ➢ name
  - ➢ parameters
    - ■ input
    - ■ output
    - ■ input-output
- associations

- ➢ direction
  - ■ directed
  - ■ undirected
- ➢ types
  - ■ inheritance
    - ▹ aggregation
  - ■ composition
- ➢ cardinality
  - ■ none-to-many `0..*`
  - ■ one-to-many `1..*`
  - ■ etc.

5.3.2 Object Diagrams

- underlined ⟹ specific instance
  - ➢ also include an instance name before a : in front of class
  - ➢ sometimes just a : if instance name is implied

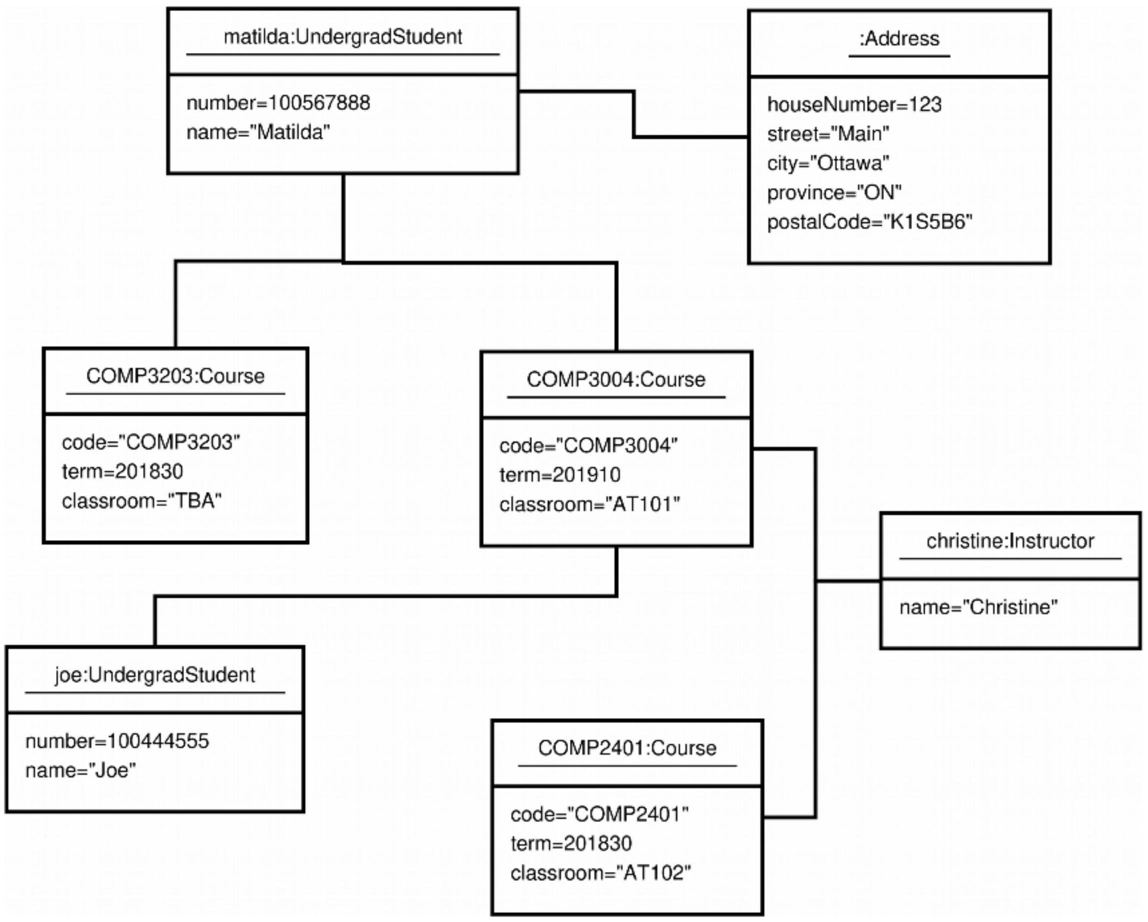

Figure 5: An example of an object diagram. Note that the object name is not always specified if it is obvious.

5.4 State Machine Diagrams

- graphical representation of the **state** of a **single objects**
  - ➢ only more complicated ones
  - ➢ some may not have any states
- purpose
  - ➢ set of states
  - ➢ transitions from one state to another
  - ➢ state:
    - ■ attribute values for an object
  - ➢ transition:
    - ■ conditions under which an object changes state

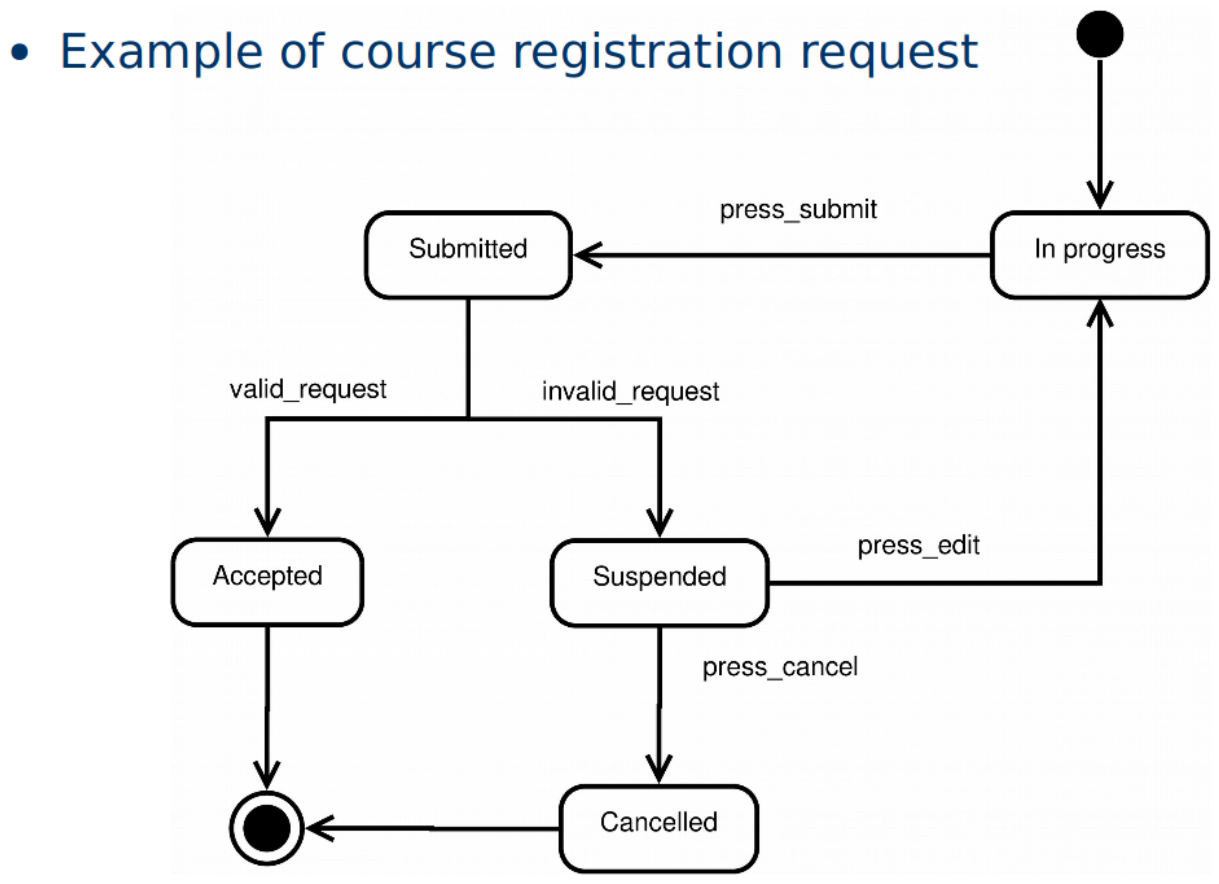

Figure 6: An example of a state machine diagram for a course registration request. The bubbles are the states and the labeled arrows are the transitions. Also note that a state can have one or more transitions to itself.

5.4.1 How it Looks

- states in bubbles
- arrows (transitions)
  - ➢ labels are mandatory
    - ■ except labels **from start** or **to end**
  - ➢ labels are the transitions
  - ➢ you can have arrows from a state to itself

5.5 Activity Diagrams

- we won't use these a lot
- what are they?
    - ➢ system behavior
        - ■ sequencing
        - ■ coordination
- purpose
    - ➢ describe sequential steps in processing
        - ■ control flow
        - ■ concurrency

- Example of course registration validation

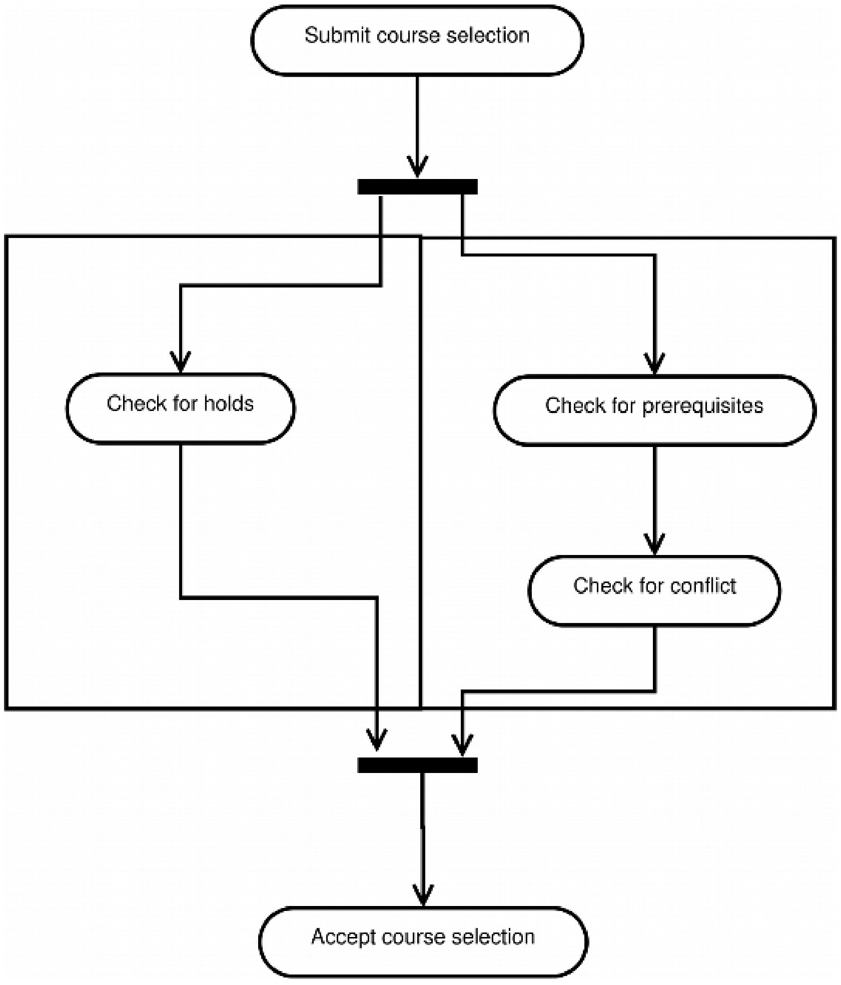

Figure 7: An example of an activity diagram. The two halves of the sqare are called "swim lanes". You can also have one swim lane or more than two swim lanes.

5.6 Sequence Diagrams

5.7 Packages

6 Requirements Analysis

- some case studies to read
  - ➢ DorcSlayer
  - ➢ arena case study from textbook

6.1 Overview

6.1.1 Purpose

6.1.2 Work Products

6.1.3 Breakdown

6.2 Requirements Elicitation

6.3 Analysis