What Does A Design Look Like?

FIT2099: SEMESTER 1 2018

What is design?

- Making decisions about how to build software.

What is conscious, good design?

- Systematically making good decisions about how to build software.
- Turning a big hard problem into lots of easy small ones.

Do we need to record design?

- Ultimately, code records design.
- Do we need an additional record/representation?

```
ublic class UnitMarks {

rivate HashMap<String, Integer> marks;

ublic UnitMarks() {
  marks = new HashMap<String,
                     Integer>();


ublic void setMark(String studentId,
                   int mark) {
  marks.put(studentId,
          new Integer(mark));
```

```
   public double getAverageMarks() {
   if (marks.size() == 0) {
      return Double.NaN;
   } else {
      int sum = 0;
      for (Integer i : marks.values())
      {
         sum += i.intValue();
      }
      return (double) sum /
             (double) marks.size();
   }
}
```

```
ublic class UnitMarks {

rivate HashMap<String, Integer> marks;

ublic UnitMarks() {
  marks = new HashMap<String,
                     Integer>();


ublic void setMark(String studentId,
                   int mark) {
  marks.put(studentId,
            new Integer(mark));
```

```
   public double getAverageMarks() {
   if (marks.size() == 0) {
      return Double.NaN;
   } else {
      int sum = 0;
      for (Integer i : marks.values())
{
         sum += i.intValue();
      }
      return (double) sum /
             (double) marks.size();
   }
}
```

```
ublic class UnitMarks {

rivate HashMap<String, Integer> marks;

ublic UnitMarks() {
  marks = new HashMap<String,
                      Integer>();


ublic void setMark(String studentId,
                   int mark) {
  marks.put(studentId,
            new Integer(mark));
```

```
   public double getAverageMarks() {
   if (marks.size() == 0) {
      return Double.NaN;
   } else {
      int sum = 0;
      for (Integer i : marks.values())
{
         sum += i.intValue();
      }
      return (double) sum /
             (double) marks.size();
   }
}
```

So when do we use models?

- When the system (or component) is big
- When the decisions are complex
- When we need to reason about big complex things.
- When we need to communicate
- When we need to record

What is a model?

- Working definition:
  - A representation of some aspect of the system we wish to model.
  - Depicts that aspect in an easier to work with way than the “real” system.

Pseudocode

```
sList = {empty list}  ## sList is the list of students we've already seen
for(each Unit u in system):
    for( each student s enrolled in u):
          if ( s is not already in sList):
              add s to sList
for (each s in sList):
    print s's details.
```

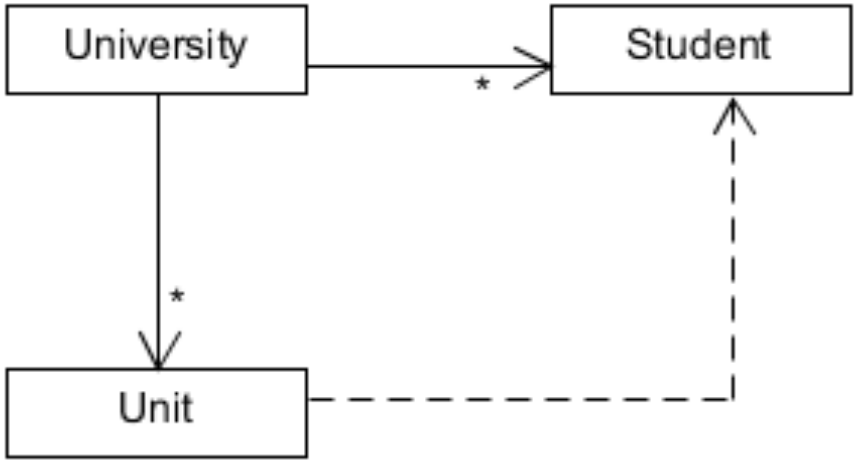
Pseudocode

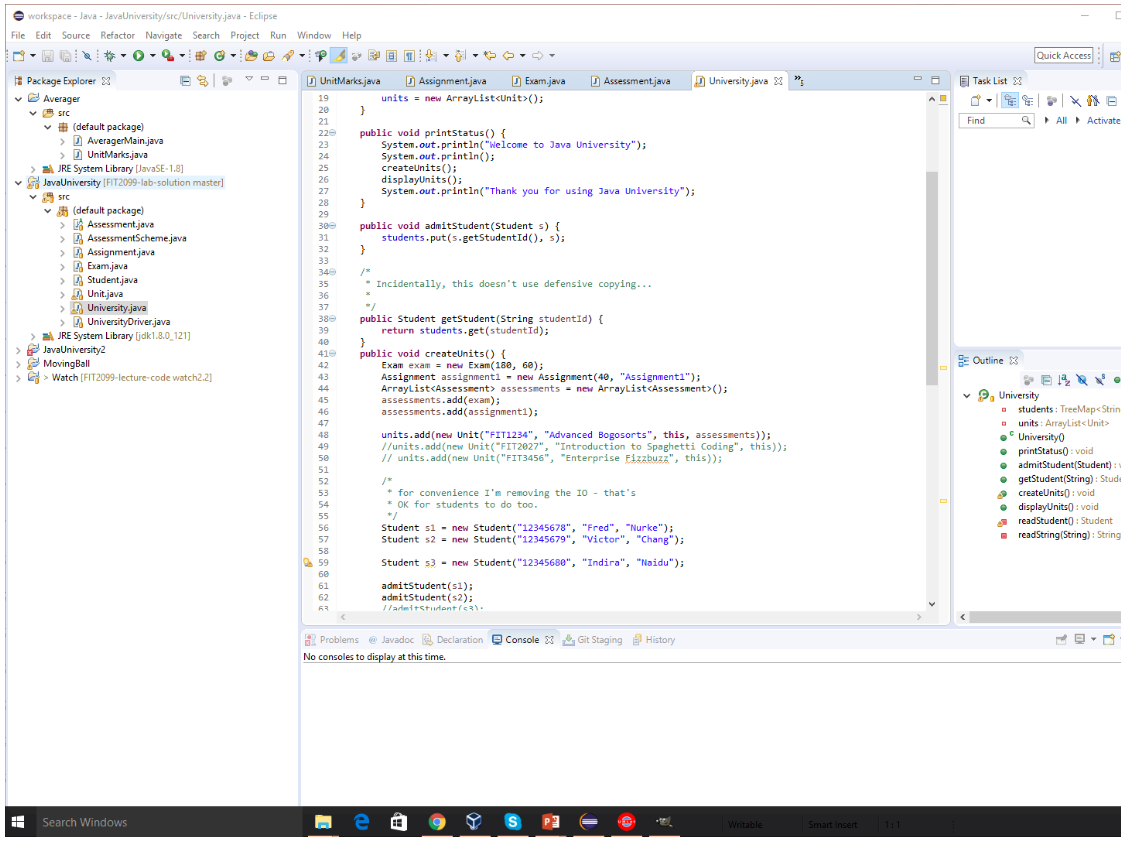
```
sList = {empty list}  ## sList is the list of students we've already seen
for(each Unit u in system):
    for( each student s enrolled in u):
        if ( s is not already in sList):
            add s to sList
sort sList by student ID
for (each s in sList):
    print s's details.
```

Pseudocode

```
 sList = {empty ordered set of students ordered by StudentID}
## sList is the set of students we’ve already seen
 for(each Unit u in system):
    for( each student s enrolled in u):
         if ( s is not already in sList):
            add s to sList
 for (each s in sList):
    print s’s details.
```

What is easier to understand?

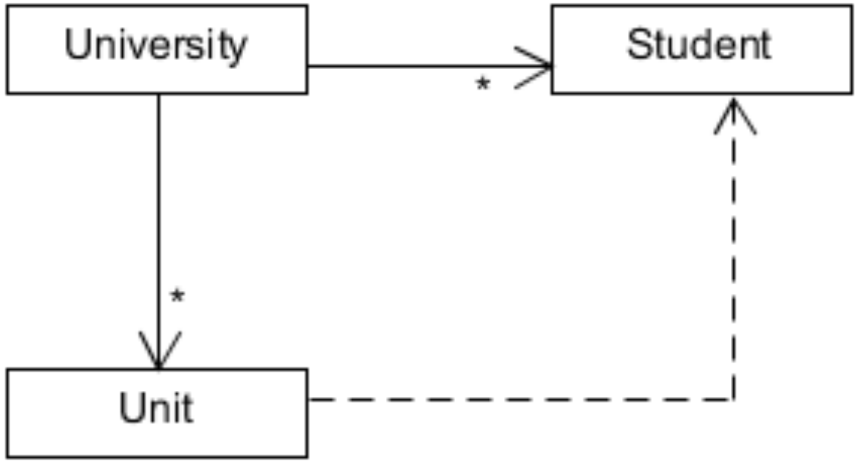


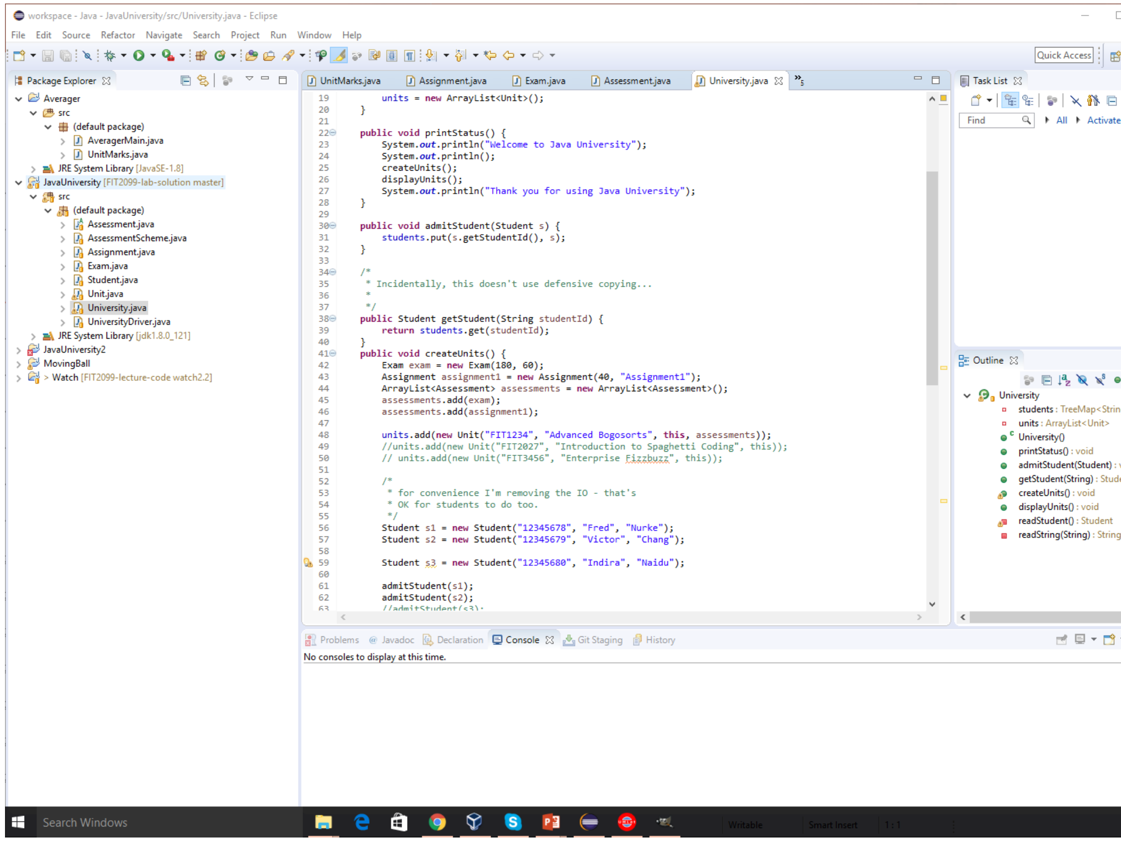


```java
        units = new ArrayList<Unit>();
    }

    public void printStatus() {
        System.out.println("Welcome to Java University");
        System.out.println();
        createUnits();
        displayUnits();
        System.out.println("Thank you for using Java University");
    }

    public void admitStudent(Student s) {
        students.put(s.getStudentId(), s);
    }

    /*
     * Incidentally, this doesn't use defensive copying...
     *
     */
    public Student getStudent(String studentId) {
        return students.get(studentId);
    }
    public void createUnits() {
        Exam exam = new Exam(180, 60);
        Assignment assignment1 = new Assignment(40, "Assignment1");
        ArrayList<Assessment> assessments = new ArrayList<Assessment>();
        assessments.add(exam);
        assessments.add(assignment1);

        units.add(new Unit("FIT1234", "Advanced Bogosorts", this, assessments));
        //units.add(new Unit("FIT2027", "Introduction to Spaghetti Coding", this));
        // units.add(new Unit("FIT3456", "Enterprise Fizzbuzz", this));

        /*
         * for convenience I'm removing the IO - that's
         * OK for students to do too.
         */
        Student s1 = new Student("12345678", "Fred", "Nurke");
        Student s2 = new Student("12345679", "Victor", "Chang");

        Student s3 = new Student("12345680", "Indira", "Naidu");

        admitStudent(s1);
        admitStudent(s2);
        //admitStudent(s3);
```

No consoles to display at this time.

What is quicker to create?





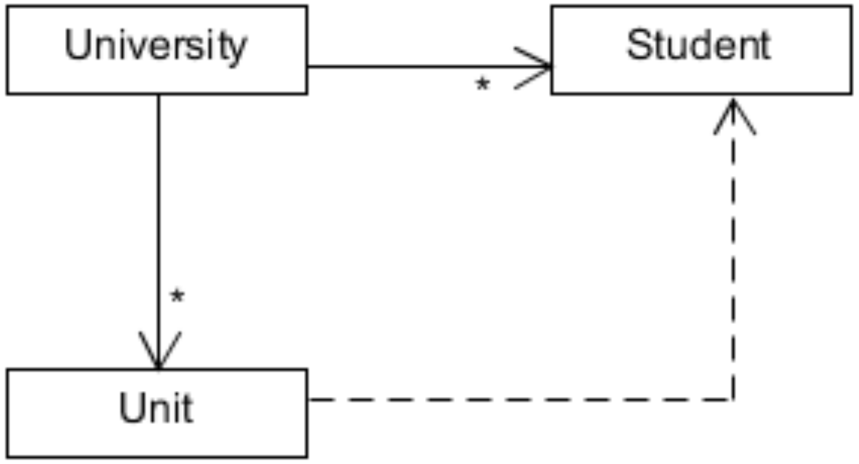
```
Averager
  src
    (default package)
      AveragerMain.java
      UnitMarks.java
  JRE System Library [JavaSE-1.8]
JavaUniversity [FIT2099-lab-solution master]
  src
    (default package)
      Assessment.java
      AssessmentScheme.java
      Assignment.java
      Exam.java
      Student.java
      Unit.java
      University.java
      UniversityDriver.java
  JRE System Library [jdk1.8.0_121]
JavaUniversity2
MovingBall
> Watch [FIT2099-lecture-code watch2.2]
```

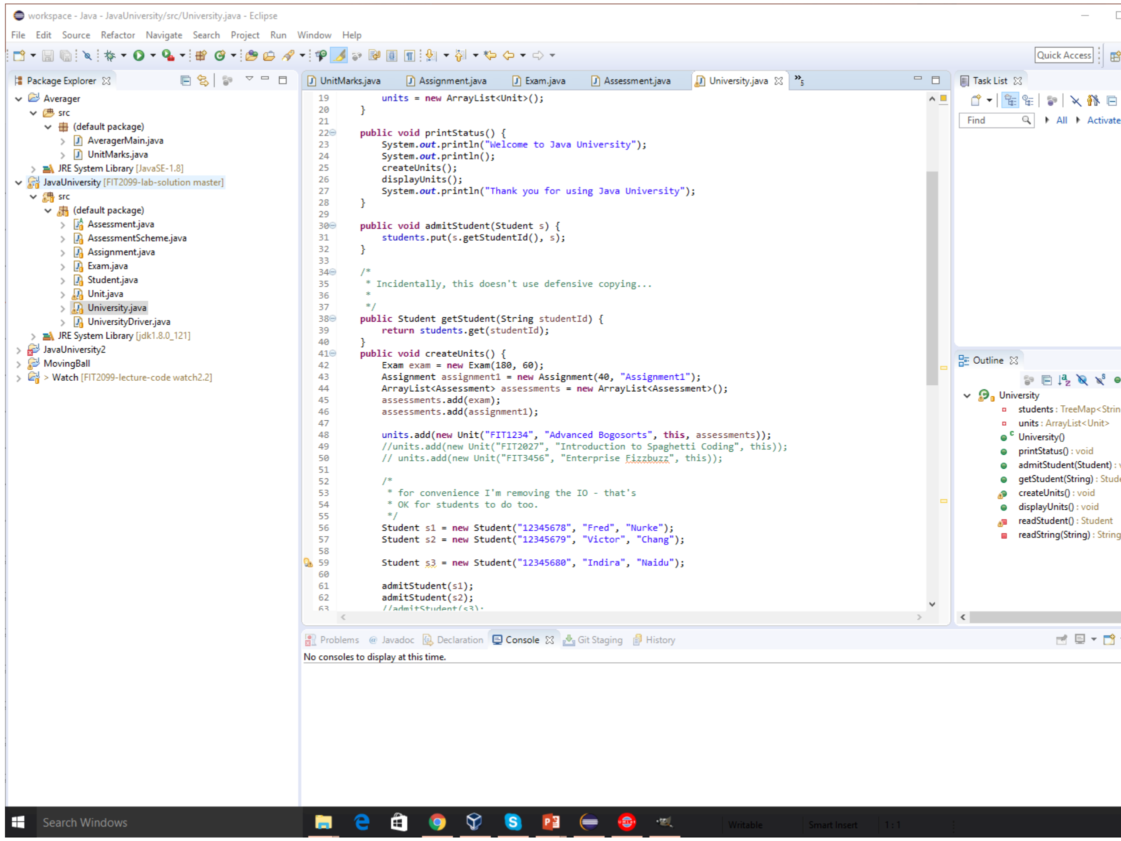
```java
        units = new ArrayList<Unit>();
    }

    public void printStatus() {
        System.out.println("Welcome to Java University");
        System.out.println();
        createUnits();
        displayUnits();
        System.out.println("Thank you for using Java University");
    }

    public void admitStudent(Student s) {
        students.put(s.getStudentId(), s);
    }

    /*
     * Incidentally, this doesn't use defensive copying...
     *
     */
    public Student getStudent(String studentId) {
        return students.get(studentId);
    }
    public void createUnits() {
        Exam exam = new Exam(180, 60);
        Assignment assignment1 = new Assignment(40, "Assignment1");
        ArrayList<Assessment> assessments = new ArrayList<Assessment>();
        assessments.add(exam);
        assessments.add(assignment1);

        units.add(new Unit("FIT1234", "Advanced Bogosorts", this, assessments));
        //units.add(new Unit("FIT2027", "Introduction to Spaghetti Coding", this));
        // units.add(new Unit("FIT3456", "Enterprise Fizzbuzz", this));

        /*
         * for convenience I'm removing the IO - that's
         * OK for students to do too.
         */
        Student s1 = new Student("12345678", "Fred", "Nurke");
        Student s2 = new Student("12345679", "Victor", "Chang");

        Student s3 = new Student("12345680", "Indira", "Naidu");

        admitStudent(s1);
        admitStudent(s2);
        //admitStudent(s3);
```

```
University
  students : TreeMap<Strin
  units : ArrayList<Unit>
  University()
  printStatus() : void
  admitStudent(Student) :
  getStudent(String) : Stude
  createUnits() : void
  displayUnits() : void
  readStudent() : Student
  readString(String) : String
```

No consoles to display at this time.

What is quicker to change?





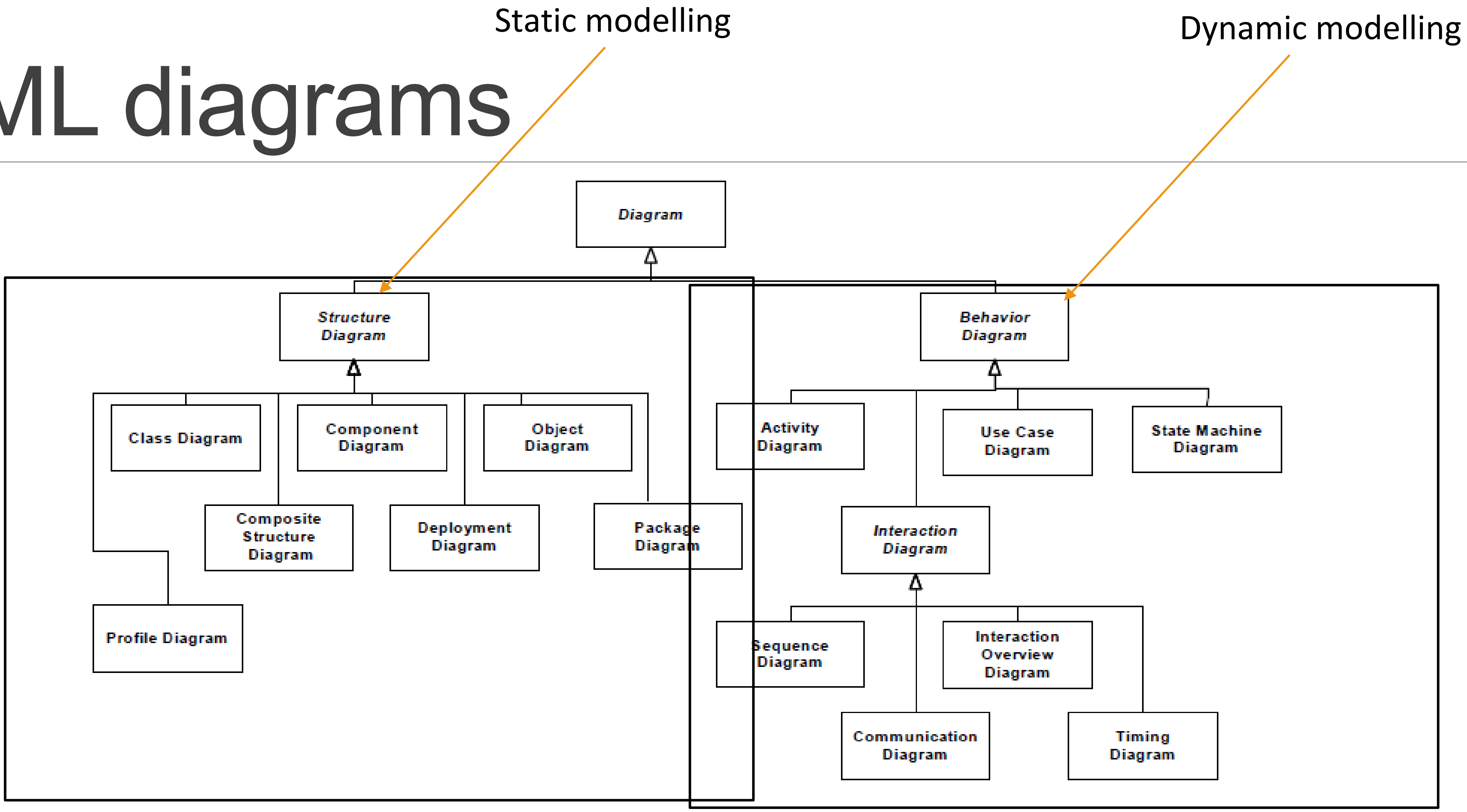
```java
        units = new ArrayList<Unit>();
    }

    public void printStatus() {
        System.out.println("Welcome to Java University");
        System.out.println();
        createUnits();
        displayUnits();
        System.out.println("Thank you for using Java University");
    }

    public void admitStudent(Student s) {
        students.put(s.getStudentId(), s);
    }

    /*
     * Incidentally, this doesn't use defensive copying...
     *
     */
    public Student getStudent(String studentId) {
        return students.get(studentId);
    }
    public void createUnits() {
        Exam exam = new Exam(180, 60);
        Assignment assignment1 = new Assignment(40, "Assignment1");
        ArrayList<Assessment> assessments = new ArrayList<Assessment>();
        assessments.add(exam);
        assessments.add(assignment1);

        units.add(new Unit("FIT1234", "Advanced Bogosorts", this, assessments));
        //units.add(new Unit("FIT2027", "Introduction to Spaghetti Coding", this));
        // units.add(new Unit("FIT3456", "Enterprise Fizzbuzz", this));

        /*
         * for convenience I'm removing the IO - that's
         * OK for students to do too.
         */
        Student s1 = new Student("12345678", "Fred", "Nurke");
        Student s2 = new Student("12345679", "Victor", "Chang");

        Student s3 = new Student("12345680", "Indira", "Naidu");

        admitStudent(s1);
        admitStudent(s2);
        //admitStudent(s3);
```

No consoles to display at this time.

What can we model?

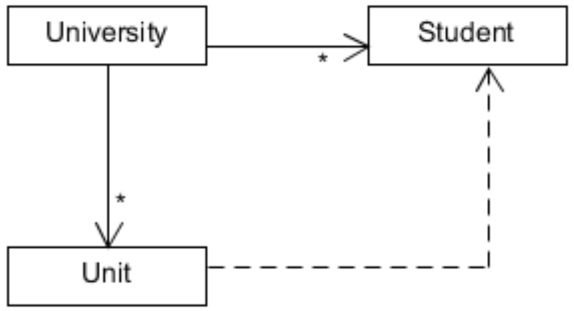
- Structure of system (static)
- Behaviour of system (dynamic)
- Use both, with feedback between each.

UML diagrams

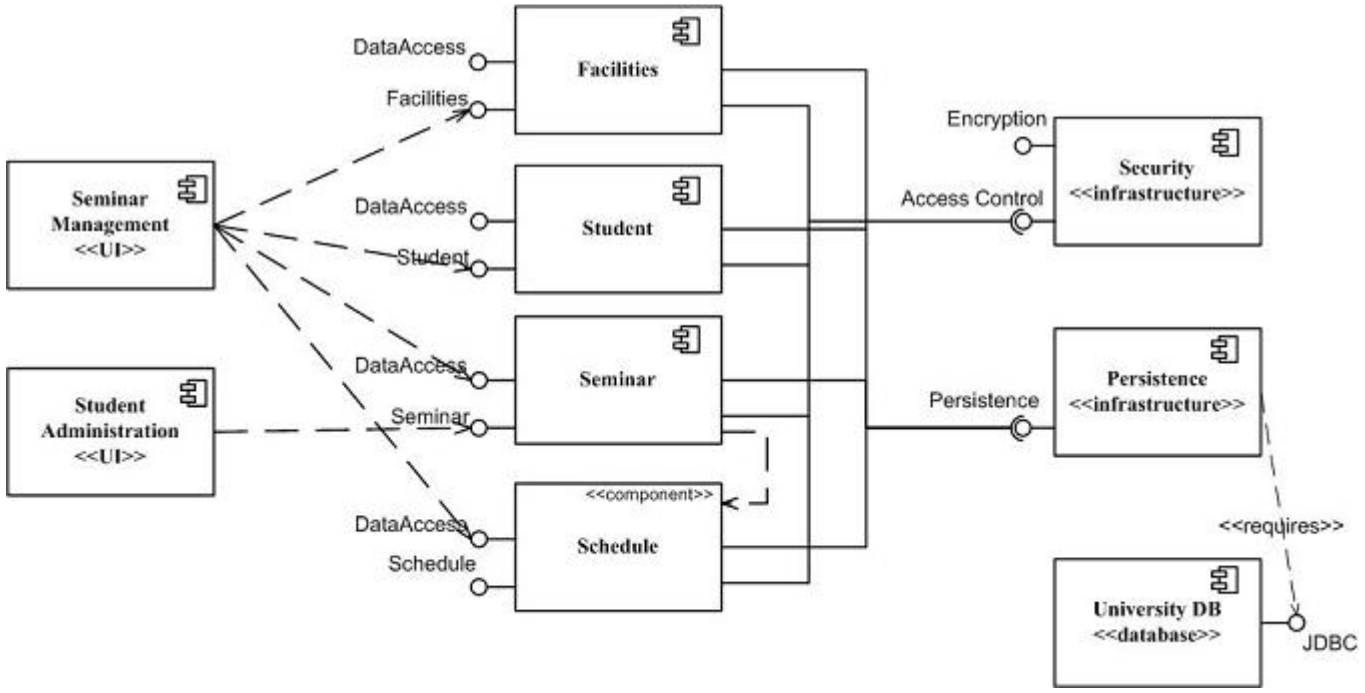


Source: UML 2.5.1 specification
http://www.omg.org/spec/UML/2.5/PDF

A static model

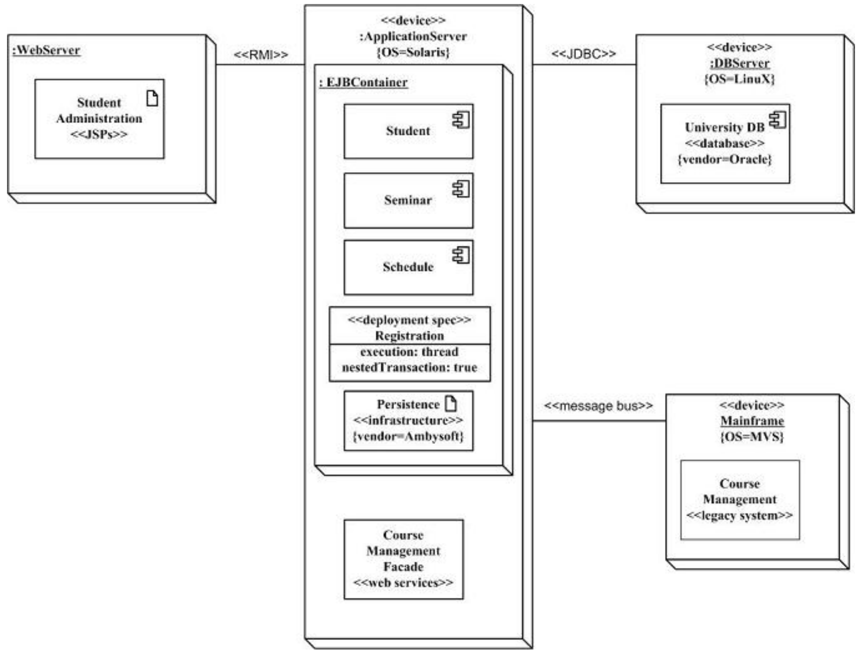


Static models



UML Component Diagram
Source: http://www.agilemodeling.com/artifacts/componentDiagram.htm

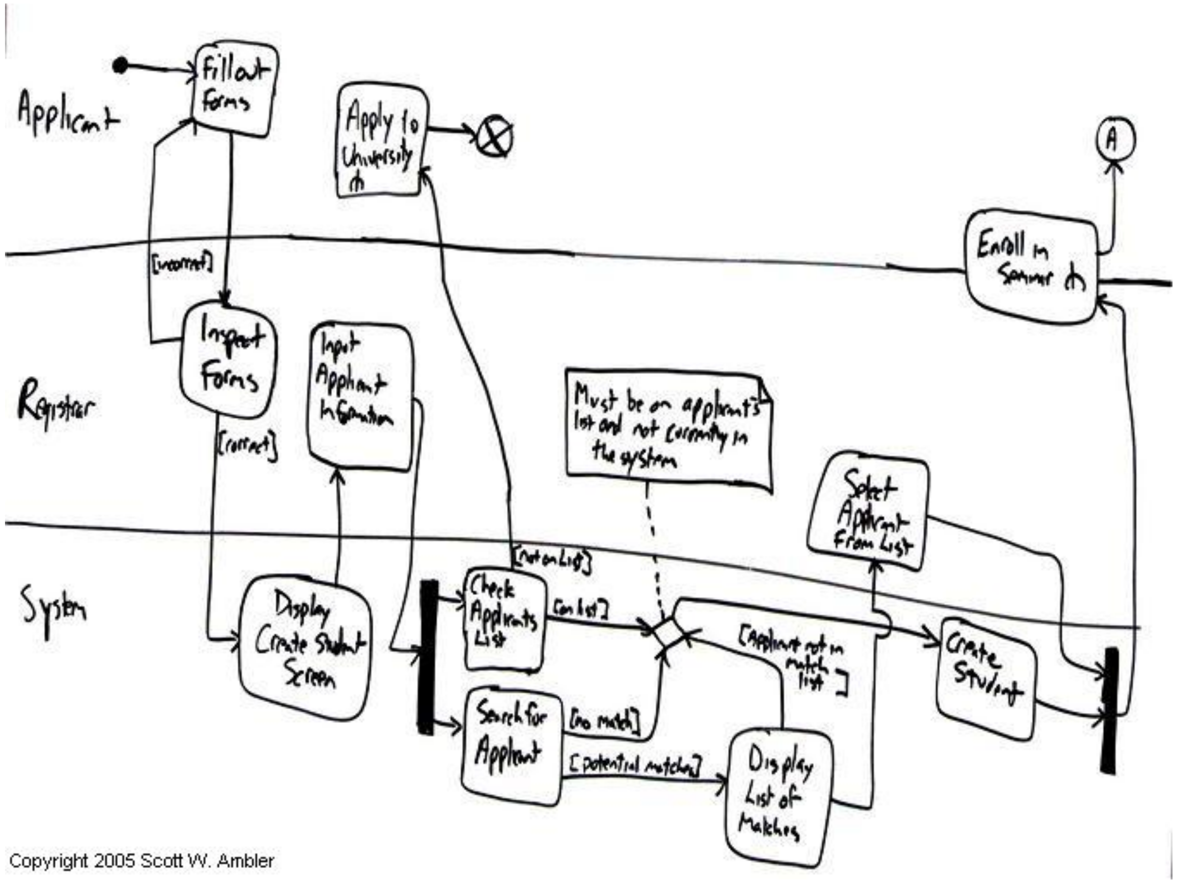
More static models


:WebServer
Student Administration <<JSPs>>
<<RMI>>
<<device>> :ApplicationServer {OS=Solaris}
: EJBContainer
Student
Seminar
Schedule
<<deployment spec>> Registration
execution: thread
nestedTransaction: true
Persistence <<infrastructure>> {vendor=Ambysoft}
Course Management Facade <<web services>>
<<JDBC>>
<<device>> :DBServer {OS=LinuX}
University DB <<database>> {vendor=Oracle}
<<message bus>>
<<device>> Mainframe {OS=MVS}
Course Management <<legacy system>>

UML Deployment Diagram

Source: http://agilemodeling.com/artifacts/deploymentDiagram.htm

Dynamic models I



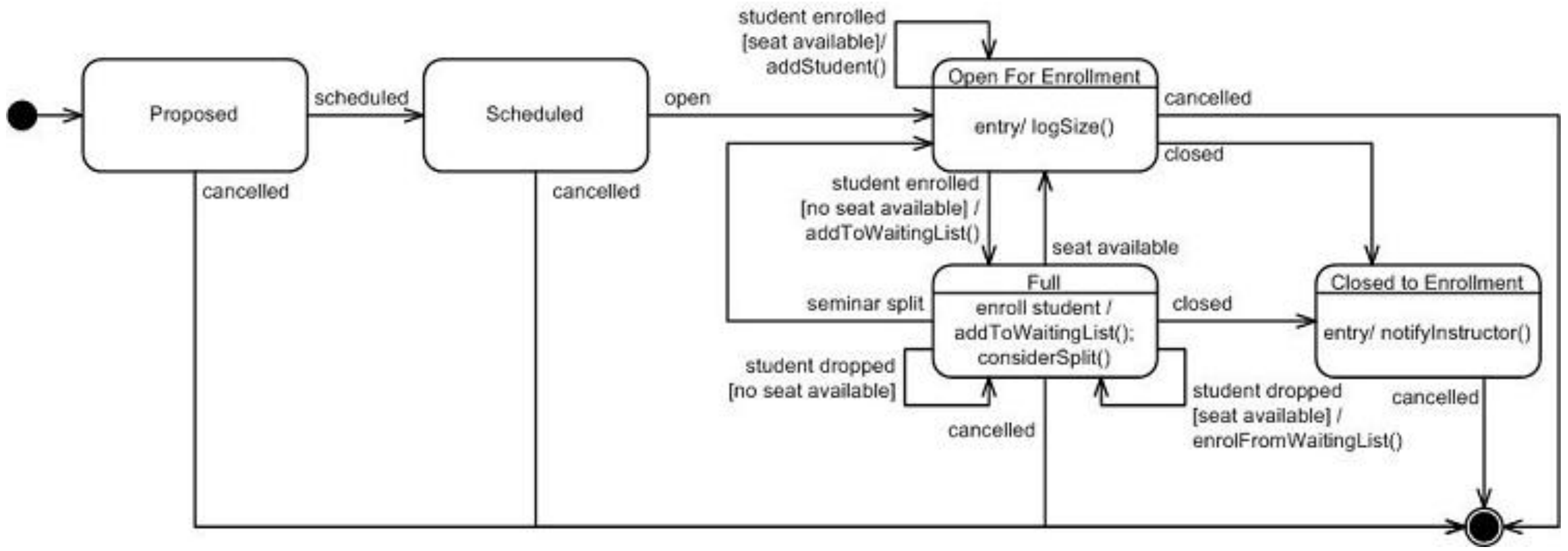
UML Activity Diagram

Source: http://www.agilemodeling.com/artifacts/activityDiagram.htm

Dynamic models II

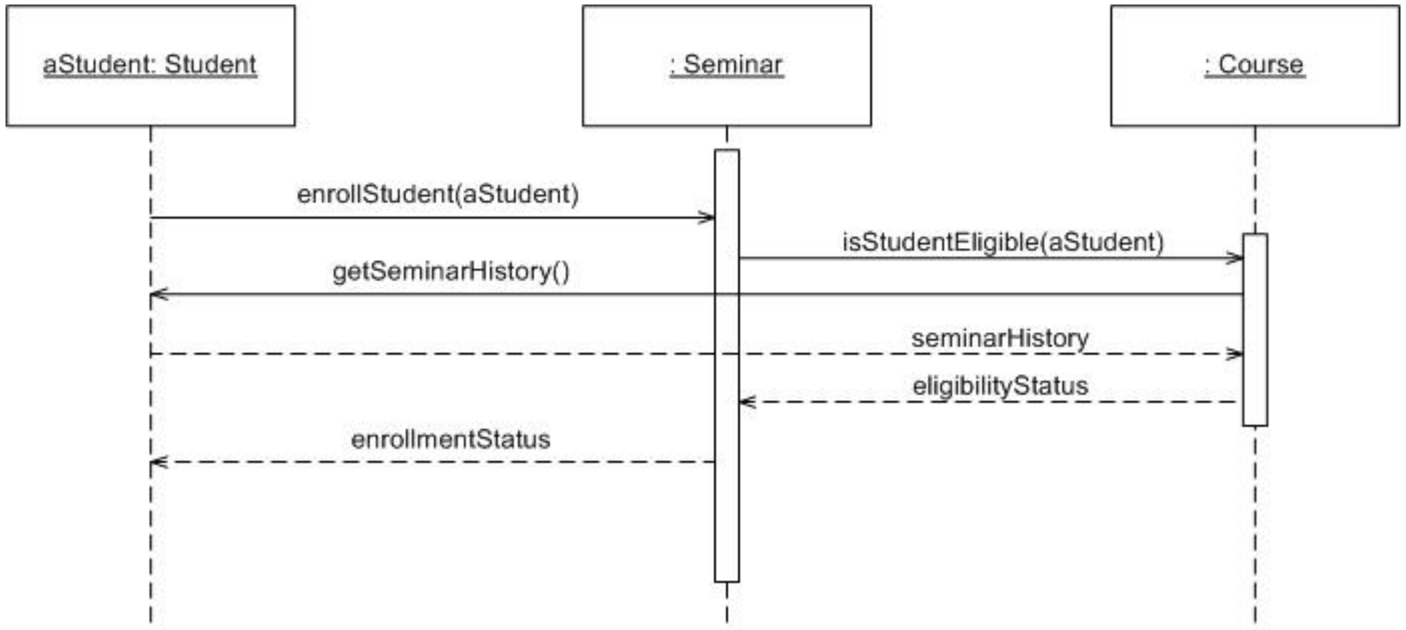
UML State Machine Diagram

Source : http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm



Proposed
scheduled
Scheduled
open
cancelled
cancelled
student enrolled [seat available]/ addStudent()
Open For Enrollment
entry/ logSize()
cancelled
closed
student enrolled [no seat available] / addToWaitingList()
seat available
seminar split
Full
enroll student / addToWaitingList(); considerSplit()
closed
Closed to Enrollment
entry/ notifyInstructor()
student dropped [no seat available]
cancelled
student dropped [seat available] / enrolFromWaitingList()
cancelled

Dynamic models III



UML Sequence Diagram

Source : http://agilemodeling.com/artifacts/sequenceDiagram.htm

Summary

- Design – making good decisions about a software system.
- Modeling – simplified representation of some aspect of the system.
  - To reason
  - To record
  - To communicate.
- Models can be in any notation that helps this.
  - Pseudocode is modeling!
- UML provides a *family* of graphical model types for capturing different things about systems:
  - Structure (static)
  - Behaviour (dynamic)

Where are we going with this?

- A smorgasboard of notations.
  - What do we do with them?
  - When do we use them?
  - How do they relate
    - to each other?
    - to other aspects of the software development process?