Software Engineering

Lecture 6

(Software Architectural Pattern)

SW Architectural Design

High Level Design

- Software Components (e.g. classes / packages / namespaces)
- Where to put each software component
- How they interact to solve the business requirements
- Roles and Responsibilities of each software component

What is Software Architecture?

- High level description of the overall system:
  - The top-level structure of subsystems.
  - The role and interaction of these subsystems.
- Grouping of classes to:
  - Improve *Cohesion.*
  - Reduce *Coupling.*
- Cohesion:
  - Set of “things” that work well together.
- Coupling:
  - Inter-Dependency between two entities.

Cohesion (= Intra-dependency)

inside

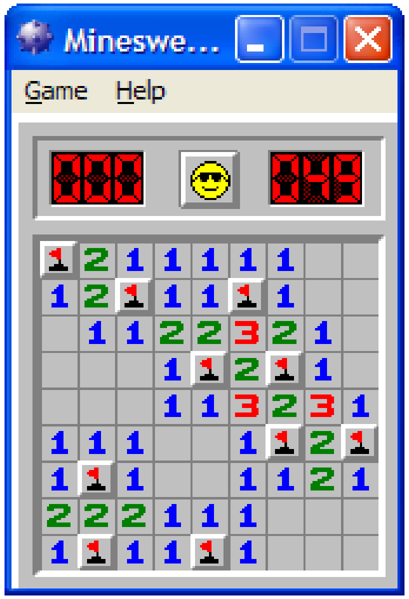
- Intra-dependencies of the components in a software unit (e.g. class, method, module)
- Want Strong cohesion
  - Meaning that “separating” these components apart to different units will cause issues
- Weak cohesion
  - Those components can be easily separated into different units without causing problems
- Refactoring (weak cohesion → stronger cohesion)

Coupling (= Inter-dependency)

- Inter-dependencies of different software units (e.g. class, method, module)
- Want Loose coupling
  - Meaning that the units do not depend on others very much
  - So replacing one unit with "a compatible one" will not cause issues
- Strong coupling
  - Those units "depend" on each other so much that replacing one with "a compatible one" will cause troubles due to some dependencies
- Refactoring (strong coupling → loose coupling)

Software Architecture: Example

- Take the *minesweeper* game as an example, and identify the high level components:



- Display:
  - Showing the game graphically.
- Application Logic:
  - Determining the flags, numbers.
  - Checking whether there are more mines, etc.
- Record Storage:
  - Storing high scores, setting, etc.

Minesweeper: System One

```
public void ClickOnSquare( ){
 if (square == bomb) {
gameState = dead
show dead icon
write high score to file
}
else if (square == number){
open neighboring squares  mark
 squares as opened
display new board
}
}
………..// some other code
}
```

- Bad Cohesion:
  - Display functions all over the place.
  - Application logic buried under other operations.
  - Storage function not clearly separated.
- Low Coupling:
  - Since there is only one class, there is no inter-dependency!

Minesweeper: System Two

```
class MSGUI {  Minesweeper msApp;
 MSStorage msStore;
public void mouseClickOnSquare( ){
 msApp.openSquare(..);
board =  msApp.getCurrentBoard(..);
 show(board);
}
public void menuExitClick( ) {  score
 = msApp.getHighScore( );
 msStore.writeHighScore(score);
}
}
```

```
 class Minesweeper {

 MSStorage msStore;

 public void openSquare(position){
if (square == bomb)
gameState = dead;
 else if (square == number){
                    open     neighboring
squares  mark squares as opened;
 }
 }

 public Board getCurrentBoard() {
..

 }
 public void saveCurrentBoard() {
         msStore.writeBoard(..);

  }
 }
```

```
 class MSStorage {
    public void writeHighScore(..){ }
 public void writeBoard(..)  { }
 }
```

Minesweeper: System Two

- Cohesion:
  - Each class groups logically similar functionality together:
    - `Class MSGUI`: user interface, input/output to screen;
    - `Class Minesweeper`: computation and logic;
    - `Class MSStorage`: file operations.
- Coupling:
  - Some interdependencies. Can be improved.
  - Observe that `MSGUI` uses `MSStorage` directly.
  - Hence, if we substitute another user interface, the high score saving functionality needs to be recorded.

Minesweeper: System Three

```
class MSGUI {
 Minesweeper msApp;
 // ~~MSStorage msStore;~~     Not needed

 .. .. ..


  public void menuExitClick( ) {
   msApp.closingDown( );
  }
}
```

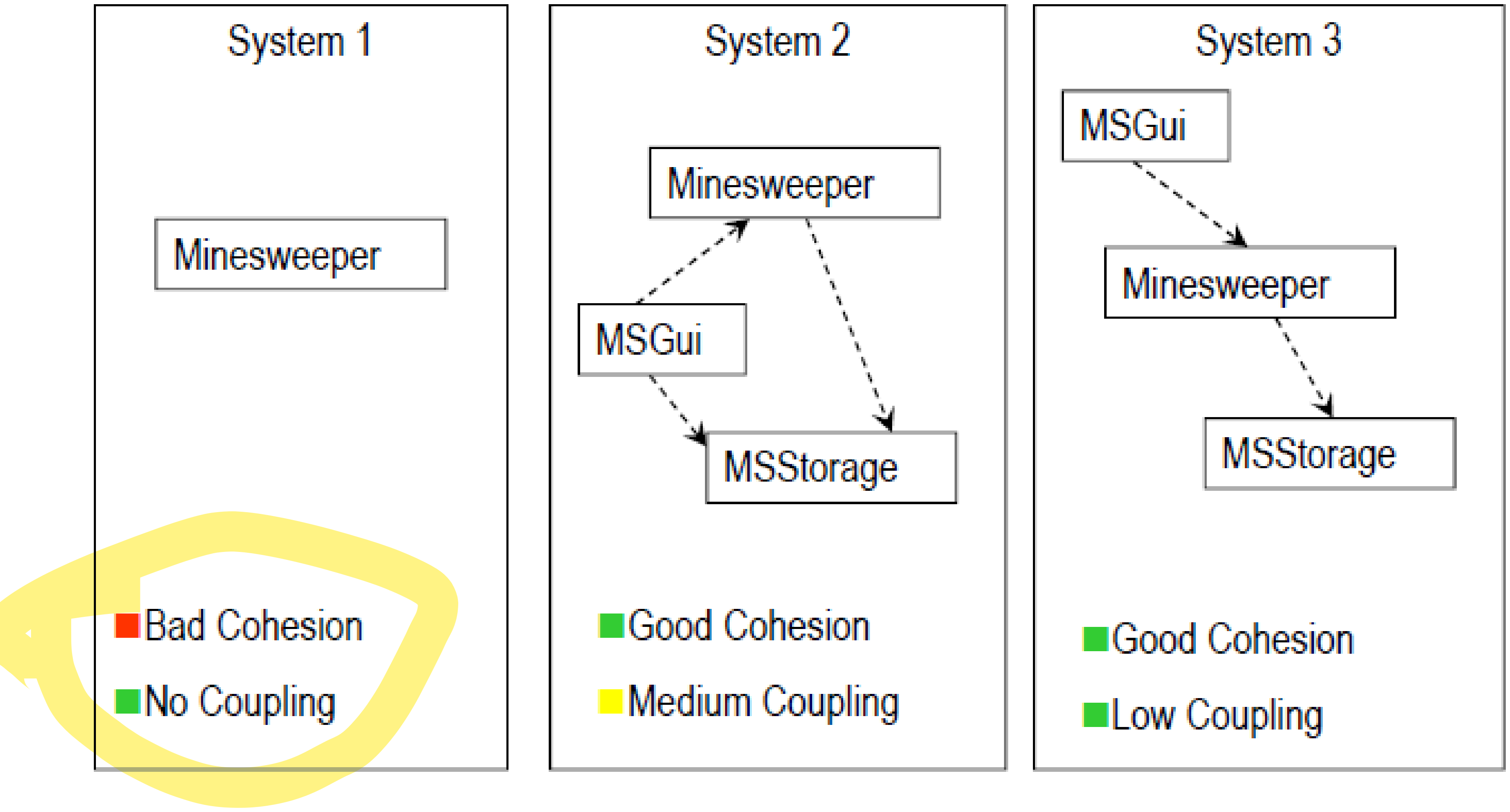
```
class Minesweeper {

 MSStorage msStore;

 .. .. ..

  public void closingDown() {
    msStore.writehighScore(..)
  }

  public void saveCurrentBoard() {
  }
}
```

```
class MSStorage {
 public void writeHighScore(..){ }
 public void writeBoard(..){ }
}
```

Minesweeper: System Three

- Coupling:
  - Reduced. `MSGUI` depends on `Minesweeper` only. `Minesweeper` depends on `MSStorage` only.
- Low coupling enables easy maintenance, e.g.:
  - Changing `MSGUI` to `MSTextUI` would not affect the main application at all.
  - Can swap in another storage class, e.g., database storage, by providing the same methods.

Minesweeper: Systems Comparison



Minesweeper: Observations

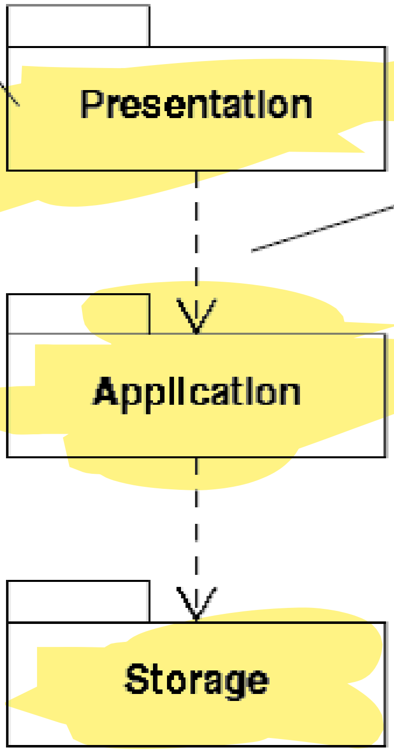
- Trade off between cohesion and coupling:
  - Improving cohesion usually implies worse (higher) coupling and vice versa.
- The three categories of functionality are quite widely applicable:
  - User Interface.
  - Main Application Logic.
  - Storage (Persistency).
- These observations help shaping Software Architecture:
  - splitting a system into sub-systems.

The Layered Architecture

- One of the oldest idea in Software Engineering.
- Split into three separate layers:
  - *Presentation Layer*
    - User Interface.
  - *Application Layer*
    - The underlying logic.
    - Implements the functionality of system.
  - *Storage Layer*
    - Deals with data storage: files, database, etc.
- The layers are higher level abstraction:
  - Each may contain several classes, or several packages (group of classes).

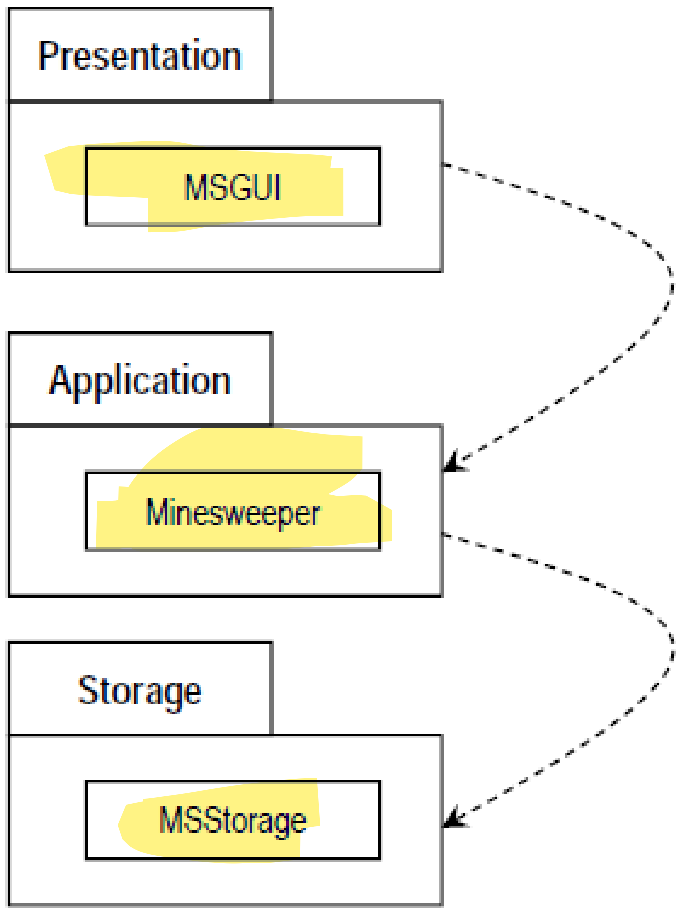
UML Package Diagram

Package:
Collection of classes or sub-packages.



Dependency:
Represent the "*make use of*" relationship .

Minesweeper: Package Diagram



- Corresponds to programming construct:
  - Java: Package.
  - C++: namespace.

Design Pattern

- Well known patterns
  - **Model View Controller (MVC)**
  - **Facade pattern**

Model-View-Controller

- **Model** – data model
- **View** – presentation of the model
- **Controller** – controls the flow / interactions of the view and model

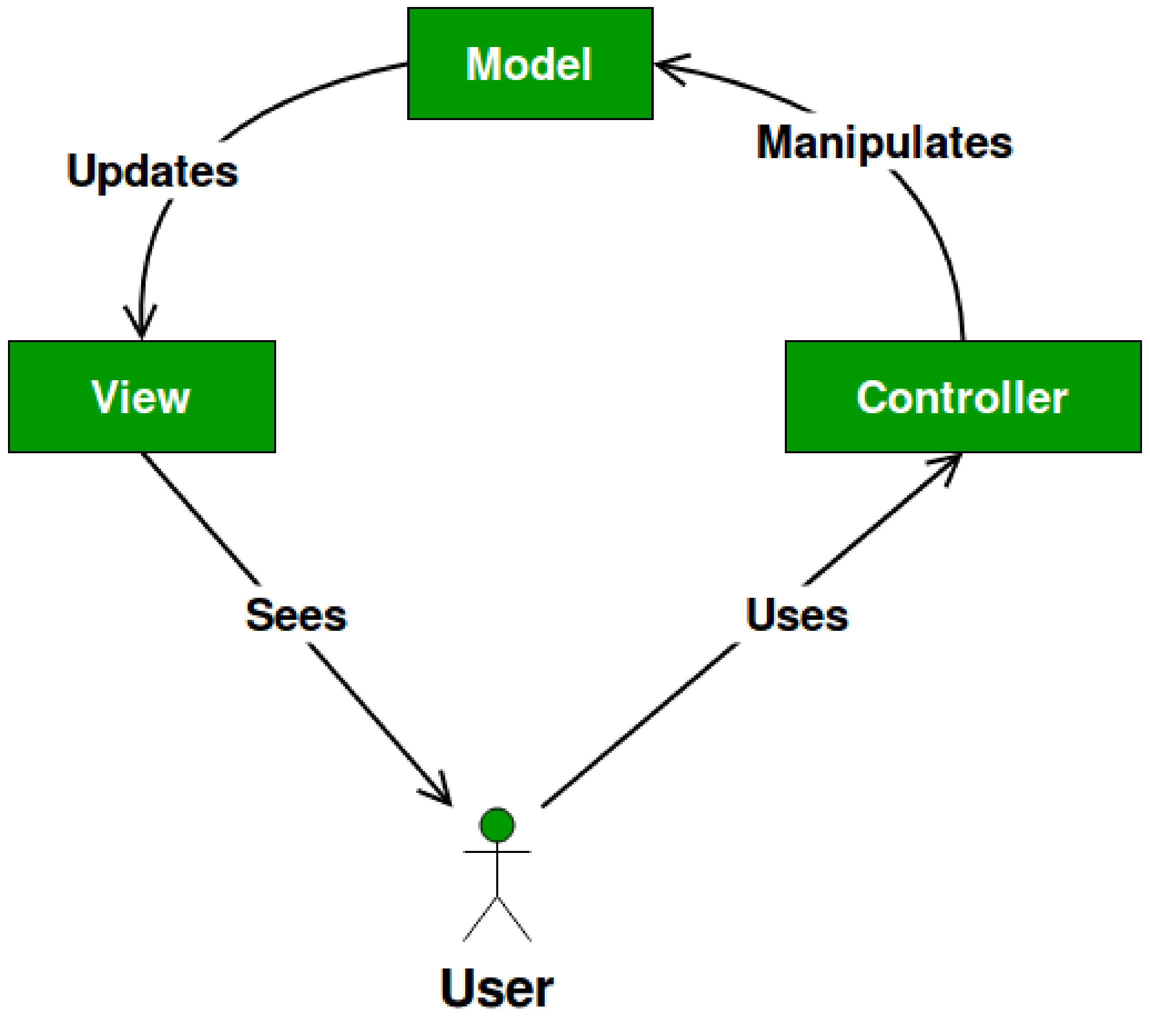
Model-View-Controller

- The model-view-controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information.
- The pattern requires that each of these be separated into different objects.

Model-View-Controller

- The ***model*** (for example, the data information) contains only the pure application data; it contains no logic describing how to present the data to a user.
- The ***view*** (for example, the presentation information) presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.
- Finally, the ***controller*** (for example, the control information) exists between the view and the model. It listens to events triggered by the view and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model.

Model-View-Controller



Model-View-Controller

```
View= You
Waiter= Controller
Cook= Model
Refrigerator= Data
```

MVC – Example – Balance Transfer

- Model: Bank_Account
- Views: Presentation Form and Result
  - Form to collect the required information
  - Responses with respect to the transfer
- Controller:
  - Check the business logic
    - Enough balance for transfer; transfer amount within daily limit; …
    - Both accounts exist and active
    - …
  - Control the process flow according to the “requirements”

Layered Architecture: Advantages

- Layers aim to insulate a system from the effects of change.
- For example, user interfaces often change:
  - but the application layer does not use the presentation layer.
  - so changes to system should be restricted to presentation layer classes.
- Similarly, details of persistent data storage are separated from the application logic.

Software is interesting …..

- Key aspects of software systems (generalized):
  - **80%** of the user tasks will involve '**reading**' or 'using' or 'getting' information FROM the system
  - **20%** of the user task will involve '**writing'**, 'inputting' or 'providing' information INTO the system
  - There are some exceptions (e.g. Word)
- Simply put -
  - Humans want to use computers mostly to 'GET' things out, not 'PUT' things in
- **But** – so far we have ignored all of this and only worried about what is happening inside the system…

Software is unique…

- Software use involves
  - Processing **information** presented on the output device
  - **Interacting** with the system (mostly involves pointing and pressing buttons)
- Two key themes
  - Information
  - Interaction

Two key themes

- Information
  - Users need to be shown Information
  - Information is rendered on the screen
  - Information rendering can be designed thoughtfully
  - Information should be shown carefully and intentionally
- Interaction
  - Inputs need to be captured & processed
  - Feedback needs to be provided to the user

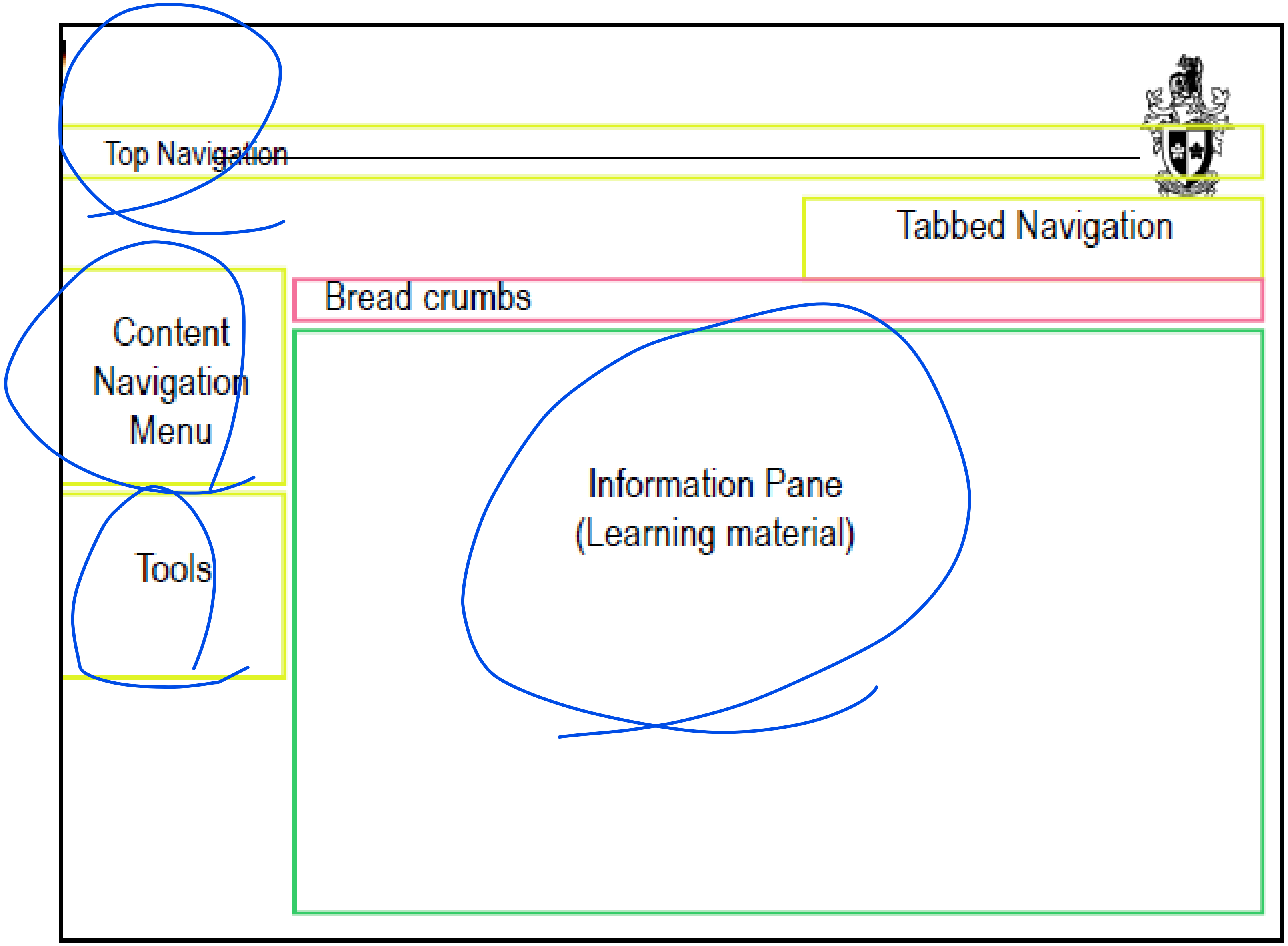
Interface design – Wireframing

- How do we present information to our users?
  - Information presentation techniques
  - Reports
- How do we decide on the structure of our web pages?
  - Wireframing

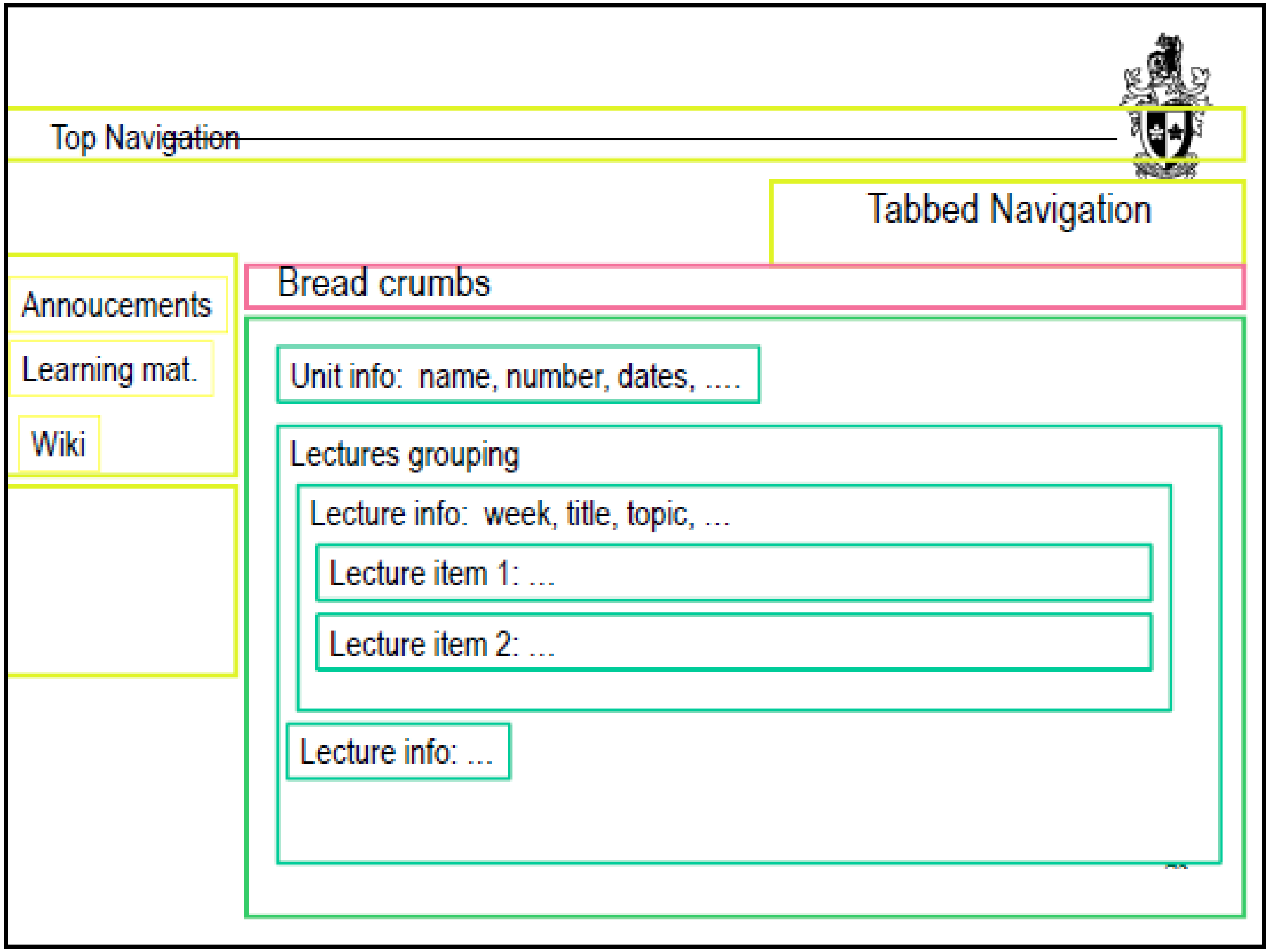
Wireframing

- Basic visual guide used in web design to suggest the structure of a website and relationships between its pages
- A **Wireframe** is a illustration of the layout of fundamental elements in an interface.
- Because of this, wireframes are often completed before any artwork is developed. When completed correctly they will provide a visual reference upon which to structure each page

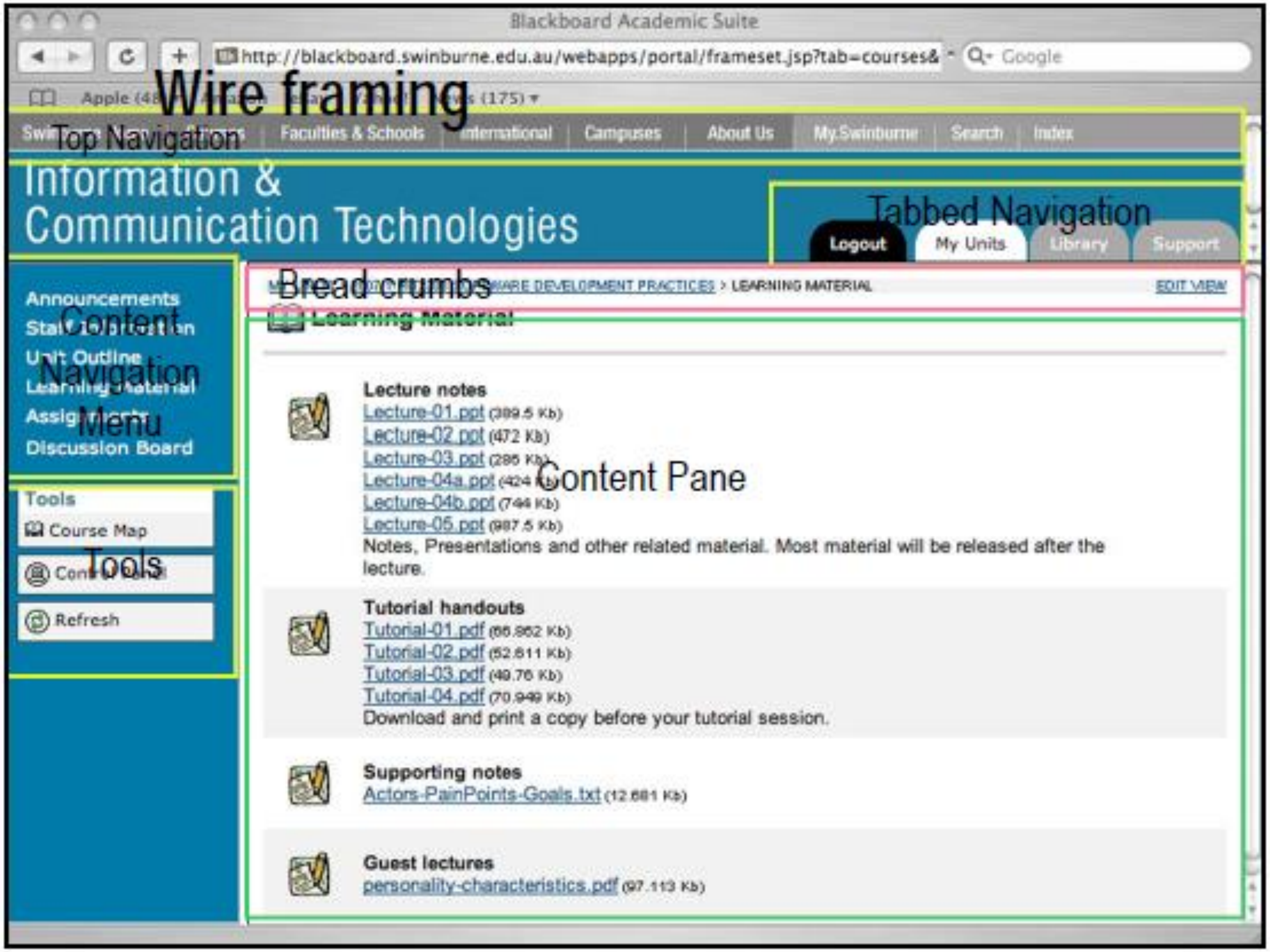
Wire framing - Bare bones



Wire framing – Fleshing out items



Wire framing



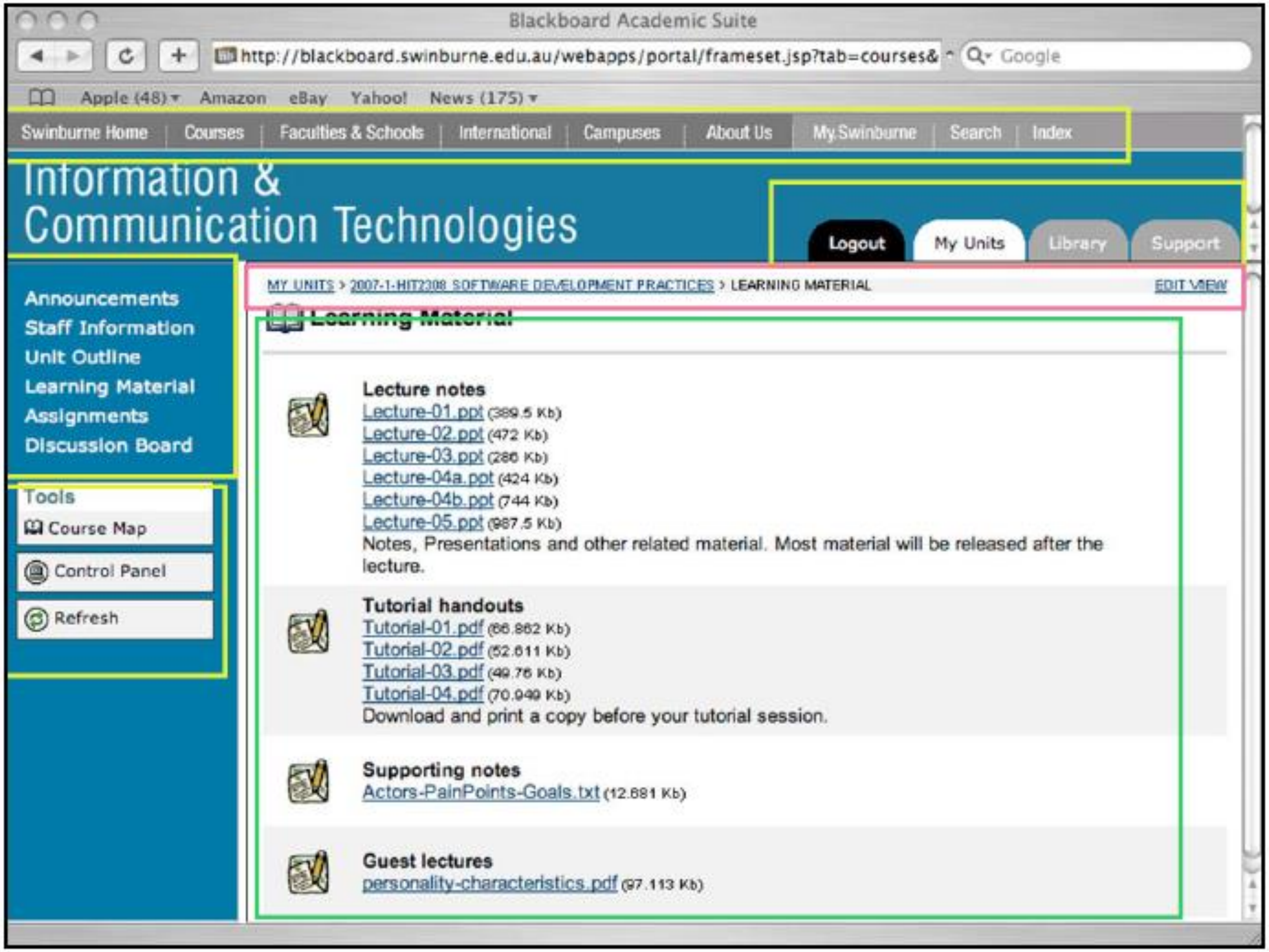
Learning Material

Lecture notes
Lecture-01.ppt (389.5 Kb)
Lecture-02.ppt (472 Kb)
Lecture-03.ppt (286 Kb)
Lecture-04a.ppt (424 Kb)
Lecture-04b.ppt (744 Kb)
Lecture-05.ppt (987.5 Kb)
Notes, Presentations and other related material. Most material will be released after the lecture.

Tutorial handouts
Tutorial-01.pdf (66.952 Kb)
Tutorial-02.pdf (52.611 Kb)
Tutorial-03.pdf (49.76 Kb)
Tutorial-04.pdf (70.949 Kb)
Download and print a copy before your tutorial session.

Supporting notes
Actors-PainPoints-Goals.txt (12.681 Kb)

Guest lectures
personality-characteristics.pdf (97.113 Kb)


Blackboard Academic Suite
http://blackboard.swinburne.edu.au/webapps/portal/frameset.jsp?tab=courses&
Google
Apple (48) Amazon eBay Yahoo! News (175)
Swinburne Home | Courses | Faculties & Schools | International | Campuses | About Us | My.Swinburne | Search | Index
Information & Communication Technologies
Logout
My Units
Library
Support
Announcements
Staff Information
Unit Outline
Learning Material
Assignments
Discussion Board
Tools
Course Map
Control Panel
Refresh

MY UNITS > 2007-1-HIT2308 SOFTWARE DEVELOPMENT PRACTICES > LEARNING MATERIAL EDIT VIEW

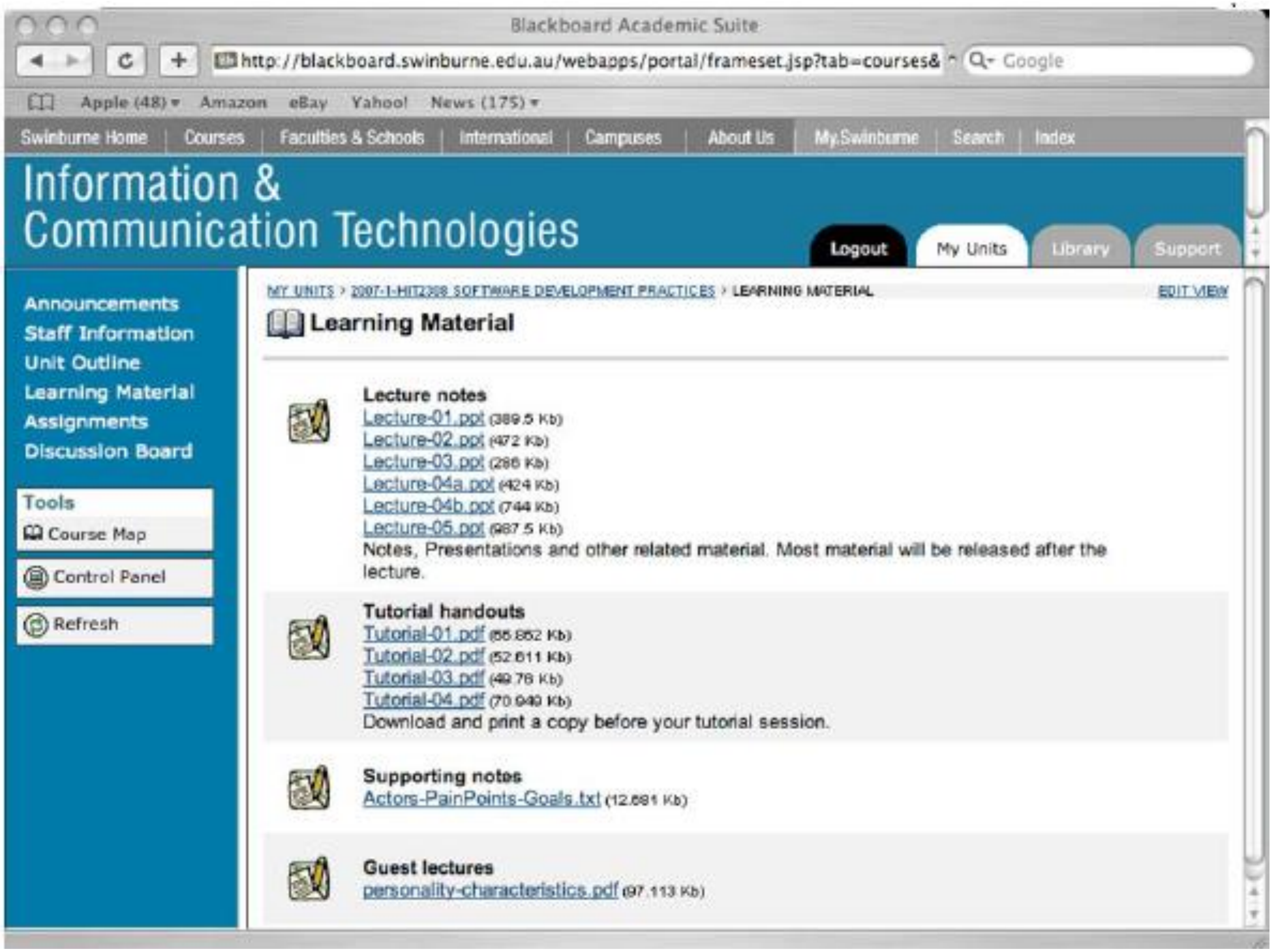
Learning Material

Lecture notes
Lecture-01.ppt (389.5 Kb)
Lecture-02.ppt (472 Kb)
Lecture-03.ppt (286 Kb)
Lecture-04a.ppt (424 Kb)
Lecture-04b.ppt (744 Kb)
Lecture-05.ppt (987.5 Kb)
Notes, Presentations and other related material. Most material will be released after the lecture.

Tutorial handouts
Tutorial-01.pdf (66.862 Kb)
Tutorial-02.pdf (52.611 Kb)
Tutorial-03.pdf (49.76 Kb)
Tutorial-04.pdf (70.949 Kb)
Download and print a copy before your tutorial session.

Supporting notes
Actors-PainPoints-Goals.txt (12.681 Kb)

Guest lectures
personality-characteristics.pdf (97.113 Kb)



Learning Material

Lecture notes
Lecture-01.ppt (389.5 Kb)
Lecture-02.ppt (472 Kb)
Lecture-03.ppt (286 Kb)
Lecture-04a.ppt (424 Kb)
Lecture-04b.ppt (744 Kb)
Lecture-05.ppt (987.5 Kb)
Notes, Presentations and other related material. Most material will be released after the lecture.

Tutorial handouts
Tutorial-01.pdf (66.862 Kb)
Tutorial-02.pdf (52.611 Kb)
Tutorial-03.pdf (49.76 Kb)
Tutorial-04.pdf (70.949 Kb)
Download and print a copy before your tutorial session.

Supporting notes
Actors-PainPoints-Goals.txt (12.681 Kb)

Guest lectures
personality-characteristics.pdf (97.113 Kb)