BUILDING BLOCKS
UML & more....
Main Sections

- Use Case Diagrams
- Sequence Diagrams

Steps:
1. Problem
2. Worth Solving?
3. Solution?
4. Steps to Solve
5. Evaluate
So, what is the problem?

- Software is extremely complex.
  - Once a structure is in place, very difficult to change.
  - Requires teamwork to build.
  - Software usually requires maintenance.
  - Requirements need to be traced.

Should we reduce ‘effective’ complexity?

Structure in place, hard to change.
Teamwork required. Team mates need to communicate.
Maintenance, hence documentation.
Traceability is important to check if the final product delivers on the functional requirements.
Why Reduce ‘Effective Complexity’?

- Software is ubiquitous. Chances are, you will encounter it.

- Will require less work from each team member to get it right the first time.

- Easier documentation and greater maintainability.

How to reduce effective complexity?

1. Definitely computer scientists.
2. If team members share a common vocabulary and can communicate, it’ll be easier for everyone.
3. Reduce risk of failure.
How to reduce ‘Effective Complexity’?

Visualize software

UML designed with the following major goals

<table>
<thead>
<tr>
<th>A Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualize different layers of detail</td>
</tr>
<tr>
<td>Apply to new and legacy systems</td>
</tr>
<tr>
<td>Universal</td>
</tr>
<tr>
<td>Support parallel dev. of large systems</td>
</tr>
</tbody>
</table>

Software construction needs a plan.
The overall scope of the software can quickly and easily be defined at the start of the project with a high level model allowing for accurate estimation. Increasing levels of detail can then be added to each part of the software as it is constructed
Universal + Unified = standard for software modelling languages.

Just like a building
UML (design and represent Building Blocks)

UML - Unified Modelling Language

“The three amigos”

James Rumbaugh (OMT + UML, RUP)

Grady Booch (Booch Method, RUP)

Ivar Jacobson (RUP, EssUP)

OMT – Object modelling technique
UML (design and represent Building Blocks)

UML - Published by the OMG

Source: omg.org

OMG – Object Modelling Group
UML 2.4 – March 2011
UML (contd.)

- **Structure diagram (not our focus!)**
  - Shows the static structure of the system.

- The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts.
- The elements in a behavior diagram represent a series of changes to the system over time.
- Use case diagrams are also known as extensions of class diagrams.
- Use case diagrams are supposed to be behavior and structure diagrams according to UML 2.4
Use case diagrams are used to specify:

- (external) requirements.
- what a system can do;
- how environment should interact with the subject so that the system will be able to perform its services.

**Use Case** - A set of actions

**Subject** - System under analysis to which a set of use cases apply.

**Actor** - external users of a system
Include similar to abstract use case defined in UML 1.xxx, UML 2.4 specifies an ‘include’ relationship, which means “what is left in the base use case is usually not complete”. Extend – open arrowhead directed from the extending use case to the extended (base) use case.

Register use case is meaningful on its own. It could be extended with optional Get Help On Registration use case.

A Plan
- Visualize different layers of detail
- Apply to new and legacy systems
- Universal
- Support parallel dev. of large systems
Sequence Diagrams

Focusses on message interchange between “lifelines”
**Lifeline:** is a named element which represents an individual participant in the interaction.

**Message:** is a named element which defines a specific kind of communication between lifelines.

Message specifies not only the kind of communication, but also the sender and the receiver. Sender and receiver are normally two occurrence specifications (points at the ends of messages).
Message Types: Synchronous Call, Asynchronous Call, Asynchronous Signal, Create, Delete, Reply

Synchronous Call – represents operation call – send message and suspend execution while waiting for response
Asynchronous Call – send message and proceed immediately without waiting for return value.
Asynchronous Signal – message corresponds to asynchronous send signal
Create message is sent to lifeline to create itself
Delete message (called stop in previous versions of UML) is sent to terminate another lifeline (x marks the destruction occurrence).
Sequence Diagrams (Simplified for this course)

Component I

Lifeline

Component I

Lifeline + Messages

Problem ➔ Worth Solving? ➔ Solution? ➔ Steps to Solve ➔ Evaluate
The large gray box is abstracted for now, basically the DOM, XML parser etc. Note that this is the partial sequence diagram when the page is not cached.
The dashed backwards arrow represents a “reply” (check earlier slides). You should use a dashed forward arrow if there is a component that is created (not shown here).