Chapter 4

**An Overview of the UML**

Stakeholders- they're the ones responsible for making vital organizational data accessible within the organization and defensibly deleting it when it's obsolete-for example:

- Analysts

- Designers

- Coders: coder needs to understand the design

of the system and be able to convert the design to a low level code.

- Testers

- QA

- The Customer

- Technical Authors: interested in the behaviour of the system as a whole, and needs to

understand how the product functions

The UML attempts to provide a language so

expressive that all stakeholders can benefit from at least one UML diagram.

**The Use Case Diagram**



The diagram is deliberately simple to understand. This enables both developers

(analysts, designers, coders, tests) *and* the customer to work with the diagram.

**The Class Diagram**

We.ll see that we can use the Class Diagram at the analysis stage as well as design . we.ll use

the Class Diagram syntax to draw a plan of the major concepts our customer understands (and we.ll call this the *Conceptual Model*). Together with Use Cases, a Conceptual Model is a powerful technique in requirements analysis

**Collaboration Diagrams**

As we are developing object-oriented software, anything our software needs to do is

going to be achieved by objects **collaborating**. We can draw a **collaboration**

**diagram** to describe how we want the objects we build to collaborate.

UML notation for the diagram is simple enough but designing effective collaborations.

**Sequence Diagram**

The sequence diagram is, in fact, directly related to the collaboration diagram and

displays the same information, but in a slightly different form. The dotted lines down

the diagram indicate *time*, so what we can see here is a description of how the objects

in our system interact over time.

**State Diagrams**



Some objects can, at any particular time, be in a certain *state*. For example, a traffic

light can be in any one of the following states:

**Off, Red, Amber, Green**

Sometimes, the sequence of transitions between states can be quite complex . in the

above example, we would not want to be able to go from the .Green. state to the

.Red. state (we.d cause accidents!).

**Package Diagrams**



smaller, easier to understand "chunks",and the UML Package Diagram enables us to model this in a simple and effective way.

**Component Diagrams**

it allows us to notate

how our system is split up, and what the dependencies between each module is. However, the Component Diagram emphasises the physical software components

(files, headers, link libraries, executables, packages) rather than the logical

partitioning of the Package Diagram.

**Deployment Diagrams**



The UML provides a model to allow us to plan how our software is going to be

deployed. The diagram above shows a simple PC configuration, for example.

**Summary**

The UML provides many different models of a system. The following is a list of

them, with a one sentence summary of the purpose of the model:

• **Use Cases** - .How will our system interact with the outside world?.

• **Class Diagram** - .What objects do we need? How will they be related?.

• **Collaboration Diagram** - .How will the objects interact?.

• **Sequence Diagram** - .How will the objects interact?.

• **State Diagram** - .What states should our objects be in?.

• **Package Diagram** - .How are we going to modularise our development?.

• **Component Diagram** - .How will our software components be related?.

• **Deployment Diagram** - .How will the software be deployed?.