A UML Introduction Tutorial

In this tutorial you will learn about the fundamentals of object oriented modelling, the Unified Modelling Language and the software development process.

Click to continue.

A UML Introduction Tutorial Copyright © 1997 - 2004 by CRaG Systems All rights reserved

Test SDLC Template Free
Use System Development Templates For Effective Software Management.
Smartsheet.com/DevLifecycleSoftware

Business Process Modeling
Business Process Modeling with UML Training and Consulting services
www.iconatg.com

Object Oriented
Get the Job You Want from Dice.com Post Your Resume Now & Get Noticed.
www.Dice.com

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
## Course Map/Table of Contents

### 1. Fundamentals of Object Oriented Modelling

- 1.1 Abstraction
- 1.2 Modelling
- 1.3 Model Organisation
- 1.4 Structured Analysis
- 1.5 Object Orientation
- 1.6 Encapsulation of Hardware
- 1.7 Encapsulation of Software
- 1.8 Summary and Test

### 2. The Unified Modelling Language - Part 1

- 2.1 UML - What It Is
- 2.2 UML What It Is - 2
- 2.3 How UML Started
- 2.4 UML Diagram Types
- 2.5 UML Diagram Types - 2
- 2.6 Business Modelling and Web Applications and extending UML
- 2.7 Use Case Diagram
- 2.8 Class Diagrams
- 2.9 Summary and Test
- 2.10

### 3. The Unified Modelling Language - Part 2

- 3.1 Sequence Diagram
- 3.2 Collaboration Diagram
- 3.3 Statechart Diagram
- 3.4 Activity Diagram
- 3.5 Component Diagram
- 3.6 Deployment Diagram
- 3.7 Modelling Architecture
- 3.8 Summary and Test

### 4. The Software Development Process

- 4.1 Business Process Modelling
- 4.2 System Requirements Definition
- 4.3 The System Analysis Model
- 4.4 System Model Information Flow
- 4.5 Screen Prototyping
- 4.6 The System Design Model
- 4.7 Overall Process Flow
- 4.8 Incremental Development
- 4.9 Summary and Test
- 4.10 Course Feedback

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
1. Fundamentals of Object Oriented Modelling

This chapter explains the principles behind modelling and abstraction and shows how they can be applied to object oriented models and systems.

1. Abstraction
2. Modelling
3. Model Organisation
4. Structured Analysis
5. Object Orientation
6. Encapsulation of Hardware
7. Encapsulation of Software
8. Summary and Test

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
Abstraction is a fundamental principle of modelling. A system model is created at different levels, starting at the higher levels and adding more levels with more detail as more is understood about the system. When complete, the model can be viewed at several levels. So abstraction is about:

- Looking only at the information that is relevant at the time
- Hiding details so as not to confuse the bigger picture
1.2 Modelling

When we model, we create a number of views of the system being described, usually 2 or 3. For a complete description 3 are normally needed. Each view is an 'orthogonal' view. Each view is needed for a full understanding of the system. Views are orthogonal when:

- Each view has information that is unique to that view
- Each view has information that appears in other views
- The information that is common between views is consistent

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
All model syntaxes provide a number of model elements which can appear on one or more diagram types. The model elements are contained with a central model, together with all their properties and connections to other model elements. The diagrams are independent views of the model just as a number of computer screens looking into different records or parts of a database shows different views.
1.3.1 How a case tool works

In a proper case tool, even though the diagrams are used to create and display the elements in the model, the model element information is not stored in the diagrams, but in a model file, or repository. Model elements can appear on more than one diagram. When the properties of an element are changed in one diagram, the changes are reflected in all other diagrams that use it. This would be difficult if each diagram had its own copy of a model element. The only information contained within the diagram is:

References to the elements on the diagram

Where the elements appear on the diagram

Diagram specific text
1.3.2 Abstraction and Modelling

1. Abstraction is about
   1. Looking at things in detail
   2. Looking at relevant information only
   3. Focusing on one detailed point

2. Hiding details is a good idea because
   1. There is no need to worry about them as they are unnecessary
   2. Details are not important
   3. It stops you worrying about them when they are not important

3. In a model made up of multiple views
   1. Each view is a transformation of another view
   2. Views contain unique information
   3. Common information must be consistent
   4. 2 and 3 are true
   5. None of the above
4 Model elements

1. appear on one diagram only
2. are distributed amongst the diagrams of the model
3. can appear on multiple diagrams of the same or different types
4. can appear on multiple diagrams of different types only
5. 1 and 2 are true
In structured analysis there are three orthogonal views:

- The functional view, made up of data flow diagrams, is the primary view of the system. It defines what is done, the flow of data between things that are done and provides the primary structure of the solution. Changes in functionality result in changes in the software structure.

- The data view, made up of entity relationship diagrams, is a record of what is in the system, or what is outside the system that is being monitored. It is the static structural view.

- The dynamic view, made up of state transition diagrams, defines when things happen and the conditions under which they happen.
1.4.1 Structured Analysis and Design is fine if the system requirements never change

Traditional Structured Analysis and Design methods tend to fall short of the goal of producing software that is easy to maintain and extend over a long period. They are able to produce a hierarchically structured model of the both the requirement and the solution that satisfies a static requirement. The model is, to a trained analyst, clear and easy to understand and, to a large degree, reasonable traceable. What structured analysis is not good at is adapting models to the continuously changing requirements that are the characteristic of modern businesses.
1.5 Object Orientation

Object Oriented software is based on the static, or object model - the things in the system, or, the things outside the system about which we need to record information, and their relationships. This is the most stable view in the system. It is why an object oriented model produces more stable software over a long period.

- Functionality from the interaction model is mapped onto the object model.
- The interaction model is, in turn, derived from the Use Case model which defines the functionality of the system from a user's perspective.
- The dynamic model defines the order and conditions under which things are done and is also mapped onto the object model.
1.5.1 An Object Oriented model produces more stable software

The object model is usually less affected by the changes to functional requirements that characterise most of the evolutionary process of the software. The things that are part of the problem and the way in which they are related tend to remain the same. It is what we want to do with them - the functional model - that changes the most. If, therefore, we model the structure of the problem and the structure of the resulting software solution around the object model instead of around the functional model, then we will produce systems that are more resilient to change over a period of time. This is what object orientation does and is the primary reason why object oriented software is more maintainable, re-useable and extensible that traditional structured software, especially over a long period.

In addition, the flexibility and resultant inherently portability of an object-oriented solution, tends to result in software that is inherently scaleable. The encapsulation of objects lends itself to redistribution across distributed platforms once the functionality of the application has been proved as a single machine solution.
The concept of encapsulation of data and functionality that belongs together is something which the hardware industry has been doing for a long time. Hardware engineers have been creating re-useable, re-configurable hardware at each level of abstraction since the early sixties. Elementary boolean functions are encapsulated together with bits and bytes of data in registers on chips. Chips are encapsulated together on circuit boards. Circuit boards are made to work together in various system boxes that make up the computer. Computer are made to work together across networks. Hardware design, therefore is totally object oriented at every level and is, as a result, maximally re-useable, extensible and maintainable. In a single word: flexible. Applying object-orientation to software, therefore, could be seen as putting the engineering into software design that has existed in hardware design for many years.

Hardware encapsulates data and function at every level of abstraction

Maximises maintainability, reuse and extension

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
In well developed object oriented software, functionality and data is encapsulated in objects. Objects are encapsulated in components. Components are encapsulated into systems. If this is done well the result is:

- Maximal coherence
- Minimal interconnection
- Solid interface definitions

- Maximum maintainability, reuse and extensibility
1.7.1 Getting the right encapsulation

Encapsulation of primitive functions, together with the data upon which they act and to which they belong, is fundamental to object orientation. The trick is to get the right functionality encapsulated with the right data. If this is achieved then the network of objects, which collaborate to provide the required functionality of the software, will have the minimum possible essential connections between objects and maximum functional cohesion of the objects. From this maximal functional cohesion can be derived maximal maintainability, reuse and extension. To achieve this we need two things. Firstly, the right objects. Secondly, to set the interface to the objects in concrete. If a change to an object requires a change to the interface to the object, then the effect of that change will escape beyond the boundaries of the object and propagate around the network. This is to allow the whole reason for encapsulation to be negated in one go. Proper Analysis and Design defines the right objects for the system and their interfaces to the point where each object can be given to a programmer and coded independently of any other object in the system. If this is done well then the resulting objects will still be around in future versions of the software, and, furthermore, their interfaces will not have changed except for additions. We aim for maximum maintainability, re-use and extension.
1.7.2 More Fundamentals

**Q** 1 In structured analysis

1. The functional view is the primary view
2. The data view records what is in the system
3. The dynamic view describes the flow of data
4. 1 and 2 are true
5. 1, 2 and 3 are true

**Q** 2 In object orientation

1. The functionality of the object model is derived directly from the use case model
2. The interaction model is mapped onto the dynamic model
3. The object model is the most stable view of the system
4. 1 and 3 are true
5. None is true

**Q** 3 Encapsulation of hardware

1. Is little used in modern electronics
2. Makes software development easier
3. Is inherent in the way most hardware engineers work

4. Object orientation
   1. Encapsulates software at object, component, and system level
   2. Maximises maintainability, reuse and extension
   3. Helps give applications a longer life
   4. All of the above
1.8 Summary and Test

- Abstraction is about looking only at the information that is relevant at the time.
- When modelling 2 or 3 orthogonal views of the system are created.
- Diagrams are independent views of the model and may be of different types.
- In structured analysis the functional view is the primary view of the system.
- In object oriented analysis, the object model, or data view, is the primary, and most stable view of the system.
- Hardware engineering produces well encapsulated hardware designs.
- Applying encapsulation techniques to software encourages maintainable, re-useable and extensible code.

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
1.8.1 Fundamentals of Object Oriented Modelling

When you are done, click the button to submit your answers and find out your score.

1. In structured analysis what word describes the view that defines the conditions under which things happen?
   
   [ ] False

2. Abstraction is about making all the information available in one succinct picture
   
   [ ] False
   [ ] True

3. What is the first requirement for maximal maintainability, re-use and extension?
   
   [ ] A The right objects
   [ ] B Well-defined interfaces
   [ ] C Maximum functional cohesion

4. What most destroys encapsulation?
   
   [ ] A Choosing the wrong objects
   [ ] B A lack of functional cohesion
   [ ] C Changes to an interface

5. How many views of a system are normally needed for the description to be complete? (type numeric integer)

   [ ]

6. Which of the following is true?
   
   [ ] A Connections between model elements are contained in diagrams
   [ ] B Model elements are contained in diagrams
   [ ] C Model elements appear on diagrams
   [ ] D Diagram text is held in a repository

Grade the Test

Your score will appear here
2. The Unified Modelling Language - Part 1

This chapter explains the origins of UML and some of the different kinds of diagrams available within it.

1. UML - What It Is
2. UML What It Is - 2
3. How UML Started
4. UML Diagram Types
5. UML Diagram Types - 2
6. Business Modelling and Web Applications and extending UML
7. Use Case Diagram
8. Class Diagrams
9. Summary and Test
10.
2.1 UML - What It Is

The Unified Modeling Language was originally developed at Rational Software but is now administered by the Object Management Group (see link). It is a modelling syntax aimed primarily at creating models of software-based systems, but can be used in a number of areas. It is:

- Syntax only - UML is just a language, it tells you what model elements and diagrams are available and the rules associated with them. It doesn't tell you what diagrams to create.

- Comprehensive - it can be used to model anything. It is designed to be user extended to fill any modelling requirement.

- Language independent - it doesn't matter what hi-level language is to be used in the code. Mapping into code is a matter for the case tool you use.
2.2 UML What It Is - 2

- **Process independent** - the process by which the models are created is separate from the definition of the language. You will need a process in addition to the use of UML itself.

- **Tool independent** - UML leaves plenty of space for tool vendors to be creative and add value to visual modelling with UML. However, some tools will be better than others for particular applications.

- **Well documented** - the UML notation guide is available as a reference to all the syntax available in the language.

- **Its application is not well understood** - the UML notation guide is not sufficient to teach you how to use the language. It is a generic modelling language and needs to be adapted by the user to particular applications.

- **Originally just for system modelling** - some user defined extensions are becoming more widely used now, for example for business modelling and modelling the design of web-based applications.

---

**Why Use UML?**

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
UML came about when James Rumbaugh joined Grady Booch at Rational Software. They both had object oriented syntaxes and needed to combine them. Semantically they were very similar, it was mainly the symbols that needed to be unified. The result was UML 1.0

Then Ivar Jacobson joined them. He brought with him the syntax for use cases which was added in UML 1.1. The Object Management Group adopted the UML1.1 specification in November 1997 making it an independent industry standard. Some small changes were made in in versions 1.3 and 1.4. Version 2.0 is currently being researched.
2.4 UML Diagram Types

UML provides a number of diagram types as a mechanism for entering model elements into the model and showing overlapping sets of models elements and their relationships. UML does not specify what diagrams should be created or what they should contain, only what they can contain and the rules for connecting the elements. The diagram types include:

- **Use Case Diagrams** - shows an outside-in view of the procedures available in the use of the system. These are summary diagrams and between them should contain all use cases available in the system and so all the available functionality of the system, represented at a high level.

- **Static Structure Diagram** - includes object and class diagrams. Most methods use class diagrams to describe the properties of the objects in the system and their relationships. Object diagrams are rarely used, except for examples of the way in which object interact, and these are normally shown on sequence or collaboration diagrams.

- **Interaction Diagrams** - these include collaboration and sequence diagrams, both of which show the way in which objects interact in order to fulfil the functionality of the use case.

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
Further diagram types include:

- **Activity Diagrams** - a generic flow chart used much in business modelling and sometimes in use case modelling to indicate the overall flow of the use case. This diagram type replaces the need for dataflow diagrams but is not a main diagram type for the purposes of analysis and design.

- **Component Diagrams** - shows the types of components, their interfaces and dependencies in the software architecture that is the solution to the application being developed.

- **Deployment Diagrams** - shows actual computing nodes, their communication relationships and the processes or components that run on them.
2.5.1 The Origins and Diagram Types of UML

1 Interaction diagrams include

1. Collaboration and class diagrams
2. Statecharts and sequence diagrams
3. Sequence and class diagrams
4. Collaboration and Sequence diagrams
5. Class and object diagrams

2 UML is

1. Language independent
2. Process independent
3. Easy to apply
4. 2 and 3
5. 1 and 2

3 The UML is currently controlled by

1. Rational Software
2. The Object Management Group
3. James Rumbaugh
4. Grady Booch
5. Ivar Jacobson

4 Which of the following statements is true?

1. Object diagrams are the most important diagram type
2. Collaboration diagrams show the relationship between states for a given object
3. Flowcharts are used to show the order in which objects interact
4. Activity diagrams can be used to show the overall flow of a use case
5. Deployment diagrams show the order in which functionality will be made available to users
2.6 Business Modelling and Web Applications and extending UML

UML can be used to model a business, prior to automating it with computers. The same basic UML syntax is used, however, a number of new symbols are added, in order to make the diagrams more relevant to the business process world. A commonly used set of these symbols is available in current versions of Rational Rose.

The most commonly used UML extensions for web applications were developed by Jim Conallen. You can access his own website to learn more about them by following the link. These symbols are also available in current versions of Rational Rose.

UML is designed to be extended in this way. Extensions to the syntax are created by adding 'stereotypes' to a model element. The stereotype creates a new model element from an existing one with an extended, user-defined, meaning. User defined symbols, which replace the original UML symbol for the model element, can then be assigned to the stereotype. UML itself uses this mechanism, so it is important to know what stereotypes are predefined in UML in order not to clash with them when creating new one.

Stereotypes allow the UML syntax to be used to model anything.
2.7 Use Case Diagram

The use case diagram show the functionality of the system from an outside-in viewpoint.

- Actors (stick men) are anything outside the system that interacts with the system.

- Use Cases (ovals) are the procedures by which the actors interact with the system
- Solid lines indicate which actors interact with the system as part of which procedures
- Dotted lines show dependencies between use cases, where one use case is ‘included’ in or ‘extends’ another.

Why Use Use Cases?

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
2.8 Class Diagrams

Class diagrams show the static structure of the systems. Classes define the properties of the objects which belong to them. These include:

- **Attributes** - (second container) the data properties of the classes including type, default value and constraints

- **Operations** - (third container) the signature of the functionality that can be applied to the objects of the classes including parameters, parameter types, parameter constraints, return types and the semantics.

- **Associations** - (solid lines between classes) the references contained within the objects of the classes to other objects enabling interaction with those objects.

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
2.8.1

Q 1 UML can be extended using stereotypes and used for

1. Business modelling
2. Web Applications
3. 1 and 2
4. Any problem domain
5. None of the above

Q 2 A use case is

1. A sequence of interactions between actors and the system
2. A procedure by which actors interact with the system
3. The way an actor to achieves the goal that is the name of the use case
4. A way of grouping together functional requirements
5. All of the above

Q 3 An association between two classes on a class diagram

1. Allows interaction between the two classes
2. Allows interaction between the objects of the two classes
3. Shows inheritance between the two classes
4. 1 and 2
5. 2 and 3
UML is just a syntax. It says nothing about how to create a model.

- UML is well documented but little understood.
- It was developed by Grady Booch, Jim Rumbaugh and Ivar Jacobon at Rational Software.
- UML specifies 8 different diagrams. Not all are used in practice.
- UML can be extended through the use of stereotypes.
- A use case diagram shows the functionality of the system from the outside.
- Class diagrams show the static structure of the systems.
2.9.1 The Unified Modelling Language - Part 1

When you are done, click the button to submit your answers and find out your score.

1. The Unified Modelling Language:
   - A Is easy to use
   - B Is designed just to model software
   - C Is process independent
   - D Is well documented
   - E Defines the order in which to create the diagrams
   - F C and D
   - G B, C and D
   - H C, D and E

2. Before UML, James Rumbaugh and Grady Boochs’ original
   - A Language semantics were similar
   - B Graphical symbols were similar
   - C A and B
   - D Neither

3. Which of the following are true?
   - A In most object modelling methods, the object diagram is the most used view of the system
   - B Collaboration and sequence diagrams show similar information
   - C A sequence diagram is a kind of interaction diagram
   - D Deployment diagrams show the types of components, their interfaces and dependencies
   - E B and C
   - F B, C and D
   - G All of the above

4. UML stereotypes:
   - A Can be used to change the original meaning of a model element
   - B Are user definable
   - C Can be used to customise the UML for any modelling purpose
   - D Extend the semantic of a model element
   - E B and C
   - F B, C and D
   - G All of the above

5. On a Use case diagram, interaction between the things outside the system and the system is shown by
   - A Solid lines between actors and use cases
   - B Dotted lines between actors and use cases
   - C Dotted lines between use cases
   - D A and B
   - E B and C
   - F All of the above
6. On class diagrams, communication between objects is enabled by
   - A Attributes on the classes
   - B Associations between classes
   - C Solid lines between classes
   - D A and B
   - E B and C
   - F All of the above

Grade the Test

Your score will appear here
3. The Unified Modelling Language - Part 2

This chapter explains the rest of the different kinds of diagrams available within UML.

1. Sequence Diagram
2. Collaboration Diagram
3. Statechart Diagram
4. Activity Diagram
5. Component Diagram
6. Deployment Diagram
7. Modelling Architecture
8. Summary and Test

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
3.1 Sequence Diagram

Sequence diagrams show potential interactions between objects in the system being defined. Normally these are specified as part of a use case or use case flow and show how the use case will be implemented in the system. They include:

- **Objects** - oblong boxes at the top or actors, named or just shown as belonging to a class from or to which messages are sent to other objects.

- **Messages** - solid lines for calls and dotted lines to data returns, showing the messages that are send between objects. This includes the order of the messages which is from top of the diagram to the bottom.

- **Object lifelines** - dotted vertical lines showing the lifetime of the objects.

- **Activation** - the vertical oblong boxes on the object lifelines showing the thread of control in a synchronous system.

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
3.2 Collaboration Diagram

Collaboration Diagrams show similar information to sequence diagrams, except that the vertical sequence is missing. In its place are:

- Object Links - solid lines between the objects. These represent the references between objects that are needed for them to interact and so show the static structure at object level. On the links are:

  - Messages - arrows with one or more message name that show the direction and names of the messages sent between objects
Statecharts, often used more in real-time embedded systems than in information systems, show, for a class, the order in which incoming calls to operations normally occur, the conditions under which the operations respond and the response. They are a class-centric view of system functionality as opposed to sequence and collaboration diagrams which are a use case-centric view of functionality. They include:

- **States** - oblong boxes which indicate the stable states of the object between events.

- **Transitions** - the solid arrows which show possible changes of state.
- **Events** - the text on the transitions before the '/' showing the incoming call to the object interface which causes the change of state.
- **Conditions** - a boolean statement in square brackets which qualifies the event.
- **Actions** - the text after the '/' which defines the objects response to the transition between states.
- **Extra syntax** which defines state centric functionality
3.3.1

1. The activation on a sequence diagram shows
   1. The messages between objects
   2. The objects involved in a use case
   3. The thread of control in the sequence
   4. The condition under which an operation is executed
   5. None of the above

2. A collaboration diagram shows
   1. The thread of control between objects
   2. References between objects
   3. Similar information to sequence diagrams
   4. 1 and 2
   5. 2 and 3

3. An event on a statechart shows
   1. What caused the transition
2. What caused the change of state
3. The action performed on the transition
4. 1 and 2
5. 2 and 3

(mouse here for the answer)
3.4 Activity Diagram

A UML Activity Diagram is as general purpose flowchart with a few extras. It can be used to detail a business process or to help define complex iteration and selection in a use case description. It includes:

- Active states - oblongs with rounded corners which describe what is done.

  ![Diagram](image)

- Transitions - which show the order in which the active states occur and represent a thread of activity.
- Conditions - (in square brackets) which qualify the transitions.
- Decisions - (nodes in the transitions) which cause the thread to select one of multiple paths.
- Swimlanes - (vertical lines the length of the diagram) which allow activities to be assigned to objects.
- Synch States - horizontal or vertical solid lines which split or merge threads (transitions)

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
Component Diagrams show the types of software components in the system, their interfaces and dependencies.
Deployment diagrams show the computing nodes in the system, their communication links, the components that run on them and their dependencies.
3.7 Modelling Architecture

Class diagrams can be used to model hi-level architecture with packages, interfaces and dependencies. Packages are used to group together a set of model elements for various purposes. The results may show:

- Subsystems - the design view of a software component or re-useable part of a component.
- Libraries of re-useable elements, usually classes.
- The hierarchic structure and layering of the system
- Client-server relationships between components and other model elements
- Logical dependencies of sub-systems and libraries on one another

Quiz

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
3.7.1

Q  1 In an activity diagram, which of the following is true?

1. A synch state defines a stable state of the system
2. A swim lane can be used to show object interaction
3. A condition qualifies a transition
4. A decision splits the thread into concurrent threads
5. A transition shows a stable state of the system

(Question Image)

Q  2 A component diagram shows:

1. Classes and their relationships
2. The types of components in the system
3. Dependencies between components
4. 1 and 2
5. 2 and 3

(Question Image)

Q  3 Which of the following is true?

1. Packages appear on sequence diagrams

(Question Image)
2. Dependencies appear on class diagrams
3. Packages show client-server relationships between components
4. 1 and 2
5. 2 and 3

**Q** 4 Computing nodes on a deployment diagram

1. Show connections between components
2. The components that run on computers
3. What communication links are available between subsystems
4. What processors are available
5. What communication links are available between computers
3.8 Summary and Test

- Sequence diagrams show potential interactions between objects in the system in the order in which they normally occur.
- Collaboration diagrams also show interaction between objects, but emphasise the structure required to support them.
- Statecharts are a class-centric view of system functionality.
- An activity diagram is as general purpose flowchart.
- Component diagrams show the types of software components in the system, their interfaces and dependencies.
- Deployment diagrams show the computing nodes in the system, their communication links, the components that run on them and their dependencies.
- Class diagrams can be used to model hi-level architecture with packages, interfaces and dependencies.

http://www.cragsystems.com/ITMUML/the03/08the03.htm
1. The vertical dotted lines on a sequence diagram show
   - A The thread of control in a synchronous system
   - B The messages sent between objects
   - C The lifetime of the object
   - D A and C
   - E A and B
   - F B and C
   - G All of the above

2. On a collaboration diagram, messages:
   - A Are shown as solid lines between objects
   - B Are shown as arrows on the links between objects
   - C Are shown as arrows between objects
   - D May be numbered
   - E Have message names
   - F C, D and E
   - G C and E
   - H B, D and E
   - I All of the above

3. A condition on a statechart
   - A Qualifies an action
   - B Appears in curly brackets
   - C Resolves to true or false
   - D Qualifies a transition
   - E Appears in square brackets
   - F A, B & C
   - G C, D & E

4. On an activity diagram, which of the following is true
   - A An activity diagram is the same as state chart, only with different symbols
   - B A decision steers the thread
   - C A swimlane can represent an activity
   - D A sync state splits or combines the thread
   - E B & C
   - F B & D
   - G C & D

5. A component diagram
   - A Shows the decomposition of the system into subsystems
   - B Shows the decomposition of the systems into software elements
   - C Shows relationships between computing nodes
   - D Shows dependencies between software elements
   - E B & D
6. A deployment diagram

- A Shows the decomposition of the system into subsystems
- B Shows the decomposition of the systems into software elements
- C Shows relationships between computing nodes
- D Shows dependencies between software elements Shows dependencies between software elements
- E B & D
- F C & D
- G B & C
- H None of the above

7. A sub-system can represent

- A Libraries of classes
- B The design of a component
- C The design of part of a component
- D A layer in the architecture
- E B & C
- F B & D
- G B, C & D

Grade the Test

Your score will appear here
4. The Software Development Process

In order to make the development of software predictable and repeatable, a process is needed which defines what is done by whom and in what order. When using UML this revolves around the creation of models at different levels of abstraction and increasing levels of detail. This chapter takes you through each step of the process, shows what models are created, and how information flows between them.

1. Business Process Modelling
2. System Requirements Definition
3. The System Analysis Model
4. System Model Information Flow
5. Screen Prototyping
6. The System Design Model
7. Overall Process Flow
8. Incremental Development
9. Summary and Test
10. Course Feedback
4.1 Business Process Modelling

A Business Model may be created to better understand the business, either for the purpose of improving the business itself, or as a way of discovering requirements for a computer system. Business processes can be modelled at various levels within the organisation. The model uses all the standard rules of UML and includes some user defined extensions. The differences include:

- Slightly different symbols for business actors and business use cases (also known as business processes)
- Some of Jacobson's stereotypes with additions to represent business entities and business workers
- The system boundary is now the business or the part of the business being modelled
4.2 System Requirements Definition

Implementation constraints are modelled separately from the use case model. The logical functional requirements of a system are modelled using use cases and use case descriptions - documents associated with the use case giving a detailed textual description of the use case flow and other logical constraints. Using use cases for the logical requirements has a number of advantages:

- It is an outside-in view and easily understood by non-technical people.

- It is more likely to be complete than a classical functionally decomposed specification.

- It directly maps into and is traceable to acceptance tests, user documentation and the analysis and design models.

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
4.3 The System Analysis Model

The System Analysis Model is made up of class diagrams, sequence or collaboration diagrams and statechart diagrams. Between them they constitute a logical, implementation-free view of the computer system that includes a detailed definitions of every aspect of functionality. This model:

► Defines what the system does not how it does it.
► Defines logical requirements in more detail than the use case model, rather than a physical solution to the requirements.
► Leaves out all technology detail, including system topology.
4.4 System Model Information Flow

The diagram illustrates the way in which the 3-dimensional system model is developed iteratively from the uses case model in terms of the information which flows between each view. Note that it is not possible fully to develop any one of the three views without the other two. They are interdependent. This is the reason why incremental and iterative development is the most efficient way of developing computer software.
4.4.1

1 Business process modelling of use cases uses
   1. The same symbols as used in analysis
   2. Similar symbols with additions
   3. Different symbols entirely
   4. Doesn't use symbols

2 System requirements defined with use cases map directly into
   1. User documentation
   2. System acceptance tests
   3. The analysis model
   4. The design model
   5. All of the above

3 The system analysis model leaves out
   1. Topology
   2. Technology
3. Functionality
4. 1 and 2
5. 2 and 3

4 Information flows from the interaction model to the

1. state model
2. object model
3. use case model
4. 1 and 2
5. 2 and 3
4.5 Screen Prototyping

Screen prototyping can be used as another useful way of getting information from the users. When integrated into a UML model,

- The flow of the screen is made consistent with the flow of the use case and the interaction model.

- The data entered and displayed on the screen is made consistent with the object model.

- The functionality in the screen is made consistent with the interaction and object models.
4.6 The System Design Model

This model is the detailed model of everything that is going to be needed to write all the code for the system components. It is the analysis model plus all the implementation detail. Preferable it should be possible automatically to generate at least frame code from this model. Then if any structural changes to the code are made in the design model first and then forward re-generated, this ensures that the design model accurately reflects the code in the components. The design model includes:

- Class, sequence or collaboration and state diagrams, as in the analysis model, but now fully defined ready for code generation
- Component diagrams defining all the software components, their interfaces and dependencies
- Deployment diagrams defining the topology of the target environment including which components will run on which computing nodes

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
The overall process flow must allow for both rework and incremental development.

- **Rework** - where changes need to be made, the earliest model that the change affects is changed first and the results flowed forward through all the other models to keep them up to date.

- **Incrementation** - increments can restart at any point, dependent on whether the work needed for this increment has already been completed in higher level models.
Incremental Development is based on use cases or use case flows which define working pieces of functionality at the user level. In an increment, the models required to develop a working software increment are each incremented until a working, tested executing piece of software is produced with incremental functionality. This approach:

- Improves estimation, planning and assessment. Use cases provide better baselines for estimation than traditionally written specifications. The estimates are continuously updated and improved throughout the project.

- Allows risks to the project to be addressed incrementally and reduced early in the lifecycle. Early increments can be scheduled to cover the most risky parts of the architecture. When the architecture is stable, development can be speeded up.

- Benefits users, managers and developers who see working functionality early in the lifecycle. Each increment is, effectively, a prototype for the next increment.
**4.8.1**

1. Screen prototyping can be integrated with
   1. the use case model
   2. the interaction model
   3. the object model
   4. 1 and 2
   5. 1, 2 and 3

2. The UML system design model may include
   1. 1 diagram type
   2. 2 diagram types
   3. 3 diagram types
   4. 4 diagram types
   5. 5 diagram types

3. An increment starts at
   1. requirements
2. analysis
3. design
4. coding
5. any point in the waterfall

4 A increment produces
1. increased system functionality
2. a working executable
3. a tested executable
4. implemented use cases
5. all of the above
4.9 Summary and Test

After taking the test, please complete the Course Feedback on the next page.

- A software process is needed to make a software development predictable and repeatable
- UML can be used to model business processes and structure
- Functional requirements are defined with use cases and use case descriptions
- A system analysis model is a technology-free model of the detail of what the system will do
- The system design model is a detailed model of the software structure of the system down to class property level
- The use case model feeds the object, interaction and state models which are interdependent
- Screen prototyping can be integrated with the other views of the system
- The classic waterfall process should be augmented by both rework (iterative) and incremental loops
- Incremental development improves estimation, planning, assessment and software quality

Display the CRaG Systems website

Copyright © 1997 - 2004 CRaG Systems All rights reserved
4.9.1

When you are done, click the button to submit your answers and find out your score.

1. The boundary of the business model is
   - A The aspects of the system model needed to define the system
   - B The business itself
   - C The part of the business being modelled
   - D A or B
   - E A or C
   - F B or C

2. The use case model includes
   - A Physical requirements
   - B Logical requirements
   - C Functional requirements
   - D Business requirements to be automated
   - E Business requirements that are not automated
   - F A, B & C
   - G B, C and D
   - H C, D & E
   - I All of them

3. The system analysis model
   - A Includes statecharts
   - B Defines what the system does
   - C Is a detailed set of logical requirements
   - D Does not contain implementation detail
   - E A, B & C
   - F All of the above

4. Which of the following statements best describes the approach to system model development?
   - A Elements of a system model are created in a particular order
   - B The object model is the only essential part of the model
   - C Each of the three orthogonal views depends on the other two
   - D The use case view is one of the three orthogonal views
   - E Any two of the three orthogonal views is a sufficient definition of the system

5. Which of the following is true
   - A Prototype screens are a useful way of eliciting information from users
   - B The screen flow must be consistent with the object model
   - C The data entered on the screen should be consistent with the object model
   - D A & B
   - E A & C
   - F B & C
   - G None of the above

6. In UML, how many diagrams types are needed for a full definition of the design model?
7. If a change is made to a model, then the models that need to be updated are
   ○ A Only the model that has been changed
   ○ B All models prior to the one that has been changed
   ○ C All models subsequent to the one that has been changed
   ○ D The earliest model affected and all subsequent models
   ○ E All the models

8. Risk-based scheduling of increments
   ○ A Uses use cases or use case flows as units of functionality
   ○ B Improves estimation
   ○ C Produces a stable architecture early in the lifecycle
   ○ D Reduces risk early in the lifecycle
   ○ E A, B & C
   ○ F B, C & D
   ○ G A, C & D

Grade the Test
Your score will appear here