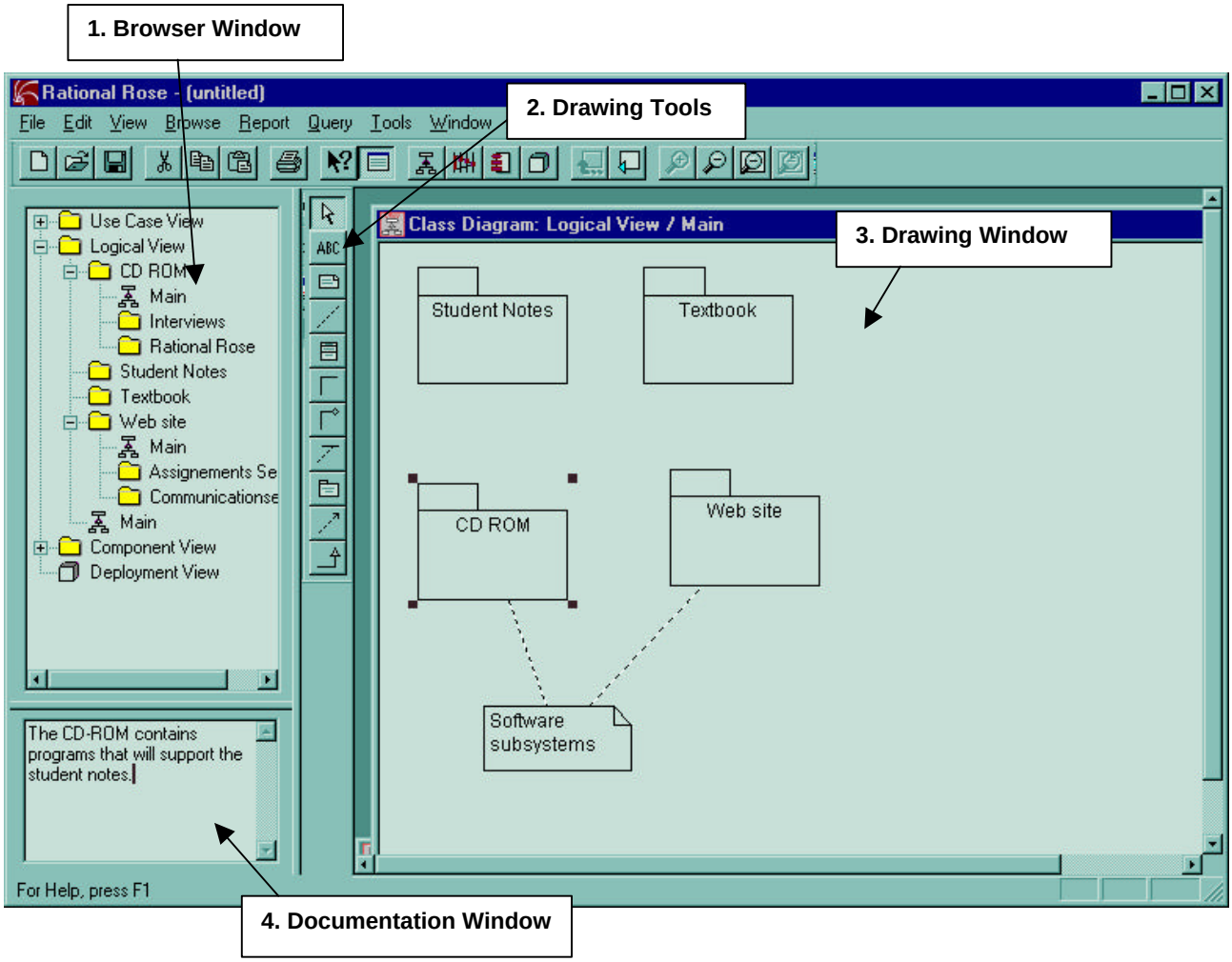
An Introduction to Rational Rose

In this course you will be learning the Unified Modeling Language (UML), which is a special notation for systems analysis and design. Creating UML diagrams requires a diagramming tool. The tool you will be using for this purpose is a student version of Rational Rose. Rational Rose is a sophisticated CASE tool with a number of automated features, including code generation and reverse engineering. The version you will use in this course is not the most recent version and there are limitations on the size of the diagrams you can save.



1. **Browser Window**

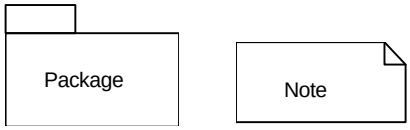
   This presents a hierarchical view of the analysis and design model, including all the diagrams and all the individual elements that make up a diagram.

2. **Drawing Tools**

   This tool presents a set of icons that indicate the different elements that can be added to a diagram. The elements that can be used will change, depending on the type of diagram being created. Different diagram types have different sets of icons. If you were creating a different diagram type, you would see a different set of icons. The above example is a class diagram in logical view.

3. **Diagram Window**

This is where the diagram is actually created. You will see that the diagram shown in the drawing window on Figure 1 represents a high-level model of this course. Course content can be seen as a system composed of four interacting subsystems, two of which involve software. We have used the Package element to represent the subsystems, and the Note element to indicate which packages contain software.



4. **Documentation Window**

It is strongly recommended that each element added to a diagram have documentation to accompany it. To add documentation, right click on the element, select `specification`, and fill in the documentation field. The documentation will then be shown in the documentation window each time the mouse is clicked on the element. Documentation can also be added directly to the documentation window.

Views in UML/Rational Rose

There are four views for a model created in Rational Rose, each representing the system from a different point of view.

The Use Case View
The use case view contains the diagrams used in analysis (use case, sequence, and collaboration), and all the elements that comprise these diagrams (e.g., actors). More recent versions of Rational Rose also allow for additional documentation in the form of word-processed documents and/or URLs to Web-based materials. The purpose of the use case view is to envisage what the system must do, without dealing with the specifics of how it will be implemented.

Logical View
The logical view contains the diagrams used in object design (class diagrams and state transition diagrams). It offers a detailed view of how the system envisaged in the use case view will be implemented. The basic element in this view is the *class*, which includes an outline of its attributes and operations. This directly corresponds to a class created in your chosen implementation language. From the logical view, skeletal code can be generated for implementation into a computer language. More recent versions of Rational Rose not only can generate skeletal code for Visual C++, Visual Java, or Visual BASIC, but also reverse engineer programs created in these languages into Rational Rose models. This allows existing components to be included in documented models, if there is access to the source code. In addition, changes that need to be made during implementation can be reflected in the documentation of the design model.

Component View
The component view is a step up from the logical view and contains diagrams used in system design (component diagrams). This includes information about the code libraries, executable programs, runtime libraries, and other software components that comprise the completed systems. Components can be pre-existing; for example, a Windows program in Visual C++ will utilize Microsoft Foundation Class to provide the framework for the Windows interface. Components that do not exist and need to be created by the developers will have to be designed in the logical view.

Deployment View
The deployment view illustrates how the completed system will be physically deployed. This view is necessary for complex applications in which a system will have different components located on different machines. For example, interface components may be located on a user machine while other components may be located on a network server.