IBM Rational University

Rational. software

DEV396 Essentials of Rational Software Architect, 2005.01.00

Student Guide

Part No. 800-027144-000

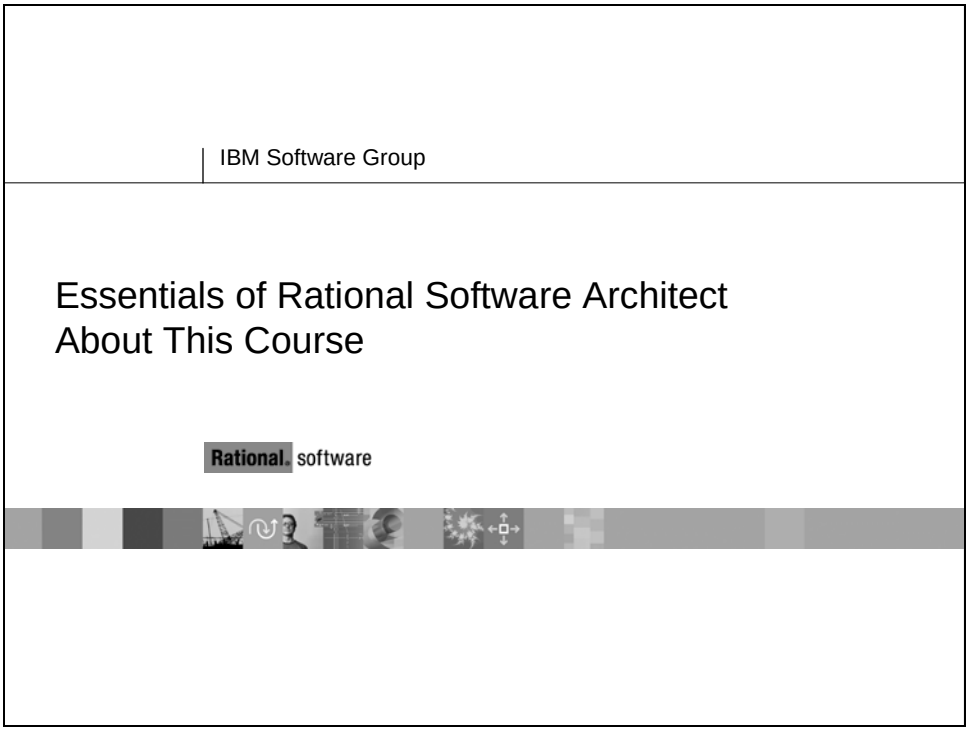
IBM Corporation
Rational University
DEV396 Essentials of Rational Software Architect
Student Manual
Version 2005.01.00

January 2005

This manual prepared by:
IBM Rational Software
18880 Homestead Road
Cupertino, CA 95014-0721
USA

Contents

▸ ▸ ▸ About This Course



Topics

Introductions

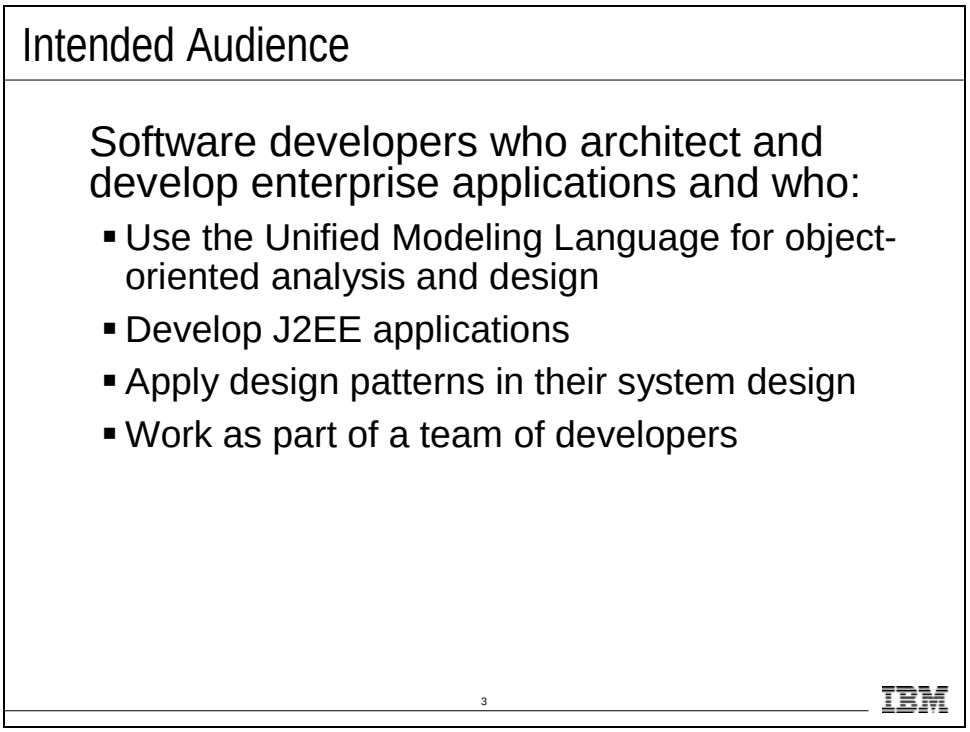
Introductions

- Your organization
- Your role
- Your background and experience
  - Software development experience
  - Object technology experience
- Course expectations

2

IBM

Intended Audience



Essentials of IBM® Rational® Software Architect™ teaches architects, designers, and lead developers how to apply Rational Software Architect to perform model-driven development, apply patterns and perform model transformations, and develop software with Software Architect as part of a development team.

Prerequisites

Prerequisites

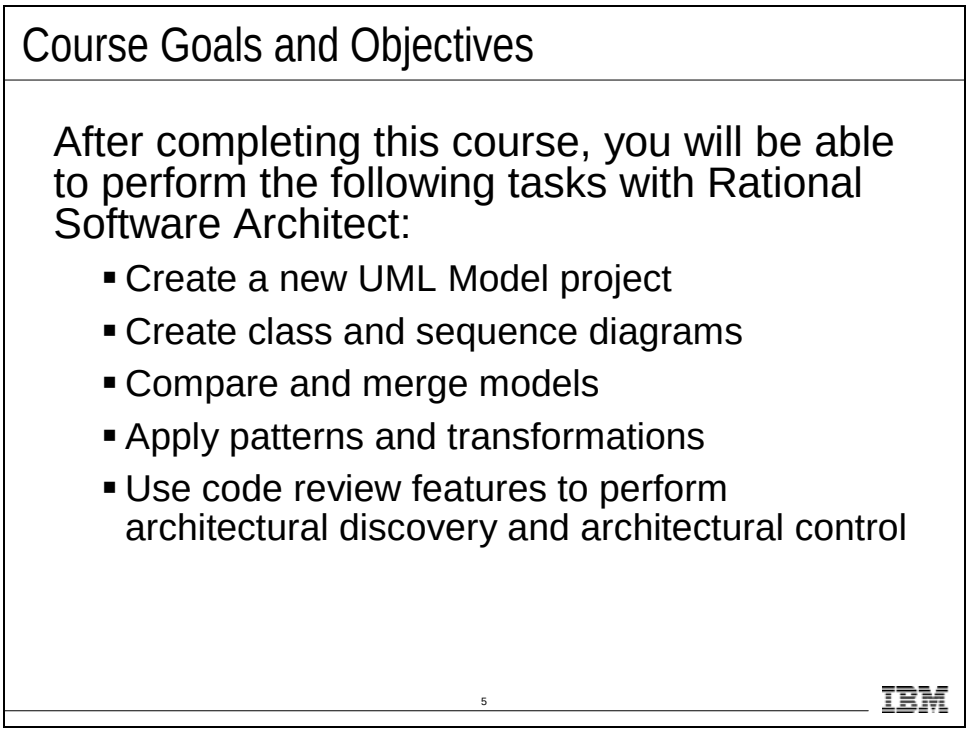
This course assumes that students:

- Have experience architecting and designing J2EE solutions
- Have taken the following IBM Rational brand courses or have equivalent experience:
  - DEV275: Essentials of Visual Modeling (ILT) or the DEV11x: Principles of Modeling (WBT) series
  - DEV475: Mastering Object-Oriented Analysis and Design Using UML

4

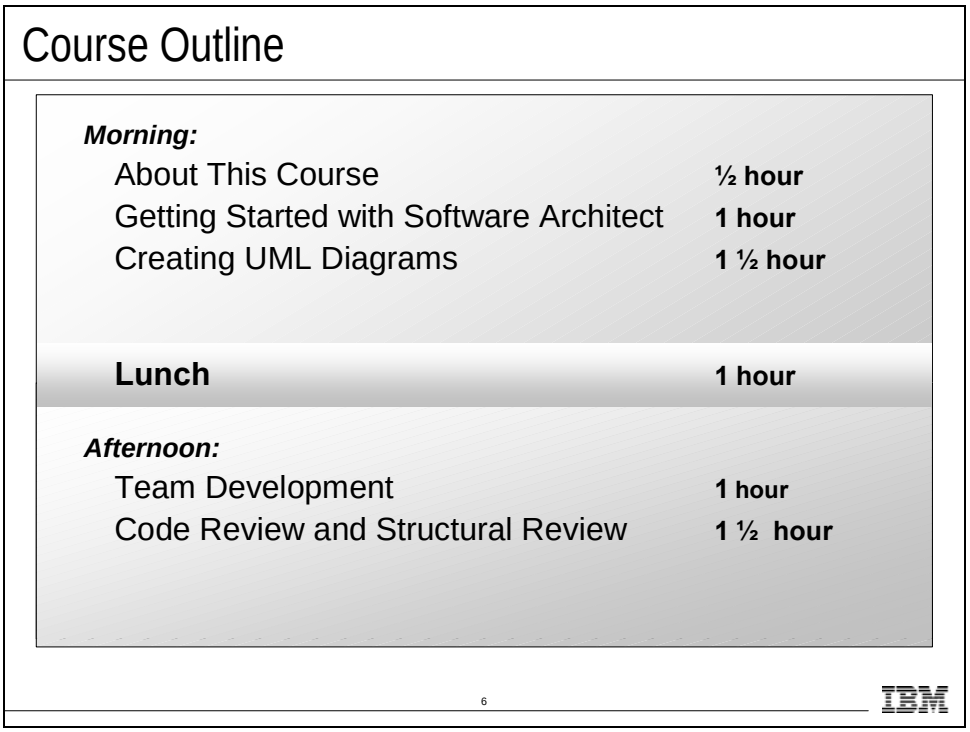
IBM

Course Goals and Objectives



This course introduces Rational Software Architect for modeling enterprise applications. The course includes discussion of team development issues, patterns, and transformations and how to use code review features for architectural control and architectural discovery.

Course Outline

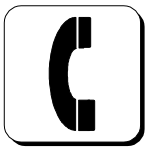


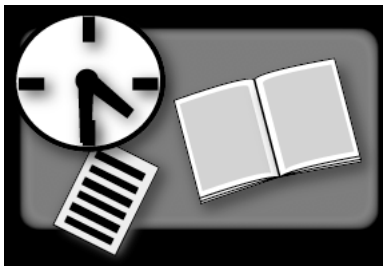
Times for modules are estimates only. Class times may vary.

Logistics

Logistics





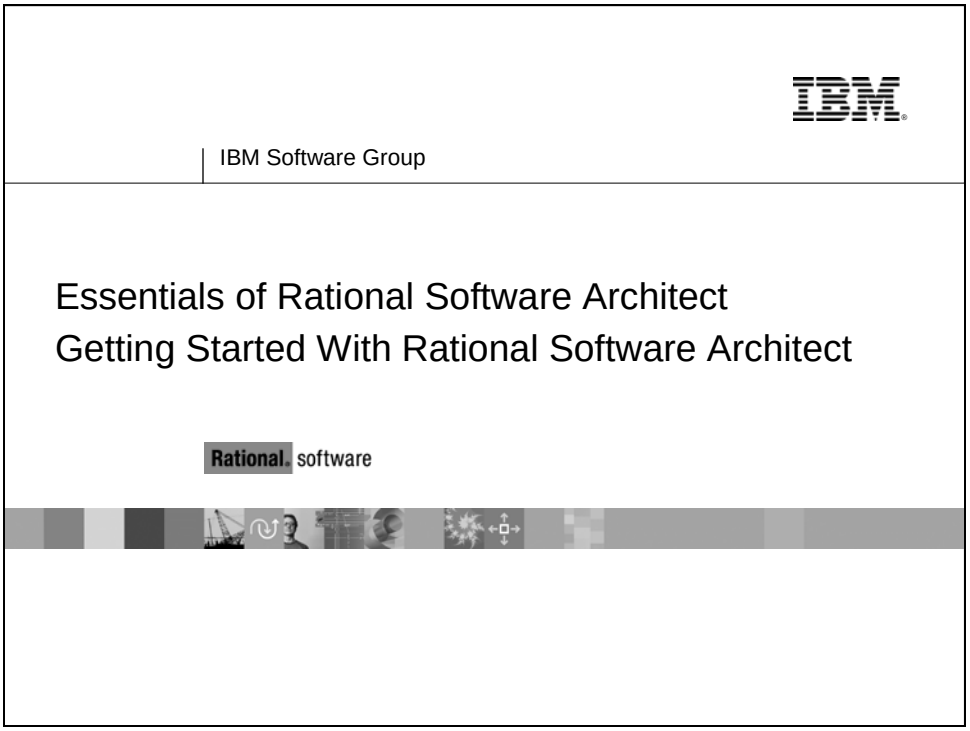


Morning
2 15-minute breaks
Lunch
1 Hour
Afternoon
2 15-minute breaks

7

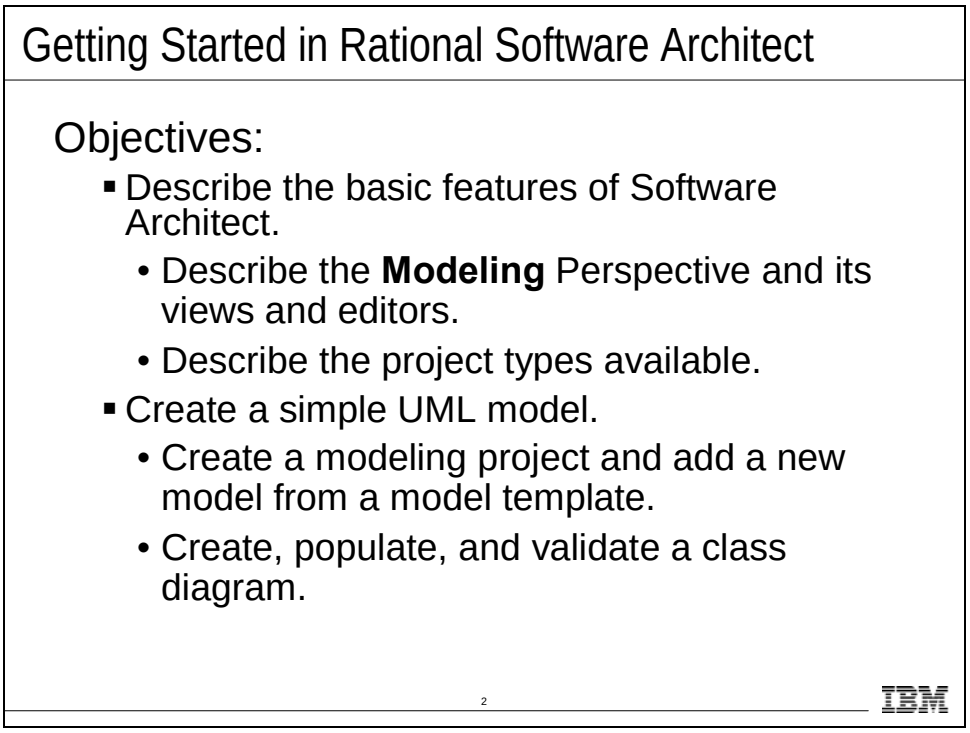
IBM

▸ ▸ ▸ Getting Started With Rational Software Architect



Topics

Module Objectives



This module describes the Rational Software Architect workbench's **Modeling** Perspective and the unique views it features.

It also presents an overview of Rational Software Architect project and model types.

Introduction to Rational Software Architect

Where Are We?

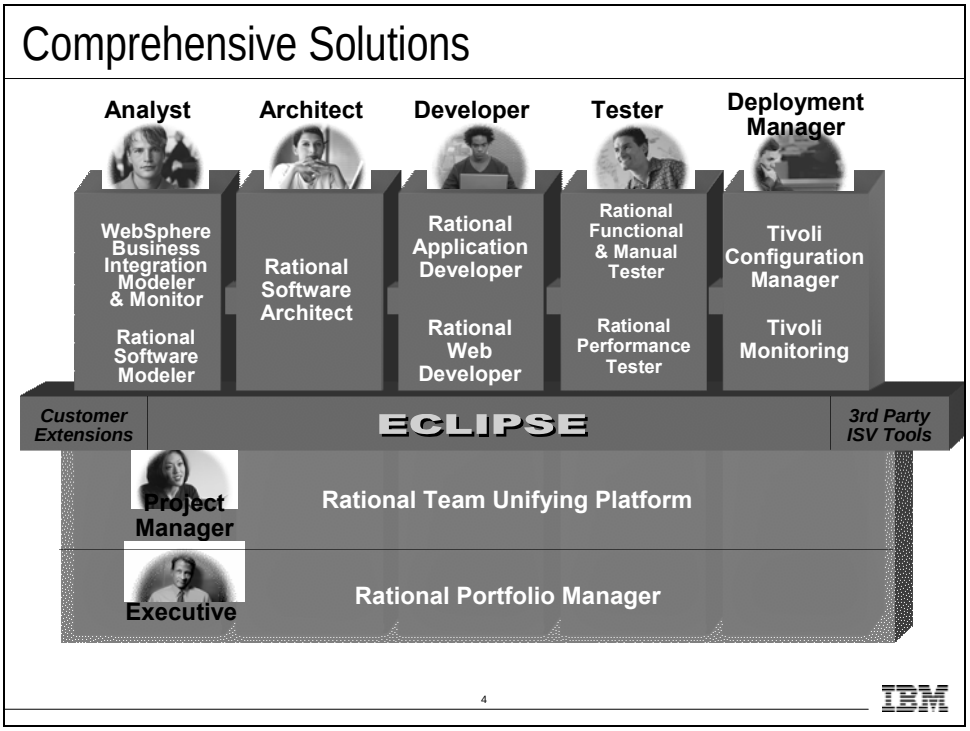
- **Introduction to Rational Software Architect**
- The Modeling Perspective
- Views in the Modeling Perspective
- Software Architect projects

3

IBM

This section introduces Rational Software Architect, its key features, and roles in software development.

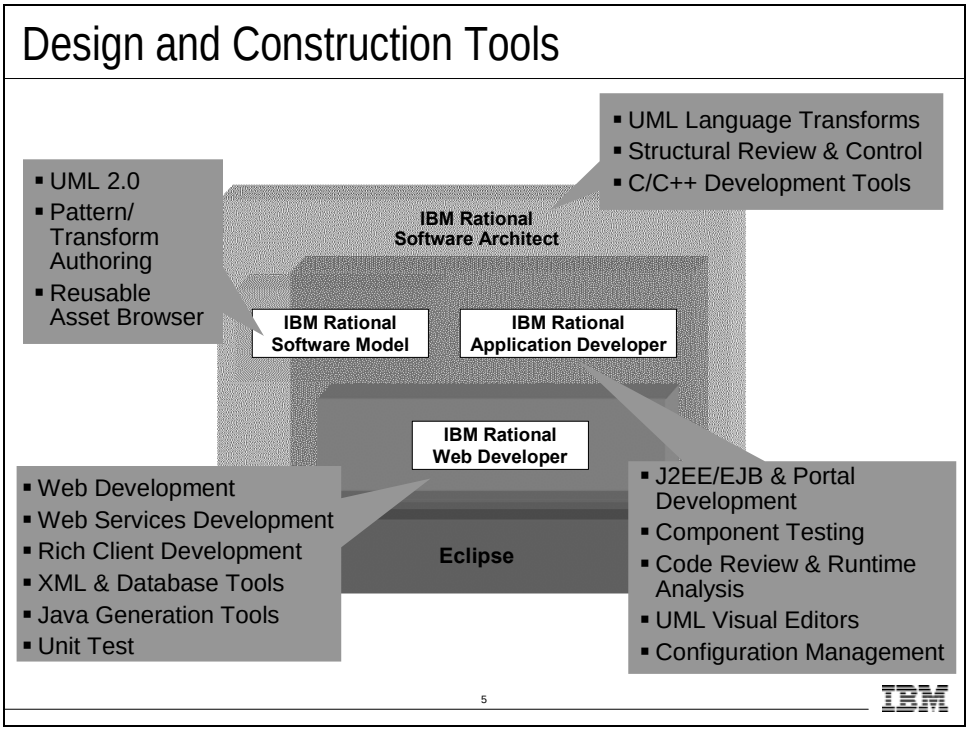
Comprehensive Solutions



The IBM Software Development Platform is based on an architectural foundation called the Eclipse universal framework. This framework enables unprecedented tool integration and artifact traceability throughout the development lifecycle.

What is Eclipse? It is both an open source software development project with over 100 active vendor participants and a rich foundation for providing shared services across an application tooling environment.

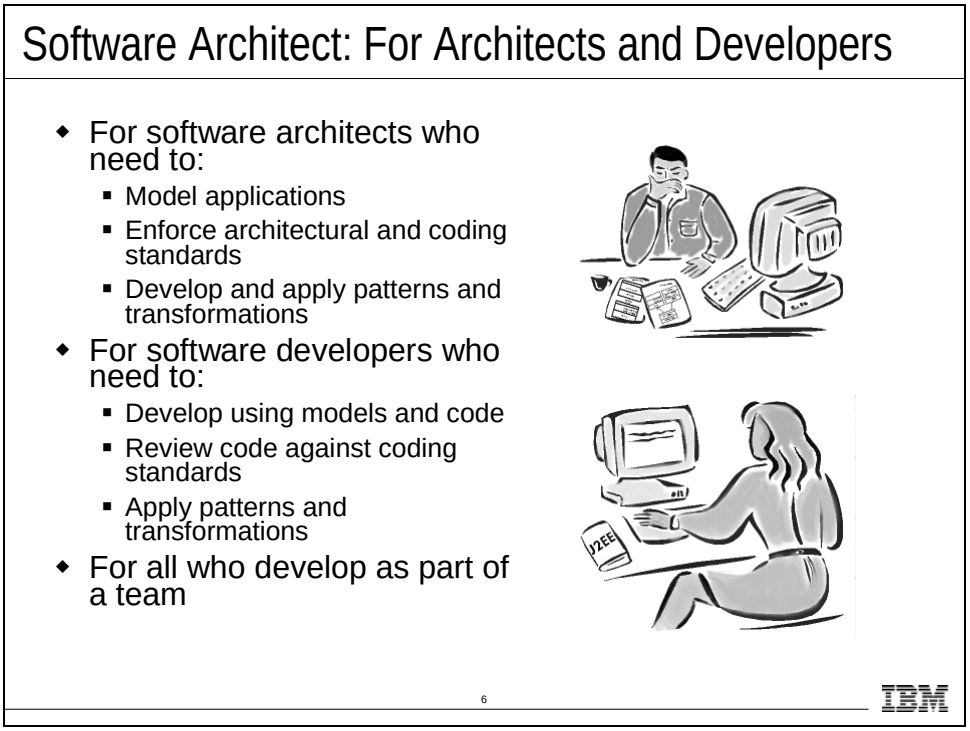
Design and Construction Tools



Rational brand promises an integrated and automated software development experience across the team. In addition to offering world class functionality, each of these solutions shares some common benefits. Each of these solutions is:

- **Flexible:** You can match the tools to diverse skill sets in enterprise development efforts: no compromise required.
- **Modular:** You can have the tools you need to precisely fit the needs of your organization.
- **Open:** Each represents the implementation of open industry standards, so you are not limited to interoperating with any set of proprietary standards.
- **WebSphere optimized:** Rational Application Developer and Rational Web Developer are optimized for WebSphere software and also provide capabilities for development on other technology platforms.

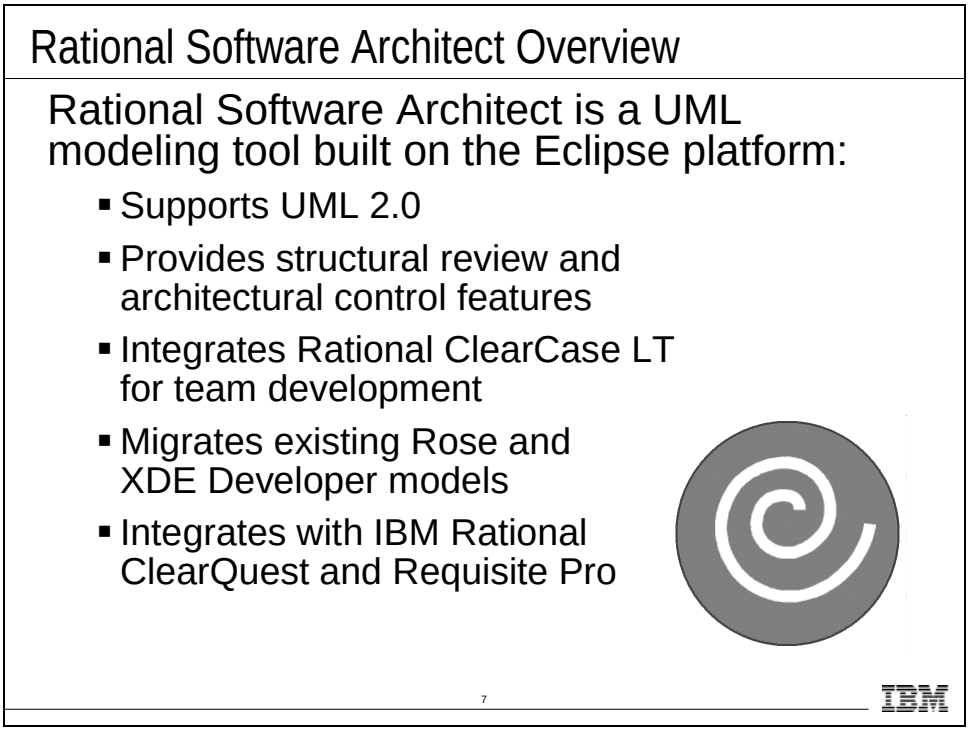
Software Architect: For Architects and Developers



Software architects and senior developers within a development team are responsible for specifying and maintaining all aspects of an application's software architecture. They need powerful and configurable tools for managing the complexity found in today's applications.

IBM Rational Software Architect is a UML model-driven design and development tool for creating well-architected applications and services.

Rational Software Architect Overview

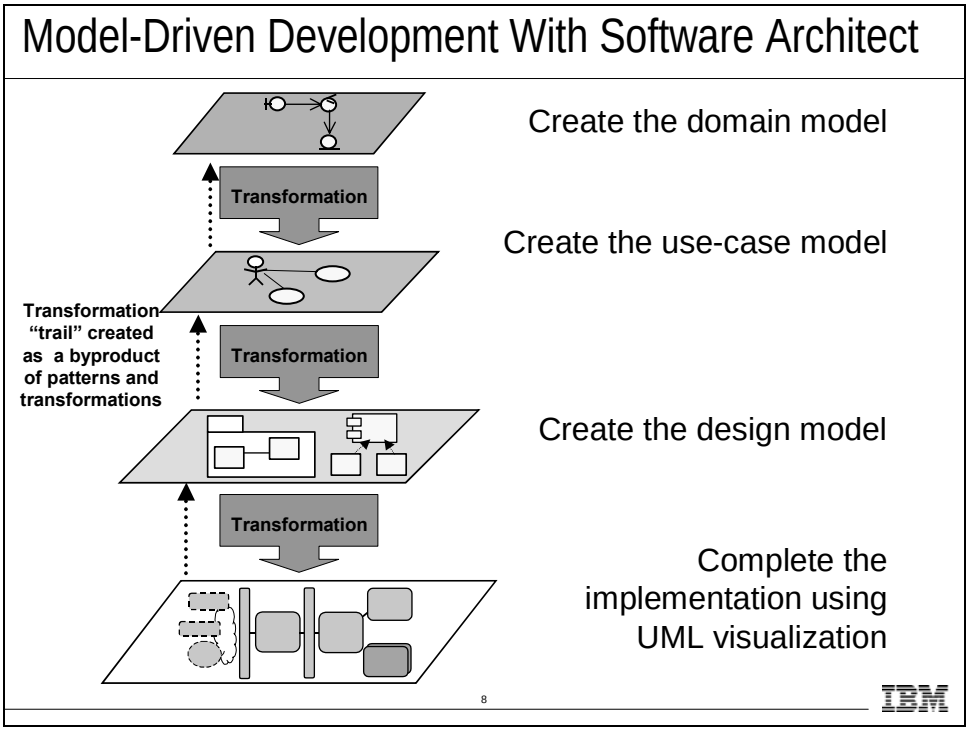


Rational Software Architect unifies UML modeling, Java structural review, and process guidance with J2EE, C++, database, XML, Web, and Web services development tools into one powerful and easy to use development environment.

Software Architect:

- Is built on the open and extensible Eclipse platform, making it ideal for creating applications optimized for IBM middleware, as well as for other vendors' technologies. The Eclipse platform also enables development on multiple platforms (Windows and Linux).
- Supports the latest UML 2 modeling constructs, allowing users to specify software architectures with more clarity and control.
- Introduces new architectural review and control features for Java applications to avoid problems with functionality, scalability, and maintainability introduced during implementation.
- Integrates with the IBM Rational team unifying platform, which provides requirements management, traceability, source code control, and other team management functions throughout the lifecycle.

Model-Driven Development With Software Architect



Software Architect is designed to support model-driven development, the development of the appropriate models to facilitate all development activities and stages in the lifecycle, plus tools to transform models into more detailed or more abstract models to move development work forward.

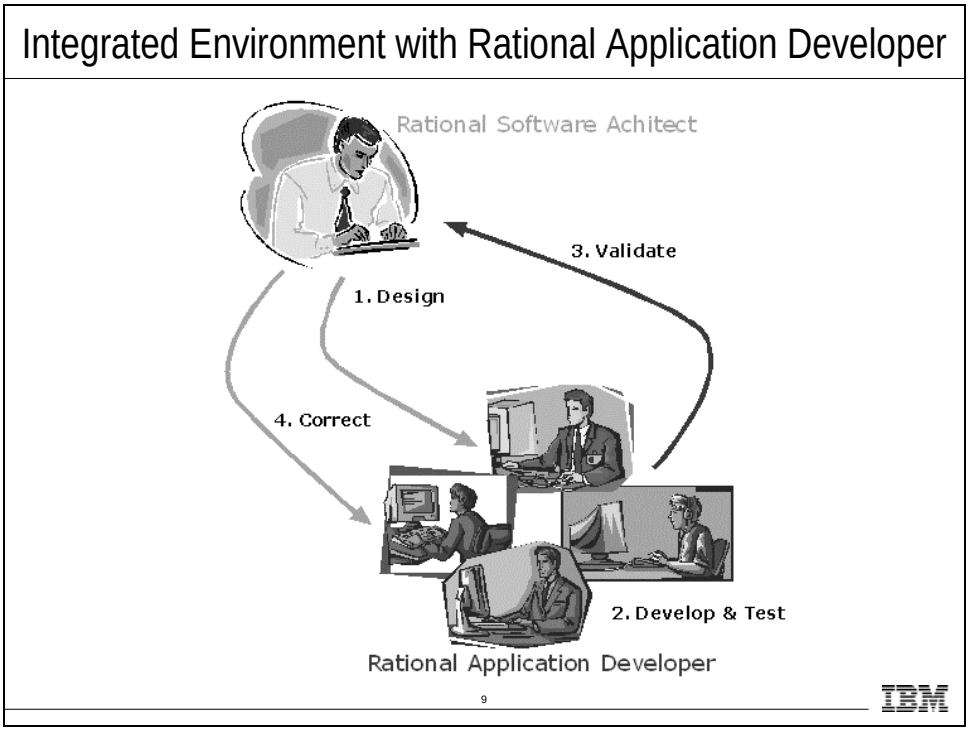
An analyst might begin by modeling the business domain in Software Architect to define the key products, deliverables, or events. The analyst can then create a use-case model to define the actors, system boundary, and use cases the system will support.

The architect then uses Software Architect to create a platform-independent design model from the use-case model. This model or set of models can be transformed in platform-dependent implementation models (including code and UML) with the assistance of visual development tools, such as:

- UML editors for Java, C++, or data
- Site Designer
- Page Designers

As each new stage of development begins, transformations can be used to create more detailed models that are incrementally closer to the target platform and infrastructure. Transformations can be designed to include traceability so that you can query the target model using elements from the source model to find elements in the target model. This feature is currently built into the UML to Java transformation that comes with Software Architect. After the transformation is complete, you can right-click a model element in the design model and perform a query to find the associated Java code.

Integrated Environment with Rational Application Developer



Rational Application Developer for WebSphere Software provides developers with all the tools they need to implement Rational Unified Process (RUP) quality guidelines in every phase of the software development lifecycle. The workspace is integrated into the source control system. Testing tools are readily available when changes have been merged.

This allows developers to work efficiently:

- Receiving designs from architects
- Developing software to design
- Testing code to validate function and verify conformance with code requirements
- Sharing code with architects during development and receive corrections
- Implementing changes and validating them prior to checking them back into the development tree

This simplifies code reviews and allows developers to concentrate more on enhancements than defects.

The Modeling Perspective

Where Are We?

- Introduction to Rational Software Architect
- **The Modeling Perspective**
- Views in the Modeling Perspective
- Software Architect Projects

10

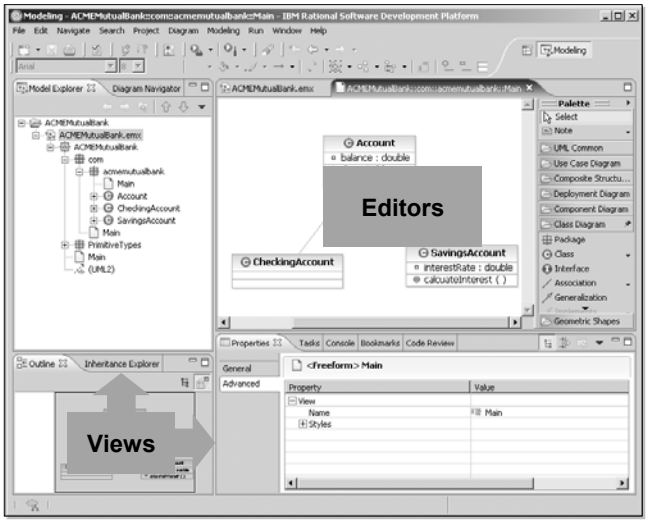
IBM

This section discusses the **Modeling** Perspective in Rational Software Architect and the views it contains.

The Modeling Perspective

The Modeling Perspective

Perspective: Defines the initial set of content editors and related views and their layout.



11

IBM

The Software Architect workbench window contains one or more perspectives. A **perspective** defines the initial set of content editors and related views (used to navigate information in the editor) and their layout in the workbench window.

Perspectives gather together sets of features for specific development activities. For example, the Java perspective gathers tools for writing code—the Package Explorer, Problems view, Outline view, and Java editor.

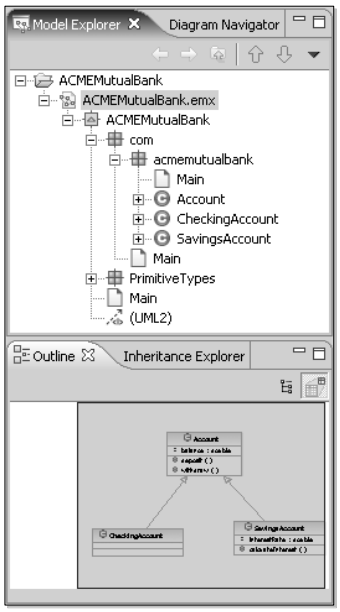
The **Modeling** Perspective contains the tools needed for building UML models and diagrams, including the diagram editor, with the following views: Model Explorer view, Pattern Explorer, Outline view, Inheritance Explorer, Properties view, Tasks view, Output window, and Bookmarks view.

Views

Views

Views in the **Modeling** Perspective:

- For exploring modeling projects:
  - Model Explorer view
  - Diagram Navigator
- For exploring resources:
  - Pattern Explorer view
  - Outline view
  - Inheritance Explorer view
  - Properties view
- For getting feedback:
  - Tasks view
  - Output window



12

IBM

Views are windows that provide different ways to navigate the information in the workspace and in the workbench. For example, the Model Explorer view displays projects and models you are working with. A diagram in the Model Explorer view can be opened into the diagram editor. The properties of the selected diagram can be modified in the Properties view.

A view might appear by itself or stacked with other views in a tabbed notebook. You can change the layout of a perspective by opening and closing views and by docking them in different positions in the Workbench window.

Views in the Modeling Perspective

Where Are We?

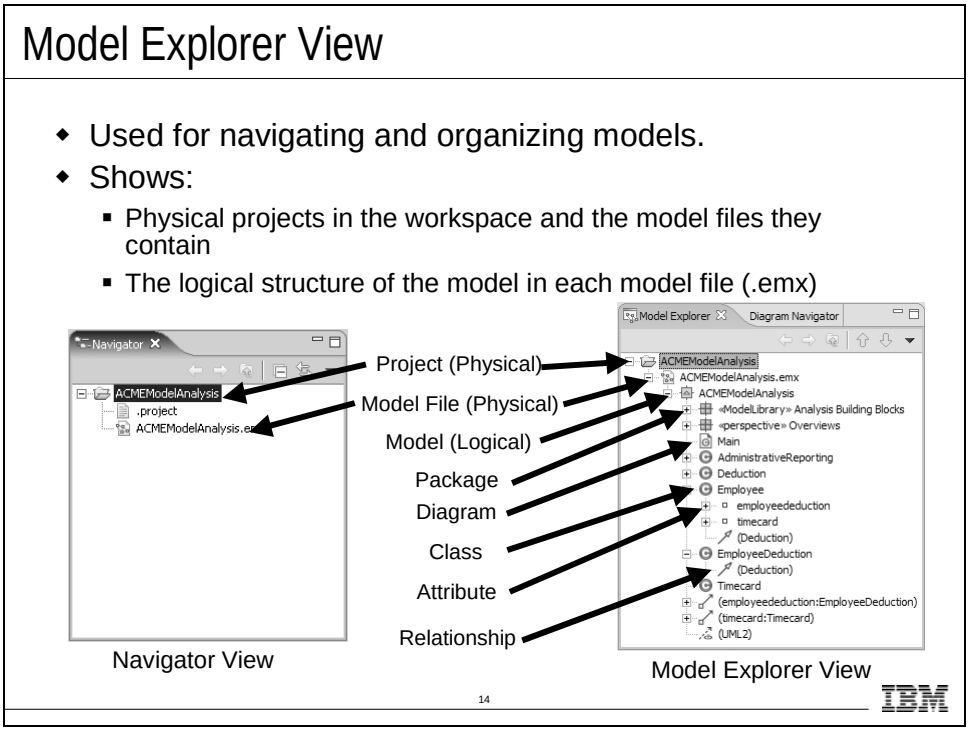
- Introduction to Rational Software Architect
- The Modeling Perspective
- **Views in the Modeling Perspective**
- Software Architect Projects

13

IBM

This section highlights the editors and views that are unique to the **Modeling** Perspective.

Model Explorer View

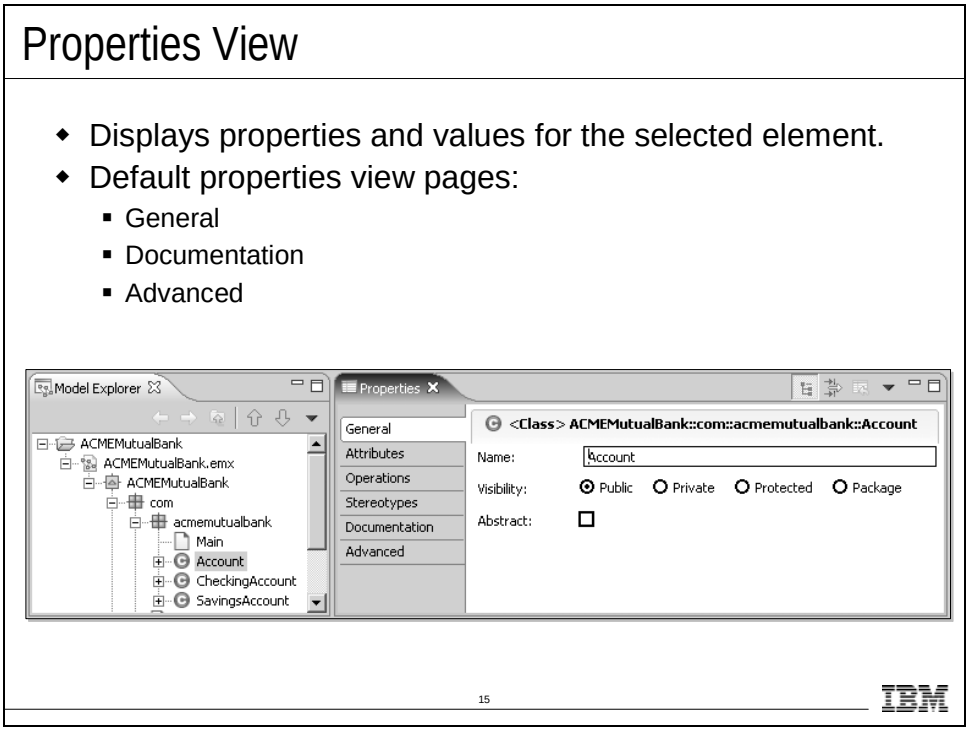


The Model Explorer view shows all the model (.emx) files in the project, along with the projects and file structures they reside in.

Under each model file, the Model Explorer view displays the logical structure of the model, including the name of the model as well as any packages, diagrams, relationships, and other model elements it contains. The model icon (light brown, containing a triangle) inside the model file allows you to add model elements at the root level of the model, and it allows you to set certain properties for the model.

You can use the Model Explorer view to add, delete, move, organize, and sort model elements for each model in your project.

Properties View

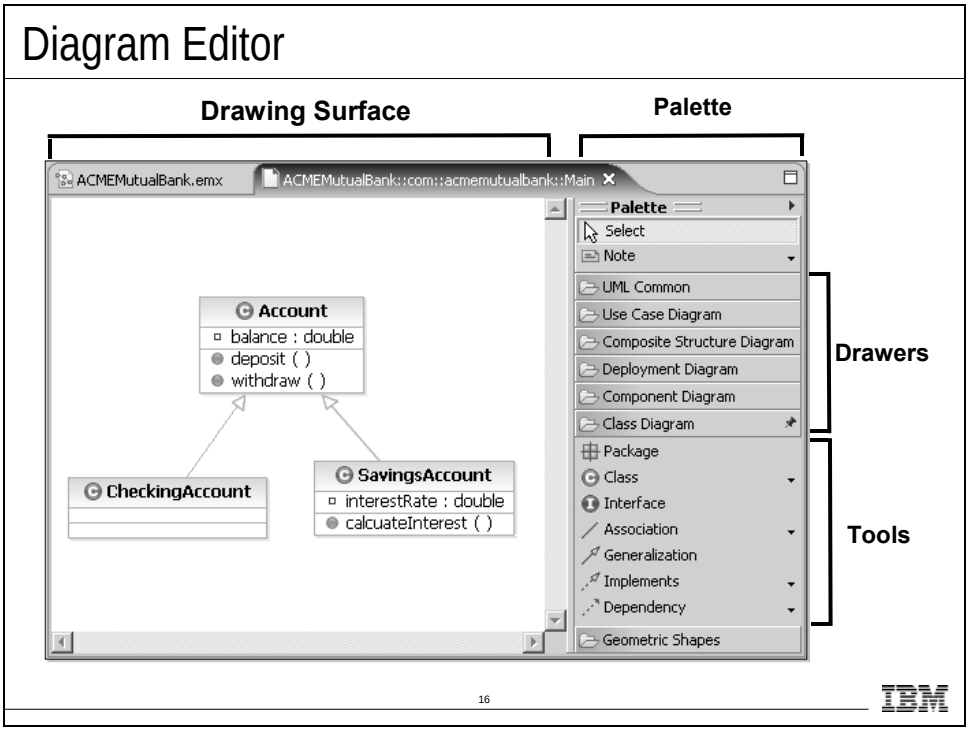


All models, diagrams, model elements, and relationships have associated properties, just like all other project resources.

In the **Modeling** Perspective, properties are organized into a series of tabbed categories. Most model elements contain the following categories:

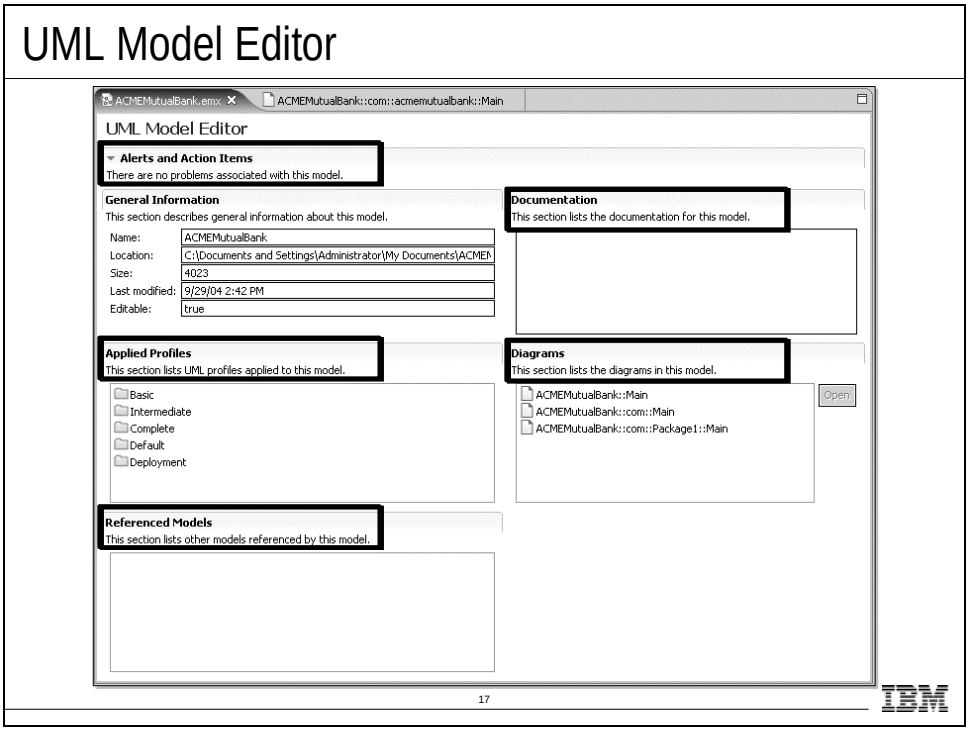
- **General:** Includes the name of the elements, qualified name, keywords for searches, visibility, and an option to make the element abstract.
- **Documentation:** Used for adding documentation to model elements.
- **Advanced:** Displays an Eclipse-style properties page.

Diagram Editor



The diagram editor contains both the drawing surface and the toolbox. The toolbox contains a basic set of tools (available by default) for selecting and annotating shapes in the diagram, along with one or more drawers of tools for various diagram types. You can customize the look and behavior of drawers in the toolbox, as well as create and customize your own drawers.

UML Model Editor



The UML Model editor appears every time you open a model. To close your model, close the UML Model editor. From the UML Model editor, you can add documentation to the model and navigate to specific diagrams in the model. It also provides the following model information:

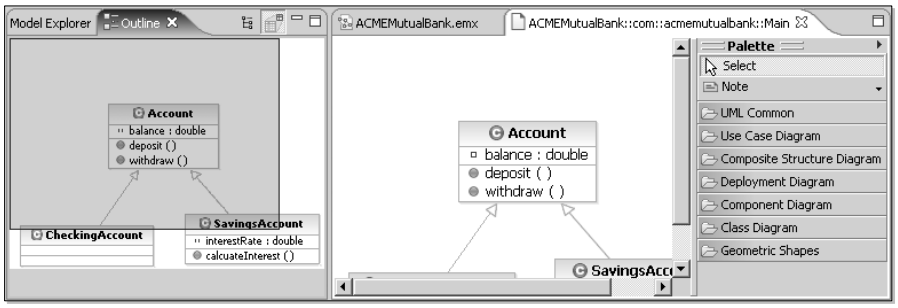
- **General Information:** Provides general model file properties, including name, location in the workspace, size of the file, modify date, and read or write status.
- **Applied Profiles:** Shows which UML profiles are currently applied to the model. Applying profiles makes available sets of stereotypes that extend the UML to create specialized models.
- **Referenced Models:** Shows which models the open model references. A referenced model contains model information from a referenced component or project.
- **Documentation:** Allows you to add documentation at the model level, just the way you would document an element in a model.
- **Diagrams:** Allows you to navigate directly to any diagram in the model and open it in the diagram editor.

Outline View

Outline View

When the diagram editor is open, the Outline view shows:

- A thumbnail view of the diagram
- A list of elements displayed in the diagram



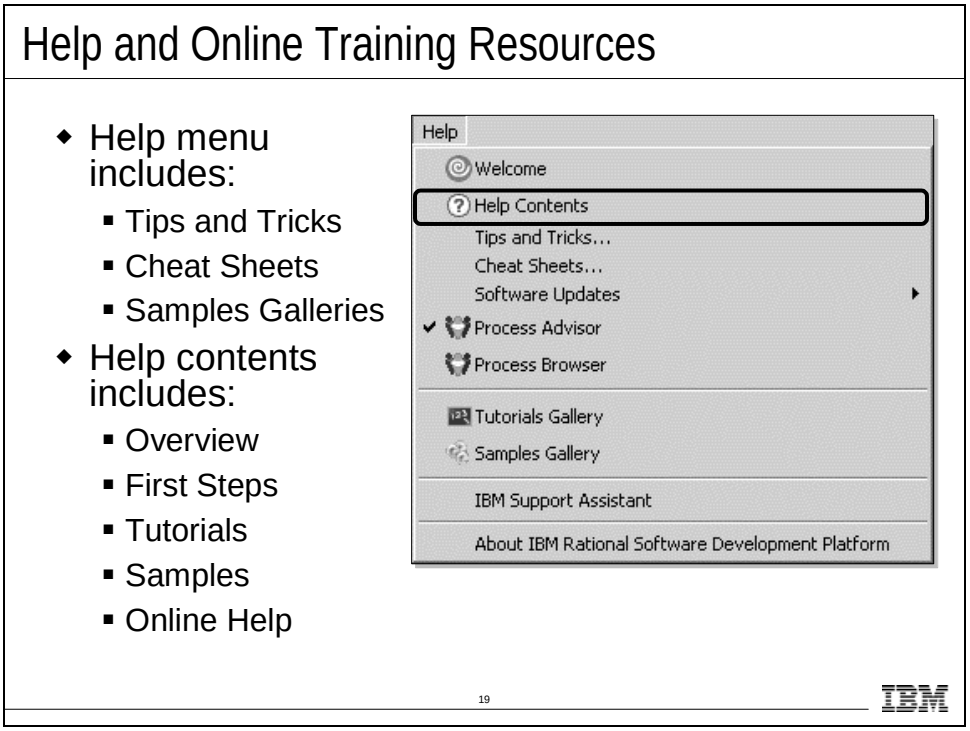
18

IBM

The Outline view displays an outline of a structured file that is currently open in the editor area and lists structural elements.

The contents of the Outline view are editor-specific. When the diagram editor is open, the Outline view can show either a tree view of the elements displayed in the diagram or provide a thumbnail of the diagram with the screen area shown in gray. You can move the active screen area in the outline view to navigate the diagram. The tree view displays an outline of a structured file that is currently open in the editor area and lists structural elements.

Help and Online Training Resources



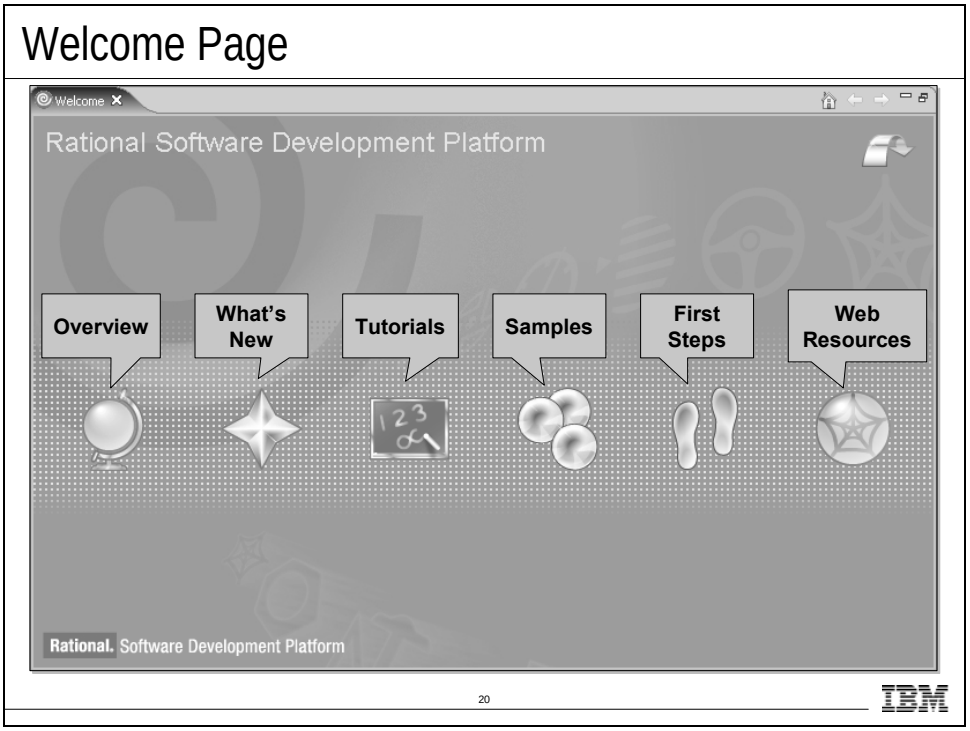
The help resources built into the Software Architect development environment include:

- **Tips and Tricks:** A list of tips for advanced users.
- **Cheat Sheets:** Step-by-step instructions for common tasks in Software Architect. Opens a Cheat Sheets view in the **Modeling** Perspective.
- **Samples Galleries:** A wide range of sample applications featuring different technologies.

From the help contents, you can find:

- **Overview:** Presents a series of animated tours of Software Architect, including presentations on modeling, developing code, best practices, and the Software Architect environment.
- **First Steps:** Provides step-by-step guidance for building an application with Software Architect.
- **Tutorials:** Provides in-depth instructions on various topics in three modes:
  - **Watch and Learn:** Provides feature demonstrations
  - **Play and Learn:** Provides interactive feature simulations
  - **Do and Learn:** Provides hands-on instructions for learning to do various tasks in Software Architect.
- **Samples:** Provides a gallery of UML models, patterns, scripts, and transforms you can use as the basis for your own projects and resources.
- **Online Help:** Contains a "tips and tricks" collection and a collection of reference "cheat sheets." We should also mention cheat sheets from the Help menu. This may require an additional slide.

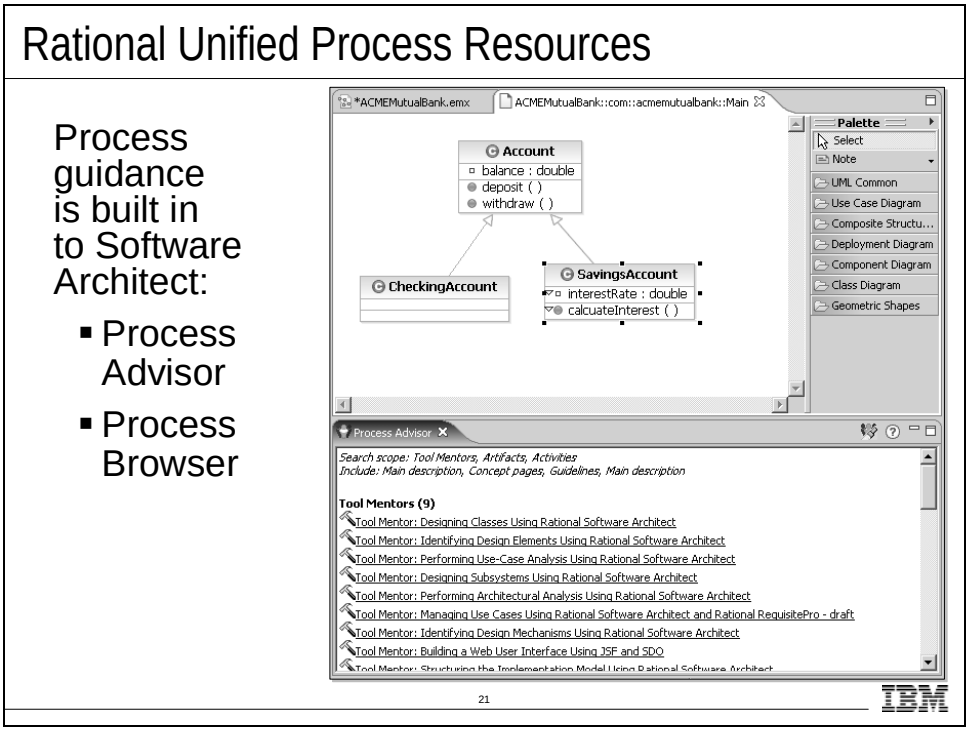
Welcome Page



The Welcome page offers the following resources on the Rational Software Development Platform (supporting Rational Software Architect and Rational Application Developer for WebSphere):

- **Overview:** A video tour of the development platform and descriptions of how to use the platform for different types of development.
- **What's New:** Descriptions of what new features have been added in the current version.
- **Tutorials:** A large gallery of tutorials of three types: "Watch and Learn," "Play and Learn," and "Do and Learn."
- **Samples:** Three types of code samples: simple application samples, applications based on specific technologies, and large and extensive showcase samples.
- **First Steps:** Instructions for getting started with various project types.
- **Web Resources:** A wide variety of IBM Rational brand Web resources.

Rational Unified Process Resources



The IBM Rational Unified Process®, or RUP®, is a software engineering process and an industry-wide platform for best practices.

There is a configuration of the RUP designed especially for Software Architect users, which is accessible inside Software Architect in two ways:

Process Advisor: The Process Advisor is a view that presents context-sensitive references to Rational Unified Process Tool Mentors, Artifacts, and Activities. From this view you can also search the RUP and access Help.

Process Browser: The RUP Process Browser in Rational Software Architect is a separate window that opens to display the RUP content selected in the Process Advisor view and also allows you to browse the RUP.

Software Architect Projects

Where Are We?

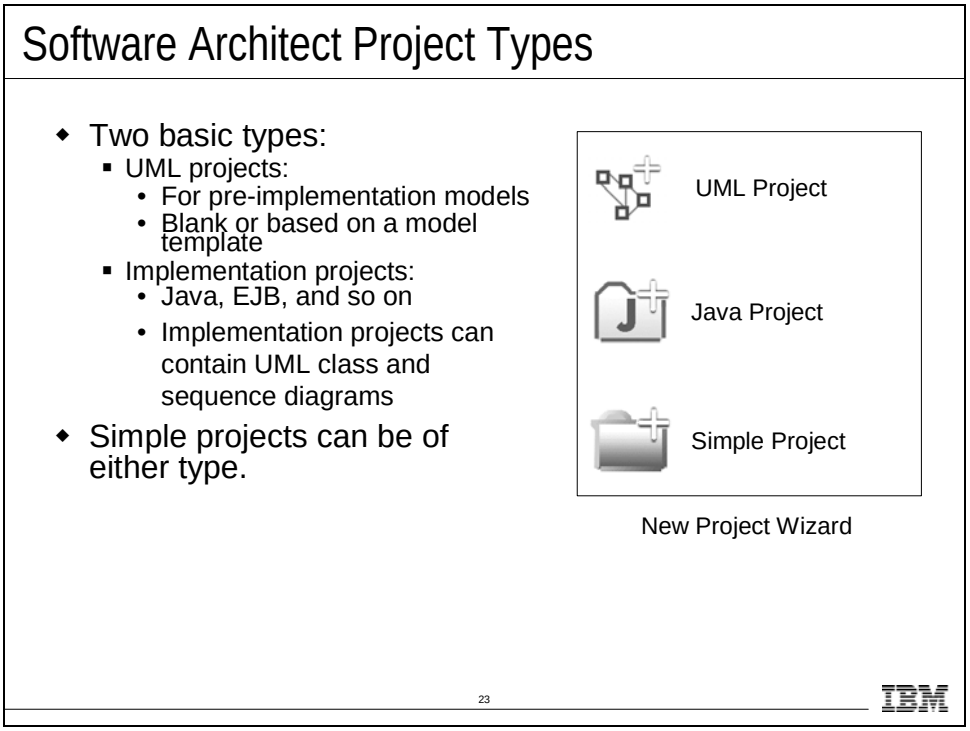
- Introduction to Rational Software Architect
- The Modeling Perspective
- Views in the Modeling Perspective
- **Software Architect Projects**

22

IBM

This section introduces the project types and model templates available in Software Architect.

Software Architect Project Types



A project in Software Architect is just a container for resources, such as model files, code files, HTML or XML files, and so on.

In Software Architect, you will develop two types of projects, **UML projects** for pre-implementation models and **implementation projects**, in which the code is the implementation model. Implementation projects can contain free-standing UML class and sequence diagrams, used to visualize the code's structure and runtime behavior.

A **Simple** project is an empty project folder, which you can populate with any file structures and resources.

A **UML project** is a Simple project that contains a model file. Models can be created either blank or based on a Software Architect UML model template.

The following implementation projects are available in Software Architect:

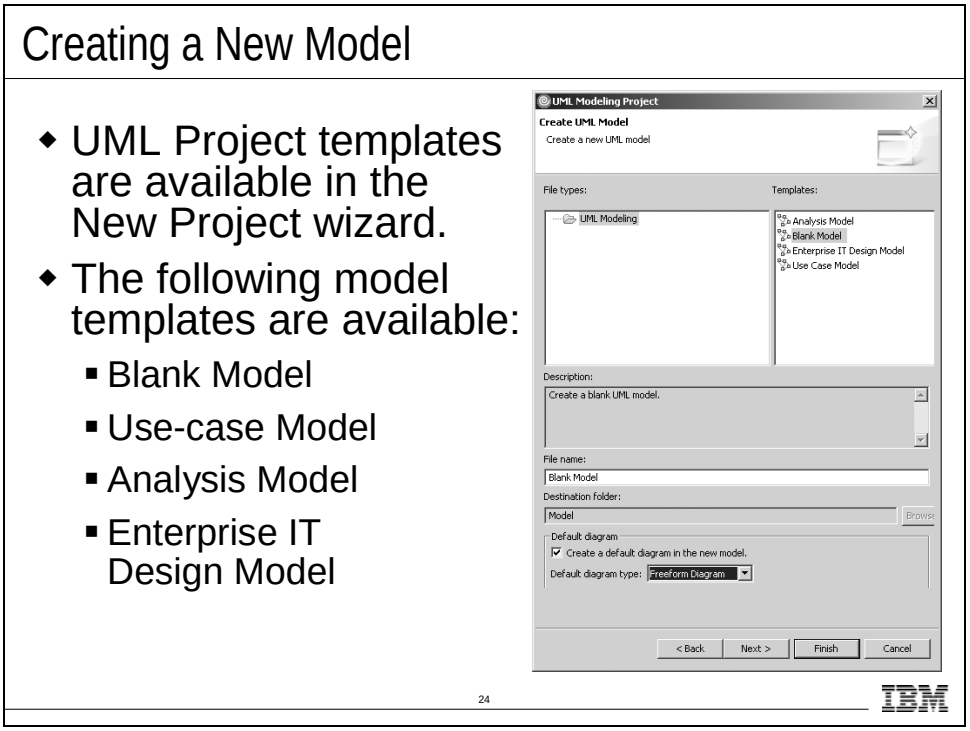
Enterprise: An enterprise application project contains the hierarchy of resources required to deploy a J2EE enterprise application, often referred to as an EAR file.

EJB: EJB projects contain the resources contained in an EJB module.

Web: Web projects are used for creating dynamic Web applications and static Web sites. Dynamic Web projects provide an accompanying EAR project.

Connector: A connector is a J2EE standard extension mechanism for containers to provide connectivity to enterprise information systems (EISs).

Creating a New Model



In Software Architect, model files are used to create **"pre-Implementation" models**, they contain conceptual UML content that does not *directly* reflect implementation artifacts. They present just enough detail so that the model can be easily understood. Model files contain both the UML semantics of the model and UML diagrams that depict the semantic elements.

When creating a UML Modeling project, a page in the New Project Wizard offers the opportunity to select a UML model template or blank model for the new project. Once the project has been created, you can add additional model resources or delete the original model resource if necessary.

Demo: Creating a Simple Model

Demo: Creating a Simple Model

- The instructor will now show you how to:
  - Create a UML project
  - Create a Blank model
  - Create a class diagram
  - Validate the diagram

25

IBM

Watch your instructor create a simple UML model.

Lab: Create a Simple Model

Lab: Create a Simple Model

Tasks:

- Create a new Simple project.
- Create a Blank model.
- Create a class diagram.
- Validate the diagram.

26

IBM

Complete the Getting Started lab.

Review

Review

- What is a “view” in Software Architect?
- What is a “perspective” in Software Architect?
- What project type is used for building models of the application before implementation?

27

IBM

Getting Started with Rational Software Architect

Objectives

After completing this lab, you will be able to:

- Create an UML project and model
- Create UML packages
- Create a class diagram
- Model a class diagram
- Validate a diagram

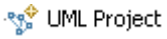
Scenario

In this lab, you will complete a simple class diagram to familiarize yourself with Rational Software Architect's basic diagramming features.

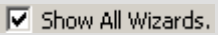
Task 1: Create a UML Project and Model

In this task, you will create a UML project containing a blank model.

a. On the menu, click **File > New > Project**.

b. In the Wizards list box, click the UML Project icon 

and then click **Next**.

TIP: Selecting the **Show all wizards** 

check box will display every Architect wizard.

c. Name the project `ACMEMutualBank` and click **Next**.

d. Complete the Create UML Model dialog with the following values and then click **Finish**:

- **Templates**: `Blank Model`
- **File name**: `ACMEMutualBank`
- Keep defaults for **Create default diagram in the new model**

TIP: If you close the UML editor window, your model will close. You can reopen a model by double-clicking the model file (`.emx`) in Model Explorer.

After you have created the UML project, your expanded Model Explorer view will appear as below:

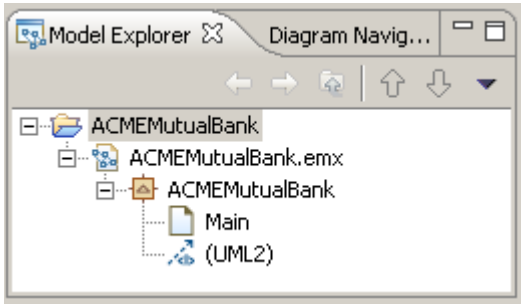
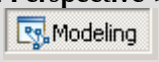


***Figure 1**: Model Explorer*

Task 2: Create a UML Package

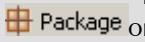
In this task, you will create a UML package for the ACMEMutualBank model.

a. Make sure you are in the **Modeling** Perspective and the `ACMEMutualBank` model is opened.

> *TIP:* Click **Window > Open Perspective > Modeling** to open the **Modeling** Perspective or click the Modeling icon 
>
> on the tool bar.

b. Create qualified UML packages:

c. Open the `Main` diagram at the root of the model, create a new package, and name it `com`.

> *TIP:* To create a package, open the **Class Diagram** drawer on the Palette and drag a package 
>
> on to the diagram.

d. Open the `Main` diagram in the `com` package.

> *TIP:* You can open the `Main` diagram in the diagram editor by double-clicking the package tab in the top left corner of the `com` package.

e. On the `Main` diagram of the `com` package, create a new package and name it `acmemutualbank`.

After you have created the UML packages, your expanded navigator will appear as in Figure 2.

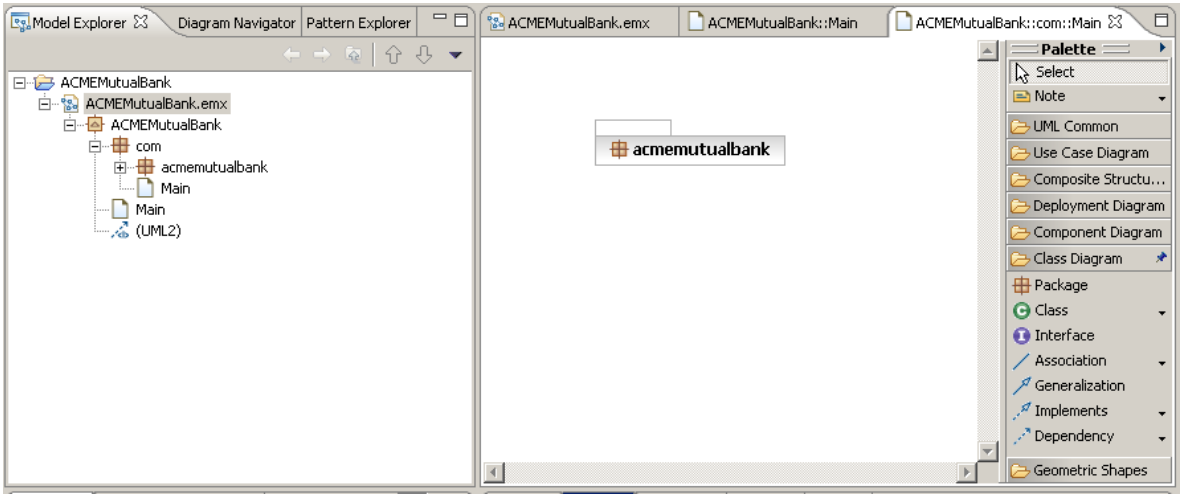


***Figure 2**: Qualified package in Model Explorer*

Task 3: Model Classes

In this task, you will model classes in the ACMEMutualBank model.

a. If it is not already open, open the `Main` diagram in the `acmemutualbank` package in the Diagram window.

b. In the `Main` diagram, create and model the `Account` class:

c. Add a new class to the `Main` diagram in the `com.acmemutualbank` package and name it `Account`.

TIP: In the Toolbox, click the **Class Diagram** drawer, click the **Class** tool Class, and then click inside the diagram to draw the class. Name the class `Account`.

d. Add an attribute to the `Account` class:

```
-balance : double
```

TIP: Right-click the `Account` class in the diagram and click **Add UML > Attribute** Attribute. Enter the attribute using this UML notation: `-balance : double`.

e. Validate the attribute is private.

TIP: The properties view may be used to validate the operation's signature by clicking the balance balance attribute and then clicking the properties view general tab General as shown in Figure 3.

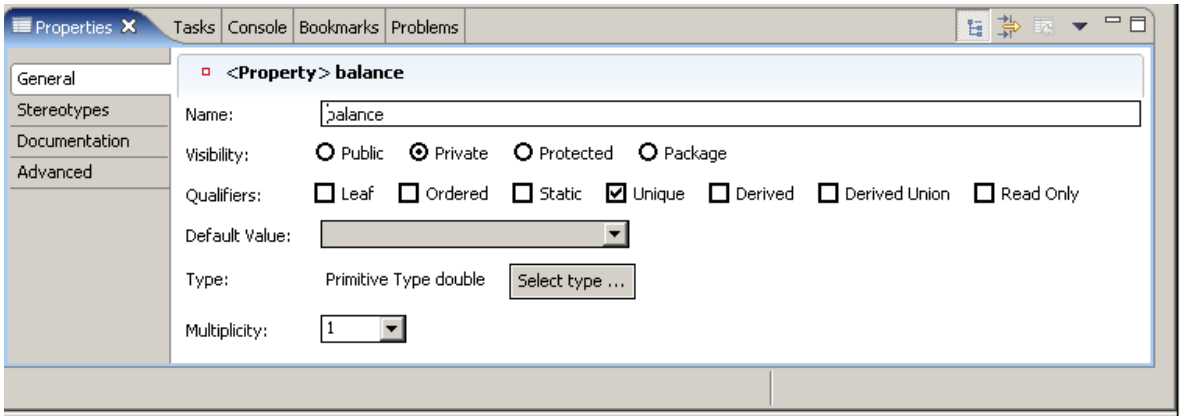


***Figure 3**: General properties for balance attribute*

f. Add the following operations to the `Account` class:

```
deposit(amount: double) : double
withdraw (amount: double) : double
```

TIP: In the diagram window, right-click the Account class and click **Add UML > Operation** Operation. Enter the operation signature using its UML notation method name(var name : var type) : returnType.

g. In the `Main` diagram of the `acmemutualbank` package, create and model the `SavingsAccount` class:

TIP: You can rename a class attribute or operation in the diagram window by clicking the attribute or operation in the class to select it and then clicking it again to make it editable.

h. Add a new class to the diagram and name it `SavingsAccount`.

i. Add an attribute to the `SavingsAccount` class:

```
-interestRate : double
```

j. Add the following operation to the `SavingsAccount` class:

```
calculateInterest() : double
```

k. Create a generalization relationship from the `SavingsAccount` class to the `Account` class (parent).

TIP: In the Toolbox, click the **Class Diagram** drawer, click the Generalization Generalization tool, click and drag the mouse on the `SavingsAccount` class to the `Account` class, and then click inside it. The arrow will then appear in the diagram.

l. Add a new class to the `Main` diagram in the `com.acmemutualbank` package and name it `CheckingAccount`.

m. Create a generalization relationship from the `CheckingAccount` class to the `Account` class (parent).

n. Select both generalizations (by clicking each one while holding the CRTL key) and then right-click the generalizations and click **Format > Line Style > Tree Style Routing**.

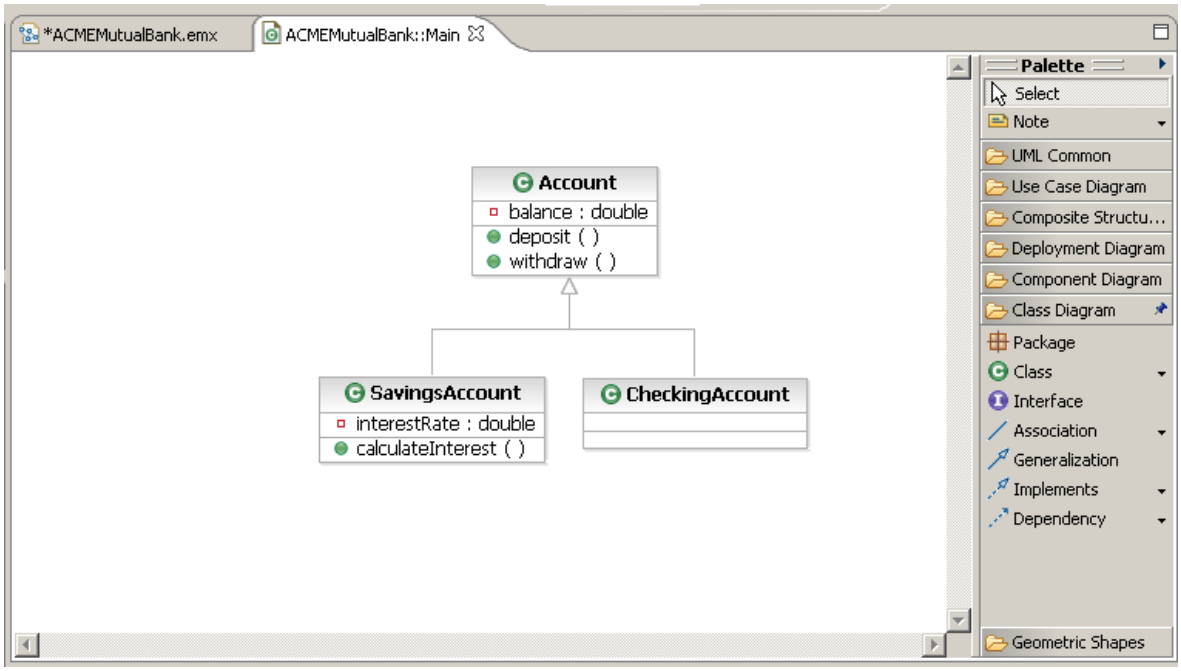


***Figure 4**: ACMEMutualBank class diagram*

o. Validate the diagram by right-clicking inside the `Main` class diagram and then clicking **Run Validation**. The results will be displayed in the Console view.

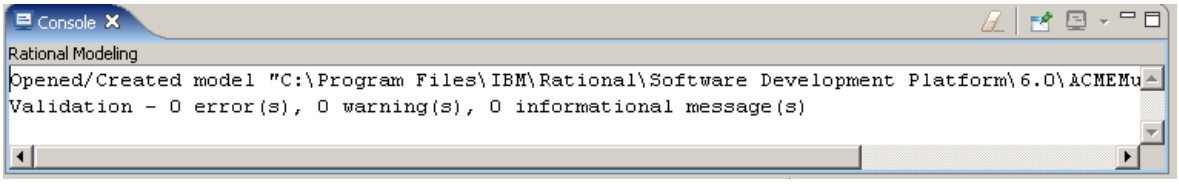
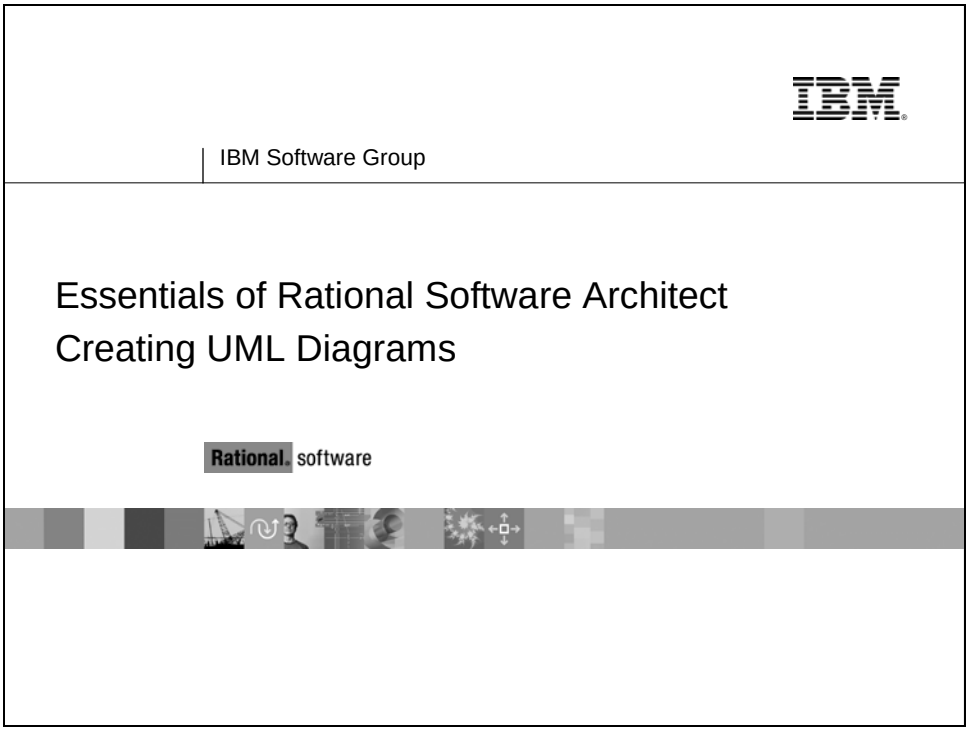


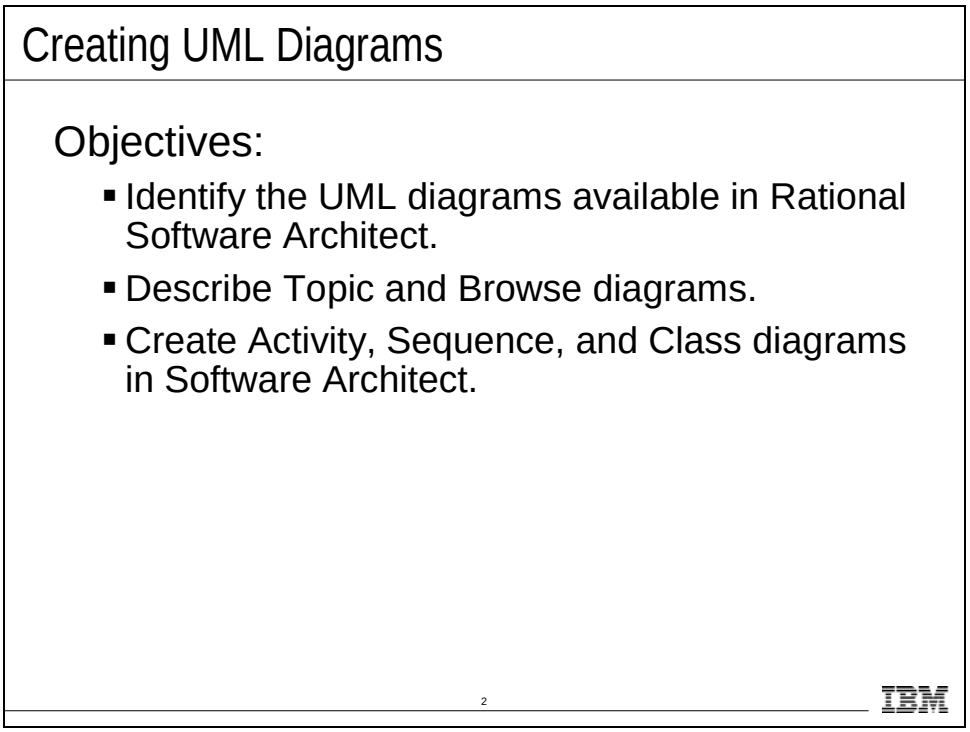
***Figure 5**: Model validation*

▸ ▸ ▸ Creating UML Diagrams



Topics

Module Objectives



This module provides an overview of the UML diagram types available in Rational Software Architect, with examples of the most commonly used diagrams, and explains how to use the diagram editor. The module also discusses model templates, which are used to create models that are structured according to the Software Architect model structure guidelines.

In the lab at the end of this module, you will add model elements and diagrams to an existing design model.

UML Diagrams

Where Are We?

- **UML Diagrams**
- Drawing UML Diagrams
- UML Model Templates

3

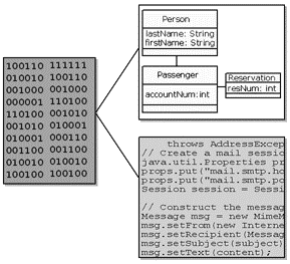
IBM

This section introduces the model templates built into Rational Software Architect for use in UML projects and describes some of the key features of each template.

What is a Model?

What is a Model?

- A model is a semantically closed abstraction of a subject system.
  - A model is defined in RUP as “a complete description of a system from a particular perspective.”
- Examples of models:
  - UML model
  - Code
  - Data model



4

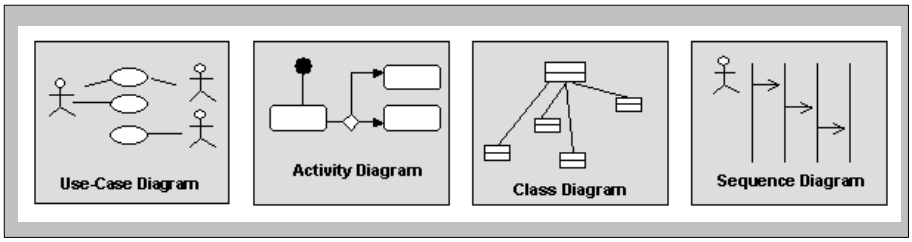
IBM

In the Rational Unified Process (RUP), a model is defined as a complete specification of a problem or solution domain from a particular perspective. Each model is *complete* in the sense that you do not need any additional information to understand the system from that perspective. A problem domain or a system may be specified by a number of models that represent different perspectives for different project stakeholders.

Diagrams

Diagrams

- Diagrams graphically depict a view of a part of your model.
- Different diagrams represent different views of the system that you are developing.
- A model element will appear on one or more diagrams.

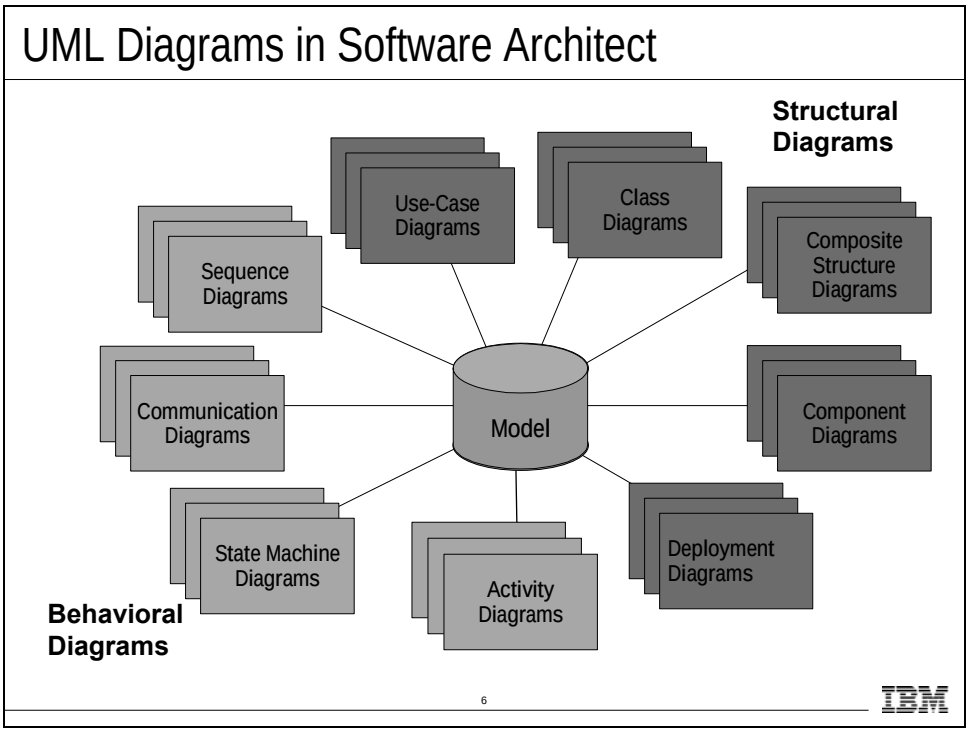


5

IBM

Diagrams provide a means of visualizing and manipulating the model elements.

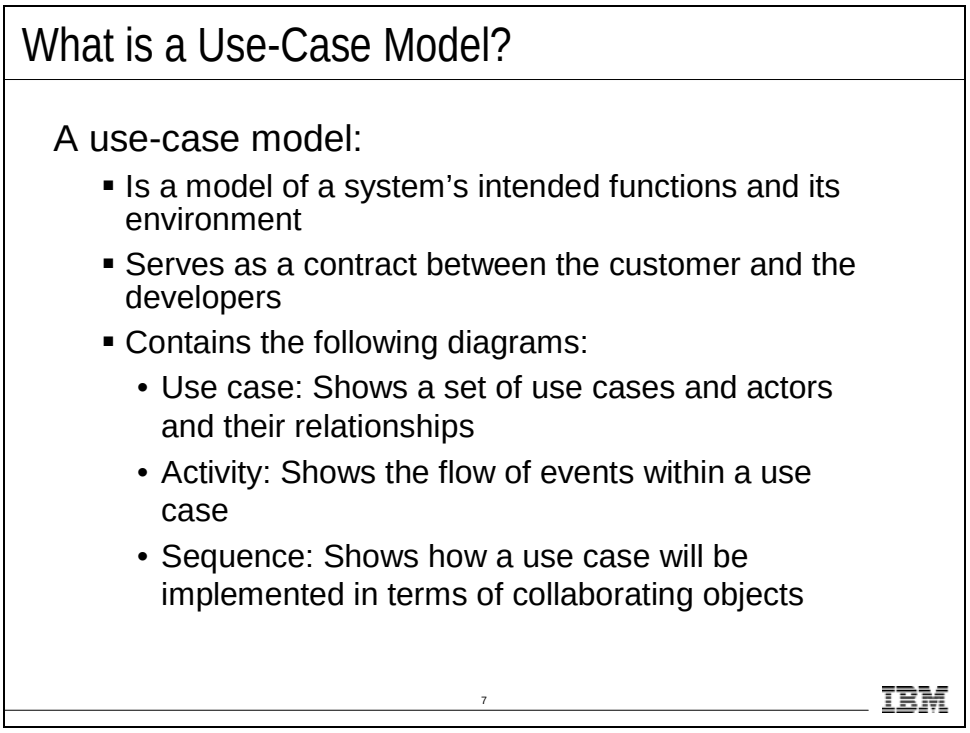
UML Diagrams in Software Architect



Visual models of a system require many different diagrams to represent different views of the system for different project stakeholders. The UML provides a rich notation for visualizing models, including the following key diagrams:

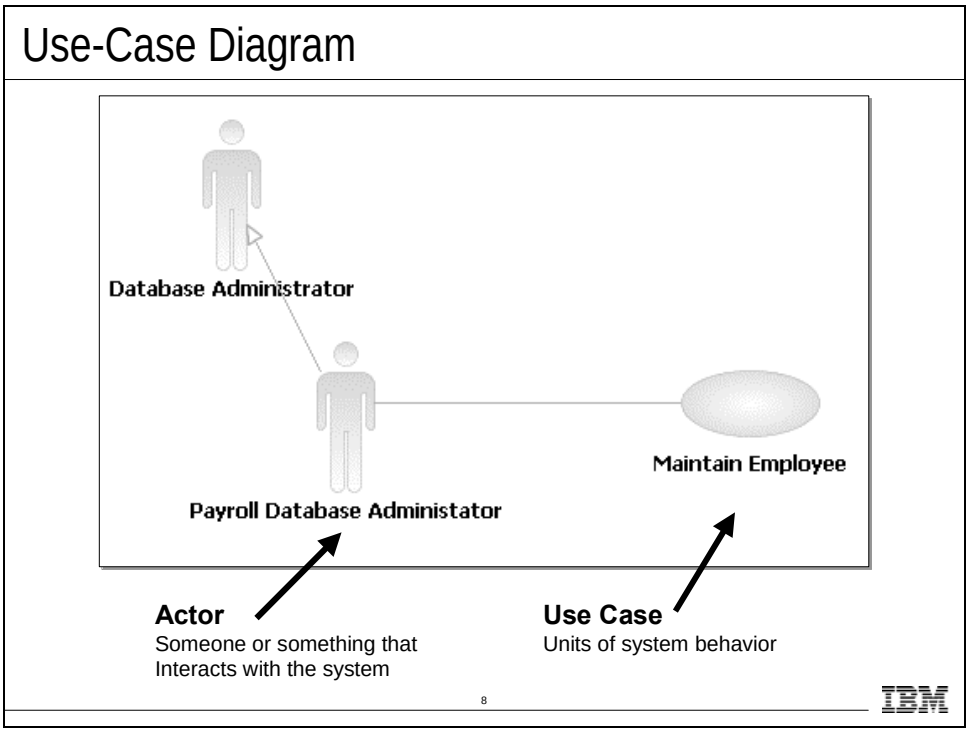
- **Use-Case diagrams** to illustrate user interactions with the system
- **Class diagrams** to illustrate logical structure
- **Composite Structure diagrams** to show the internal structure of a class or component at runtime
- **Component diagrams** to illustrate the organization and dependencies among modular parts of the system
- **Deployment diagrams** to show the mapping of software to hardware configurations
- **Activity diagrams** to illustrate flows of events
- **State Machine diagrams** to illustrate the series of states an object can have
- **Communication diagrams** to illustrate behavior in terms of how objects interact
- **Sequence diagrams** to illustrate behavior in terms of the sequence of interactions between objects

What is a Use-Case Model?



In the Rational Unified Process (RUP), a model is defined as a complete specification of a problem or solution domain from a particular perspective. Each model is *complete* in the sense that you do not need any additional information to understand the system from that perspective. A problem domain or a system may be specified by a number of models that represent different perspectives for different project stakeholders.

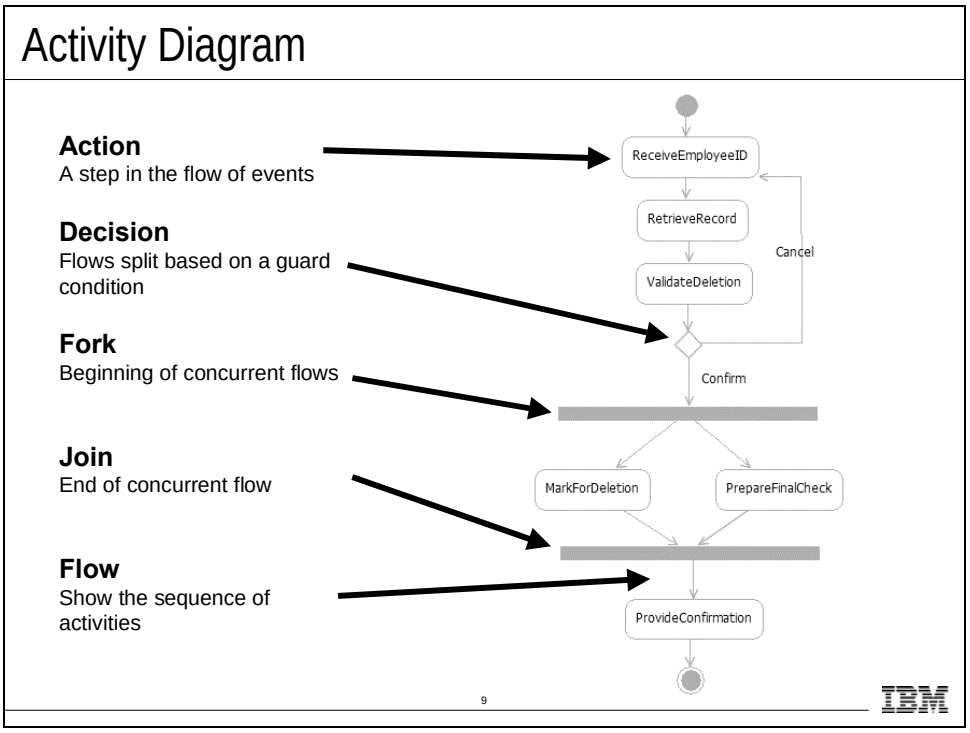
Use-Case Diagram



The use-case model contains the following elements, which are displayed in a use-case diagram:

- **Actors:** Used to represent someone or something outside the system that interacts with the system.
- **Use Case:** Used to represent a unit of system behavior that comprises several possible sequences of actions.

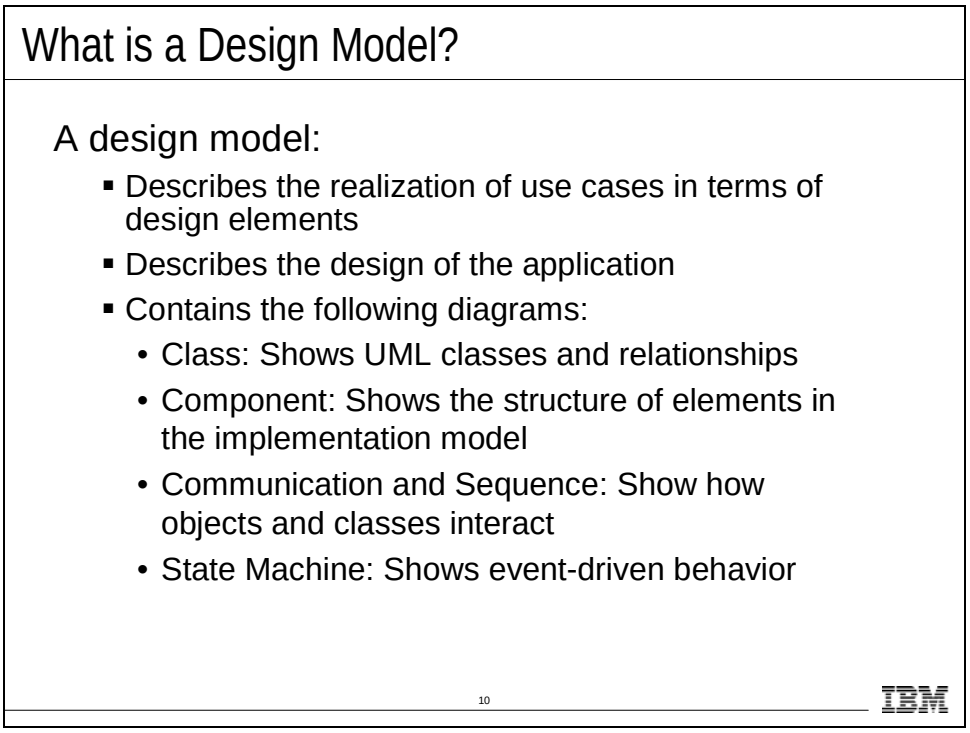
Activity Diagram



Activities describe graphically the flow of events of a use case. The flow of events consists of a sequence of activities that together produce something of value for the actor. The flow of events consists of a basic flow and one or several alternative flows.

- **Actions:** Represent the performance of an activity or step within the flow of events.
- **Flow/Edge:** Show what activity state follows after another.
- **Decision/Merge** Control which flow (of a set of alternative flows) follows once the activity has been completed, based on a guard condition. Decisions are used to show alternative threads in the flow of events of a use case.
- **Forks/Joins:** Show the beginnings and ends of parallel subflows. Forks and joins are used to show concurrent threads in the flow of events of a use case.

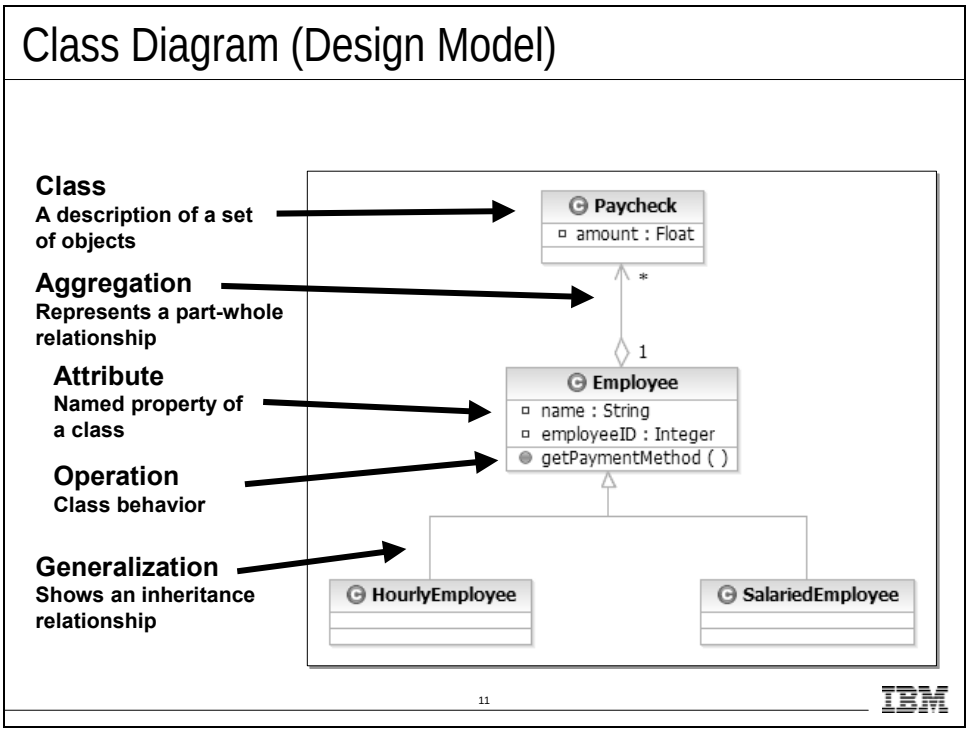
What is a Design Model?



The design model may include the following diagrams:

- **Class diagrams:** Show the classes, their internal structure, and their relationships to other classes. May also show the package structure of the design model (package diagram).
- **Composite Structure diagrams:** Show the runtime decomposition of a class.
- **Component diagrams:** Used, in the design model, to show the high-level structure of the implementation model.
- **State Machine diagrams:** Used to model the event-driven aspects of an object's behavior.
- **Interaction diagrams:** Used, in the design model, in use-case realizations to show how behavior is assigned to classes in the design model. The two interaction diagrams are semantically equivalent, so in Software Architect you can generate a sequence diagram under an interaction automatically from the interaction's communication diagram or vice versa.

Class Diagram (Design Model)

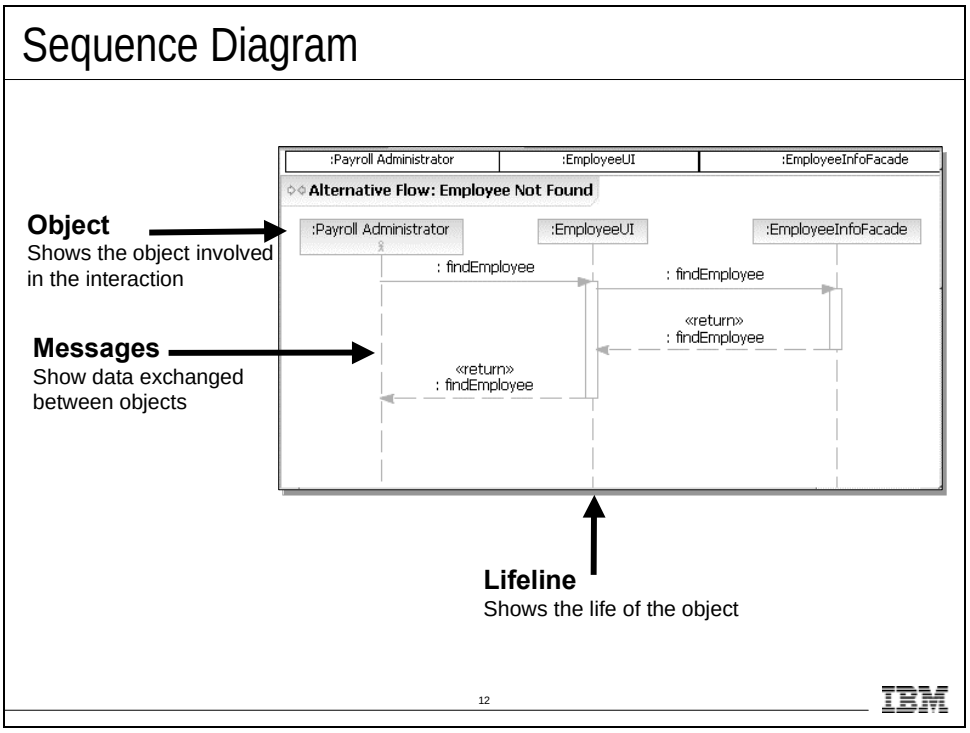


Class diagrams show the static structure of the model, in particular, its classes, their internal structure, and their relationships to other classes. Class diagrams do not show temporal information.

A class diagram typically displays the following elements and the relationships among them:

- **Classes:** Describe sets of objects that share the same responsibilities, relationships, operations, attributes, and semantics.
  - **Attributes:** Named properties of the class or its objects. An attribute also defines the type of its instances.
  - **Operations:** Services that can be requested (the behavior of the class).
- **Packages**: Can be included for the sake of context although UML 2.0 has brought package diagrams into the mainstream.

Sequence Diagram

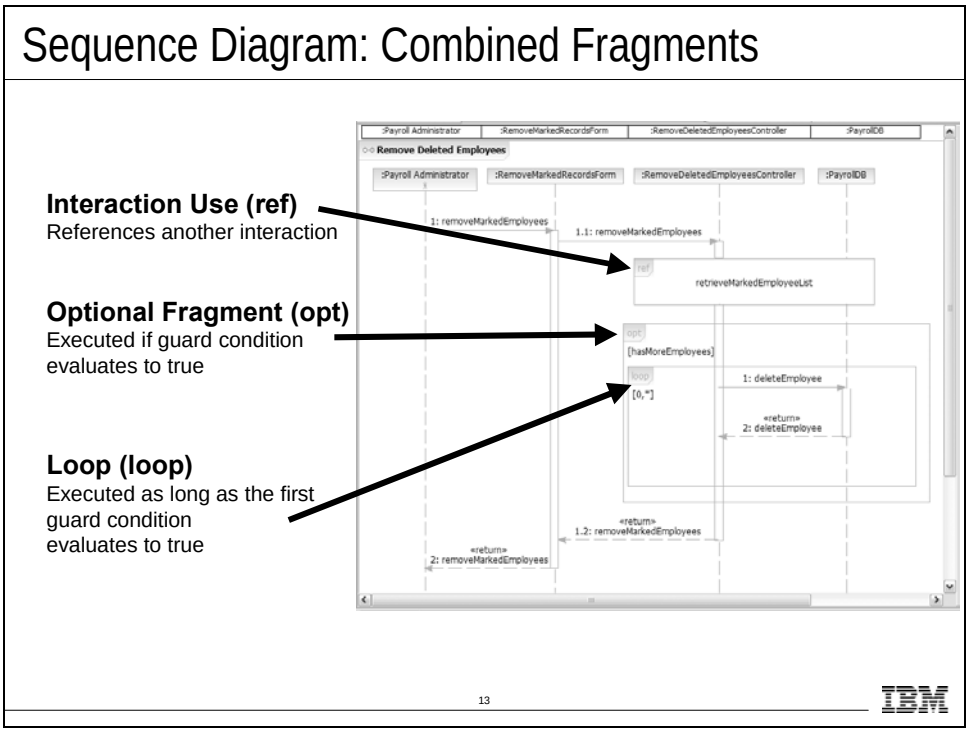


In the design model, sequence diagrams are used to show how objects interact to perform the behavior of all or part of a use case as part of a use-case realization. The sequence diagram clarifies the roles of objects in a flow of events, and thus it provides basic input for determining class responsibilities and interfaces.

Sequence diagrams display the following model elements:

- **Lifelines:** Represent the existence of the object at a particular time
- **Actors:** Can be included in the diagram when you want to show that an actor initiates the interaction
- **Messages:** Communicate between objects that conveys information with the expectation that activity will ensue

Sequence Diagram: Combined Fragments



To help with the modeling of, for example, multiple flows of events, which in the past always meant creating many interaction diagrams with only slight differences among them, you can use combined fragment notation—new with UML 2.0 and supported in Software Architect—which allows you to embed multiple paths through an interaction inside the same interaction diagram.

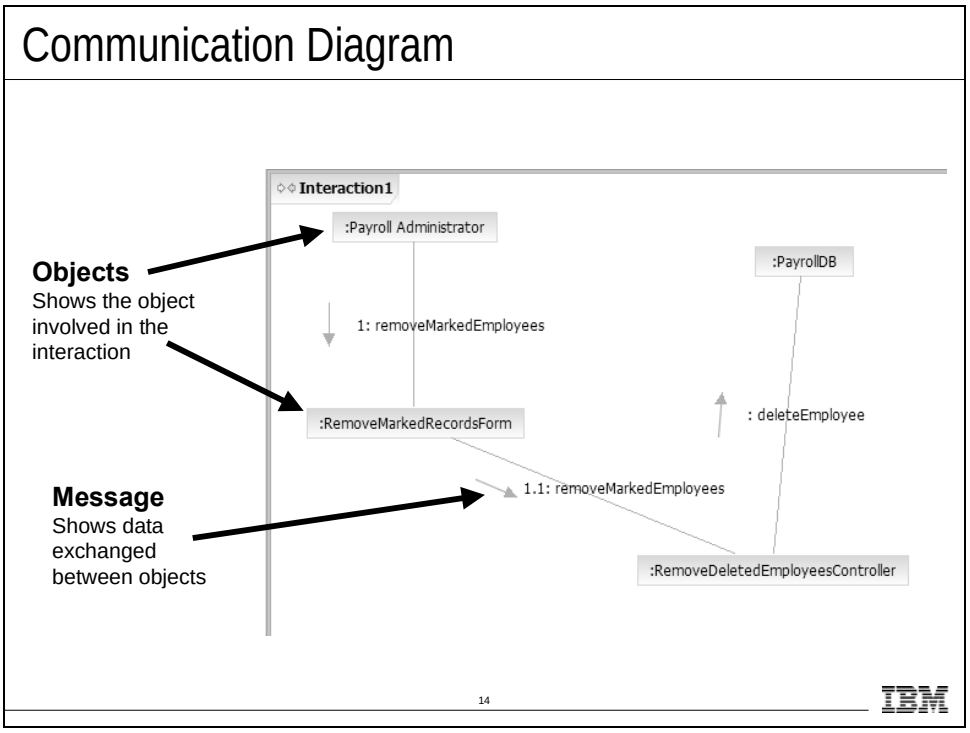
A combined fragment in a sequence diagram has an operator and one or more subinteractions in a nested region of the diagram. Gates may be defined on the combined fragment so that messages can be sent from the combined fragment to other parts of the main interaction.

Combined fragments have a keyword associated with them, shown on the frame surrounding the combined fragment. The more common combined fragment types include:

- **`loop`**: The loop fragment contains one subfragment that executes as long as a Boolean guard condition evaluates to `true`.
- **`ref`**: The combined fragment references another sequence diagram. In Software Architect, adding the `ref` operator from the toolbox automatically generates a new interaction instance in the model.
- **`opt`**: The opt fragment type contains one subfragment that executes only when the guard condition evaluates to true.

See Software Architect Help for discussion of other keywords.

Communication Diagram



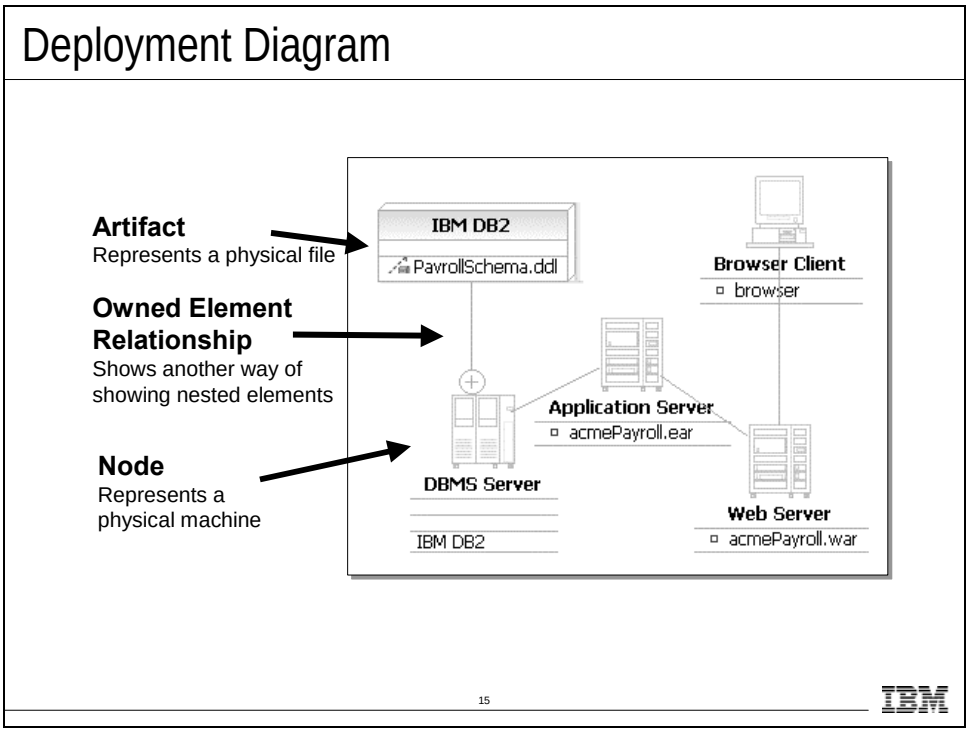
Communication diagrams provide another way to show how objects interact to perform the behavior of a particular use case or a part of a use case. Where sequence diagrams emphasize the interactions of objects over time, communication diagrams are designed to emphasize the relationships between objects.

- **Objects:** Objects show the name of the object and its class, separated by a colon:
  `objectname : classname`
- **Links:** Links are relationships among objects across which messages can be sent, shown as a solid line between two objects. A link can be an instance of an association, or it can be anonymous, meaning that its association is unspecified. Message flows are attached to links.

The diagram shown in the slide is based on the example sequence diagram from the previous slide. Two other elements that may appear in a communication diagram but do not appear in the example are:

- **Actors:** Actor instances show the invoker of the interaction.
- **Messages:** A message is a communication between objects that conveys information with the expectation that activity will ensue.

Deployment Diagram

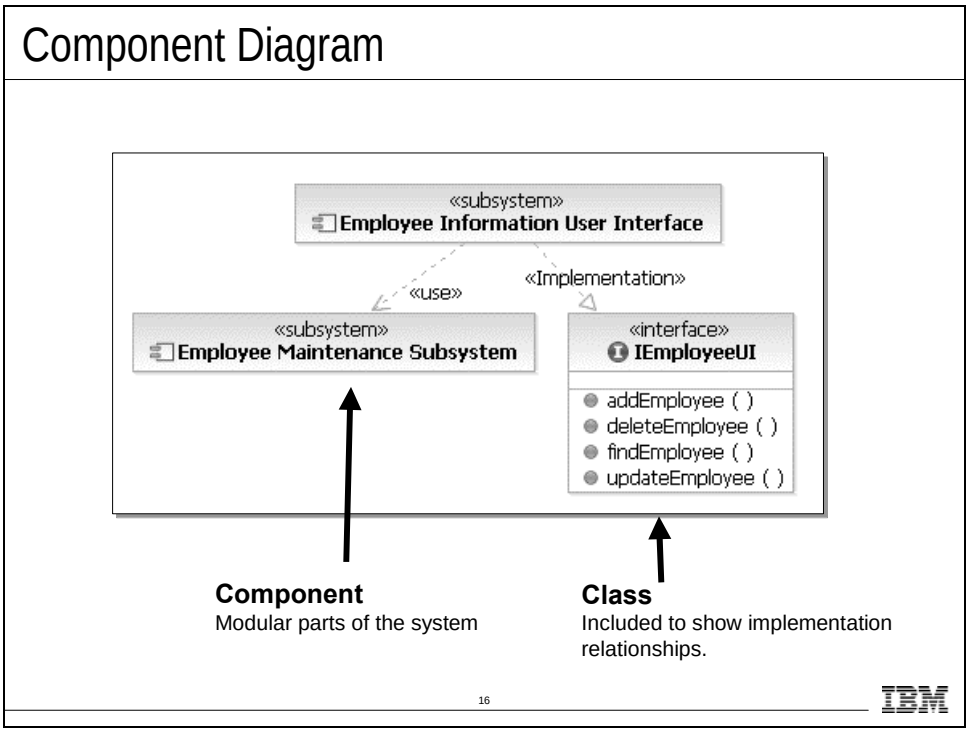


Deployment diagrams show the deployment architecture of the system, that is, which of the system's software artifacts reside on which pieces of hardware.

Deployment diagrams display the following model elements:

- **Artifacts:** Represent physical files (executable files, binaries, archives, and so on). In previous versions of UML, artifacts were called "components."
- **Nodes:** Represent any physical machine that can host software, such as a server or workstation. A node could also be used to model deployment to a virtual machine, for example, used as a test environment.

Component Diagram



Component diagrams show the runtime structure of the system at the level of software components. Components are the modular parts of the system and are made up of groups of related objects that are hidden behind an external interface.

- Component notation is based on class notation, so a component can have attributes and operations.
- Relationships among components can include connectors or dependencies.

Drawing UML Diagrams

Where Are We?

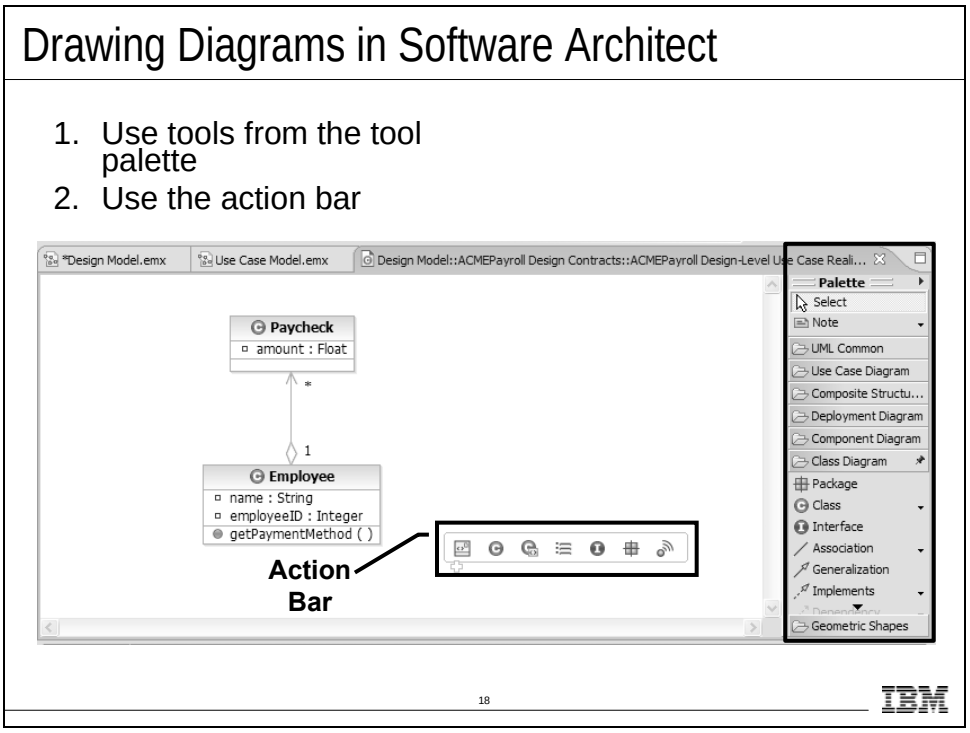
- UML Diagrams
- **Drawing UML Diagrams**
- UML Model Templates

17

IBM

This section examines the diagramming tools in the Software Architect **Modeling** Perspective: the diagram editor and the Diagram Navigator view.

Drawing Diagrams in Software Architect



The diagram editor window contains both the drawing surface and the palette. The palette contains a basic set of tools for selecting and annotating shapes in the diagram, along with one or more drawers. You can customize the look and behavior of drawers in the palette, as well as create and customize your own drawers.

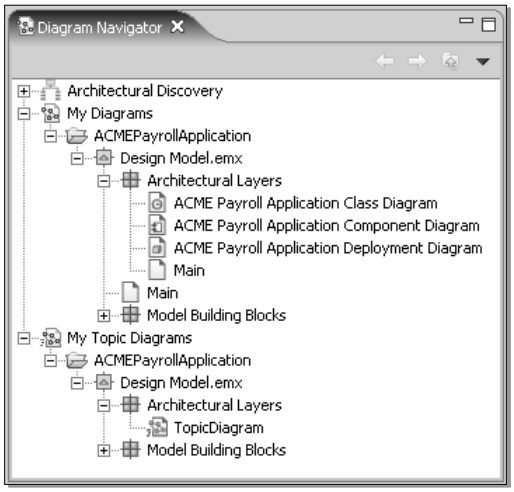
You can also add shapes to the model by right-clicking the diagram surface or by right-clicking a package in the Model Explorer and then clicking **Add UML** and selecting the appropriate shape.

Diagram Navigator

Diagram Navigator

Diagrams organized by type:

- My Diagrams
- My Topic Diagrams
- Architectural Discovery
  - Generated Topic diagrams



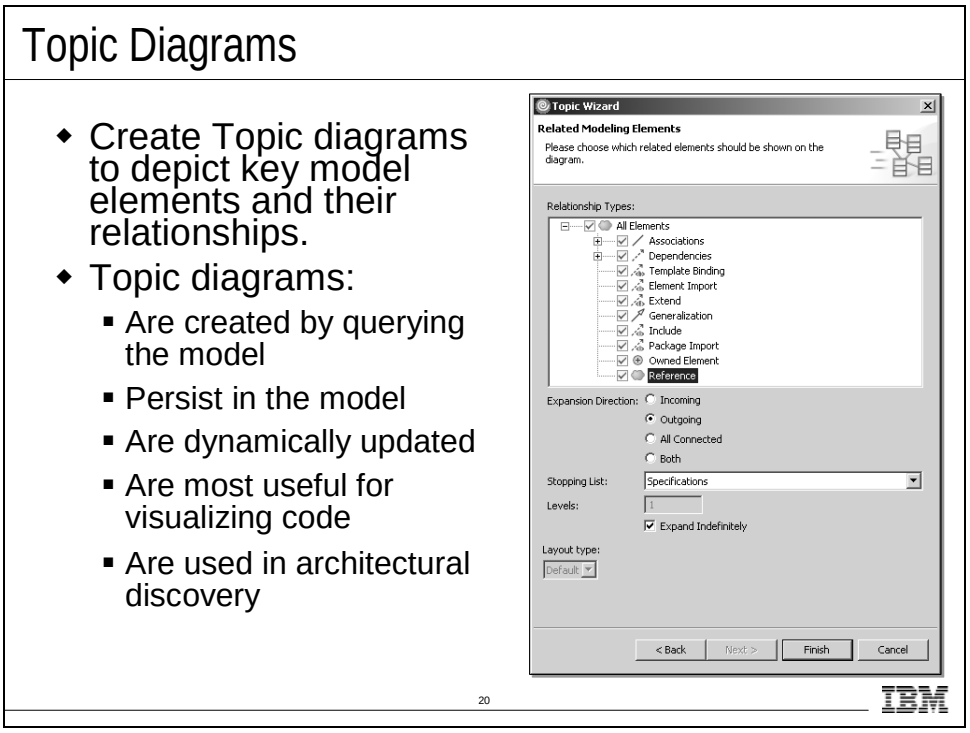
19

IBM

The Diagram Navigator shows the structure of all the diagrams in the workspace. As projects and model structures evolve and become more elaborate, the Diagram Navigator becomes useful for finding and opening the appropriate diagram quickly.

If you were new to an ongoing development project and had to examine an existing set of highly elaborated projects in Software Architect, you might use the Diagram Navigator to examine the high-level diagrams for each model to get familiar with key abstractions and their relationships.

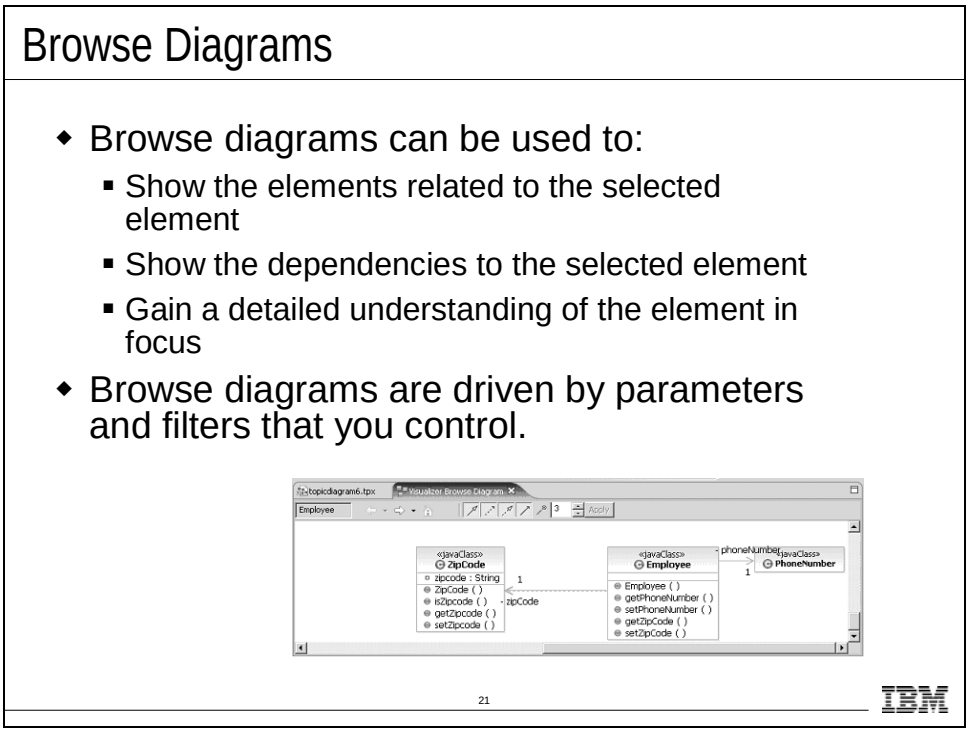
Topic Diagrams



Software Architect can generate and arrange topic diagrams for you automatically. These diagrams are useful when working in large models that contain many elements with complicated relationships. They are also invaluable when trying to help discover the architecture of an application.

Topic diagrams are used for both discovery and as visual documentation since they can be saved in the model. Developing a Topic diagram involves running a query against the existing model contents. You select a key model element and then define what other elements you want to appear in the diagram, based upon the types of relationships they have to the topical element. As the content of the model changes, topic diagrams adjust accordingly.

Browse Diagrams



Using browse diagrams to view elements in your project can help you to understand the relationships between elements in your code. There are times when you need to gain perspective and understand how an element is used and accessed by other objects.

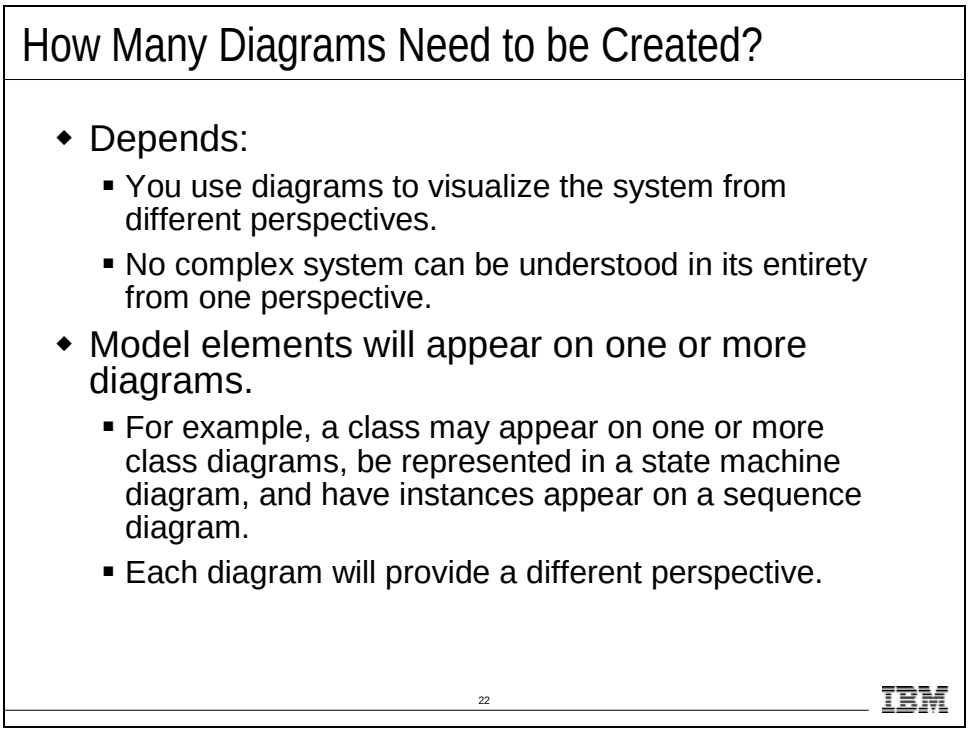
You can change the level of relationships shown in the browse diagram and use filters to choose specific relationship types. The layout of the browse diagram is based on a radial pattern with the selected element at the center of the dial.

Browse diagrams are not persisted by Software Architect. However, you can save browse diagrams as a class diagram, .gif, .jpeg, or .bmp file.

To create a browse diagram:

1. In the **Project Explorer**, right-click a Java element.
2. Select **Visualize** > **Explore in Browse diagram**.

How Many Diagrams Need to be Created?



Good diagrams make the system you are developing understandable and approachable. Choosing the right set of diagrams to model your system forces you to ask the right questions about your system and helps to illuminate the implications of your design.

Demo: Drawing UML Diagrams

Demo: Drawing UML Diagrams

The instructor will now show you how to:

- Add a diagram
- Add elements to a diagram using the tool palette
- Add elements or relationships using the action bar

23

IBM

Watch your instructor demonstrate how to create diagrams in Software Architect.

UML Diagram Templates

Where Are We?

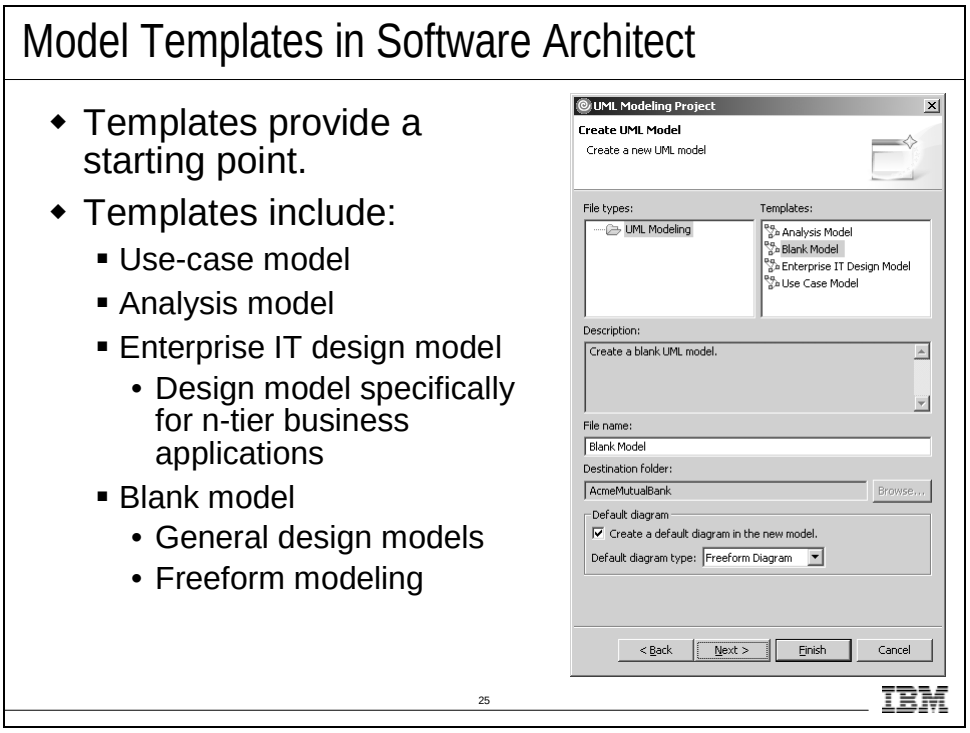
- UML Diagrams
- Drawing UML Diagrams
- **UML Diagram Templates**

24

IBM

This section introduces the model templates built into Rational Software Architect for use in UML projects and describes some of the key features of each template.

Model Templates in Software Architect

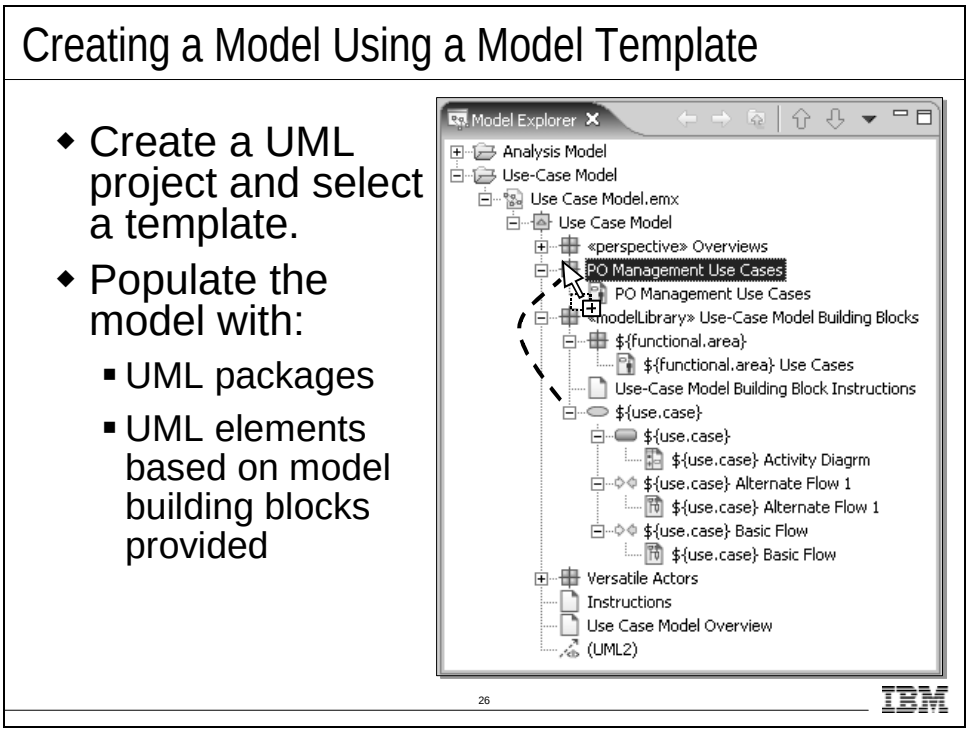


Software Architect provides the following model templates:

- **Blank model:** Includes no default content other than a Main freeform diagram and no profiles applied. The blank model template allows you to create your own model types and structures.

  In the design phase, you can also use the blank template to construct an **Implementation Overview** model to capture the high-level organization of the implementation or planned dependencies among implementation projects and packages. An Implementation Overview model in Software Architect corresponds with the Implementation model in the RUP.
- **Use-case model:** Includes blank model with default content (use cases, package structure, diagrams) for building a use-case model.
- **Analysis Model:** Contributes the Software Engineering Process profile, and some default content, such as packages and diagrams.
- **Enterprise IT design model (EITDM):** Contributes default content, including a profile that supports the modeling of services and transformations of designs into code. Use this template for design (and optionally for analysis) when targeting business applications. To create a design model for a simpler application, start with a blank model instead and add the appropriate package structure.

Creating a Model Using a Model Template



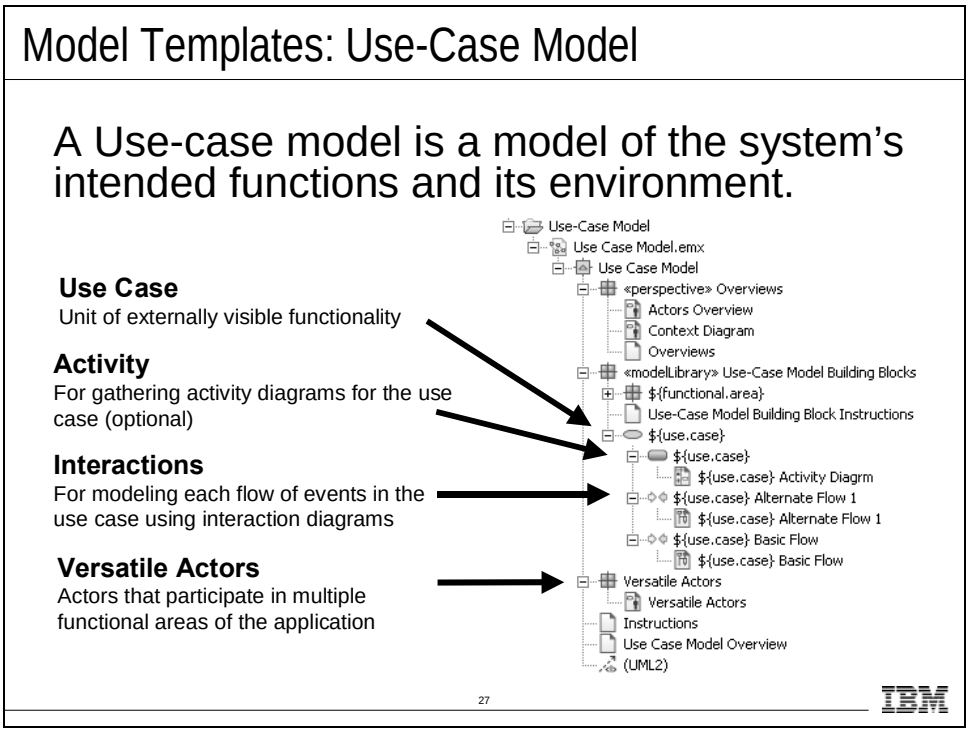
Every template contains a «modelLibrary» package called *`TemplateName`* `Building Blocks`. This package contains chunks of model content that you can use to build the design model more quickly. Building blocks act as "template" model elements. You can copy and paste the building block elements to create new elements for your model.

To use a building block element:

- In the Model Explorer, copy and paste or CTRL-drag a building block element from the building blocks package to the desired location in the model
- Right-click the new element and choose Find/Replace to change the placeholder name (`$`*`name`*) to the desired name

A best practice for naming diagrams is to come up with a descriptive name and then add the diagram type. For example, the use-case diagram above is called "PO Management Use Cases." It might also be called "PO Management Use-Case Diagram."

Model Templates: Use-Case Model



The use-case model is a model of the system's intended functions and its environment. The use-case model template in Software Architect contributes the default content shown in the slide. The model is divided with packages into the functional areas within the application. Each functional area package contains actors and use cases. Each use case can optionally contain interactions that link directly to use-case specifications and contain activity diagrams showing each flow of events graphically. The use-case model is used as an essential input to activities in analysis, design, and test.

The model contains a «perspective» package containing overview diagrams of all the functional areas in the model and a package containing "versatile actors," which are actors that appear in multiple functional areas in the application.

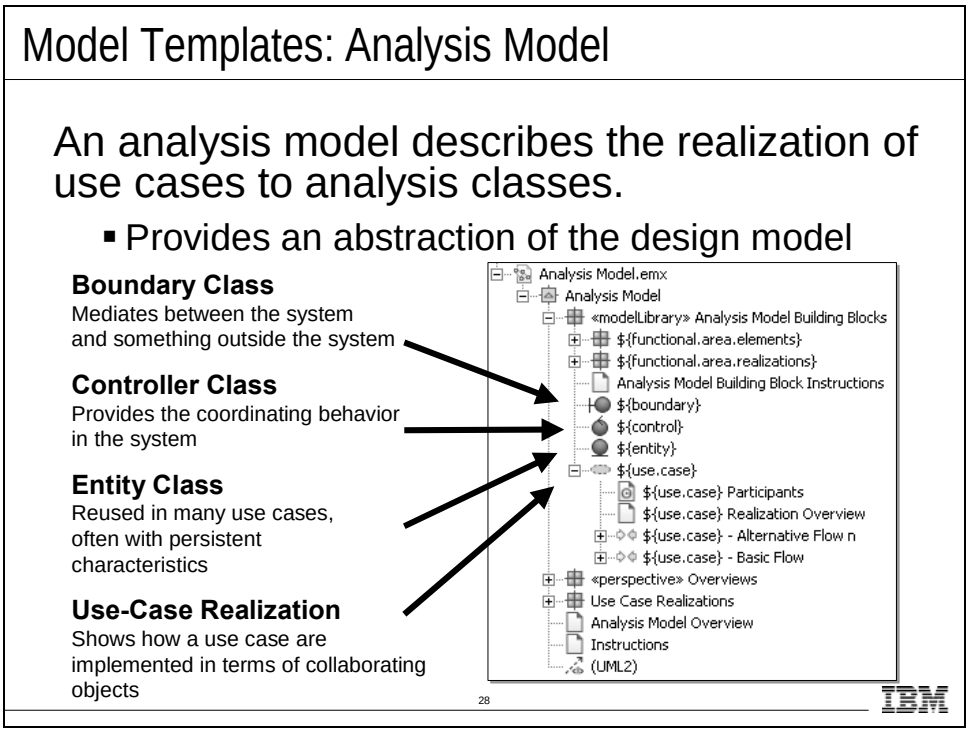
To create your use-case model using the building blocks you will perform these activities iteratively:

- Create "functional area" packages using the package building block provided
- Populate each package with the use cases for that functional grouping using the use-case building block provided

You can then perform the other activities of use-case modeling, such as gathering use cases into use-case diagrams, adding actors, drawing extend or include relationships among the use cases, drawing associations from the use cases to the actors that participate in them, writing descriptions for the use cases and actors, and (where it adds value) composing the high-level activity and "black box" sequence diagrams for each use case.

For information on the use-case model, see the Rational Unified Process: **Artifacts > Requirements Artifact Set > Use-Case Model**.

Model Templates: Analysis Model



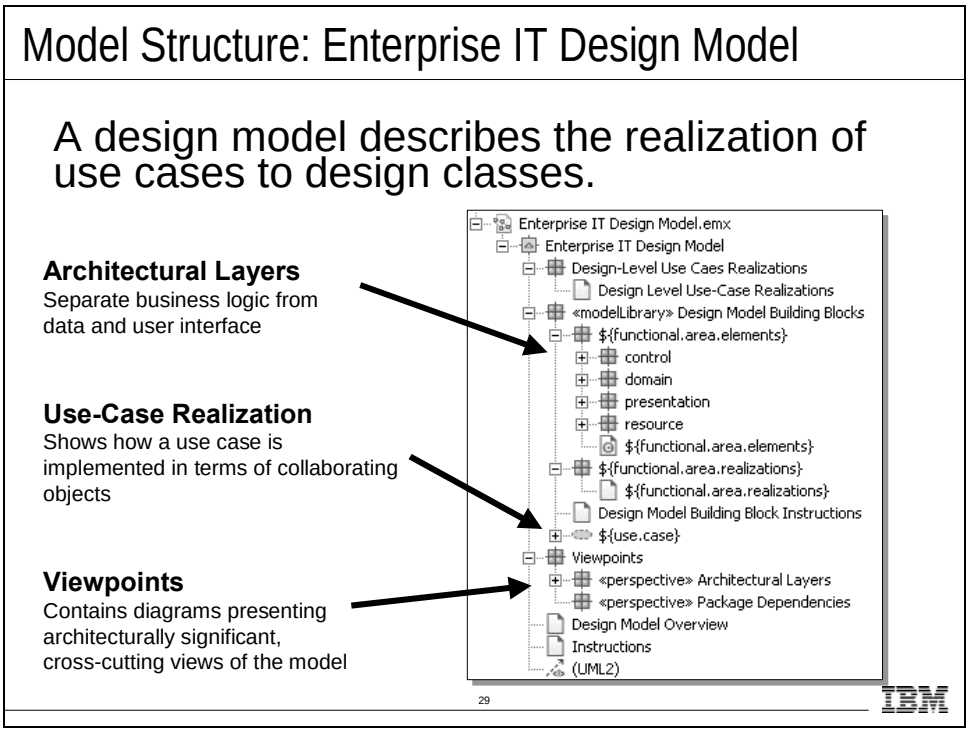
The analysis model is an object model that describes the realization of use cases (shows the collaborating objects derived from flows of events modeled in the use-case model). The Analysis model contains the results of use-case analysis. The analysis model serves as an abstraction of the design model.

In Software Architect, the Analysis model template contributes the default content shown in the slide. An “Analysis” profile is applied to model files created from this template.

The analysis model includes packages for the same functional areas defined in the use-case model, containing analysis classes and diagrams, derived from use-case realizations.

The process of use-case realization is beyond the scope of this course, though it is described in the Rational Unified Process (located in **Designer role > Use-Case Design > Guidelines > Use-Case Realization**). For a review of the analysis model and analysis classes, see **Artifacts > Analysis & Design Artifact Set > Analysis Model**.

Model Structure: Enterprise IT Design Model



The design model is an object model describing the realization of use cases and serves as an abstraction of the implementation model and its source code. The design model is used as essential input to activities in implementation and test.

The "Enterprise IT Design Model" (EITDM) template contributes default content, including a profile that supports modeling of services and transformations of design model elements into code. This is the appropriate model type to use for design (and optionally for analysis) when targeting business applications and using code-generating transformations to support creation of such applications.

To create your design model using the building blocks, you perform these activities iteratively:

- Create functional area element and functional area realization packages.
- Populate the realization packages with specification elements as you specify the behaviors (typically interactions) that express the design-level realizations of your use cases. The "realization" packages define the usage contracts for your components and services.
- Populate the element packages with implementation elements that realize the specification elements in the realization packages.

For more information on the design model, see the Rational Unified Process, **Artifacts > Analysis & Design Artifact Set > Design Model**.

Lab: Create UML Diagrams

Lab: Create UML Diagrams

- Given:
  - Payroll application problem statement
  - Maintain Employee Information Use-Case Specification
  - Payroll Application Case Study
- Complete the following tasks:
  - Create an activity diagram
  - Create a sequence diagram

30

IBM

Complete the Creating UML Diagrams lab.

Review

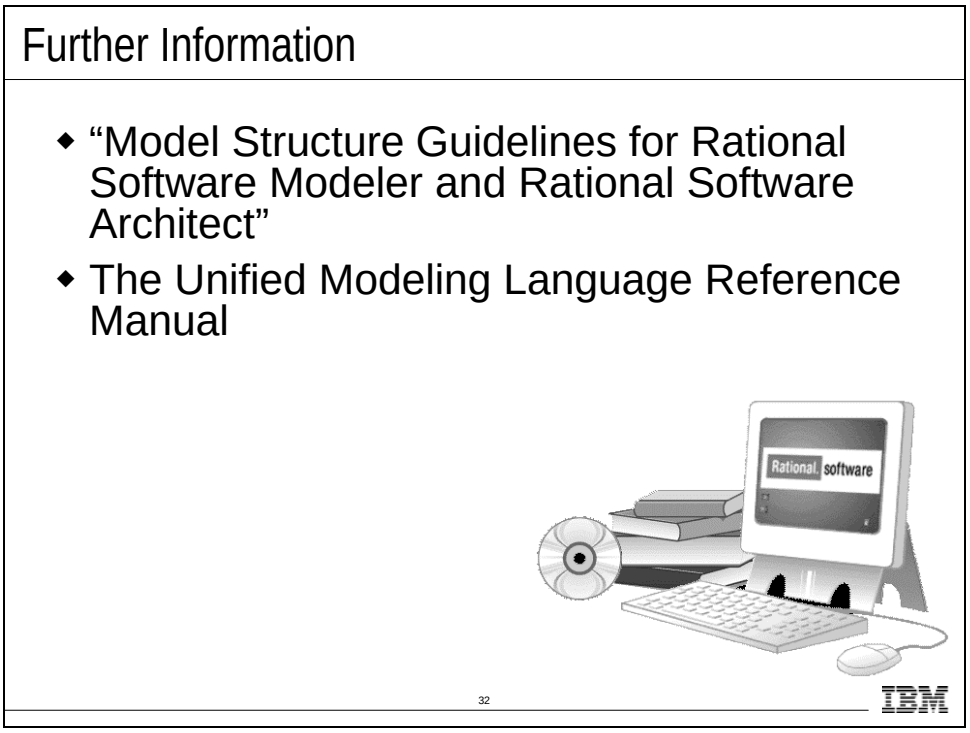
Review

- Which UML diagrams are used to model system structure?
- What are the two types of interaction diagrams?
- Describe two ways to add shapes and connectors to diagrams in Software Architect.

31

IBM

Further Information



- IBM Rational Software. “Model Structure Guidelines for Rational Software Modeler and Rational Software Architect” (2004 Release).
- James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. 2nd Ed. New York: Pearson Education, 2004.

Related IBM Rational Training

Related IBM Rational Training

- DEV275: Essentials of Visual Modeling with UML 2.0 (2004)
- DEV475: Mastering Object-Oriented Analysis and Design with UML 2.0 (2004)

33

IBM

Creating UML Diagrams

Objectives

After completing this lab, you will be able to:

- Review a problem statement and case study
- Import an UML2 model
- Draw an activity diagram
- Draw a sequence diagram in the context of a use-case realization
- Populate a class diagram

Given

The following lab artifacts can found in the `DEV396` folder:

- Payroll problem statement (`Payroll Problem Statement.doc`)
- `Maintain Employee Information` use-case specification (`MaintainEmployeeInfo.doc`)
- Payroll application design and use-case models (`ACMEPayrollModel.zip`)

Scenario

In this lab, you will create various diagrams using the Payroll application use-case and design models.

Task 1: Review Payroll Problem Statement and Use-Case Specification

In this task, you will study the Payroll problem statement and its use-case specification.

1. Open and read the problem statement artifact (`Payroll Problem Statement.doc`) located in the `DEV396` folder.
2. Review the use-case specification document (`MaintainEmployeeInfo.doc`) located in the `DEV396` folder:
   - How many use cases are described?
   - How many actors?
   - List the flows of events.

Task 2: Import the Payroll Application Project

In this task, you will import the payroll application project saved in Project Interchange format. The Project Interchange feature provides new import and export wizards for sharing a set of projects easily in one step.

1. Close all open models.
2. Import the `ACMEPayrollModel` project:
   a. On the **File** menu, click **Import**.
   b. In the list box, select **Project Interchange** Project Interchange as the import source and then click **Next**.
   c. Beside the `From Zip file` field, click the **Browse** button and find `ACMEPayrollModel.zip` in the `DEV396` folder. Select the file and then click **Open**.
   d. Select `ACMEPayrollApplication` in the **Import Projects** list.
   e. Click **Finish**.
1. Open the project and explore its contents.

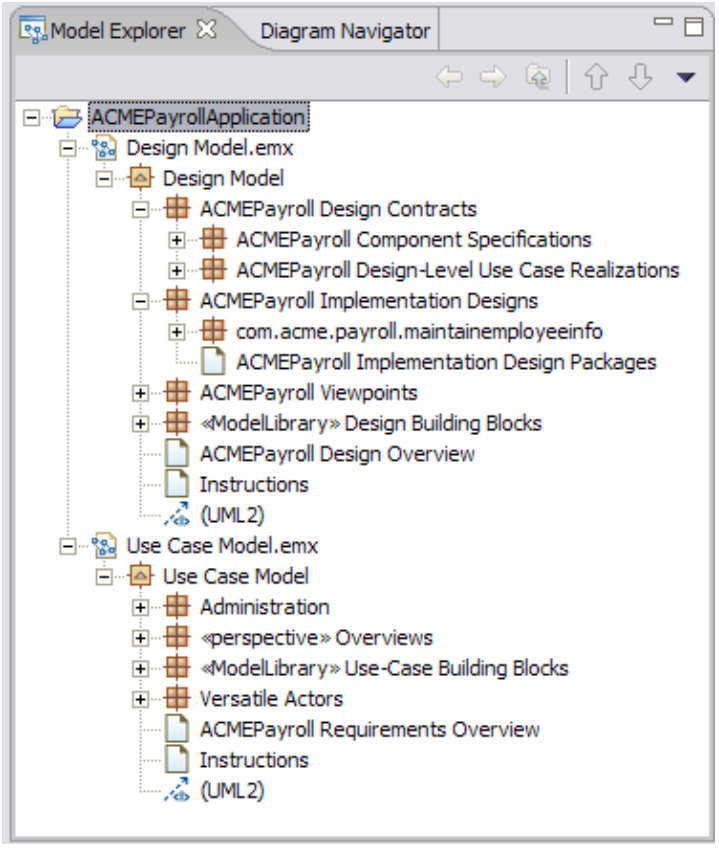
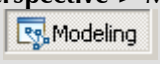


***Figure 1**: ACMEPayrollApplication in Model Explorer view*

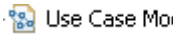
Task 3: Create an Activity Diagram

In this task, you will create an activity diagram in the `ACMEPayrollApplication` project to illustrate the flow in a use case.

1. If not already open, open the **Modeling** Perspective. With the `ACMEPayrollApplication` project open.

   TIP: Click **Window > Open Perspective > Modeling** to open the Modeling perspective or click on the Modeling icon 

on the tool bar.

2. Open the Use Case Model by double-clicking the 

icon.
3. In `Use Case Model`, open the `Administration` package**.**
4. Right-click the `Maintain Employee Information` use-case and then click **Add Diagram > Activity Diagram**.
5. Name both the activity and the diagram `AddEmployeeActivities` (as shown in Figure 2).

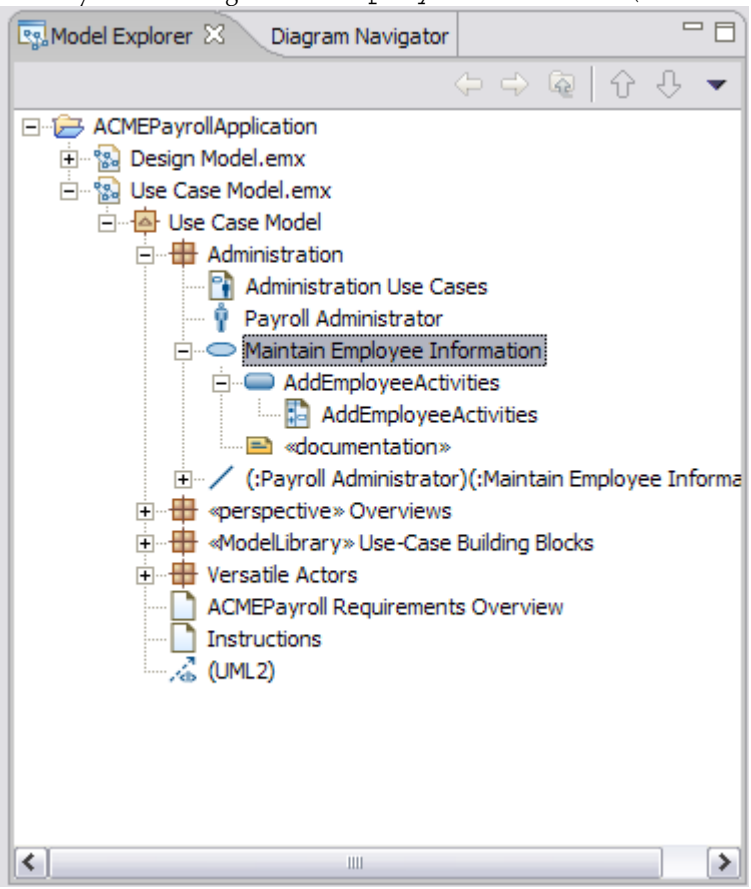
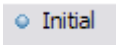
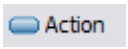


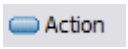
***Figure 2**: Activity Diagram in Model Explorer*

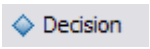
Task 4: Draw an Activity Diagram

In this task, you will draw a simple activity diagram for the `Add an Employee` flow of the `Maintain Employee Information` use-case (shown in Figure 2).

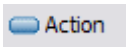
1. Open the activity diagram in the Diagram editor.
2. Use the palette or action bars on the diagram surface to add the following nodes from top to bottom down the diagram:
   - An **Initial** 

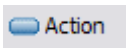
node
   - An **Action** 

node named **Receive Employee Information**
   - An **Action** 

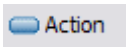
node named **Validate Employee Information**
   - A **Decision** 

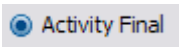
node
   - A **Fork** 

node and size appropriately
   - An **Action** 

node named **Assign Paycheck Delivery**
   - An **Action** 

node named **Generate Employee ID**
   - A **Join** 

node and size appropriately
   - An **Action** 

node named **Add Employee**
   - An **Activity Final** 

node

   Arrange the nodes as shown in Figure 3.

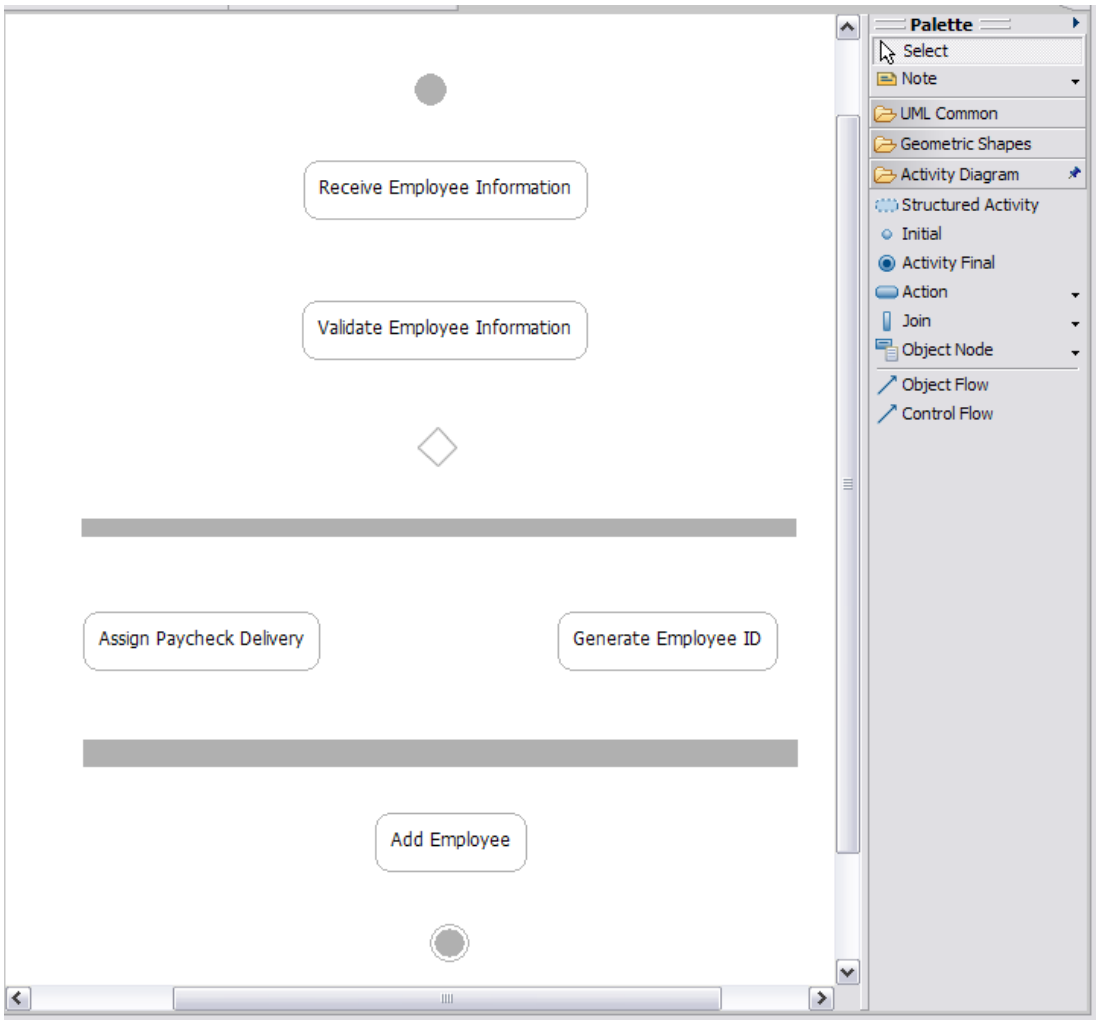
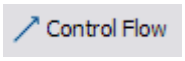
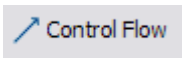
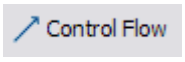
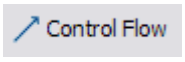


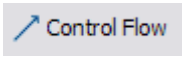
***Figure 3**: Layout of elements*

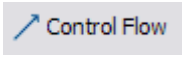
3. Use the palette or action bars on the diagram surface to add the following flow relationships:
   - A **Control flow** 

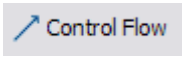
from the `Initial` node to the `Receive Employee Information` action.
   - A **Control flow** 

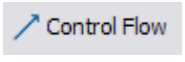
from the `Receive Employee Information` action to the `Validate Employee Information` node.
   - A **Control flow** 

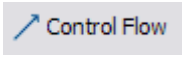
connection from `Validate Employee Information` action to the `Decision` node.
   - A **Control flow** 

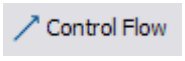
connection from the `Decision` node to the `Receive Employee Information` node.
   - A **Control flow** 

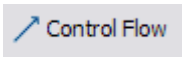
connection from the `Decision` node to the `Fork` node.
   - A **Control flow** 

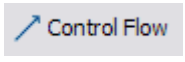
connection from the `Fork` node to the `Assign Paycheck Delivery` action.
   - A **Control flow** 

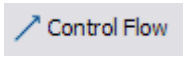
connection from the `Fork` node to the `Generate Employee ID` action.
   - A **Control flow** 

connection from the `Assign Paycheck Delivery` action to the `Join` node.
   - A **Control flow** 

connection from the `Generate Employee ID` action to the `Join` node.
   - A **Control flow** 

connection from the `Join` node to the `Add Employee` action.
   - A **Control flow** 

connection from the `Add Employee` action to the `Activity Final` node.
   - Label the **Control flow** 

from the `Decision` node to the `Receive Employee Information` action as “Invalid Information Provided.”
   - Label the **Control flow** 

from the `Decision` node to the `Fork` node as “Valid Employee Information.”

After arranging the diagram for presentation, it appears as in Figure 4.

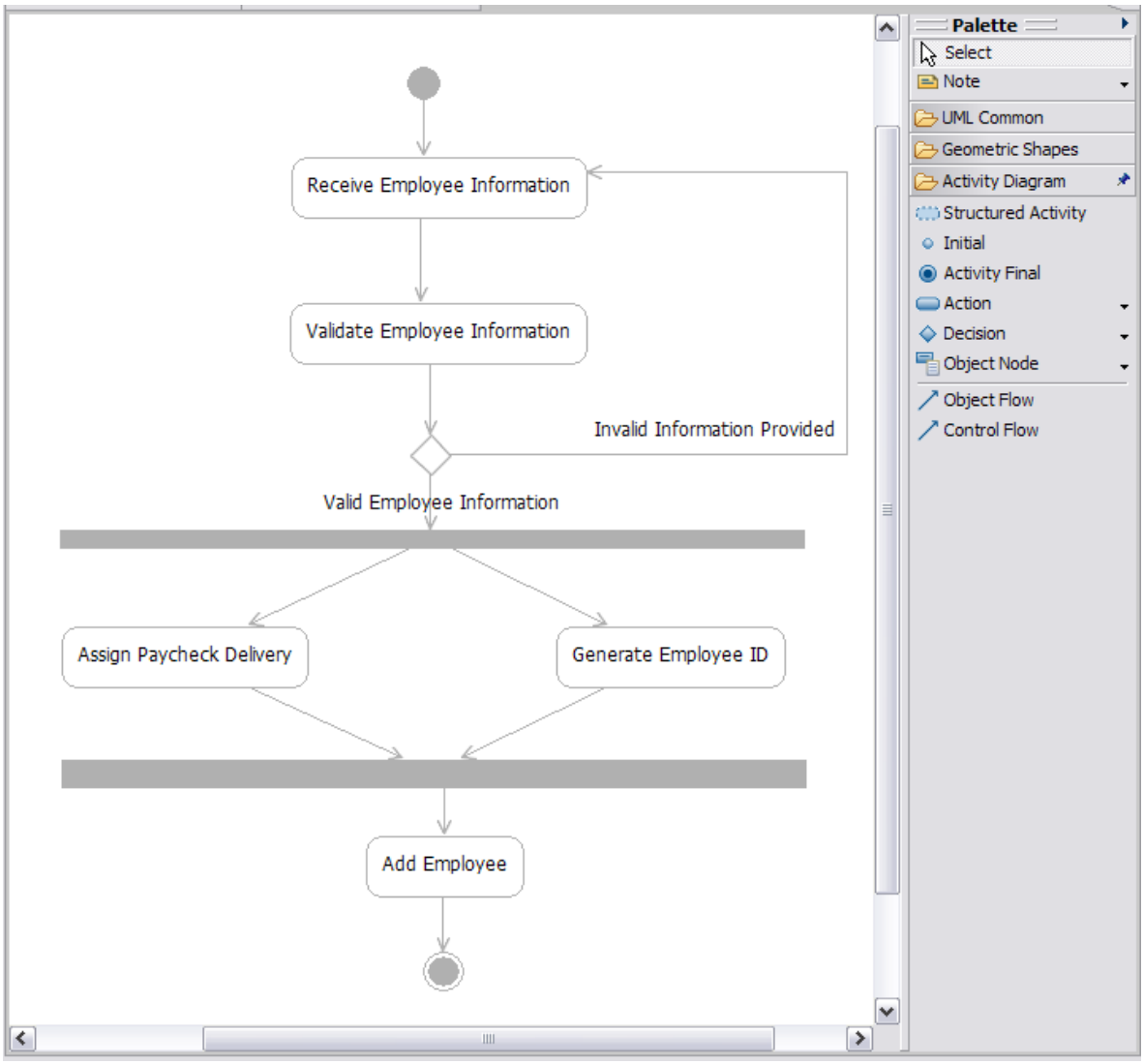


***Figure 4**: AddEmployeeActivities diagram*

Task 5: Create a Sequence Diagram

In this task, you will draw a sequence diagram for the `Find Employee` flow as part of the `Maintain Employee Information` use-case realization.

1. In the Model Explorer view, expand **Design Model > ACMEPayroll Design Contracts > ACMEPayroll Design-Level Use Case Realizations > Administration.**
2. Right-click `<<use-case realization>> Maintain Employee Information` and select **Add UML Diagram > Sequence Diagram**. Name both the interaction and the diagram `Subflow: Remove Deleted Employees`.
3. Add a new `Payroll Administrator` object to the sequence diagram as an unnamed object.

> *TIP:* From the use-case model, navigate to the Administration package, drag the Payroll Administrator actor, and drop it on the sequence diagram. When prompted for a name for the element, press **Delete** and then **Enter**.

4. Add the following unnamed objects to the diagram:
   - RemoveMarkedRecordsForm
   - RemoveDeletedEmployeesController
   - PayrollDB

> *TIP:* On the palette, select the Sequence Diagram drawer, select the Lifeline tool, and click on the diagram. Select **Create New Class**. Enter a name for the new class and click **OK**. When prompted for a name for the element, press **Delete** and then **Enter**.

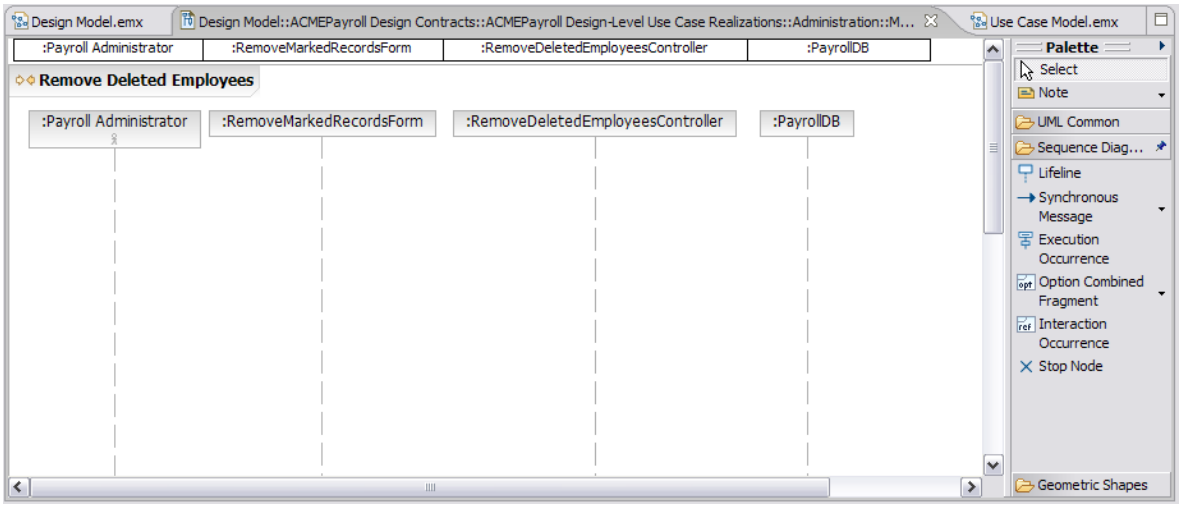


***Figure 5**: Lifelines created on sequence diagram*

5. Create the `removeMarkedEmployees` message from `Payroll Administrator` object to the `RemoveMarkedEmployeesForm` object.

> *TIP:* In the **Sequence Diagram** drawer, select the **Synchronous Message** tool, click on the `Payroll Administrator` object's lifeline and drag it to `RemoveMarkedEmployeesForm` lifeline, release, and in the drop-down list click the `findEmployee` operation. Name the operation `removeMarkedEmployees` and click **OK**.

6. Create the `removeMarkedEmployees` message from `RemoveMarkedEmployeesForm` object to the `RemoveDeletedEmployeesController` object. Ensure that the message starts from the current execution occurrence on the Payroll Administrator lifeline.

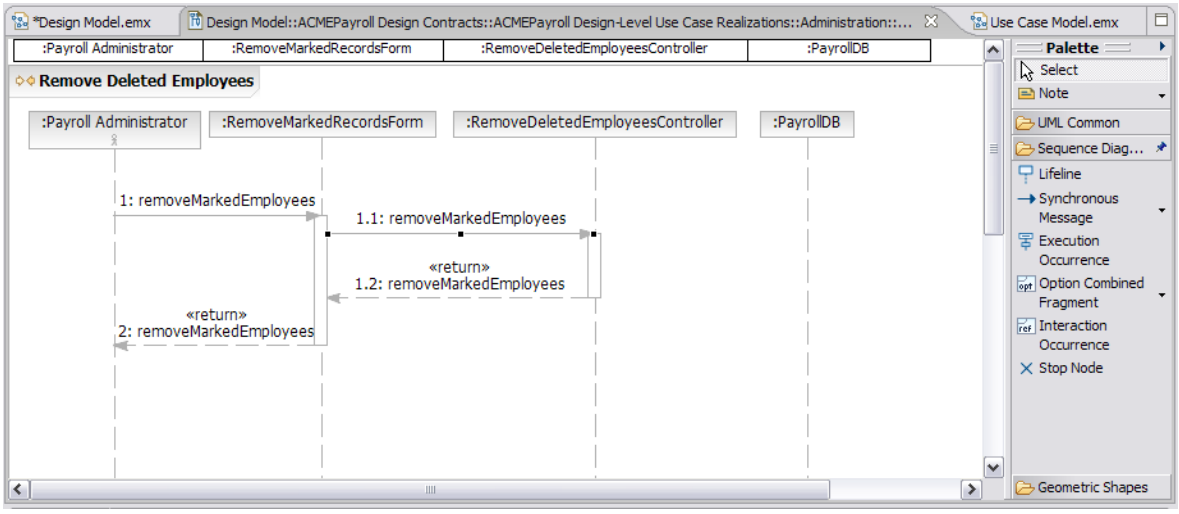


***Figure 6**: Messages created on sequence diagram*

7. Add an Interaction Occurrence that references another sequence diagram:
   - From the palette, click **Interaction Occurrence**. Then on the sequence diagram, click on the execution occurrence on the `RemoveDeletedEmployeesController` lifeline.
   - Select **Create New Interaction.**
   - Name the **ref** `retrieveMarkedEmployeeList`.
   - Configure the **Interaction Occurrence** to span the `RemoveDeletedEmployeesController` and `PayrollDB` lifelines.

> *TIP:* On the sequence diagram drawer, single click the interaction occurrence. Then click and hold the handle that appears on the right-hand side of the element. Drag the handle so that the interaction occurrence now covers the `PayrollDB` lifeline and release the mouse. In the **Add Covered Lifelines** dialog, select `PayrollDB` and then click **OK**.

8. Add an Option Combined Fragment:
   - From the palette, click **Option Combined Fragment**. Then on the sequence diagram, click on the execution occurrence on the `RemoveDeletedEmployeesController` lifeline below the interaction occurrence that was placed previously.
   - Enter `hasMoreEmployees` as the guard condition.
   - Configure the Option Combined Fragment to span the `RemoveDeletedEmployeesController` and the `PayrollDB` lifelines.

9. Add a Loop Combined Fragment:
   - From the palette, click the arrow beside the Option Combined Fragment tool and select Loop Combined Fragment.

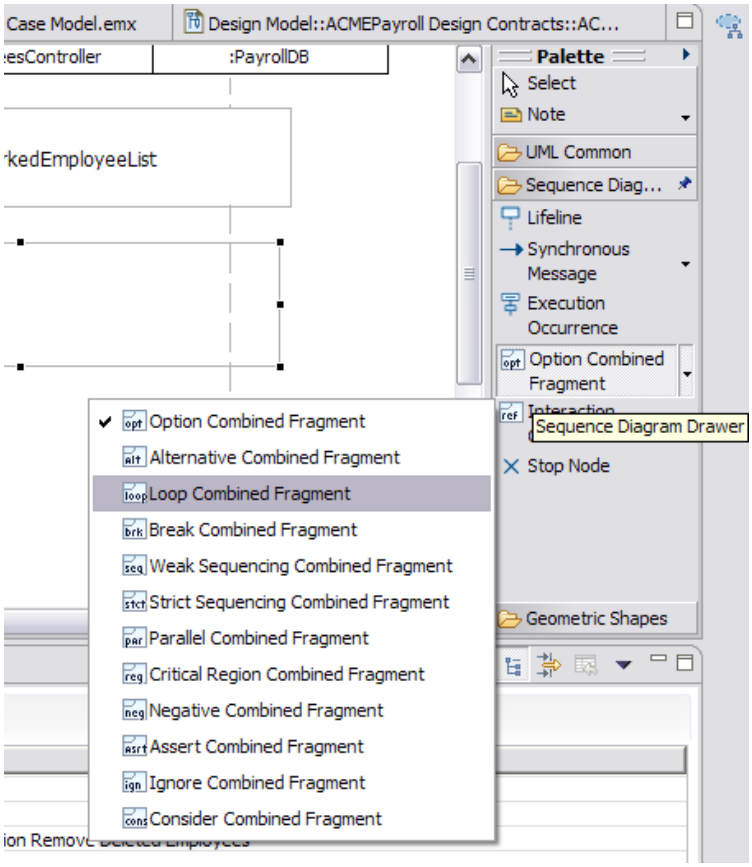


Figure 7: *Selecting a Loop Combined Fragment*

   - Click on the execution occurrence for **RemoveDeletedEmployeeController** within the area of the Option Combined Fragment placed previously.
   - Click **Enter** to accept the default parameters of **0,***.
   - Configure the **Loop Combined Fragment** to span the `RemoveDeletedEmployeesController` and `PayrollDB` lifelines.

10. Within the **Loop Combined Fragment**, create the `deleteEmployee` message from `RemoveDeletedEmployeesController` object to the `PayrollDB` object.

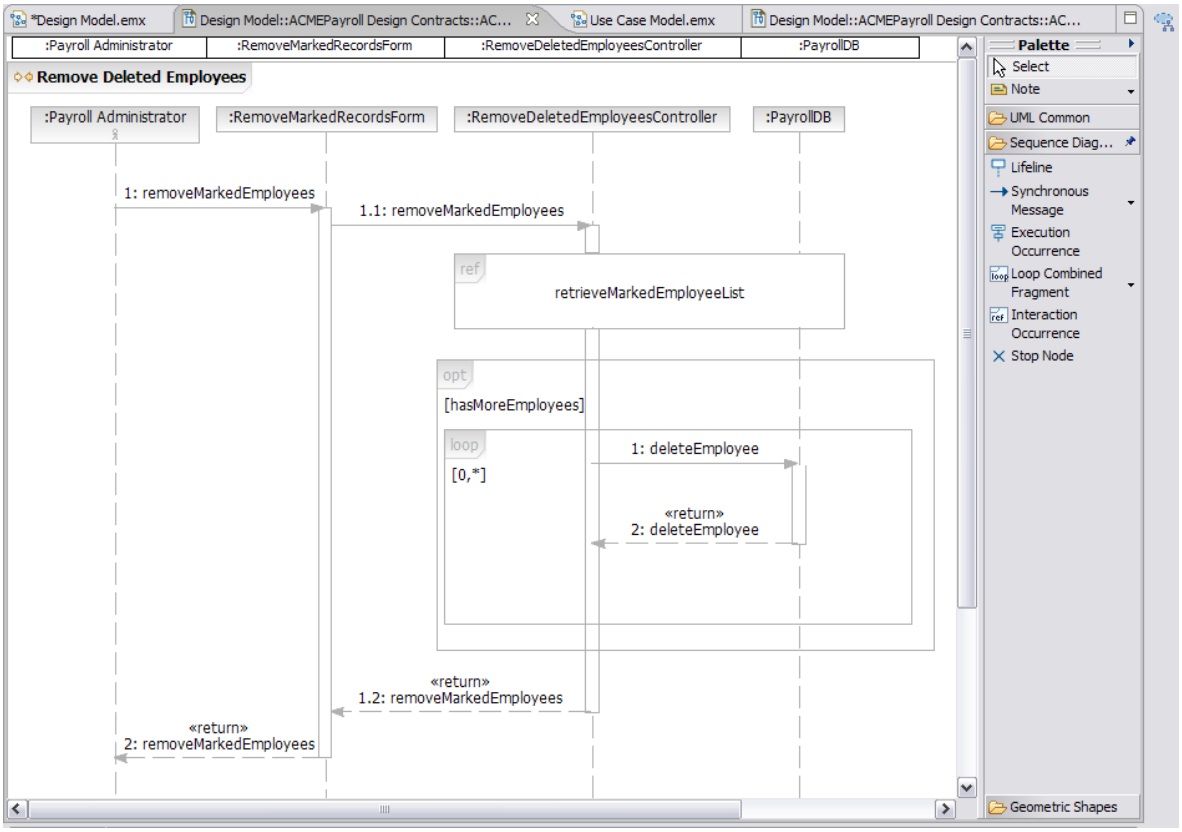


***Figure 8**: Completed sequence diagram*

Task 6: Populate a Class Diagram

In this task, you will create a class diagram that shows the classes that participate in the use-case realization.

1. Open the `Participants` diagram found under the `<<use-case realization>> Maintain Employee Information` node.
2. From the Model Explorer, drag the following classes onto the diagram:
   - PayrollDB
   - RemoveMarkedRecordsForm
   - RemoveDeletedRecordsController

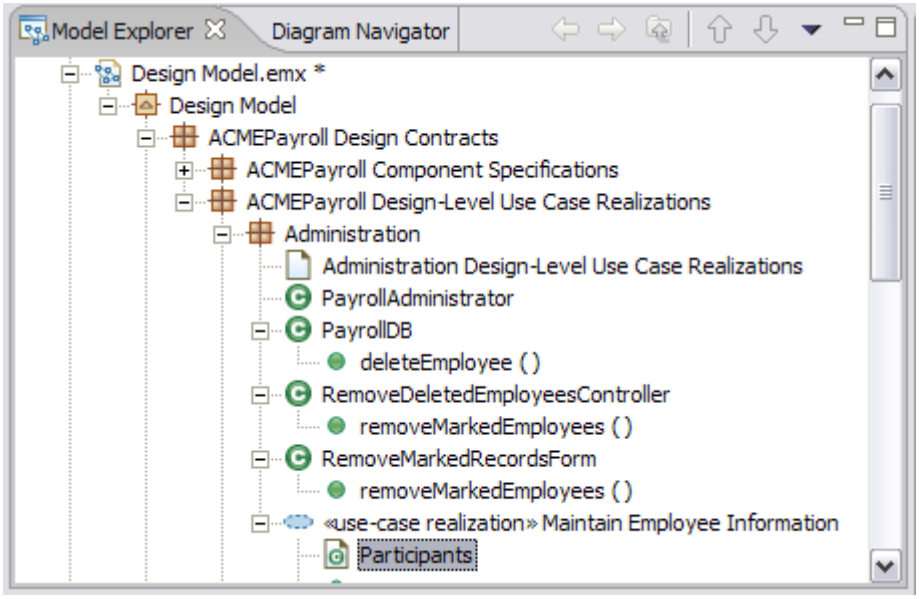


***Figure 9**: Participants diagram and the classes to add*

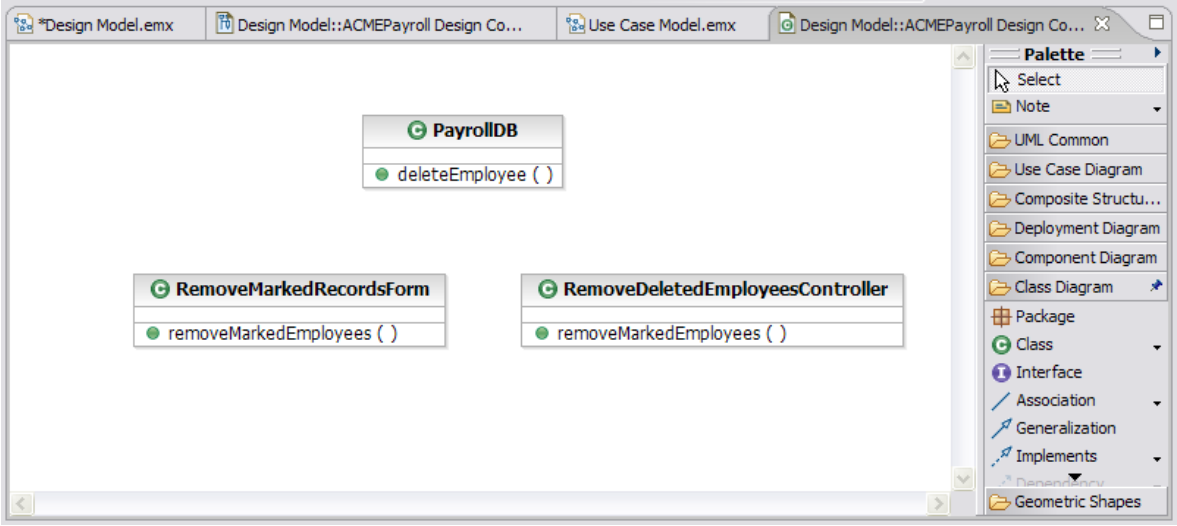
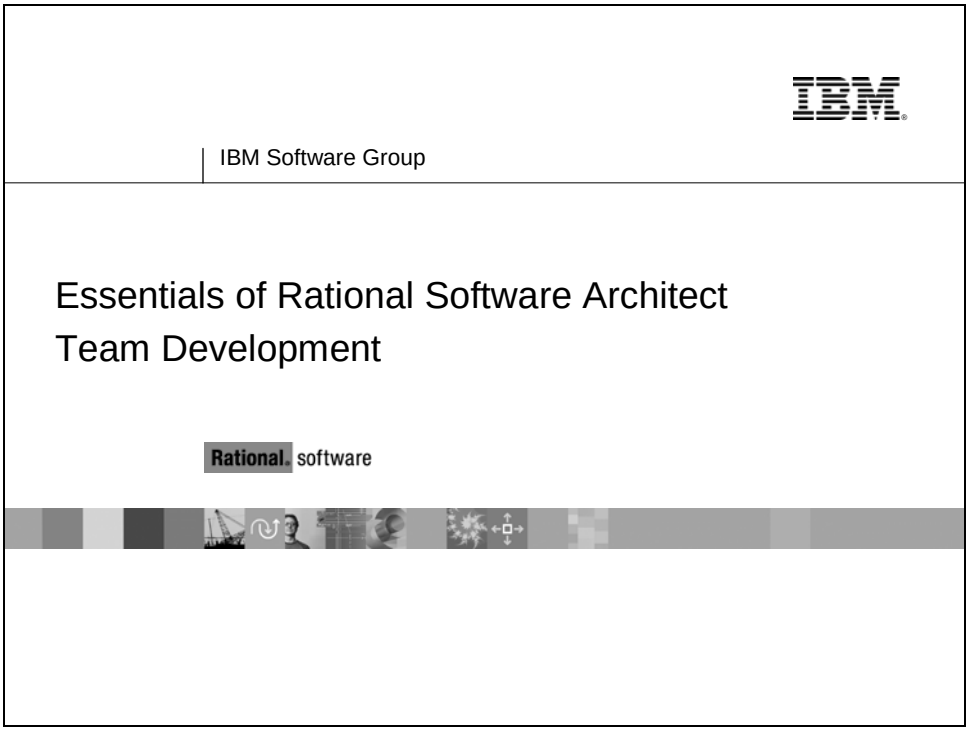


***Figure 10**: Participants diagram with classes shown*

▸ ▸ ▸ Team Development

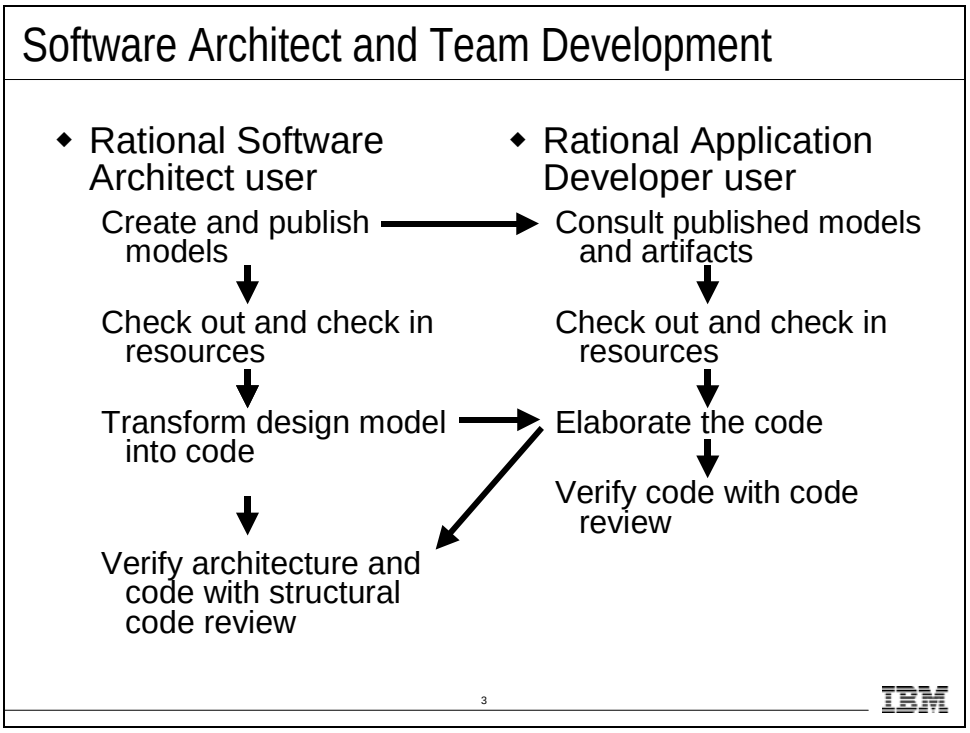


Topics

Module Objectives

This module introduces team development with Rational Software Architect, including configuration management features and best practices.

Software Architect and Team Development



The following Software Architect features facilitate collaboration between architects and developers:

Transformation

Software Architect's model transformation features allow architects to create basic code structures from high-level UML models, which can be used as a starting point for developers.

Code Review

The code review feature is used to analyze code against the project's architectural guidelines. Code review includes:

- Structural anti-pattern detection to assist in Java refactoring
- Structural rules (including user-defined rules) to enforce architectural control of Java code

Web Publisher

While creating detailed implementation code, the developer can refer to Web-published models and UML-injected Javadocs, which Software Architect can generate automatically.

Configuration Management in Software Architect

Where Are We?

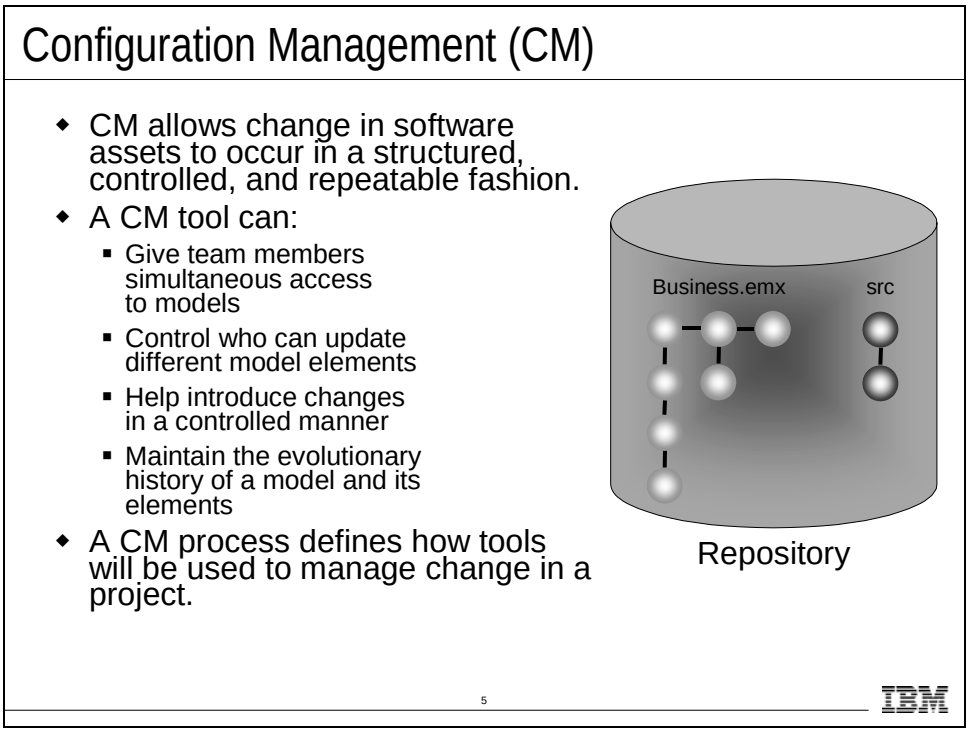
- **Configuration Management in Software Architect**
- Compare and Merge
- Model Publishing

4

IBM

This section provides an overview of configuration management issues with Software Architect.

Configuration Management (CM)



Configuration management (CM) is a software engineering discipline that comprises the tools and techniques (processes or methodology) that a company uses to manage change to its software assets. The goal of successful CM is to allow change in a structured, controlled, and repeatable fashion.

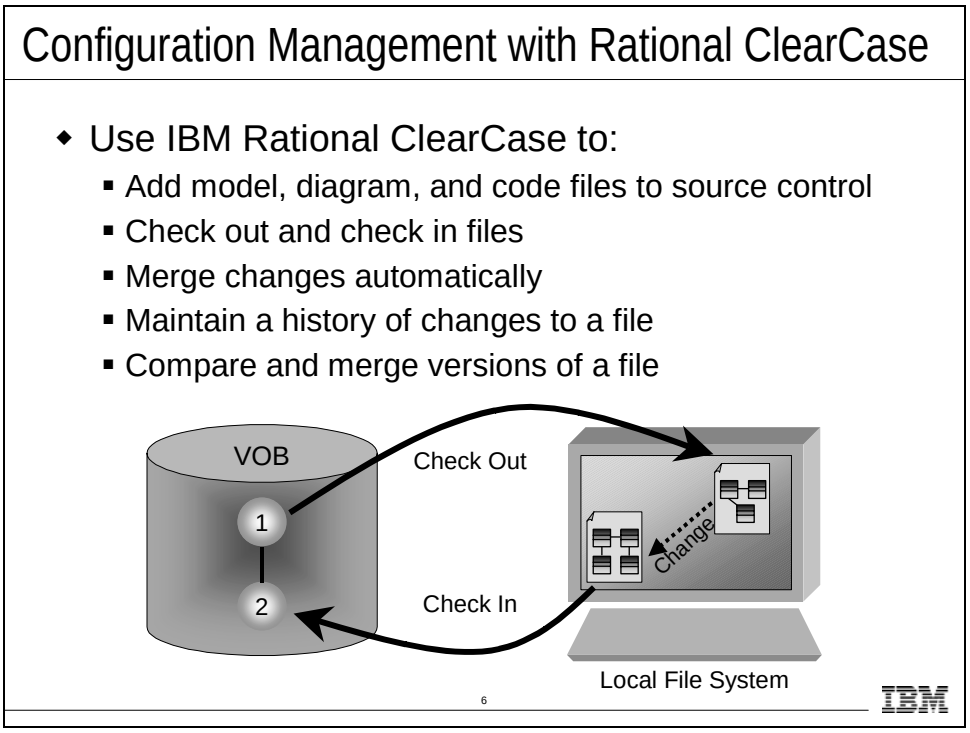
Successful CM is achieved by applying both tools and processes to a software development project.

CM tools are software tools that automate and facilitate CM activities.

A CM process is the way CM is performed on your project; specifically, how a CM tool is applied to accomplish a task. The key to a good process is in tailoring it to be as simple as possible, following standard industry practices.

Another important strategy for team development is to facilitate the reuse and sharing of existing project assets, such as patterns and components. Remember that changes to reusable assets need to be controlled the way changes to any other project artifacts are controlled.

Configuration Management with Rational ClearCase



IBM® Rational® ClearCase® provides software configuration management for medium- to large-sized teams. ClearCase can manage all the artifacts in the development process from requirements documents and design models to code and tests. ClearCase supports:

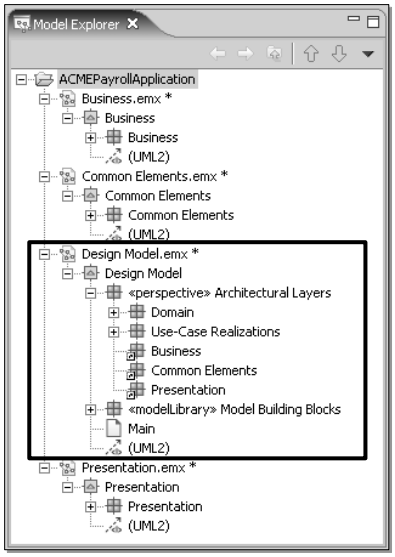
- Parallel development
- Advanced workspace management, including dynamic views
- Build management features, including mainframe connectors

ClearCase integrates with Software Architect, including Software Architect's compare and merge features. When team members work with version-controlled artifacts, they frequently need to compare file versions to understand their history and development. Team members may need to merge files because a merge result generally contains all the work that multiple team members complete in parallel.

SCM Best Practices: Model Partitioning

SCM Best Practices: Model Partitioning

- Partition the model to avoid unnecessary merges
- Factors to consider when deciding how to partition a model:
  - Stabilize abstraction levels
  - Minimize dependencies between models
  - Establish ownership policies
  - Avoid broken references



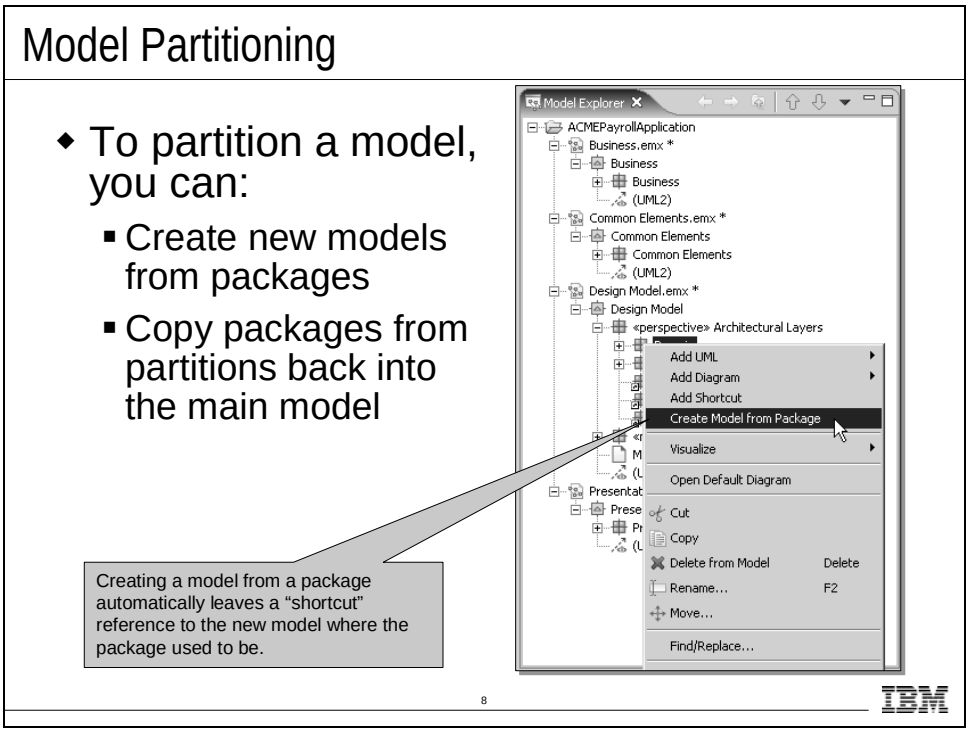
7

IBM

While the compare and merge features in Rational Software Architect are advanced and easy to use, model merge scenarios can become complex and prone to user error if the models involved are needlessly complicated and disorganized. To optimize your models for configuration management and team development, you should observe the following principles:

- **Stabilize abstraction levels:** Do not partition the model until the top-level subsystems are clearly defined. Once the subsystems are mature and stable, you can separate them to enable parallel development and to improve the speed with which the model opens. When an individual subsystem's contents stabilize, you can then separate the subsystem from the model.
- **Minimize dependencies between models:** By minimizing dependencies between models, you can reduce the likelihood that a change in one model will affect another. If you allow dependencies between models, widespread conflicts can occur that must be resolved with out of context model merges (which are very challenging).
- **Establish ownership policies:** Assigning different model files to different team members can help avoid conflicts when you deliver the model to a shared work area and can help speed integration. You should establish the size and scope of each model so that a single person can work on it.
- **Avoid broken references:** Whenever you move a model partition outside of your configuration management system, you break that model file's references. You have to repair these broken references whenever you reintegrate the model back into the CM system. It can be time-consuming and error-prone to resolve a large number of references in a complicated model.

Model Partitioning



You can refactor an existing model file into several, smaller model files in Software Architect.

To separate a package from a model into a new model file in the Model Explorer, right-click the package and then click **Create Model From Package**.

To reassemble the model after it has been partitioned, you must copy and paste the packages back into the main model.

Compare and Merge

Where Are We?

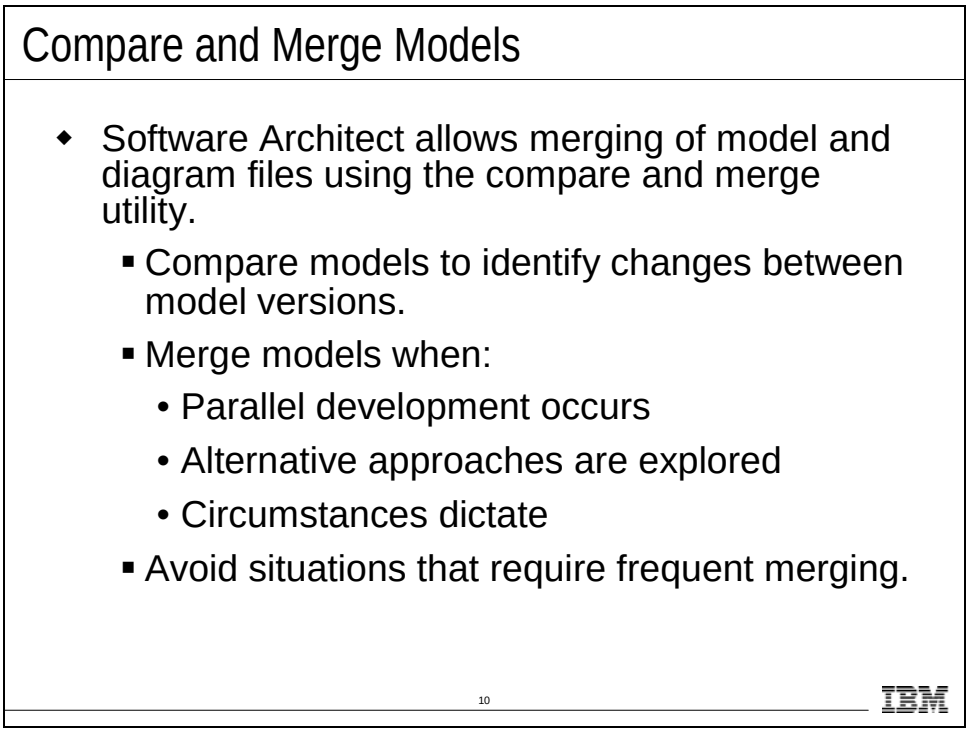
- Configuration Management in Software Architect
- **Compare and Merge**
- Model Publishing

9

IBM

This section shows how to compare and merge versions of a model.

Compare and Merge Models

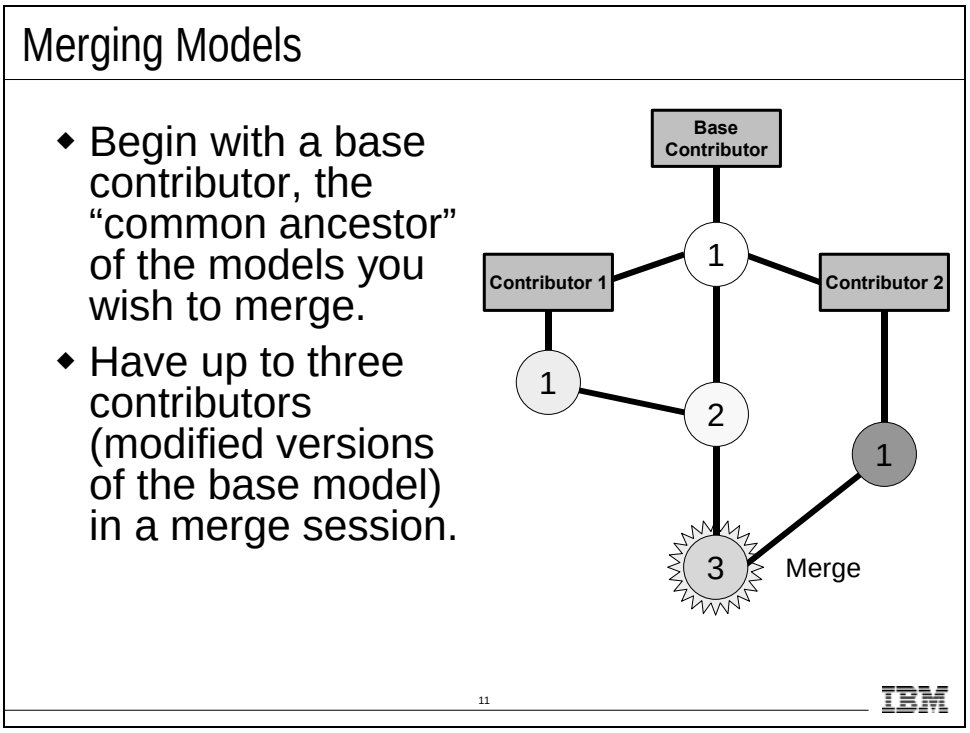


You can compare two or three models with a common ancestor to find the differences among them outside a source-controlled environment or within a team environment, using a configuration management system, such as IBM Rational ClearCase.

The compare feature can be used to compare versions of models just to see what has changed between versions as part of everyday modeling work, particularly in collaborative settings. Compare and merge becomes essential in more complicated team development settings, such as parallel development.

Compare and merge scenarios can often become very complicated and difficult to resolve if not carefully controlled. When working with large and complicated models, compare and merge should be avoided except when absolutely necessary.

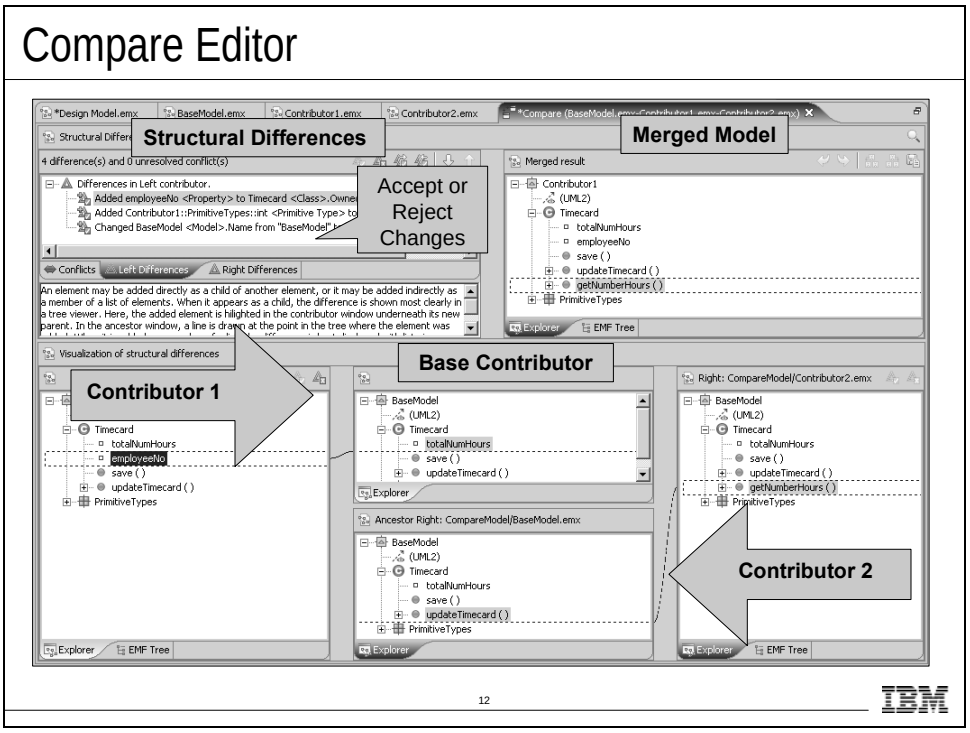
Merging Models



The slide shows the simplest and most common compare and merge scenario:

- You and a colleague work on copies (Contributor 1 and Contributor 2) of a model file (Base Contributor) that is under source control.
- You both make changes to your copies.
- Your colleague checks in changes (creating Base Contributor version 2).
- You try to check in changes.
- The SCM system detects that the latest version in the repository (BC version 2) is not the version you checked out and modified (BC version 1).
- The SCM system detects that your version cannot replace the version in the repository, or else you lose your colleague's changes.
- The SCM system attempts to merge your changes into the latest version (BC version 3) in the repository.

Compare Editor



Given multiple versions of a model that were modified in parallel, you can merge the models by starting a merge session and using the Compare editor. Once you resolve all conflicts among the contributors, you can save the final merged model as a new version. When you compare two versions of a model, their differences are displayed in the Differences editor. When you compare three models, two sets of differences are displayed:

- Differences between one version (the left contributor) and the common ancestor (the base contributor)
- Differences between the other version (the right contributor) and the common ancestor

When working with a version control system, such as ClearCase, a merge session may start automatically when a user checks in a file under version control. After merging models, saving the merged model, and closing the session, the process that initiated the check-in process continues.

Automatic merges

During an automatic merge, all differences and trivial conflicts are merged automatically. If the software automatically makes a resolution, a resolution icon is displayed beside the resolved difference in the Structural Differences view. The software also generates a merged model for these resolutions.

If the merged model contains conflicts, a conflict icon is displayed beside each conflicting element in the Structural Differences view.

Manual merges

During a manual merge, you resolve the differences or conflicts. You can right-click a difference or conflict and then accept or reject it. You can also accept or reject groups of differences or conflicts.

You can start a manual merge session in the following ways:

- Merge models in a ClearCase version or history interface
- Compare models in an Eclipse resource view that has one or more modifiable resources

Demo: Comparing and Merging Models

Demo: Comparing and Merging Models

- The instructor will now show you how to:
  - Merge two contributors
  - Merge three contributors

13

IBM

Watch your instructor demonstrate how to merge models in Software Architect.

Lab: Compare and Merge Models

Lab: Compare and Merge Models

- Given:
  - Base model
  - Two contributor models
- Complete the following tasks:
  - Merge two contributors
  - Merge three contributors

14

IBM

Complete the Team Development lab.

Model Publishing

Where Are We?

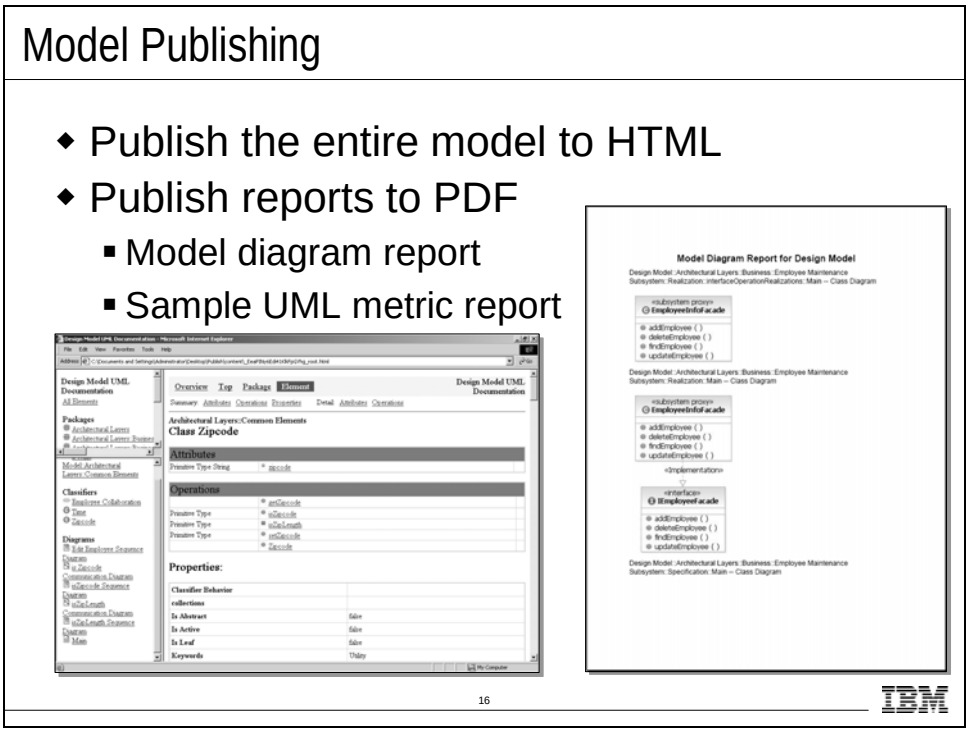
- Configuration Management in Software Architect
- Compare and Merge
- **Model Publishing**

15

IBM

This section discusses the model Web publishing features of Software Architect. Publishing is important for sharing models and diagrams with team members, collaborators, and customers.

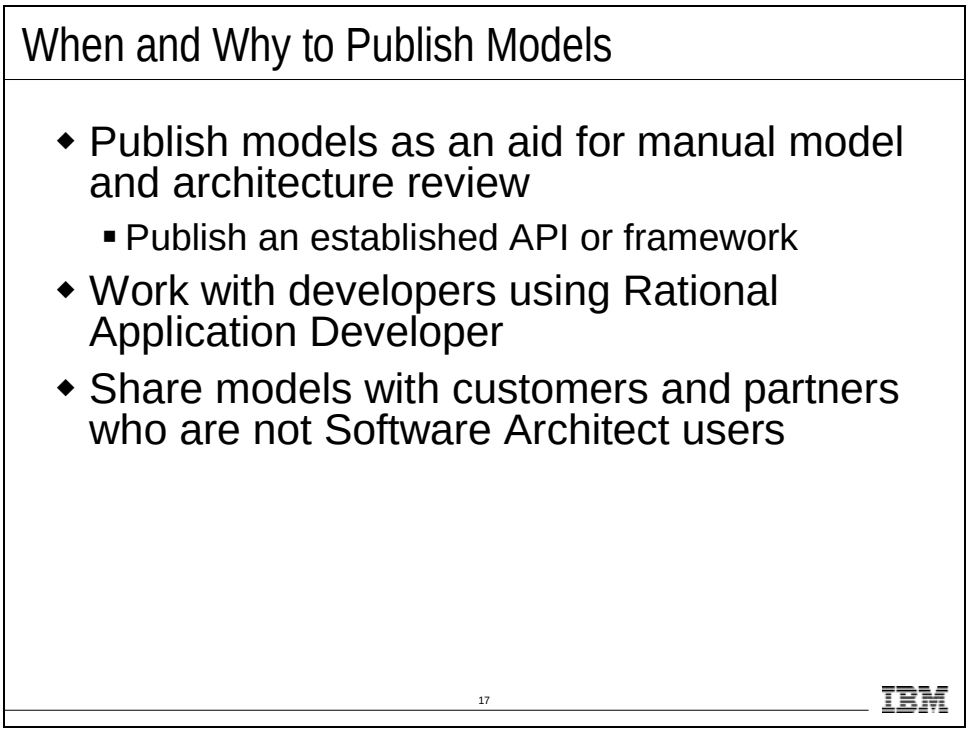
Model Publishing



When you publish a model as a Web site, you indicate the scope of data to include, how much about that data to include, and where you want to save the published model.

Each published model is a separate, wholly contained entity with its own root page. When you publish, the root page has a link to each included model.

When and Why to Publish Models

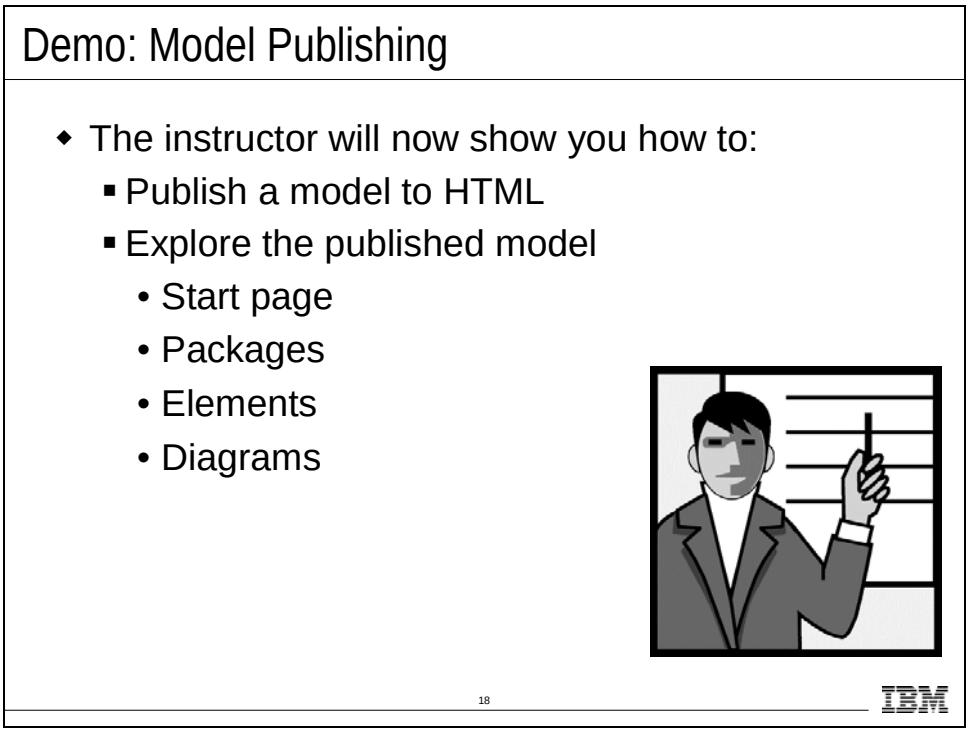


Publishing extracts model information and uses it to convert model elements, including diagrams, classes, packages, relationships, attributes, and operations, to images and model information to HTML for viewing outside the modeling tool.

In the published model, hypertext links let viewers navigate the model much as they would within Software Architect.

You can publish successive iterations of an evolving model for review or to share information with team members or customers. Another potential use is to publish documentation for a frozen API or framework.

Demo: Model Publishing



Watch your instructor publish a model in Software Architect.

Review

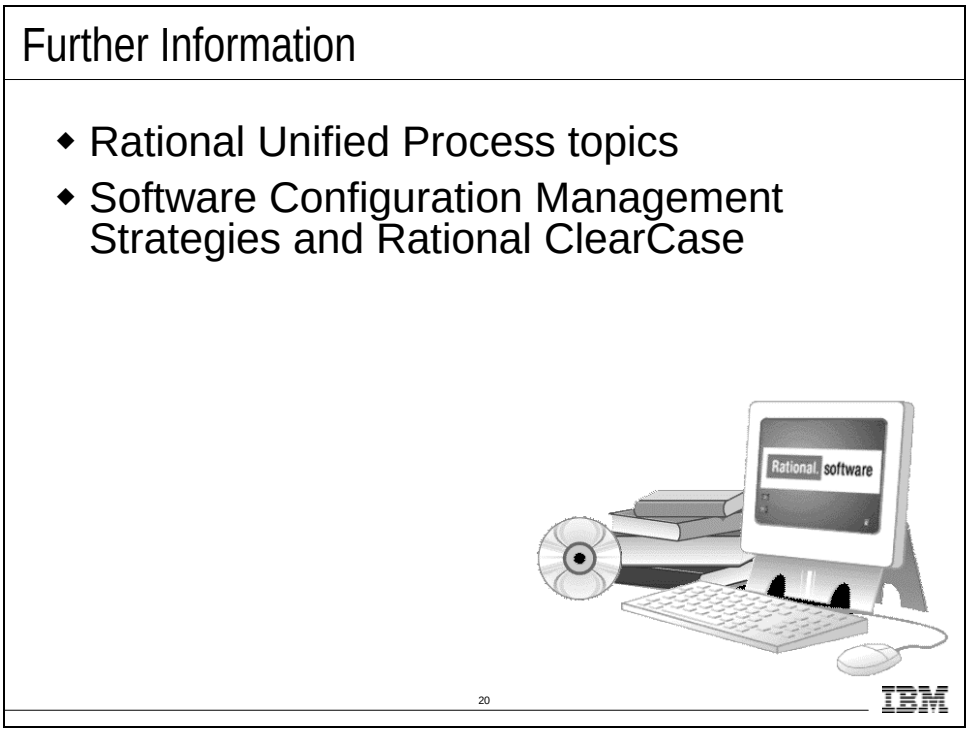
Review

- Why should you partition model files?
- What are the four best practices for model partitioning?
- Name some examples of when you might need to Web-publish a model.

19

IBM

Further Information



- Rational Unified Process, developer topics:
  - **Designer** > **Whitepapers** > **Developing Large-Scale Systems with the Rational Unified Process**
  - **Designer** > **Implementation** > **Workflow**
  - **Designer** > **Configuration and Change Management: Overview**
  - **Designer** > **Tool Mentors** > **Rational ClearCase**
- Brian A. White, *Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction*. Boston: Addison-Wesley, 2000.

Team Development

Objectives

After completing this lab, you will be able to:

- Compare and merge model files in Software Architect

Given

The following lab artifacts can found in the `DEV396` folder:

- UML2 models with a base model and 2 contributors (`ACMECompareModel.zip`)

Scenario

In this lab, you will compare and merge two models that have been modified in separate development streams. Normally, when a developer checks in the model and discovers that another developer made changes to the same model, the first developer's changes are overwritten. However, with the compare and merge capability of Software Architect, the changes from both developers are merged into a single model.

Task 1: Import the Merge Model

In this task, you will import the `CompareModel` project containing the models to compare and merge.

1. On the **File** menu, click **Import**.
2. In the list box, select **Project Interchange** as the import source and then click **Next**.
3. Next to the **From zip file** field, click the **Browse** button and find `ACMECompareModel.zip` in the `DEV396` folder. Select the file and click **Open**.
4. Select `CompareModel` in the `Import Projects` list.
5. Click **Finish**.

Task 2: Compare Two Contributors

In this task, you will use Software Architect's compare and merge feature to compare two models.

1. Create the comparison by using the models you imported by completing the following steps:
   a. Select the `Contributor1.emx` and `Contributor2.emx` files in the Model Explorer.
   b. Right-click `Contributor2.emx` and click **Compare with > Each other**. Software Architect compares the two models.

TIP: You can double-click the Compare Editor tab to expand it to the full screen.

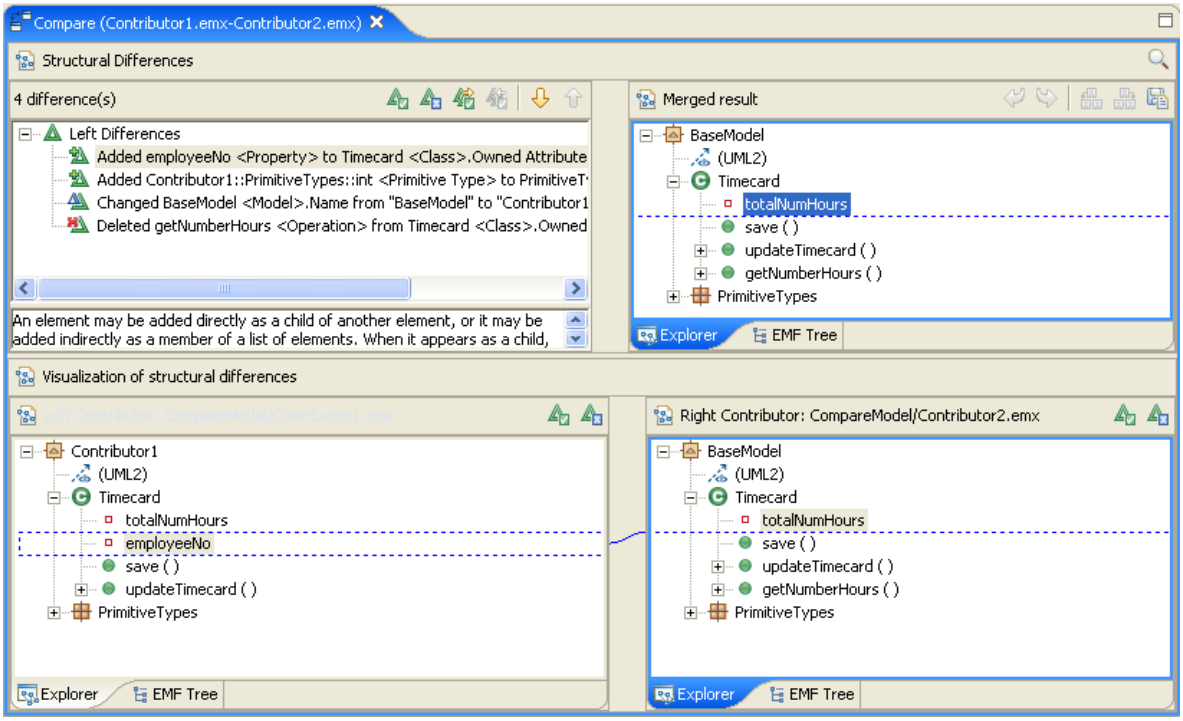


Figure 1: *Merge-Compare visualization*

2. Edit the merged model:
   - In the **Structural Differences** list box, right-click `Changed BaseModel <Model>.Name from "Base Model" to "Contributor1"` difference and then click **reject**. The new merge result is automatically updated.

> *TIP:* Right-click the difference and click **Reject** to omit differences in the new the merged model.

After you have completed this merge, the results appear as follows:

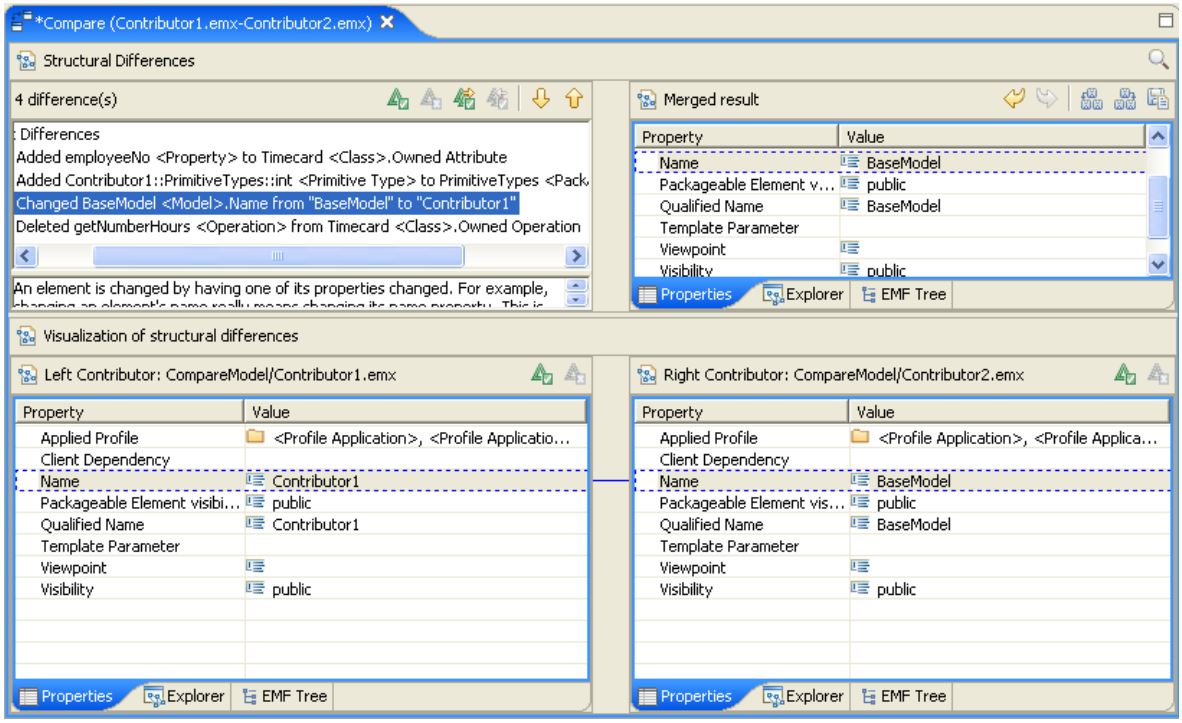


Figure 2: *Merge-Compare visualization after changes*

3. Examine the other differences in the merged model. After you are finished, close the Compare editor without saving.

Task 3: Merge Three Contributors

In this task, you use Software Architect's compare and merge feature to merge model elements from three contributors. You will use two models (`Contributor1.emx` and `Contributor2.emx`), and their common ancestor (`BaseModel.emx`), compare the differences, and then combine them into a single model.

1. Create the comparison between the three models you imported into `CompareMerge` project.

> *TIP:* Select the `BaseModel.emx`, `Contributor1.emx`, and `Contributor2.emx` files in the Model Explorer, right-click them, and then click **Compare with > Each other**.

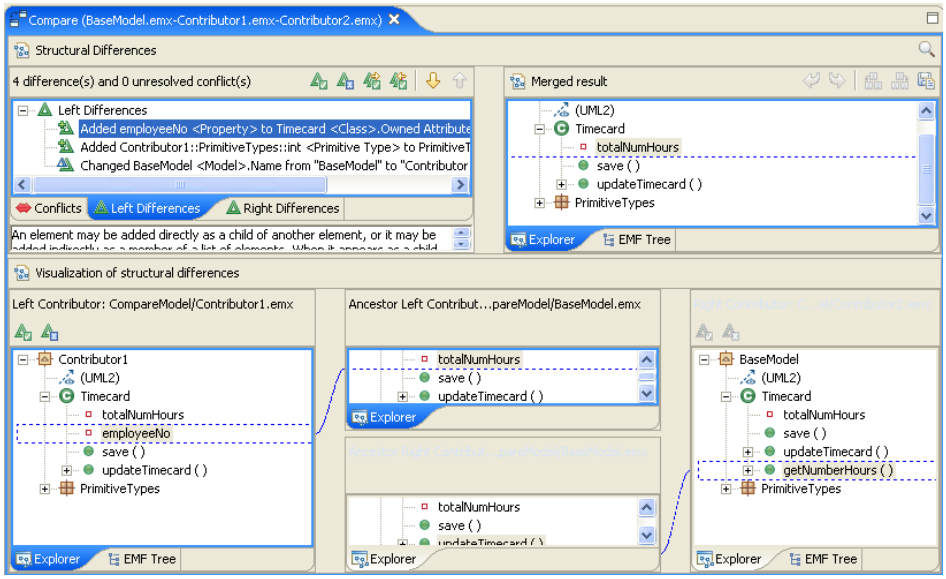


***Figure 3**: Three contributors compare merge visualization*

2. Merge changes by choosing desired differences and save a copy of the merged result:
   a. Compare model differences by toggling the **Conflicts, Left Differences,** and **Right Differences** tabs in the Structural Differences view. As we begin the merge process, note:
      - No conflicts exist between the three models.
      - Contributor1 model has three differences.
      - Contributor2 model has one difference.
   c. In the **Left Differences** tab of the **Structural Differences** view, reject the `BaseModel` name change by right-clicking the difference and clicking **Reject**.

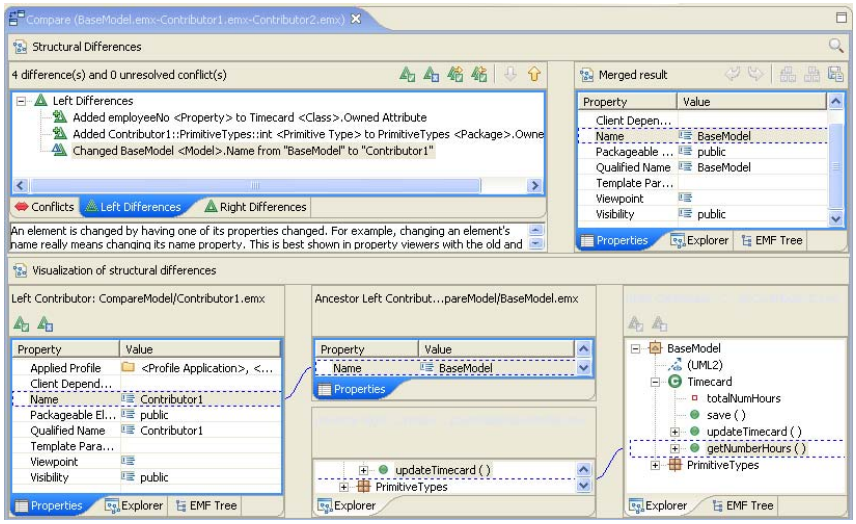


***Figure 4**: BaseModel name change difference*

d. In the **Left Differences** tab of the **Structural Differences** view, accept the following changes by right-clicking the difference and clicking **Accept:**

- Added employeeNo <Property> to Timecard <Class>.Owned Attribute.
- Added Contributor1::PrimitiveTypes::int <Primitive Type> to PrimitiveTypes <Package>.Owned Member.

e. In the **Right Differences** tab of the **Structural Differences** view, accept the following changes by right-clicking the difference and clicking **Accept:**

- Added getNumberHours <Operation> to Timecard <Class> Owned Operation.

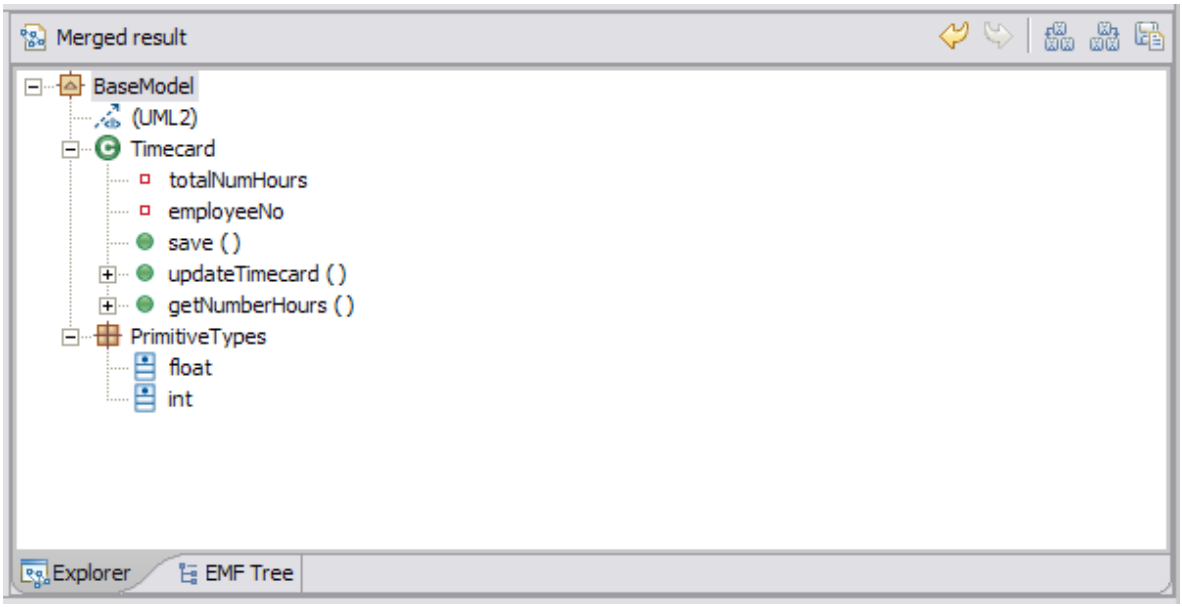


***Figure 5**: Explorer listing in the Merged Result view*

f. As a result of the merging, the model has changed as follows:

- A new attribute `employeeNo` has been added to the base model.
- A new primitive type `int` has been added to the base model.
- A new method `getNumberHours` has been added to the base model.

g. Save the results of the merge:

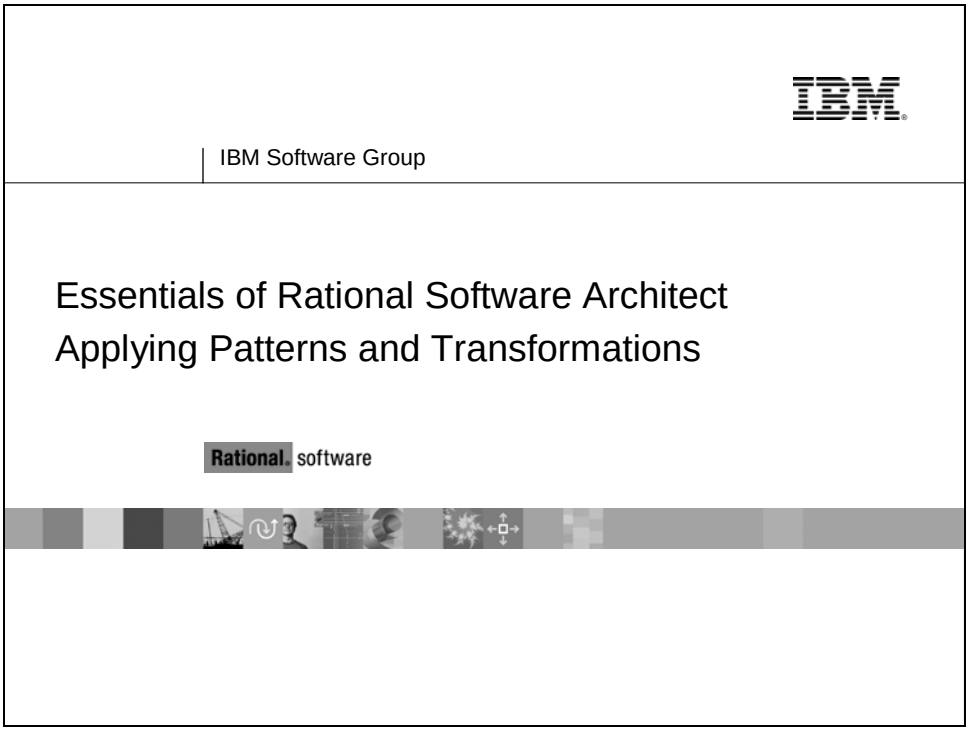
- Note that we have decided that `Contributor2.emx` is the model file that we will use moving forward. As such, we will want to save our merged results to that file.
- Click **File** > **Save**.
- At the **Save** dialog, select **Right** as the `Contributor2.emx` file is the right contributor.

h. Close the **Compare Editor**.

i. In the **Model Explorer**, open the `Contributer2.emx` model.

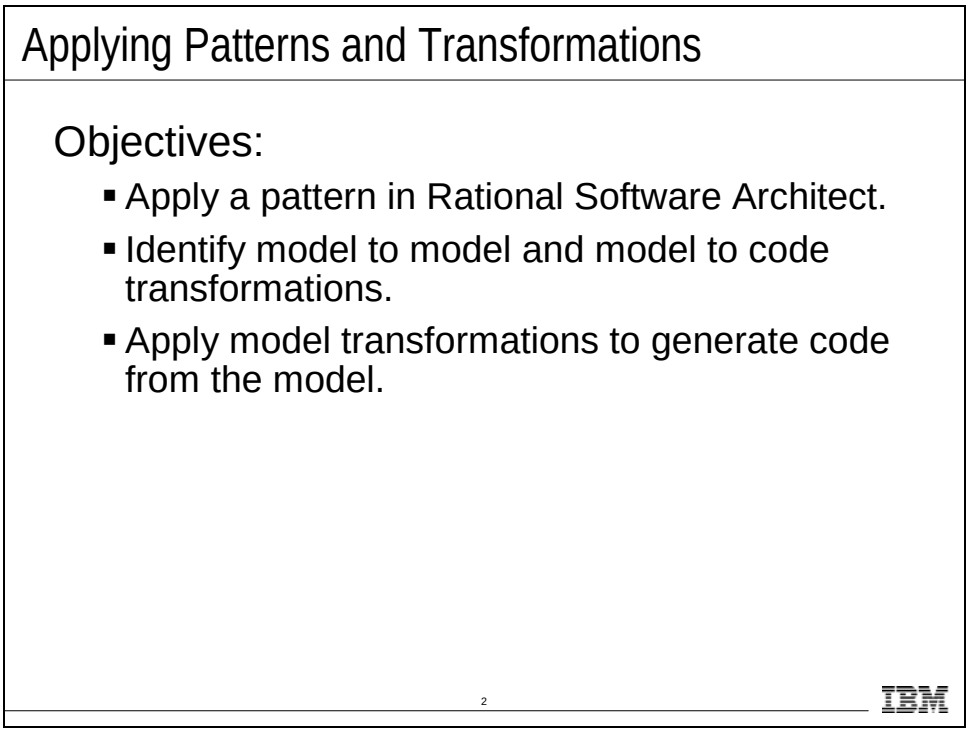
j. Review the elements in the `Contributor2` model and compare to the results from the **Merge Editor** shown in Figure 5.

▸ ▸ ▸ Applying Patterns and Transformations



Topics

Module Objectives



This module introduces patterns and transformations in Rational Software Architect. Patterns offer a way to capture, reuse, and share solutions to common problems. Transformations offer a way to transform an entire model or part of a model (down to a single class) of one type to another.

These features, when used within a larger development team, help automate routine modeling and coding tasks. Patterns also allow developers to share solutions and best practices.

Introduction to Model Transformation

Where Are We?

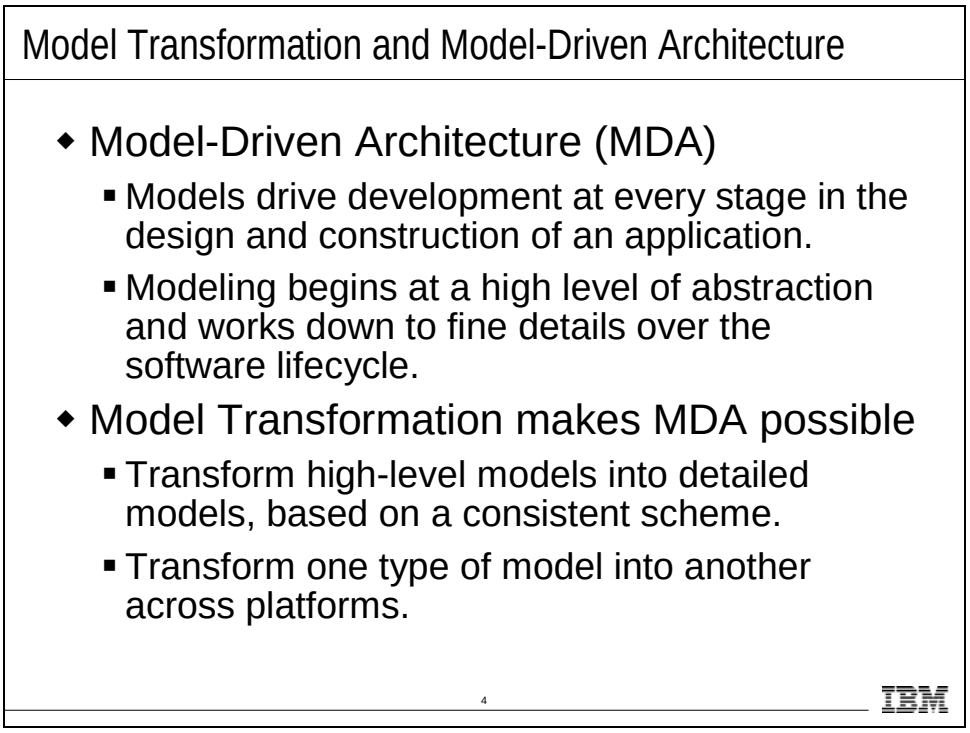
- **Introduction to Model Transformation**
- Applying Transformations
- Applying Patterns

3

IBM

This section introduces transformations in Software Architect.

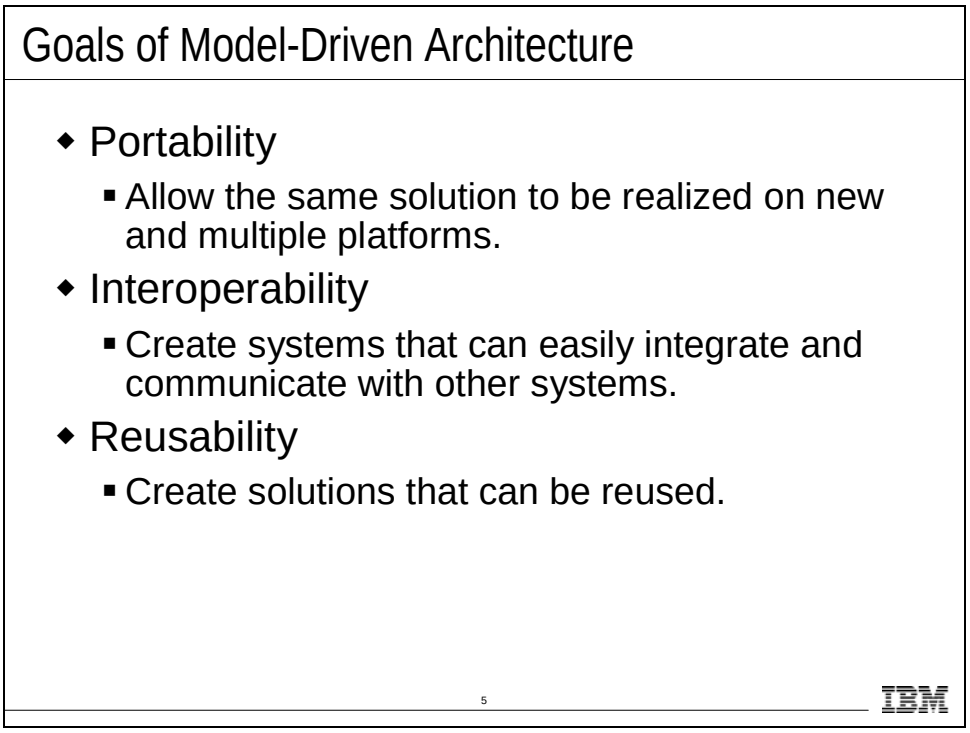
Model Transformation and Model-Driven Architecture



The concept of **model transformation** (see Kleppe et al. p. 24), using a tool to transform one model into another based on a standard scheme (a transformation definition), makes model-driven architecture possible:

- You take an early, platform-independent model and transform it into potentially multiple platform-dependent models.
- Abstract, platform-independent models describe solutions that may be reused in many contexts.
- Finally, transformation definitions can be standardized to help automate implementation and make the executable code more consistent across the application.

Goals of Model-Driven Architecture



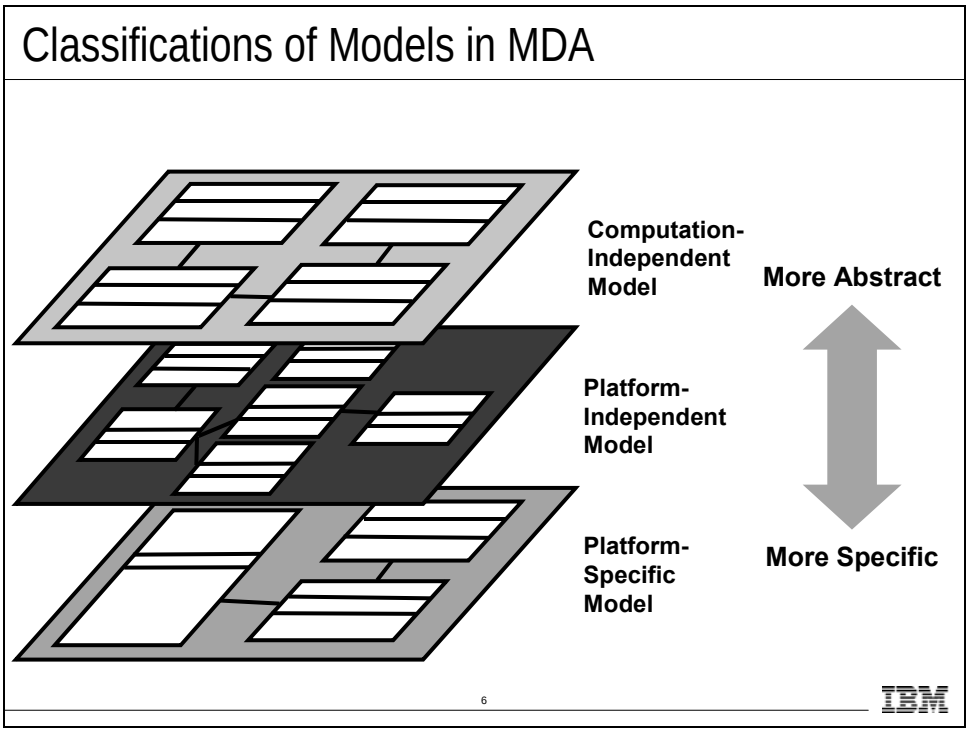
There are three primary goals for MDA, which make it well-adapted for enterprise application development:

Portability: To allow the same solution to be realized on new or multiple platforms.

Interoperability: To create systems that can easily integrate and communicate with other systems and use a variety of resource applications.

Reusability: To create solutions that can be reused in many different applications in different contexts.

Classifications of Models in MDA



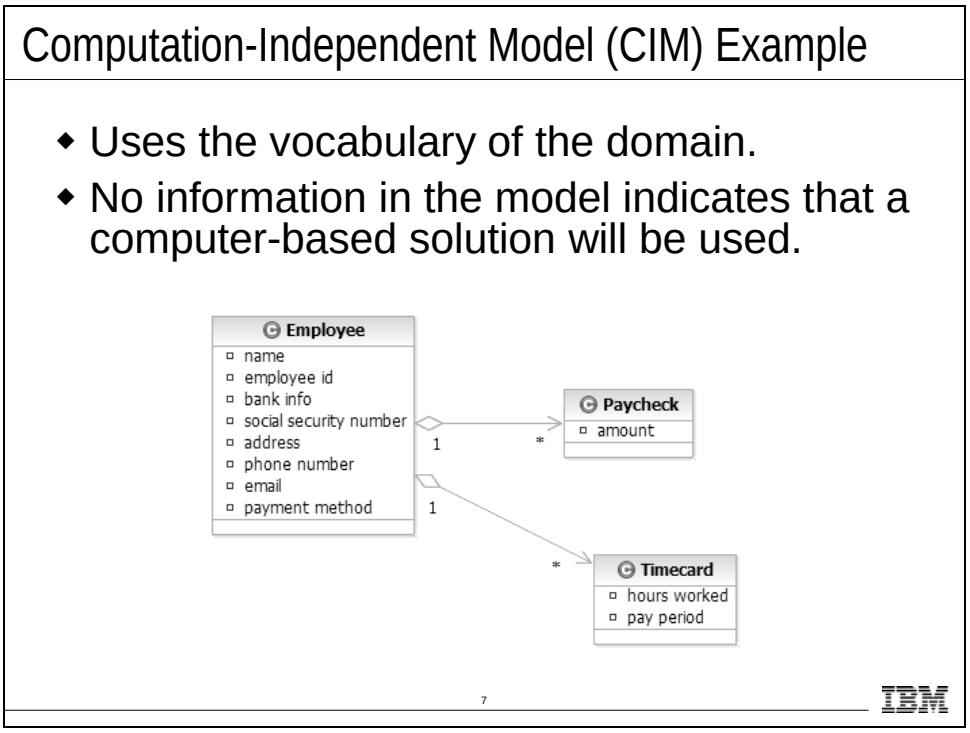
Computation-Independent Model (CIM): Also referred to as a domain or business model, the CIM presents the system at the highest level of abstraction. The goal of the CIM is to model the problem entirely in business terms and without getting into the solution or how it might be implemented. CIMs help domain experts share business requirements with software development experts.

Platform-Independent Model (PIM): A PIM is used by software architects and designers to describe the software solution at a high level, independent of the solution's deployment platform. This high-level definition of the solution can then be translated into multiple platform-specific models.

Platform-Specific Model (PSM): A PSM specifies a combination between the details found in the PIM with the details representing how a solution can be implemented on a platform. As the name implies, we are now including details about a specific platform in the model.

Code: Application code is also a model of the application at a relatively low level of abstraction.

Computation-Independent Model (CIM) Example

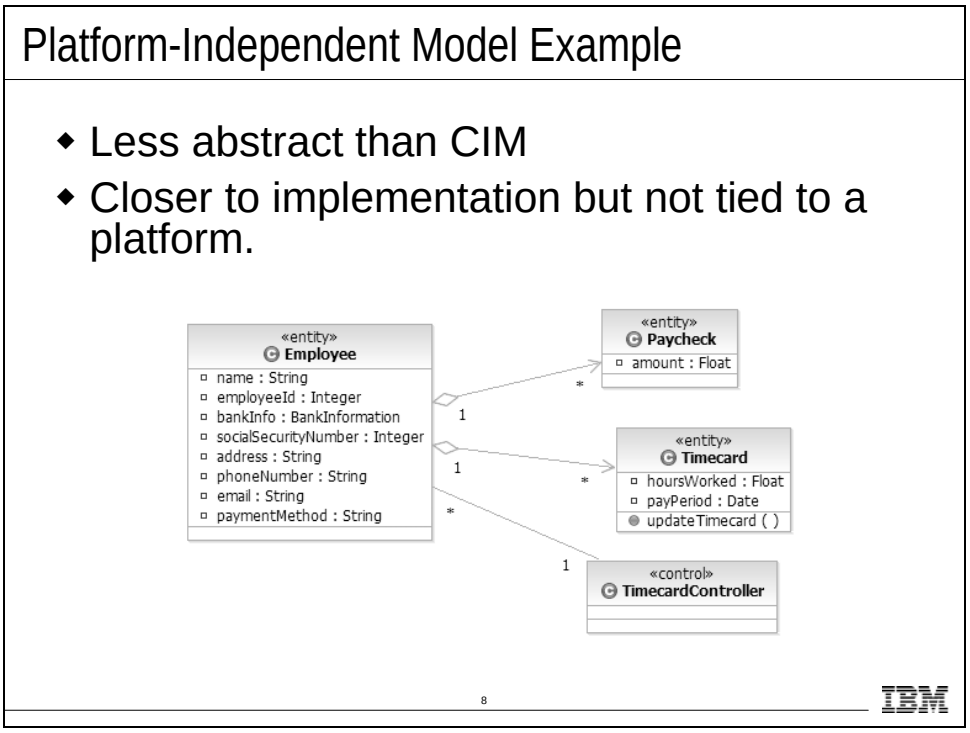


The slide presents a portion of a CIM. The CIM is based on the information from the Payroll application example used throughout the course.

The CIM uses the vocabulary of the domain. For example, a manufacturing domain model might include terms such as “supply chain,” “employee,” and “purchase order” to describe business elements in that domain. You can model some elements in the CIM in a software model, but the CIM must represent the business system, independent of the software system.

An important aspect to note is that the model introduces the main concepts found in our problem domain, but it does not tie us to any type of solution. There is no information found in the model that indicates that a computer-based solution will be used.

Platform-Independent Model Example



The slide presents a platform-independent model (PIM) for the portion of the payroll application presented in the previous slide.

A target platform for a software system can be one or more of the following:

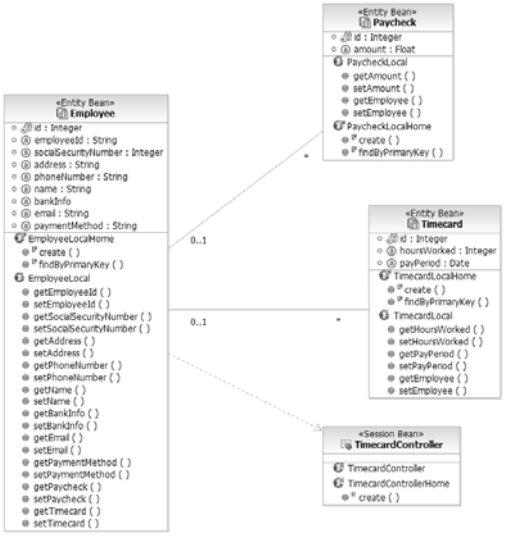
- A specific operating system
- A specific computer processor
- A specific language
- A specific code library

By creating a PIM for the domain, a software architect and designer can better understand the domain to develop repeatable solutions, saving the business time and money.

Platform-Specific Model Example

Platform-Specific Model Example

- Less abstract than PIM
- Closer to implementation
- J2EE elements captured in model



9

The slide presents a platform-specific model (PSM) for the portion of the payroll application presented in the previous slide.

The PSM expresses the PIM using the terms and constructs of the specific technology or platform in which the solution will be implemented. Compare this model of the application in J2EE with the platform-independent model on the previous slide.

Applying Transformations

Where Are We?

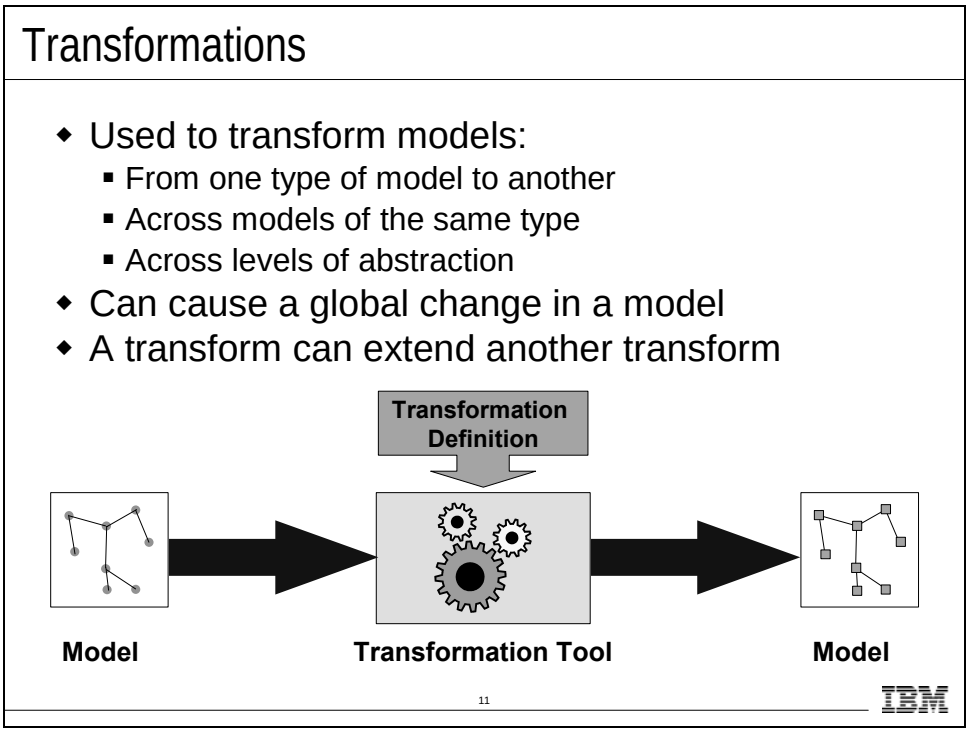
- Introduction to Model Transformation
- **Applying Transformations**
- Applying Patterns

10

IBM

This section introduces applying transformations in Software Architect

Transformations



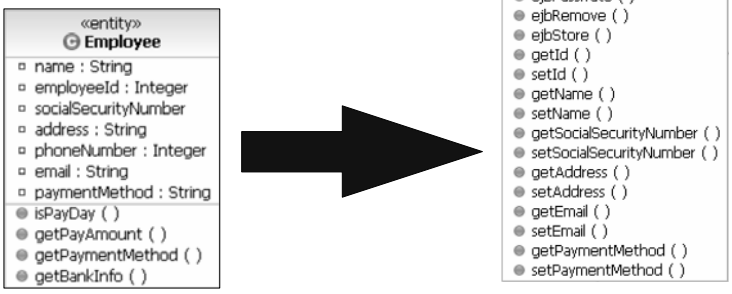
The most familiar type of transformation, already available in some form in many development environments, is automatic code generation from a visual model or "roundtrip engineering." However, there are other potential transformations, such as transformations from one model to another of the same model type, transformations between models within the same level of abstraction, and even transformations within the same model.

Transformations can cross metamodels, models, and levels of abstraction (such as from a high-level design model to implementation model). In Software Architect, one transformation can be developed to extend or to specialize another for specific applications.

Applying Transformations in Software Architect

Applying Transformations in Software Architect

- The following transformations are built into Software Architect:
  - UML to C++
  - UML to EJB
  - UML to Java
- Model-to-model transformations can be custom built.

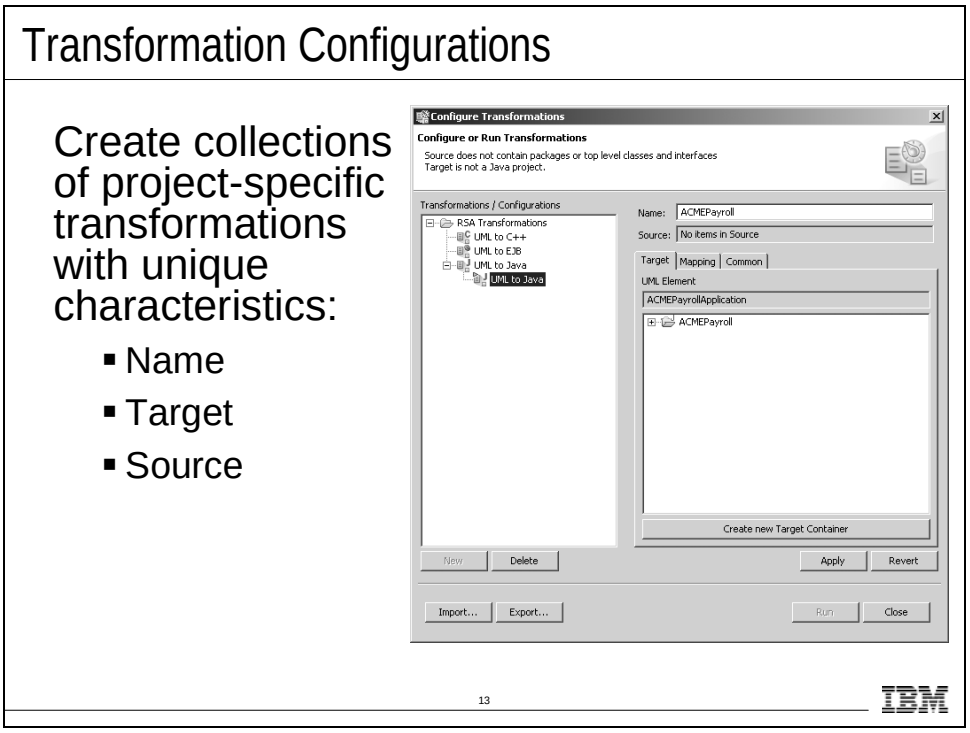


Platform-Independent Model

Platform-Specific Model

12

Transformation Configurations



A transformation configuration is an instance of a transformation that includes information that is used by all transformations, such as a unique name, the source, and the target of the transformation. A transformation uses the information you provide in a transformation configuration when it executes.

A transformation configuration can also include properties specific to a given transformation. When a configuration is run, an instance of the transformation is created and is executed with the properties defined in that configuration.

Demo: Apply a Transformation

Demo: Apply a Transformation

The instructor will now show you how to:

- Create a UML to Java transformation
- Transform UML classes to Java classes
- Review the code

14

IBM

Watch your instructor perform a UML to Java code transformation.

Lab: Apply a Transformation

Lab: Apply a Transformation

- Complete the following tasks:
  - Create UML project
  - Create UML Classes
  - Transform UML classes to Java classes
  - Review the code

15

IBM

Complete the Apply Transformation lab.

Applying Patterns

Where Are We?

- Introduction to Model Transformation
- Applying Transformations
- **Applying Patterns**

16

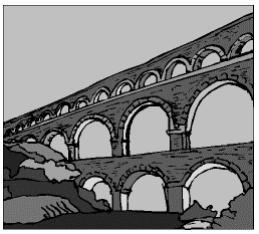
IBM

This section introduces applying patterns in Software Architect.

Design Patterns

Design Patterns

- A design pattern is a named solution to a problem that can be applied in certain contexts.
- Patterns:
  - Can make developers more productive
  - Promote learning and the reuse of effective solutions
  - Can be used in all phases of software development



17

IBM

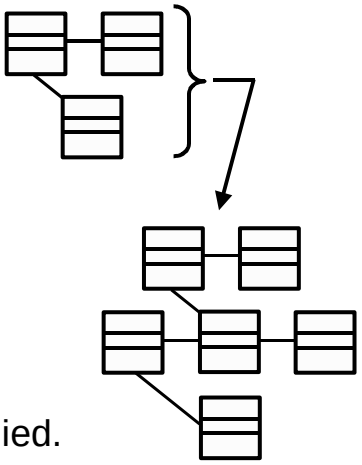
Craig Larman defines a design pattern as "a named problem/solution pair that can be applied in new contexts, with advice on how to apply it in novel situations and discussion of its tradeoffs" (p. 218). Patterns provide a standard way of capturing and naming solutions, programming idioms, and best practices. As more developers research and understand patterns, patterns become a standard way for practitioners to communicate and share what they know with others.

For an architect leading a large team of developers, with tools to share and apply patterns automatically, patterns can become a way to enforce coding standards and ensure consistency in the way a design for a system is implemented. For the developer, a set of carefully selected patterns, customized for a specific project or application, can reduce mundane coding tasks and head off confusion about how to approach the implementation of design elements in the system.

Patterns in Software Architect

Patterns in Software Architect

- A design pattern in Software Architect is a transformation that occurs:
  - Within a single metamodel
    - UML to UML
    - Java to Java
  - Within the same model and at the same level of abstraction:
    - Design model to design model
    - Code to code
- Patterns usually affect small areas of the model when applied.



18

IBM

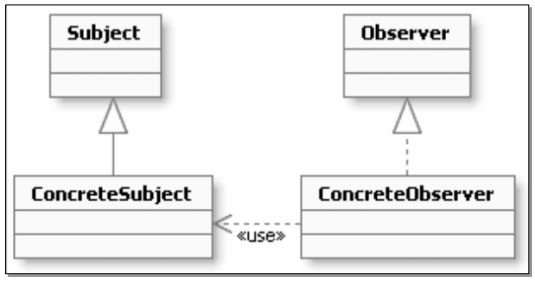
Patterns, which can be developed and applied in Software Architect, are an example of a model transformation. We can think of patterns as a transformation that occurs primarily within the same metamodel (UML to UML), within the same model, and at the same level of abstraction. So if you have a common design pattern, such as the observer pattern, you can apply it as written inside a high-level design model or add implementation details to the pattern and apply it in a platform-specific model.

Patterns in Software Architect (cont.)

Patterns in Software Architect (cont.)

In Software Architect, patterns:

- Are embodied by executable code (in an Eclipse plug-in)
- Can be shared as Reusable Asset Specification (RAS) assets
- Can have associated transformation extensions
  - Transform model elements to code.



19

IBM

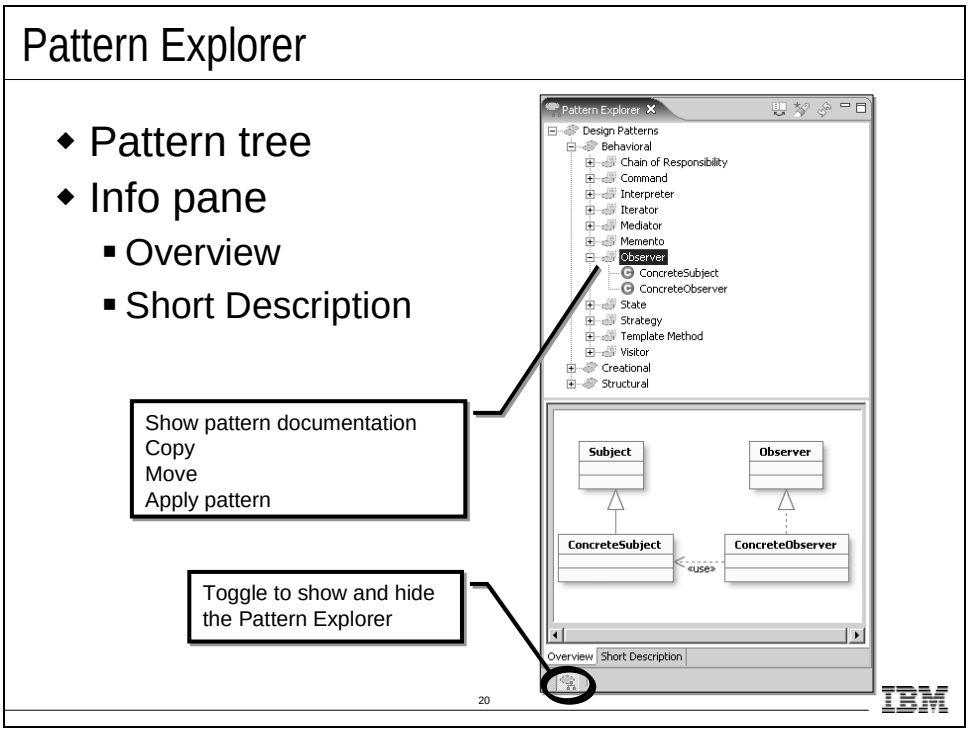
In Software Architect, a pattern is optimized to be applied and elaborated step by step, by the user, within the same model and within the same level of abstraction. Patterns can be applied to change the structure of existing elements within a model or used as templates to create new elements.

In Software Architect, patterns are typically specialized, and their applications localized within a model. Transformations often involve global changes to a model, such as the case of converting code from one language to another or changing a UML visual model to code or to a different model type.

Patterns are created and stored in Eclipse plug-in type projects, which can be deployed and shared as Reusable Asset Specification (RAS) asset (.ras) files. The RAS is an Object Management Group (OMG) standard for sharing reusable assets.

The RAS file format is based on the compressed archive (.zip) format but packaged with the artifacts is a manifest including metadata in XML format for storage and retrieval in RAS repositories.

Pattern Explorer



The Pattern Explorer, associated with the **Modeling** Perspective, is split into two panes:

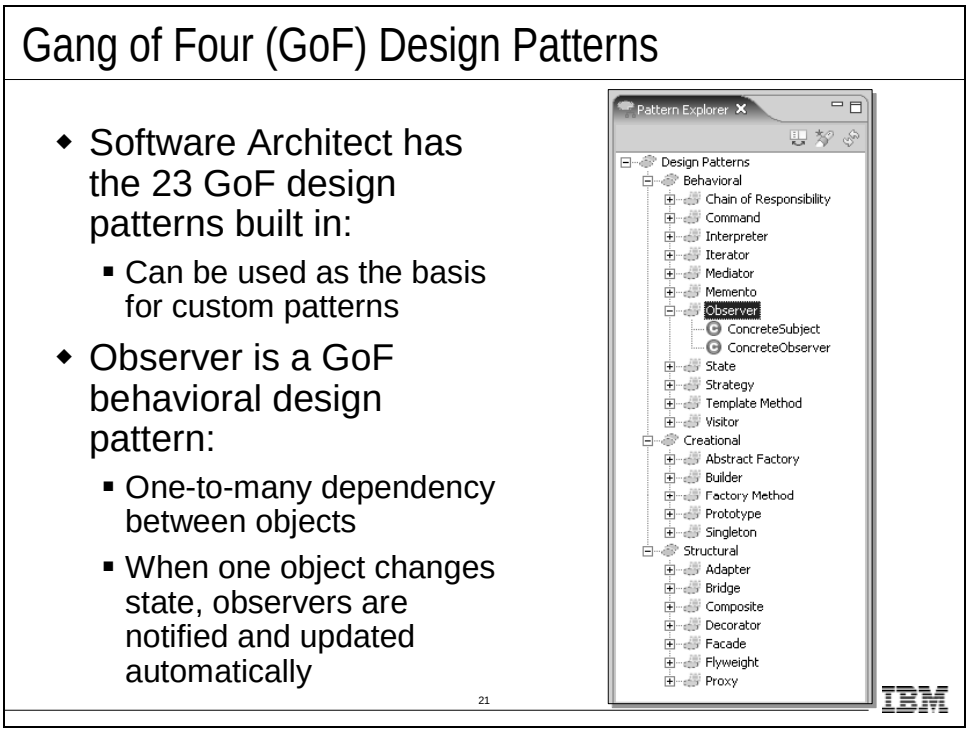
Pattern tree: Presents a hierarchical list of patterns, organized into pattern groups.

- By right-clicking items in the pattern tree, you can copy and move patterns and groups and delete and rename groups.
- By right-clicking a pattern, you can choose to apply the pattern using the Apply Pattern wizard or open the pattern's full documentation in Help.
- By clicking and dragging a pattern onto the drawing surface, you can create a pattern instance that can have its parameters bound by drag and drop actions.

Info pane: The information pane has two tabs which provide information about the pattern selected in the pattern tree:

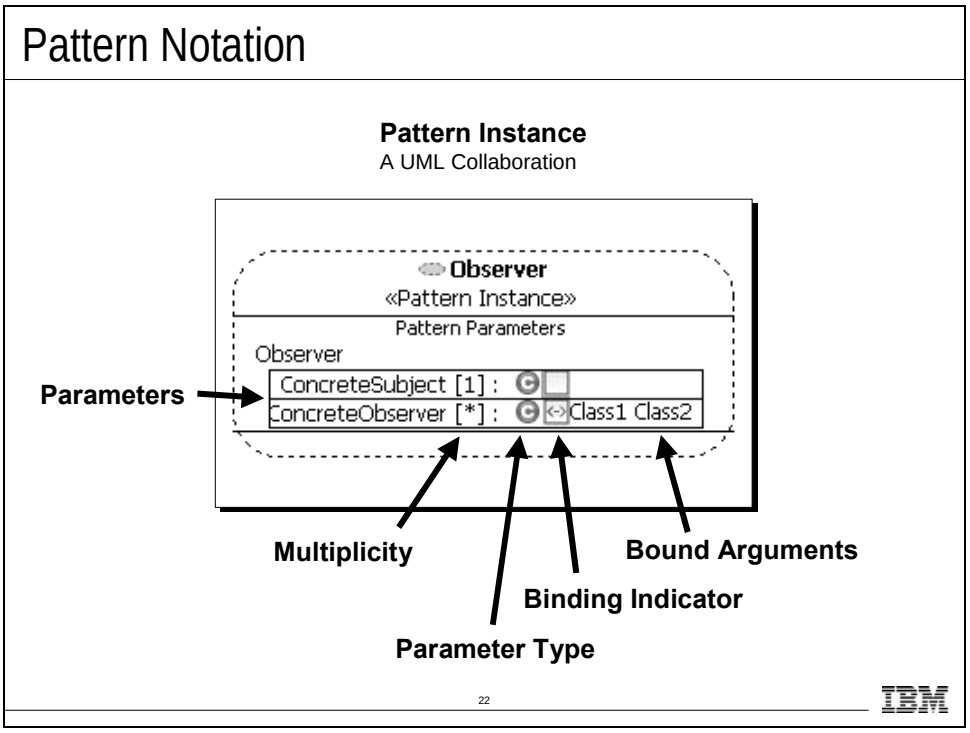
- The **Overview** presents a participants class diagram, showing what objects participate in the pattern.
- The **Short Description** provides a brief summary of the problem and the solution the pattern provides. If you have a pattern participant selected, the Short Description describes the participant.

Gang of Four (GoF) Design Patterns



The observer pattern, one of 23 Gang of Four patterns included with Software Architect, is used in cases when the ConcreteSubject object needs to inform other objects that have registered an interest that some change inside it has occurred. When these objects have been notified, they interrogate the ConcreteSubject object to see if the change requires them to take action (see Gamma et al., pp. 293–303).

Pattern Notation

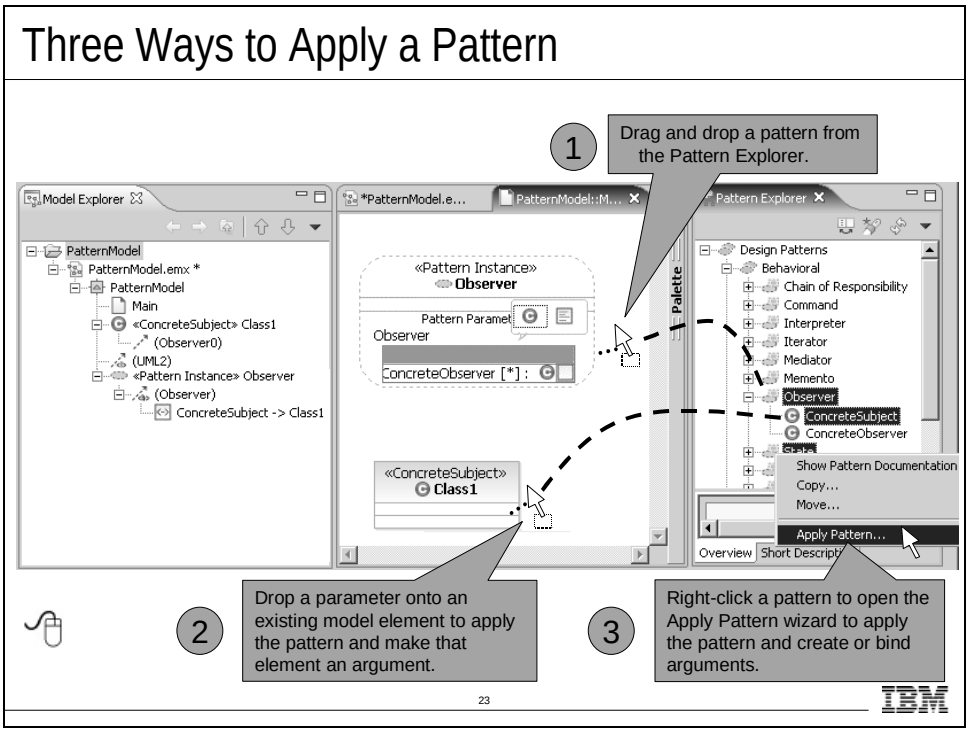


Pattern instances in Software Architect are represented with UML collaborations stereotyped «pattern instance».

The pattern instance includes the following features:

- **Parameters:** Each parameter in the pattern instance takes an argument. When the pattern instance is created, its parameters show the unbound parameter icon as an empty blue box. You can add or create an argument using the action bar or by dragging an existing element from the diagram or Model Explorer and dropping it on the parameter. When bound, the icon changes to a blue box containing a double arrow.
- **Parameter Multiplicity:** The parameter's multiplicity is shown in brackets after the parameter name.
- **Parameter Type:** After the multiplicity, an Eclipse-style icon or text shows the parameter type (for example, class, interface, or operation).
- **Binding Indicator:** An icon or text that shows whether the parameter has an argument bound to it. An empty blue box indicates that no arguments are bound to the parameter. A binding icon shows that arguments are bound to the parameter.
- **Arguments:** One (or more, if the pattern allows it) arguments may be bound to the parameter.

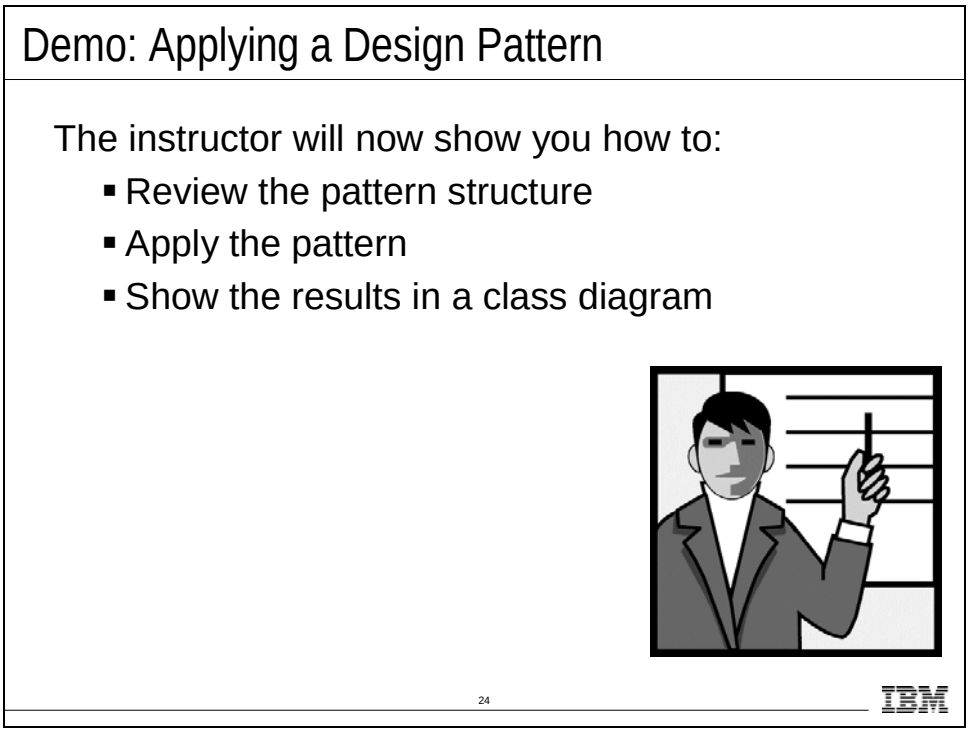
Three Ways to Apply a Pattern



The most intuitive way to create a pattern instance in your model is to the drag the pattern from the Pattern Explorer and drop it onto an open diagram. If you click or hover the mouse over a parameter in the pattern instance, the action bar will appear, allowing you either to choose either to select an existing class in the model or to create a new argument. You can also drag and drop an existing model element, either from the diagram or from the Model Explorer view, onto a pattern instance's parameter to bind that element to the parameter.

If you have an existing element in a diagram that you would like to make into an argument for one of the pattern's parameters, drag the appropriate participant from the Pattern Explorer and drop it on to the element in the open diagram. Doing this will create a new pattern instance in the diagram containing the selected parameter and the model element it was dropped on set as its argument.

Demo: Applying a Design Pattern



Watch your instructor demonstrate how to apply a pattern in Software Architect.

Lab: Apply a Design Pattern

Lab: Apply a Design Pattern

Tasks:

- Create a UML modeling project
- Select a pattern
- Apply a pattern
- Create a class diagram

25

IBM

Complete the Apply a Design Pattern lab.

Review

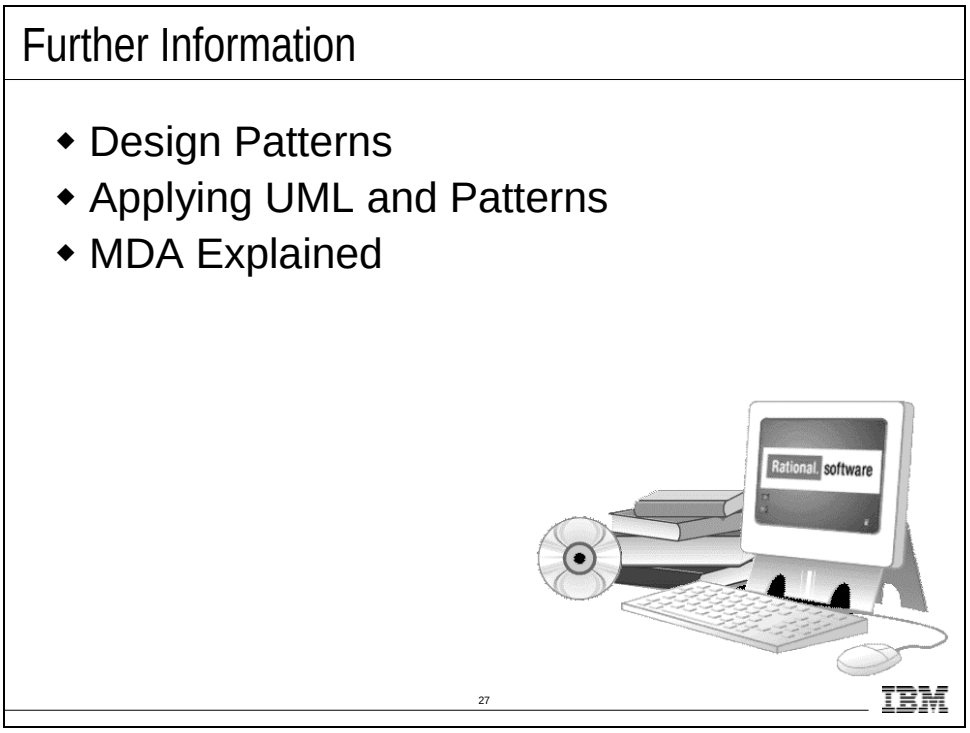
Review

- In MDA, what kind of model shows the technology-specific implementation details for the system?
- What is a pattern instance?
- Describe three ways to apply a pattern in Software Architect.

26

IBM

Further Information



- Erich Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1995.
- Anneke Kleppe et al. MDA Explained The Model Driven Architecture: Practice and Promise. Boston: Addison-Wesley, 2003.
- Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process. Upper Saddle River, NJ: Prentice Hall, 2002.

Related Rational Training

Related Rational Training

DEV325: Essentials of Model-Driven Architecture

28

IBM

Apply Transformation

Objectives

After completing this lab, you will be able to:

- Transform UML classes to Java classes
- Explore the transformation results

Given

No lab artifacts are required.

Scenario

In this lab, you will model a number of classes and then utilize Software Architect's Transformation capabilities to generate Java code.

Task 1: Create the UML Project

In this task, you will create a UML Modeling project to for the application example.

1. Close all open models and open the **Modeling** Perspective.

   TIP: On the **File** menu, click **Close All** to close all the open models and clear the Model Explorer.

2. Create a new UML project and name it `ACMETransformProject`.
   a. Enter the following values in the UML Modeling dialog and then click **Finish**:
   b. On the **File** menu, click **New > Project**.
   c. In the Wizards list box, click **UML Project** and then click **Next**.
   d. Name the project `ACMETransformProject` and then click **Next**.
      - Complete the Create **Templates**: `Blank Model`
      - **File name**: `TransformModel`
      - Select **Create default diagram in the new model** and set the diagram type as **Class Diagram**

Task 2: Create UML Classes

In this task, you will add a number of classes to the Main class diagram in the `TransformModel` model.

1. Open the **Main** class diagram in the `TransformModel` model.
2. Right-click the `TransformModel` and select **Add UML > Class**. Name the class `Employee`.
3. Right-click the `TransformModel` and select **Add UML > Class**. Name the class `PhoneNumber`.
4. Right-click the `TransformModel` and select **Add UML > Class**. Name the class `Zipcode`.
5. Add the classes to the **Main** class diagram.
6. Add directed associations from the Employee Class to the `Zipcode` class and from the `EmployeeClass` to the `PhoneNumber` class.

> *TIP:* In the Toolbox, click the **Class Diagram** tab, click on the arrow beside the **Association Arrow** tool and select **Directed Association**. Click on the `Employee` class, drag the connector to the `Zipcode` class, and click inside it.

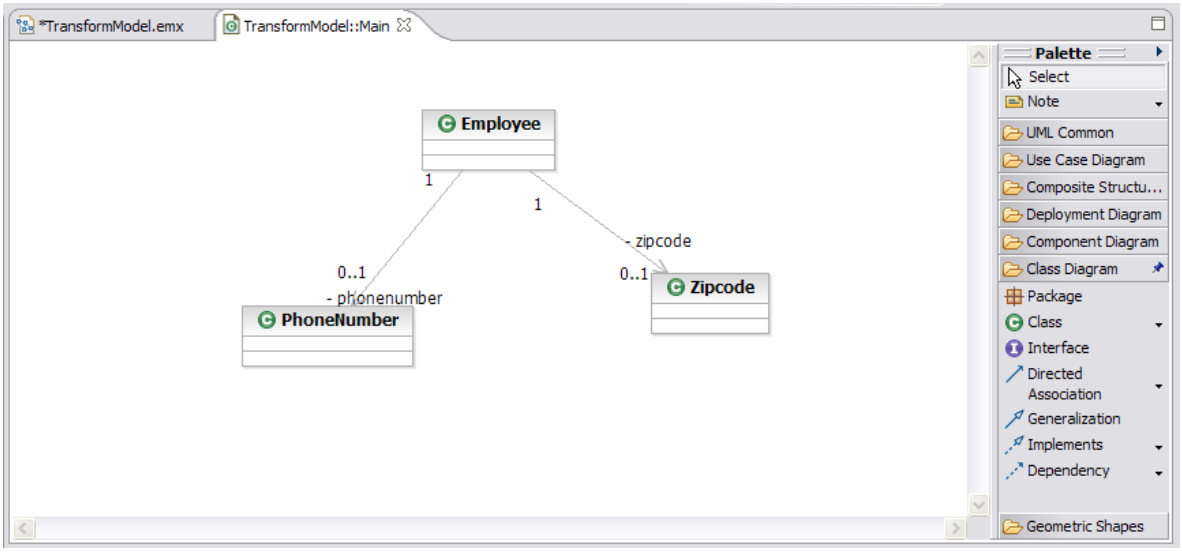


***Figure 1**: Classes with Directed Associations*

7. Add a new attribute to the PhoneNumber class:

- On the Main class diagram, right-click the PhoneNumber class and select **Add UML > Attribute**.
- Name the attribute **theNumber** and set its type to **String** using the following format:
  - – theNumber : String
- Add a new operation to the PhoneNumber class:
- On the Main class diagram, right-click the PhoneNumber class and select Add UML > Operation.
- Name the operation isPhoneNumber accepting a String parameter and having a return type of Boolean using the following format:
  - + isPhoneNumber (aNumber : String) : Boolean

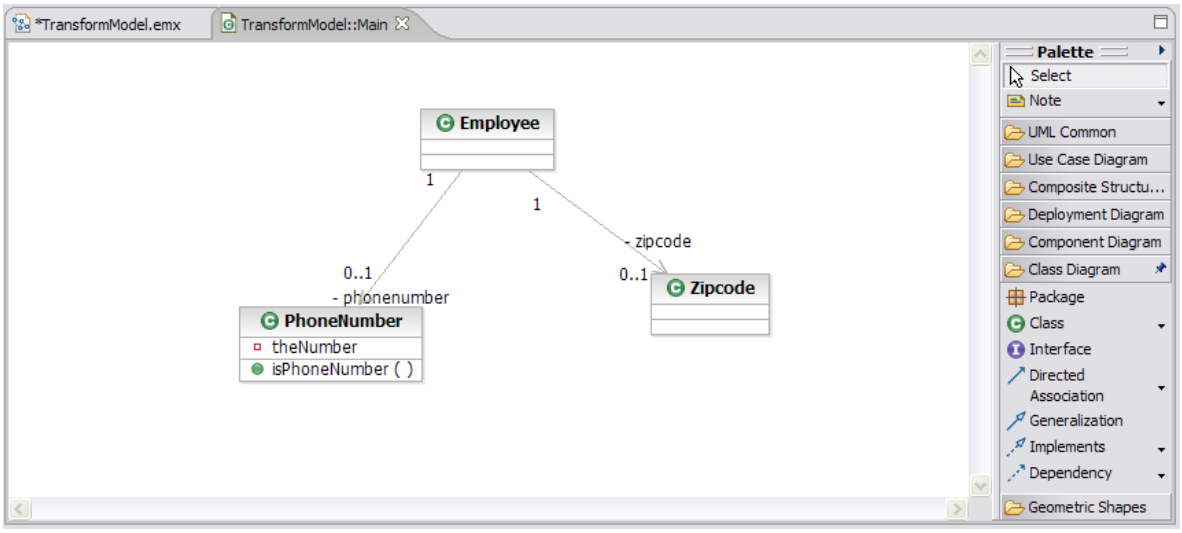


***Figure 2**: Attribute and Operation added to PhoneNumber*

8. Save the project.

Task 3: Transform UML Classes to Java Classes

In this task, you will transform the UML classes in the UML project into Java classes in the Java project.

1. With the **Modeling** Perspective open, expand the `TransformModel` model.
2. In the Model Explorer, right-click the `Employee`, `CPhoneNumber` and `Zipcode` classes and click **Transform >Run Transformation > UML to Java**.
3. In the Run Transformation dialog, click **Create new Target Container**. Enter `TransformTarget` as the project name and then click **Finish**.
4. In the Run Transformation dialog, ensure that `TransformTarget` is selected and then click **Run**.

   The transformation will process and create Java classes in the `TransformTarget` project as appears in Figure 3.

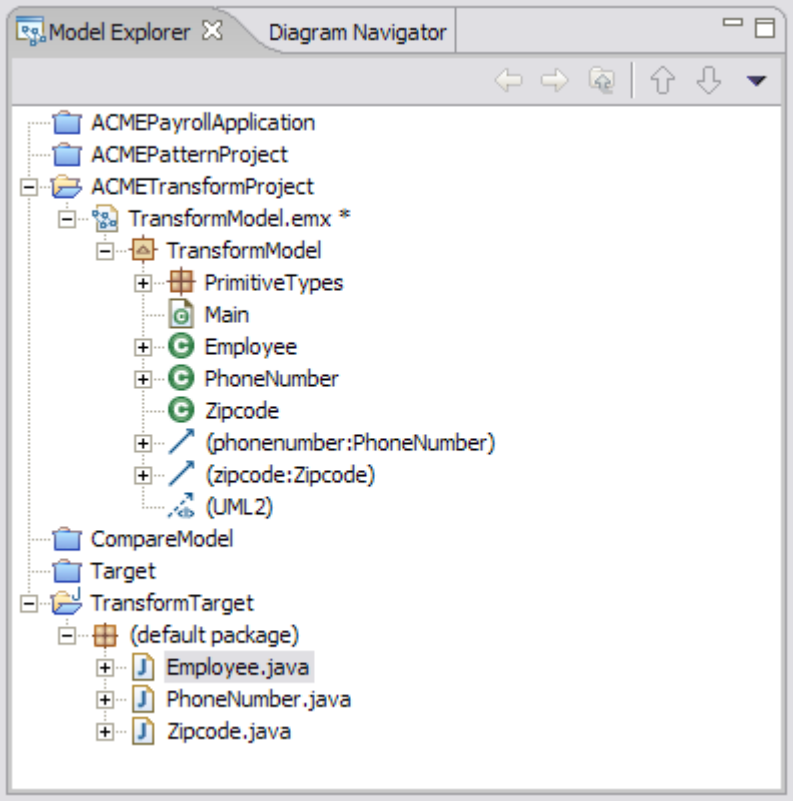


***Figure 3**: Transformed Java objects in Model Explorer*

Task 4: Review the Code

In this task, you will review the Java code generated by a UML to Java transformation.

1. Open the `Employee.java` file in the Java editor.

   TIP: In the Model Explorer, double-click on the `Employee.java` to open the Java editor.

2. Open the `PhoneNumber.java` file.
3. Answer the following questions:
   - Why are there attributes in the Employee class?
   - What value does isPhoneNumber return?

   A sample of `ConcretePayrollObserver.java` file appears in Figure 4.

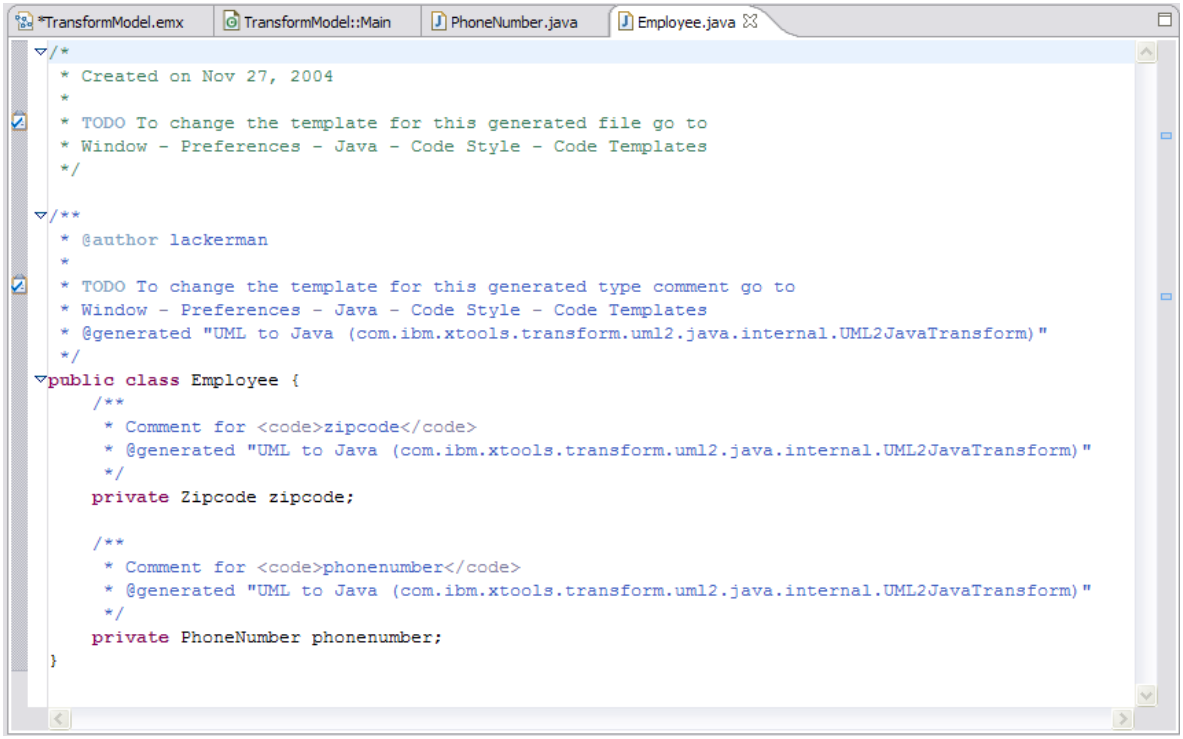


***Figure 4**: Transformation source code in Code Editor*

Apply a Design Pattern

Objectives

After completing this lab, you will be able to:

- Apply a design pattern to a model
- Transform UML classes to Java classes
- Explore the transformation results

Given

No lab artifacts are required.

Scenario

In this lab, you will use Rational Software Architect to apply the Observer design pattern. The Observer pattern will assist the application in publishing ACME new hire employee information to interested subscribers, such as the Payroll Processing and Management Reporting systems.

The Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The pattern contains the following participants and relationships:

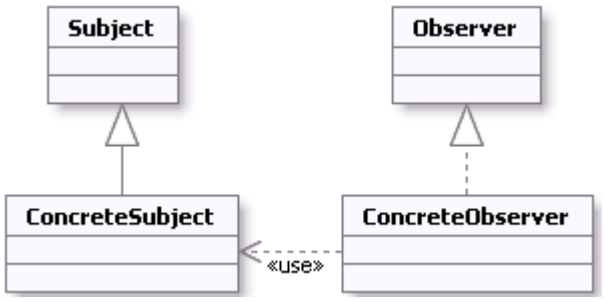


***Figure 1**: Observer pattern class diagram*

- **ConcreteSubject:** Stores the state of `ConcreteObserver` objects and sends a notification to its observers when its state changes.
- **ConcreteObserver:** Maintains a reference to a `ConcreteSubject` object and stores the state

that should stay consistent with the subject's. You will explore the ConcreteObserver's one to many relationship with the ConcreteSubject feature by creating and binding two ConcreteObserver classes.

Task 1: Create the UML Modeling Project

In this task, you will create a UML modeling project for the application example.

1. Close all open models and open the **Modeling** Perspective.

   TIP: On the **File** menu, click **Close All** to close all the open models and clear the Model Explorer.

2. Create a new UML project and name it `ACMEPatternProject`.
   a. On the **File** menu, click **New > Project**.
   b. In the Wizards list box, click **UML Project** and then click **Next**.

   TIP: Selecting the **Show all wizards** check box will display every Architect wizard.

   c. Name the project `ACMEPatternProject` and then click **Next**.
   d. Complete the Create UML Model dialog with the following values and then click **Finish**:
      - **Templates**: `Blank Model`
      - **File name**: `PatternModel`
      - Select **Create default diagram in the new model** and leave the diagram type as **Freeform Diagram**

After you have successfully created the project, the model will contain the following elements:

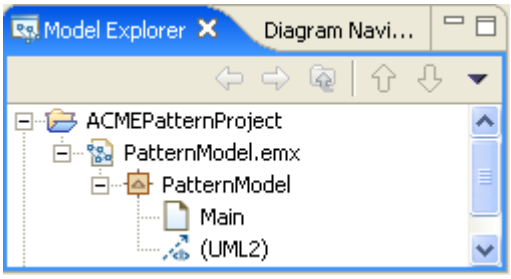


***Figure 2**: Model Explorer with PatternModel model*

Task 2: Select a Pattern

In this task, you will use Software Architect's diagram editor to select the Observer design pattern and place it on a diagram.

1. Open the Pattern Explorer view and then click **Design Patterns > Behavioral > Observer**.

   TIP: On the **Window** menu, click **Show View > Pattern Explorer**.

2. Drag the Observer icon to the `Main` diagram of the `PatternModel`.

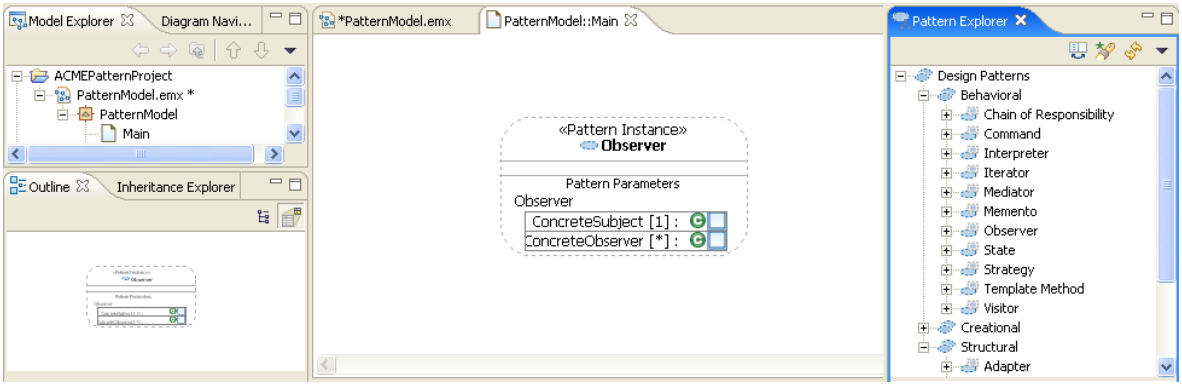


***Figure 3**: The Observer pattern in the diagram editor*

Task 3: Apply a Pattern

In this task, you will use Software Architect's diagram tool to create concrete subject and observer class structures and bind them to the pattern instance. A **concrete class** is a class that implements an abstract data type.

1. Create the `ConcreteSubject` and `Observer` classes automatically. A `ConcreteSubject` or `ConcreteObserver` class may be dynamically created by entering the new class names as its parameters.

   a. Bind an argument to the `ConcreteSubject` parameter and name it `ConcreteEmployeeStatus`.

   TIP: In the diagram editor, hover the mouse on the `ConcreteSubject` parameter so that the action bar appears. In the action bar, click the text icon and then type `ConcreteEmployeeStatus`.

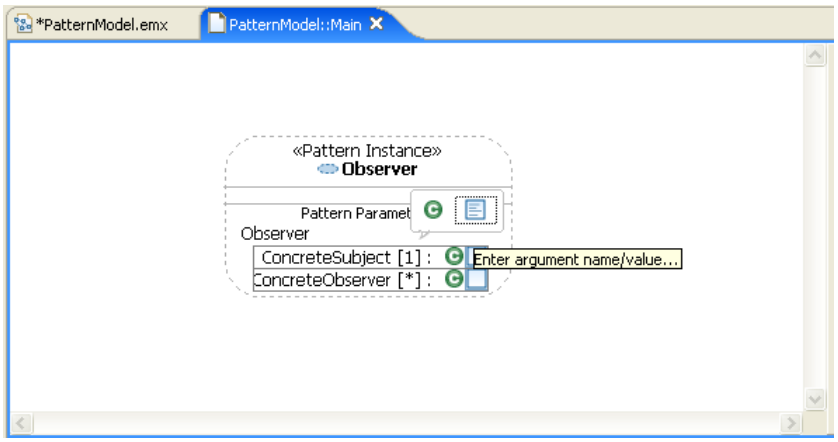


***Figure 4**: Binding action bar*

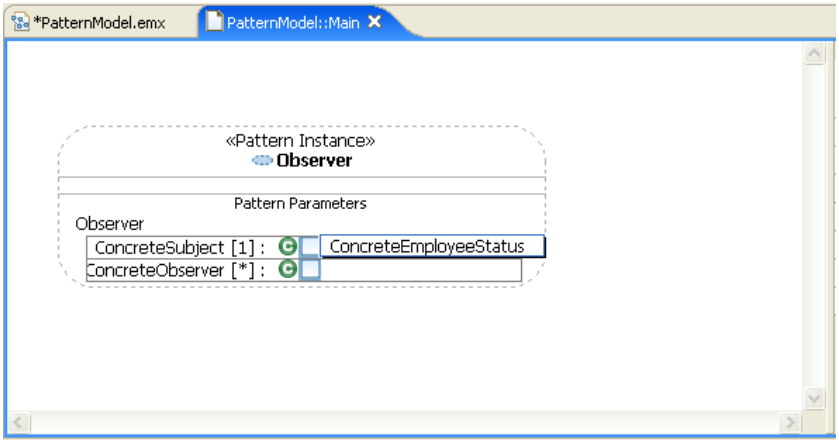


***Figure 5**: Argument name entry*

b. Bind an argument named `ConcreteReportingObserver` to the `ConcreteObserver` parameter**.**

c. Bind an argument named `ConcretePayrollObserver` to the `ConcreteObserver` parameter.

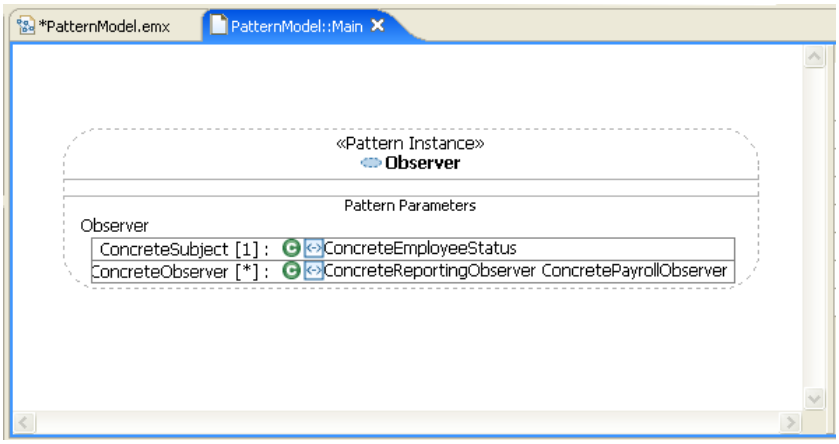


***Figure 6**: Observer pattern bindings*

2. Alternatively, a pattern can be applied with existing classes:
   a. Add a new class to the `Main` diagram and name it `ConcreteHRObserver`.
   b. Drag and drop the `ConcreteHRObserver` on the Observer pattern's `ConcreteObserver` parameter asterisk [*].

   After you have successfully applied the pattern, examine the diagram editor as shown below:

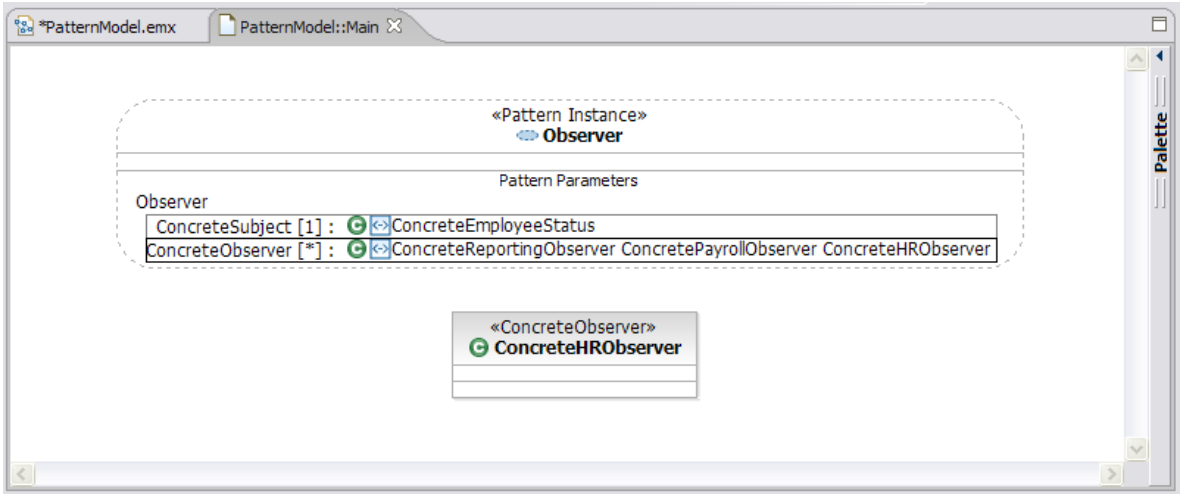


***Figure 7**: Bound pattern*

Task 4: View Pattern Relationships

In this task, you will view the relationships of the classes that comprise the pattern.

1. Right-click in the `Main` diagram and click **Select > All Shapes**.
2. Then right-click on any model element in the diagram and click **Filters > Show Related Elements**.
3. In the **Custom Query** list, click `Show All Relationships [Default]` and click **OK**.

After adding the objects to the diagram, it should appear as follows:

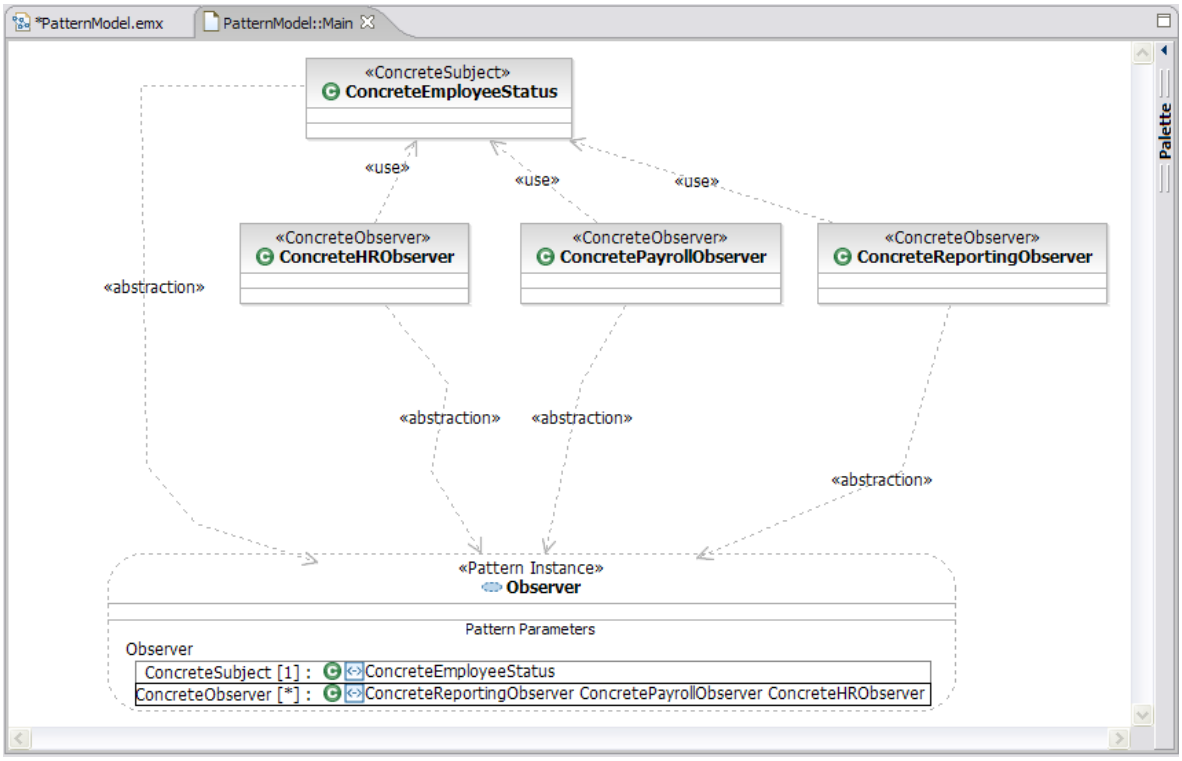


***Figure 8**: Pattern relationships*

Task 5: Create Target Project

As part of the transformation process, you need to create a Java project where Software Architect can put the generated code.

1. Create a new Java project and name it `Target`.
   a. On the **File** menu, click **New > Project**.
   b. In the Wizards list box, click **Java Project** and then click **Next**.
   c. If asked to enable the Java Development capability, click **OK**.
   d. For the **Project name,** enter `Target`.
   e. Click **Finish**.
   f. On the **Confirm Perspective Switch** dialog, click **No**.

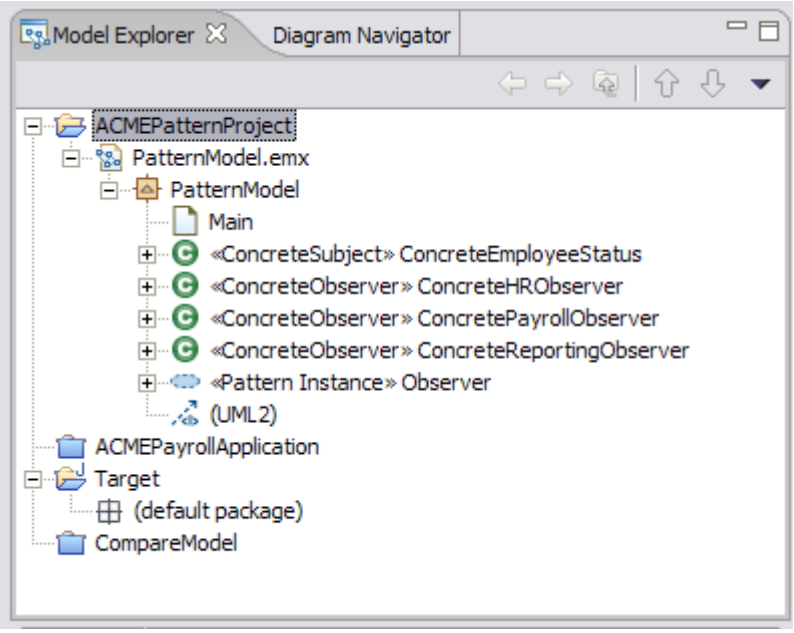


Figure 9: *Model Explorer after creating target*

Task 6: Apply Transformation and Review Code

In this task, you will transform the UML classes in the UML project into Java classes in the Java project.

1. With the Modeling perspective open, expand the PatternModel model.
2. In the Model Explorer, right-click the `ConcreteEmployeeStatus`, `ConcreteHRObserver`, `ConcretePayrollObserver` and `ConcreteReportingObserver` classes and click **Transform >Run Transformation > UML to Java**.
3. Select the `Target` project in the UML Element list under the Target tab and then click **Run**.

   The transformation will process and create Java classes in the `Target` project as appears in Figure 10.

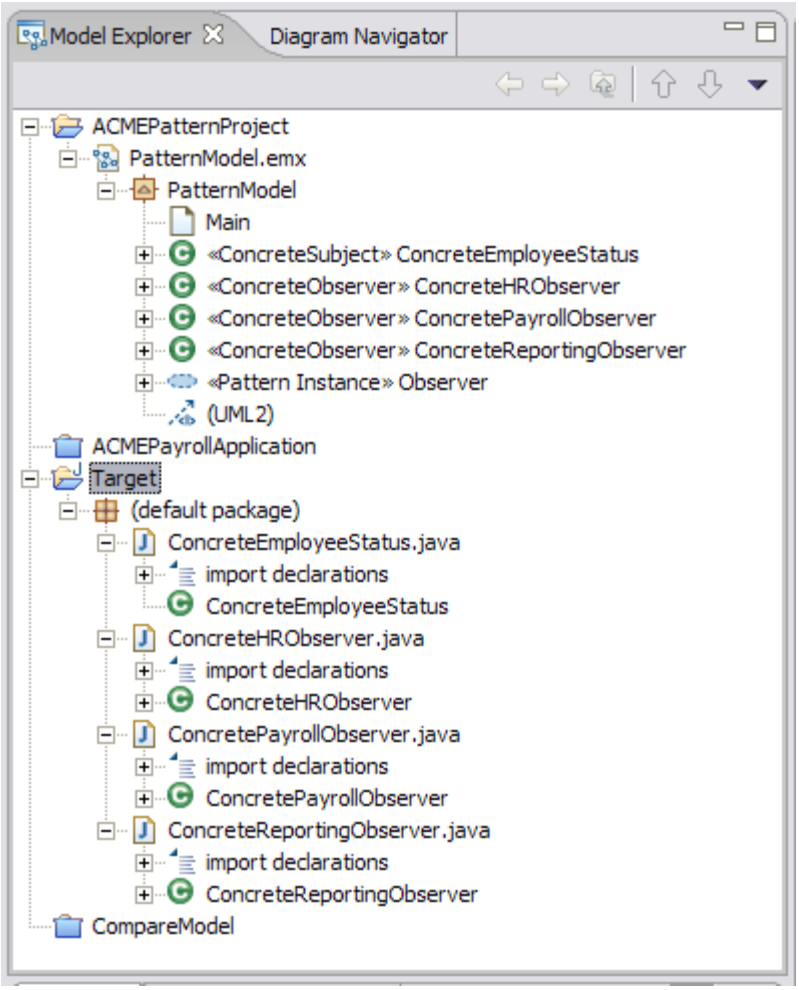


***Figure 10**: Transformed Java objects in Model Explorer*

Task 7: Review the Code

In this task, you will review the Java code generated by a UML to Java transformation. You will validate that the transformation created Java source and class files from UML classes and applied patterns, converted attributes to members and created prototyped stub methods from the UML operations.

1. Open the `ConcreteEmployeeStatus.java` file in the Java editor.

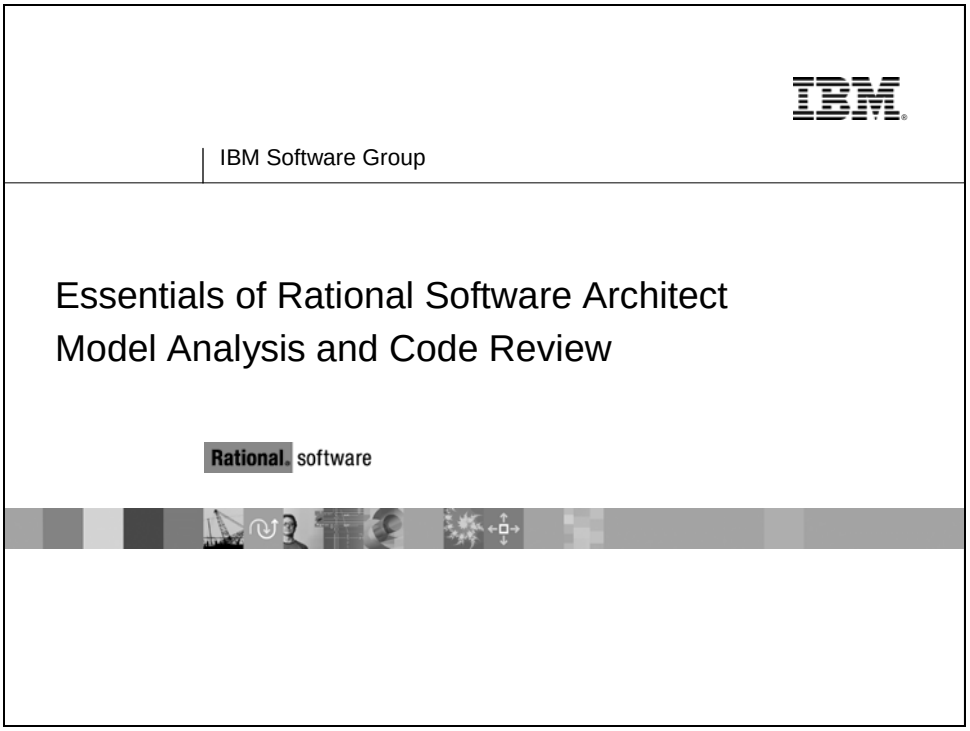
   TIP: In the Model Explorer, double-click on the `ConcreteEmployeeStatus.java` to open the Java editor.

2. Open the `ConcretePayrollObserver.java` file.

   A sample of `ConcretePayrollObserver.java` file appears in Figure 11.

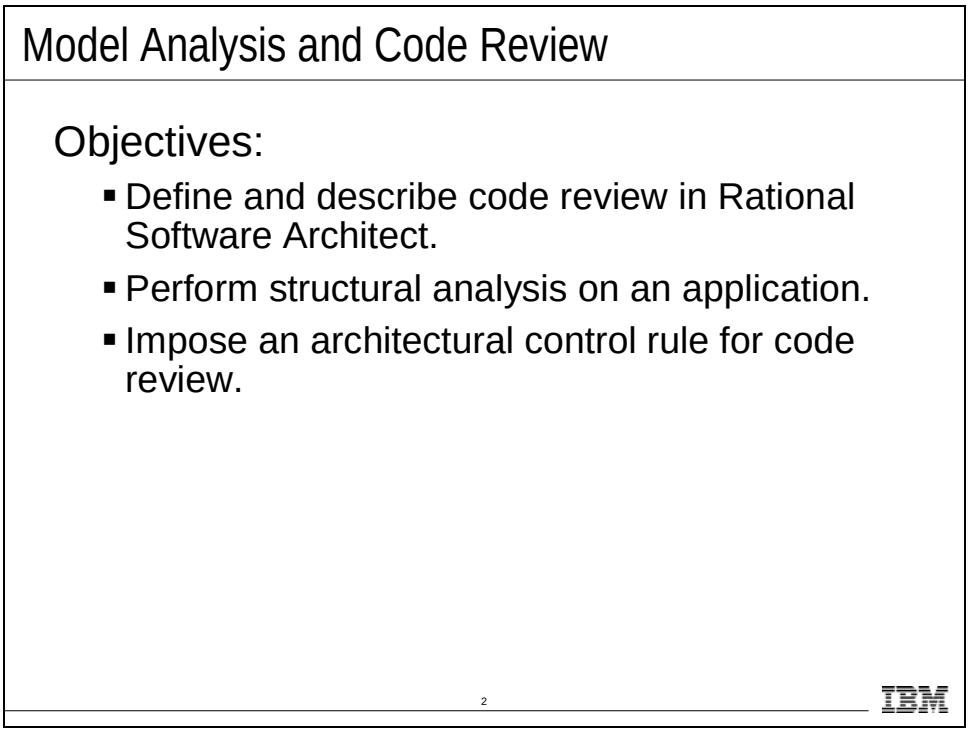
Figure 11: *Transformation source code in Code Editor*

▸ ▸ ▸ Model Analysis and Code Review



Topics

Module Objectives



This module introduces the automated code review features in Rational Software Architect.

Validating UML Models

Where Are We?

- **Validating UML Models**
- Code Review
  - Introduction to Code Review
  - Structural Review

3

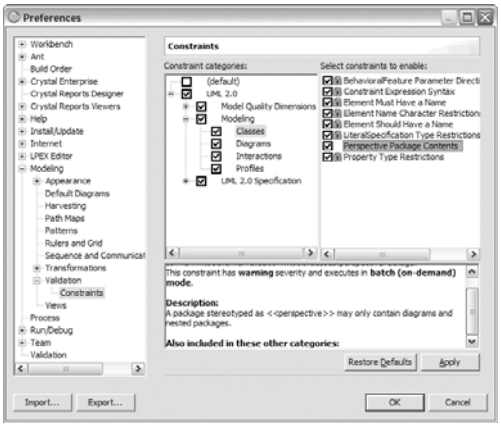
IBM

This section introduces model validation in Software Architect, including model analysis.

Model Validation: Validating Model Semantics

Model Validation: Validating Model Semantics

- Check for broken semantic rules
  - According to model's language
  - Adherence to UML requirements, logic, structures, and usage
- To validate the model:
  - On the **Modeling** menu, click **Run Validation.**
  - Right-click the model in the Model Explorer view or diagram and click **Run Validation.**

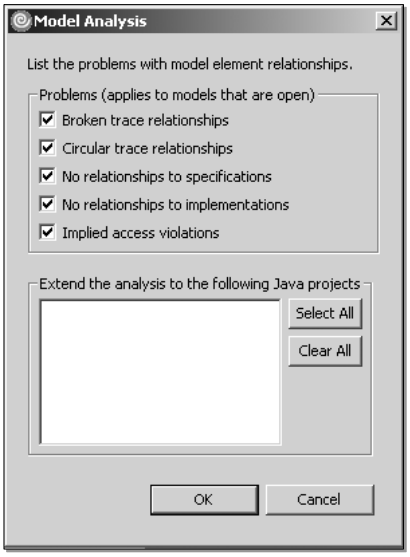


4

Model Analysis: Validating Traceability

Model Analysis: Validating Traceability

- Use Model Analysis to discover issues with traceability relationships between model elements:
  - No traceability path to an implementation or a specification
  - Broken trace relationships
  - Circular trace relationships
  - Implied access violations



5

IBM

Broken Trace Relationships

The target or source of a trace dependency that cannot be found or accessed generates a broken trace relationship. You can view broken element trace relationships and their causes in the Model Report view.

A trace relationship is broken if the trace relationship of the client or supplier:

- Is an element from another model, but the model is not found or is not open
- Is an artifact and its `fileName` property contains a file that does not exist.
- Is a visualized element that does not exist

Circular Trace Relationships

Circular trace relationships occur when one or more model elements have abstraction dependencies on each other.

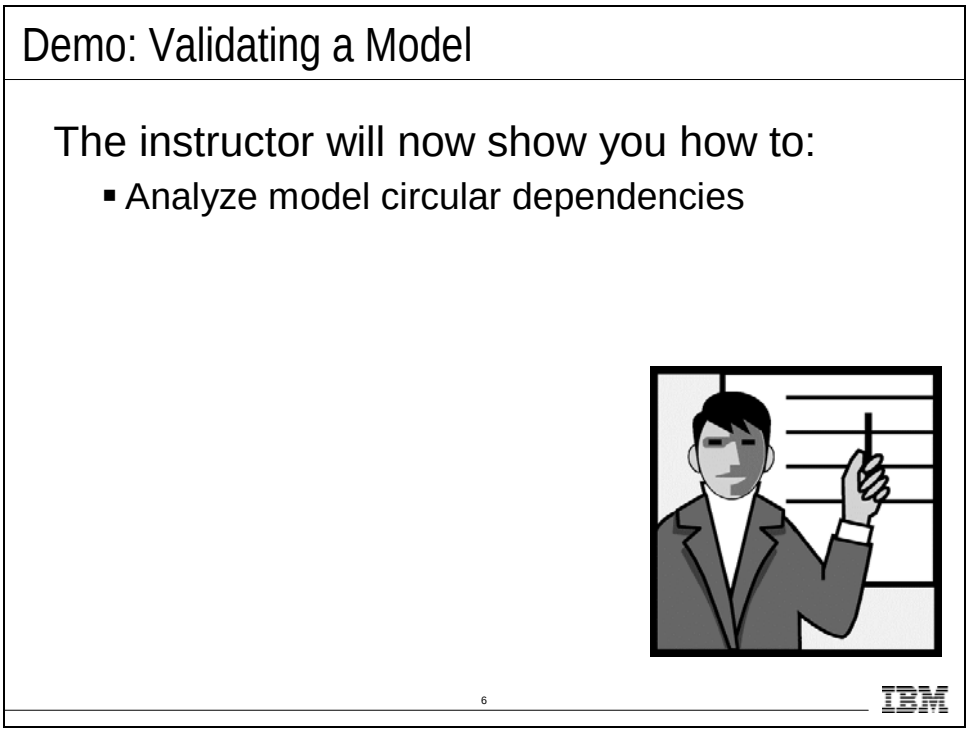
Access Violations Between Model Elements With Implied Dependency Relationships

An implied dependency between two UML elements occurs if any of the following items are true:

- Both model elements are in the same package.
- Each model element is in a different package and the source model element's package has a Permission, Import Package, or Import Element relationship to the target model element's package.
- Each model element is in a different package and the source model element's package has an Import Element or Permission relationship to the target model element.

Note: If one of the above is not true, an access violation error is listed in the Problems view. A model element that accesses a model element that is in another package without a Permission, Import Package, or Import Element relationship to the target model element's package is an access violation that might not be resolvable.

Demo: Validating a Model



Watch your instructor demonstrate how to validate a model in Software Architect.

Lab: Validate a Model

Lab: Validate a Model

- Given:
  - ACMEModelAnalysis model
- Complete the following tasks:
  - Import the Model Analysis model.
  - Analyze model circular dependencies.
  - Correct the model.

7

IBM

Complete the Model Analysis lab.

Introduction to Code Review

Where Are We?

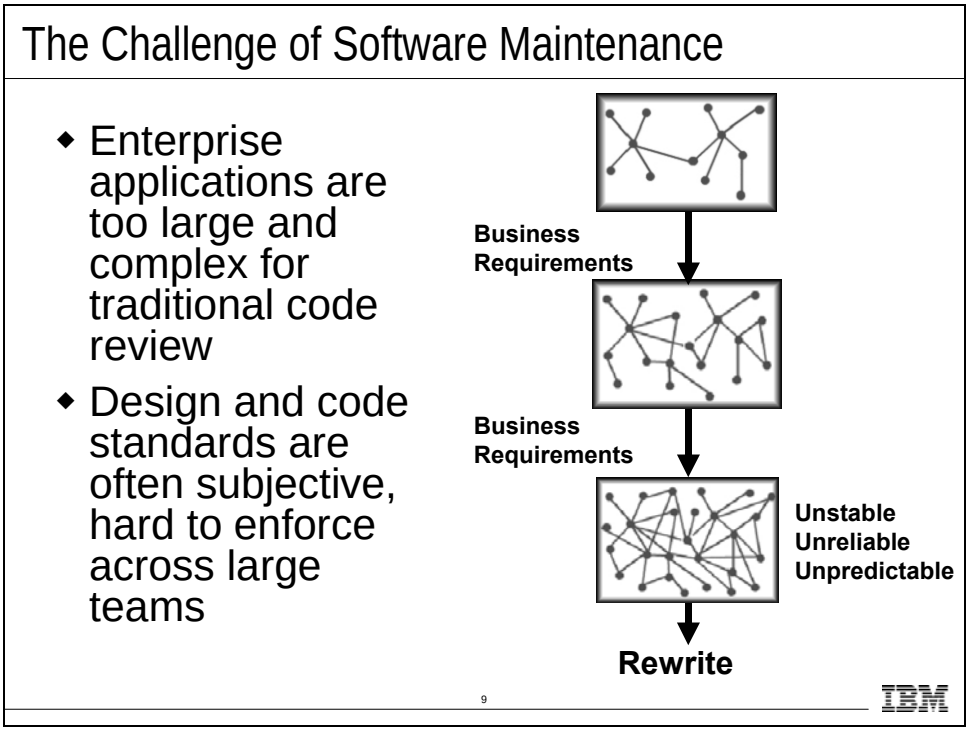
- Validating UML Models
- **Code Review**
  - **Introduction to Code Review**
  - Structural Review

8

IBM

This section introduces the code review features in Rational Software Architect, the Code Review view, and how to use the interface.

The Challenge of Software Maintenance

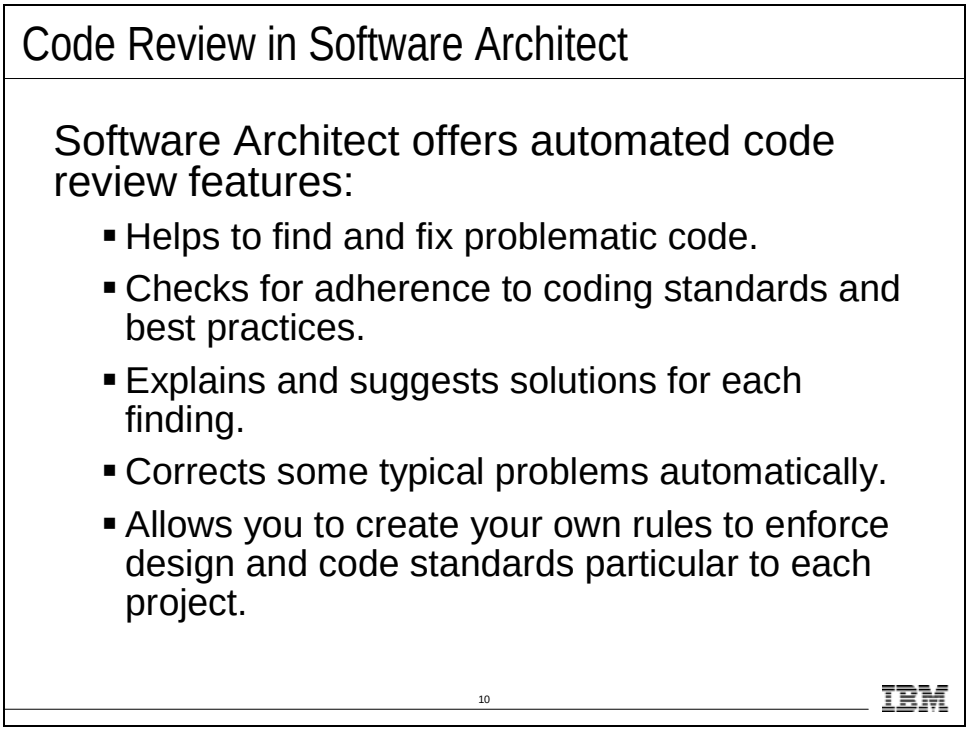


It is in the nature of software to change over many releases to keep up with evolving business needs. Cycles of change and evolution in an application, along with the usual challenges of large team development, eventually result in greater size and complexity. Over time, new technologies and new and third-party components may be integrated; older components may be discarded, updated, or rewritten.

Size and complexity in an application multiplies over time until eventually the code becomes so difficult to understand, so full of errors, and so costly to maintain, that it would be easier simply to start over and rewrite the application.

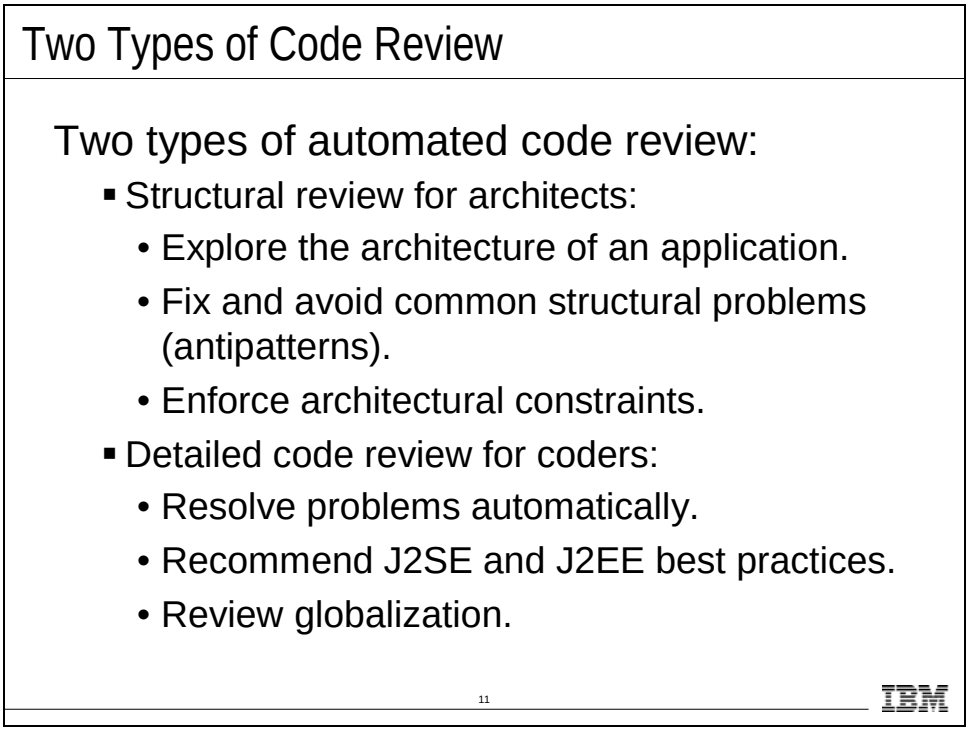
Component-based architecture and visual modeling techniques can be used to develop a resilient and extensible software architecture. However, the added challenges of software maintenance call for automated tools and processes to enforce adherence to architectural and coding standards over the long term as new requirements arise and more and more developers have a hand in making the needed changes.

Code Review in Software Architect



While the manual code review process can require many hours of detailed code inspection and discussion, Software Architect's Code Review features are easy to apply and to customize for a new project. As a result, you can run routine code reviews during development as often as you please.

Two Types of Code Review



Code review supports process improvement by helping developers use information gained from software testing, measurement, and monitoring to improve the application lifecycle and prevent software errors.

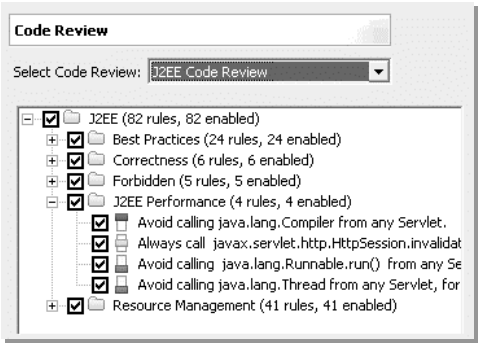
For the architect, code review features are used to analyze the high-level structure of the application, for example, to find recurring patterns that can be harvested from an application, to find anti-patterns to be repaired or redesigned, and to ensure that the implementation is faithful to the application's design.

For the developer, code review automates the code inspection process, making sure that the code is presented in a consistent way and that the code follows best practices consistent with the J2SE and J2EE specifications. Code review will also make sure that natural language-dependent code elements are optimized for translation into different languages for different locales.

Code Review

Code Review

- Code review analyses a selected code base, based on a set of configurable rules.
- Add and configure Code Review settings in Preferences.
  - Rules are organized into categories and category sets.
  - New rules and categories can be created and customized.
  - Select only rules you wish to apply during code review.



Code Review Preferences Dialog Box

12

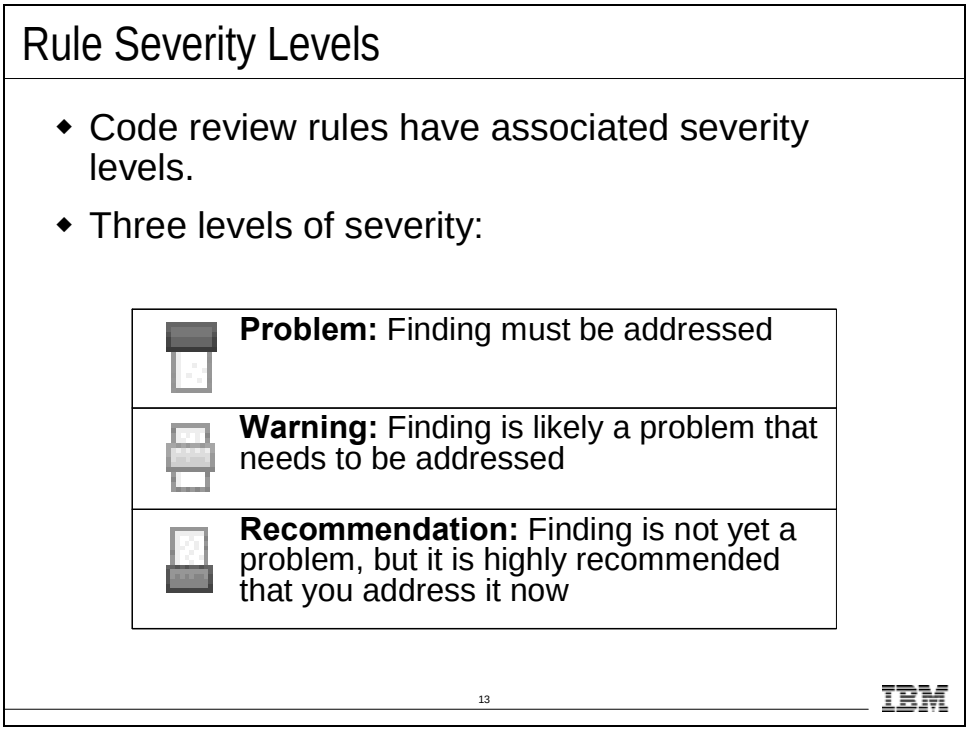
IBM

Code review in Software Architect is a set of rules that automates the process for a software developer or architect to analyze code for errors, optimize code based on coding best practices, and ensure that implementation code satisfies the design specification for the application.

You can set Code Review options in the Code Review preferences dialog box. There rules are organized into categories, and the categories are gathered into various code review sets, which provide selections of rules for different types of code reviews (and are available in the Select Code Review box).

The software architect can define rule sets to enforce project-specific coding standards and architectural constraints. Architectural constraints will be covered in the next section.

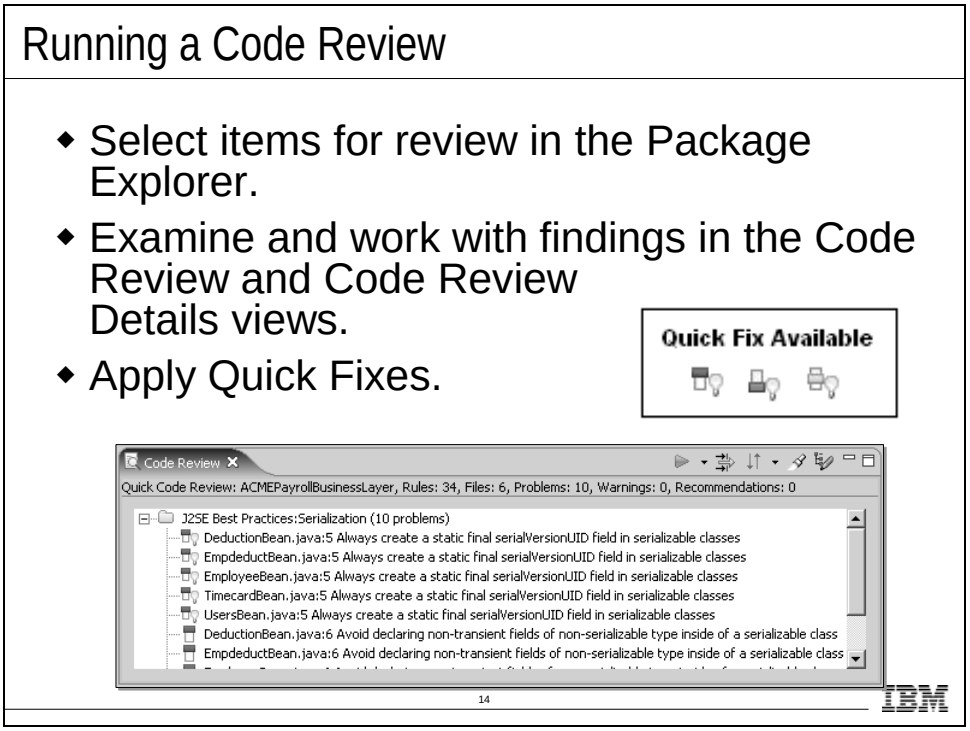
Rule Severity Levels



Every rule has one of the three severity levels associated with it. When creating a new rule, you specify a severity level that appears in code review results whenever the rule appears in code review results called **findings**. You can also modify the severity level assigned to rules that are shipped with Software Architect.

It is best not to discount the importance of recommendation-level rules. They take into account best practices and industry standards that engineering teams should adhere to. These issues, while not a serious problem now, could lead to problems in the future with software maintenance and extensibility.

Running a Code Review

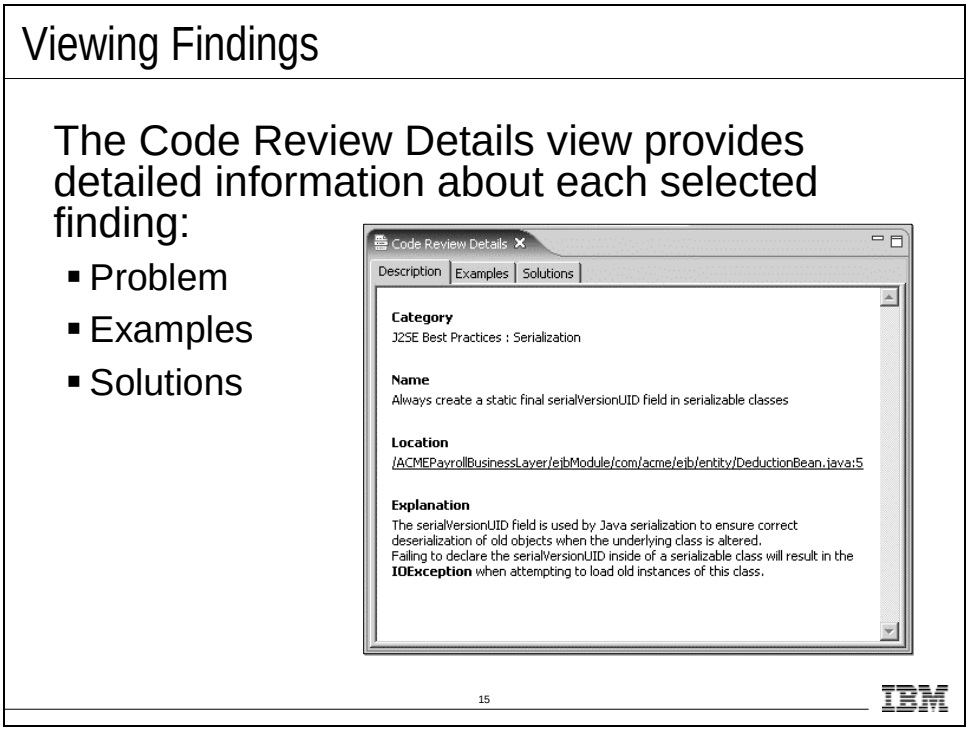


Run a code review by right-clicking a selected file, package, entire project, or range of projects in the Package Explorer, depending on the scope of the review you wish to run, and clicking **Code Review > Review**. Findings will appear in the Code Review window.

To view the details of a finding, double-click the finding and the details will open in the Code Review details view.

Many of the simpler rules have quick fixes available. After learning about the problem and solution, the developer can simply right-click the finding in the Code Review window and then click **Quick Fix**.

Viewing Findings



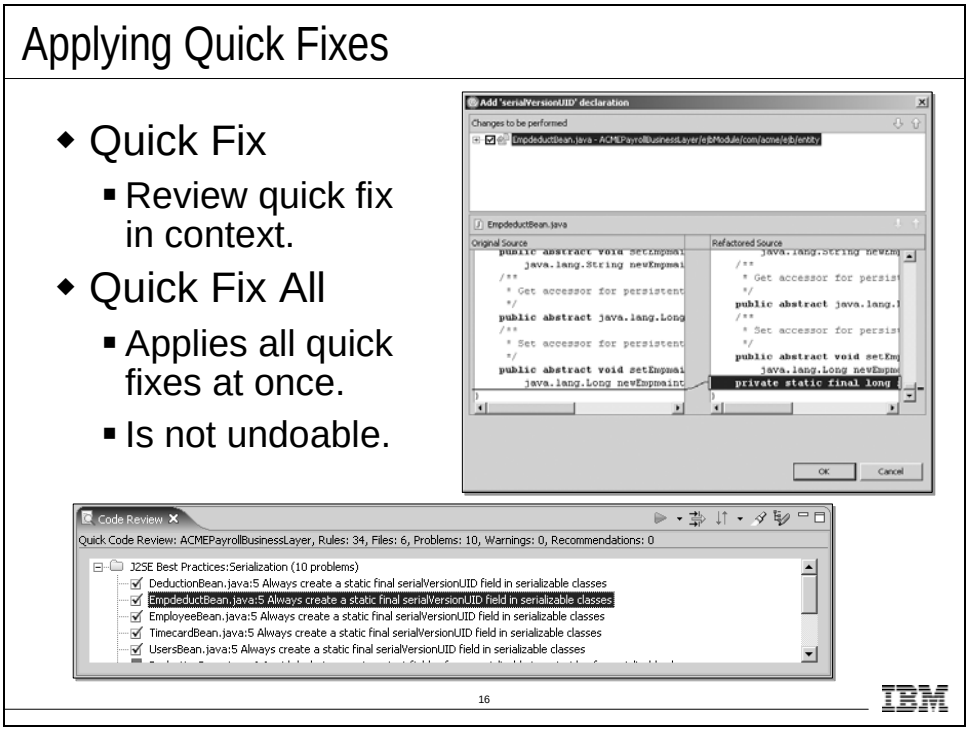
When working with predefined rules, the Code Review Details view provides a definition of the problem, examples to help the developer understand the issue, and (in many cases) a solution that can be applied to the code if the developer chooses to use it:

Problem: Describes the problem category, problem name, location, JDK, and a short description.

Examples: Describes a generic example of this kind of problem.

Solutions: Provides the current code contrasted with the suggested code.

Applying Quick Fixes



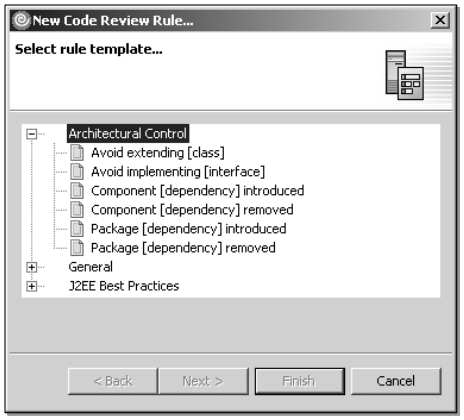
Some findings in a code review have quick fixes. A quick fix is an automated solution for a common problem. You apply a quick fix to save time and ensure that a problem is fixed consistently each time. Applying a quick fix ensures that the underlying issue of a finding will be eliminated according to best practices guidelines.

New Code Review Rule Wizard

New Code Review Rule Wizard

Process for creating a custom rule:

- Select Code Review type.
- Select Rule template.
- Select Basic Properties:
  - Category
  - Severity
- Select Specific Properties.
- Define Resource Filter.



17

IBM

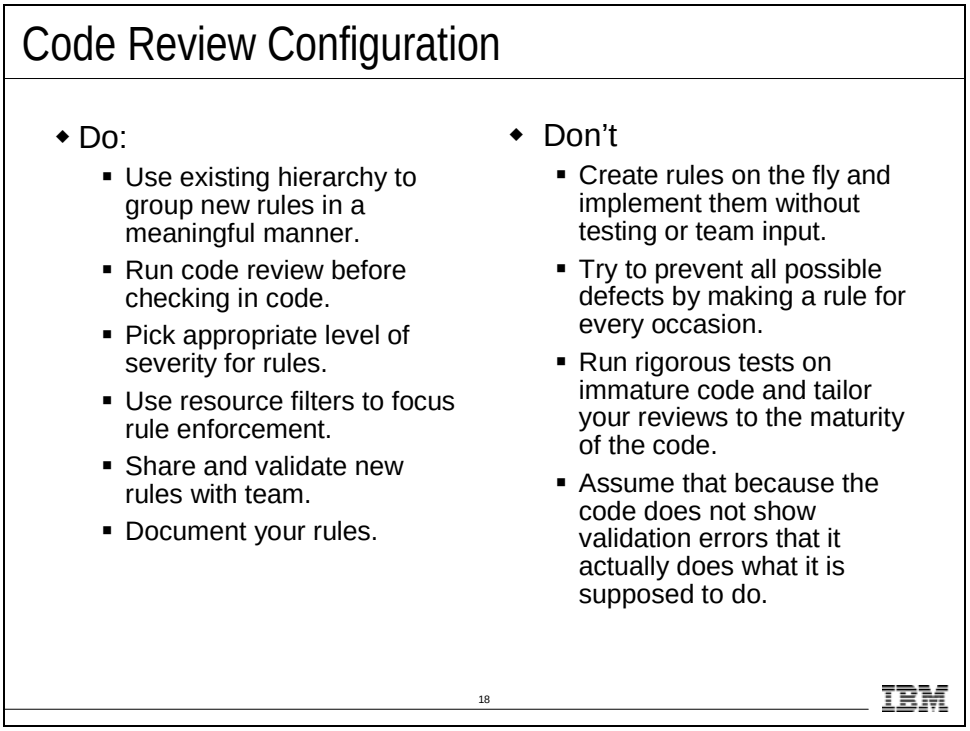
To define a custom code review rule, you can either click the Manage Rules button in the Code Review view or right-click an item in the Package Explorer and then click **Code Review > New Rule**. The New Code Review Rule Wizard opens.

All custom rules must be based on rule templates, which you select from the available categories in the New Code Review Rule Wizard.

- The **Basic Properties** settings for the rule include the rule category (in which group of rules it will be available in the Code Review list) and one of the three possible levels of severity.
- The **Specific Properties** setting defines the variable upon which the rule template operates.

The **Resource Filters** setting describes which filter is used to narrow the choice of resources.

Code Review Configuration



There are rules that are very useful to test in some code that are totally useless in other code. Rational Software Architect can keep a catalog of rule sets available for the developer to apply as applicable.

The results display includes the following tabs:

Example: Avoid using synchronized modifier in method declaration

Explanation: Synchronized blocks are computationally expensive; it is desirable to place as little code as possible inside of these blocks

Solution: Create synchronized block inside of the method that only locks necessary statements

Demo: Code Review

Demo: Code Review

The instructor will now show you how to:

- Configure Code Review preferences to run a Quick code review.
- Perform a Quick code review.
- Examine findings.
- Apply a Quick Fix.

19

IBM

Watch your instructor perform a code review and apply a Quick Fix.

Structural Review

Where Are We?

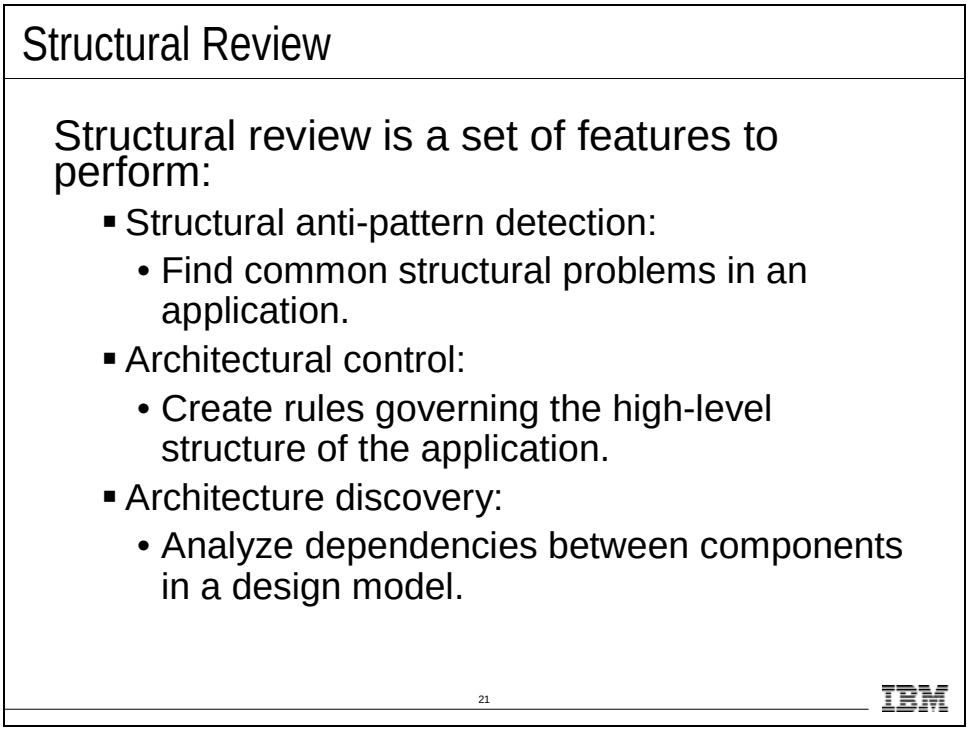
- Introduction to Code Review
- **Code Review**
  - Introduction to Code Review
  - **Structural Review**

20

IBM

This section introduces structural review in Software Architect. Structural review is the set of code review features that address the structure of the application and the fidelity of the implementation to the architecture.

Structural Review



The development and implementation of software may result in structures and dependencies that are significantly different than the original design. These dependencies can be hard to detect and manage manually and with conventional analysis tools because such changes may not cause errors or affect the behavior of the software.

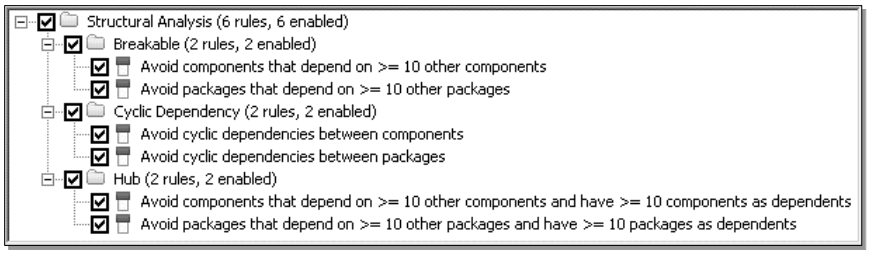
Code Review in Software Architect offers features (available through the Diagram Navigator view) that will enable the user to define rules to verify design integrity during the design of the application and apply these rules on the code created in the implementation. They can be applied during any other activities that result in the modification of the application's structure.

Structural Analysis

Structural Analysis

Antipattern: Common, recurring “worst practices” in software design and implementation.

- May be detected in the code through code review.
- May be detected in the architecture through architectural discovery.

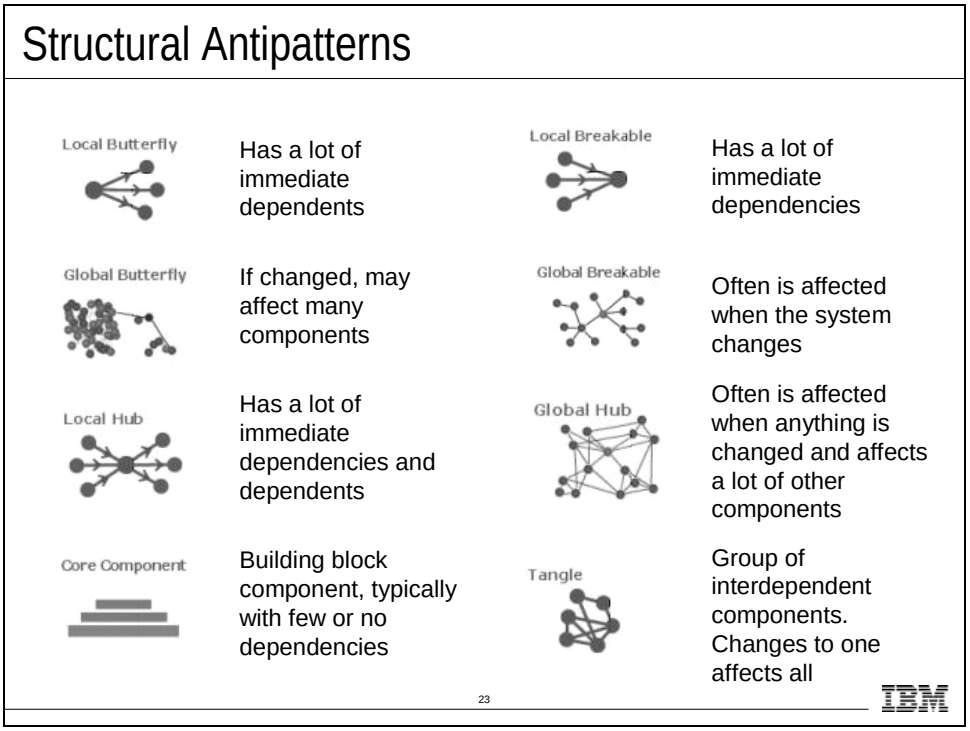


22

IBM

Structural analysis examines dependencies inside the application in order to measure their stability. It detects structural antipatterns, which are suspicious design elements in software. Antipatterns violate programming best practices. Antipattern structures need to be refactored because they make the application either less stable or harder to maintain.

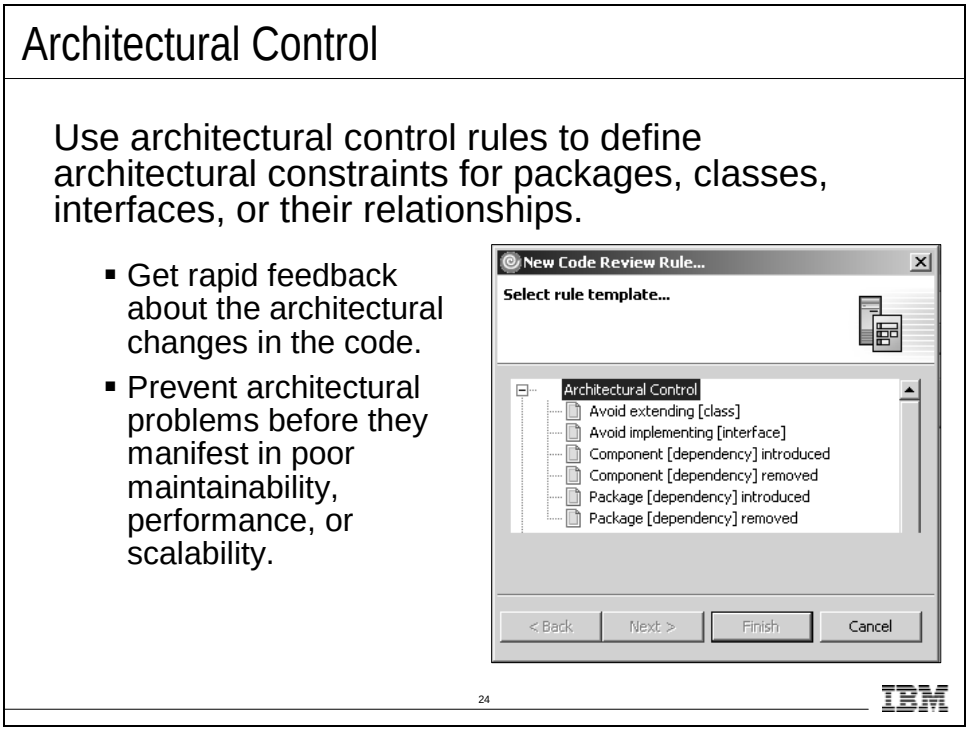
Structural Antipatterns



The antipatterns supported in structural analysis are:

- **Cyclic Dependency (Tangle):** A group of objects is so interconnected that a change to any object could affect all of the others. Cyclic dependencies need to be broken at the weakest link(s) within among them.
- **Local Breakable:** The class has so many dependencies that it is likely to break when another object is changed. The class should be broken into many smaller classes to distribute dependencies and make the system more stable.
- **Global Breakable:** Like a local breakable but with global dependencies. Usually occurs with the highest-level concepts in the system. Having many global breakables in your system implies high instability.
- **Local Hub:** The object has both many dependencies and many dependents. It is affected when another object is changed. Likewise, when it is changed, other objects are affected. Local hubs need to be refactored into several smaller classes.
- **Global Hub:** Like a local hub but with global dependents and dependencies. Often basic interfaces, abstract base classes, and utilities. Global hubs indicate that a system is not conceptualized well and that it is highly unstable.
- **Local Butterfly:** The object has many immediate dependents. Typically, they would be basic interfaces, abstract base classes, and utilities. Local butterflies present risk. Any changes you make to one will have significant impact on its dependents.
- **Global Butterfly:** Like local butterflies but with global dependents.

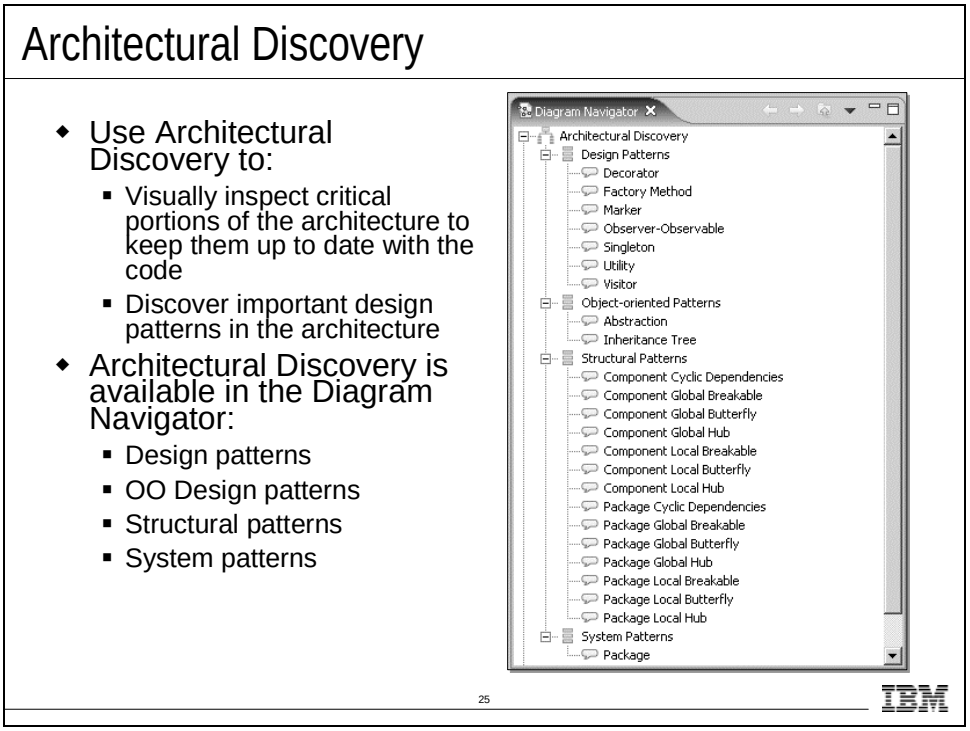
Architectural Control



The goal of architectural control is to track any changes that occur in the structure of the application, using code review rules to catch unwanted dependency, inheritance, or implementation relationships.

Rules for architectural control are created from the Architectural Control templates available in the New Code Review Rule Wizard. Rules are then made available in the Structural Analysis (default) or User Defined category, for example.

Architectural Discovery



Architectural discovery is a set of features used to visually inspect an application's architecture, based on a variety of possible criteria:

Design patterns: Finds elements that participate in collaborations that match the structure of common design patterns.

Object-Oriented Design patterns: Finds examples of abstraction (for example, interfaces and abstract classes) and inheritance in the application.

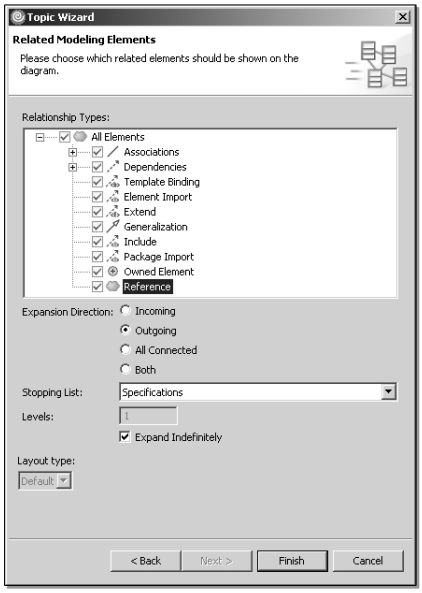
Structural patterns: Examines the design for structural antipatterns, the same antipatterns that have been described for structural analysis code review.

System patterns: Finds and displays all the packages in the application.

Review: Topic Diagrams

Review: Topic Diagrams

- Create topic diagrams to depict key model elements and their relationships.
- Topic diagrams:
  - Are created by querying the model
  - Persist in the model
  - Are dynamically updated
  - Are most useful for visualizing code
  - Are used in architectural discovery



26

IBM

Software Architect can generate and arrange topic diagrams for you automatically. These diagrams are useful when working in large models that contain many elements with complicated relationships. They are also invaluable when trying to help discover the architecture of an application.

Topic diagrams are used for both discovery and as visual documentation since they can be saved in the model. Developing a topic diagram involves running a query against the existing model contents. You select a key model element and then define what other elements you wish to appear in the diagram, based upon the types of relationships they have to the topical element. As the content of the model changes, topic diagrams adjust accordingly.

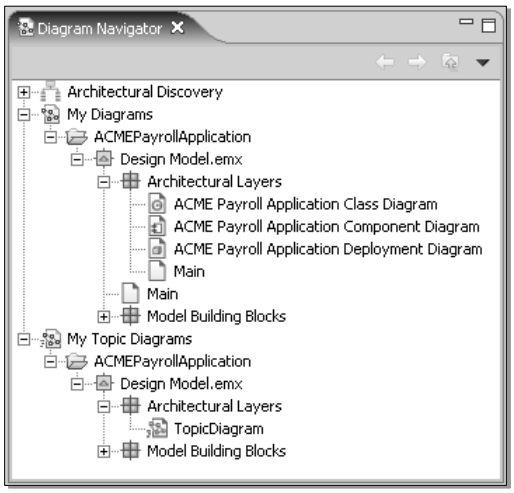
Topic diagrams, along with browse diagrams, which are similar to topic diagrams except that they are generated once and conveniently discarded, are most useful when trying to understand or document an existing code base when you want to understand the structure of artifacts within an existing application.

Diagram Navigator

Diagram Navigator

Diagrams organized by type:

- My Diagrams
- My Topic Diagrams
- Architectural Discovery
  - Generated Topic diagrams



27

IBM

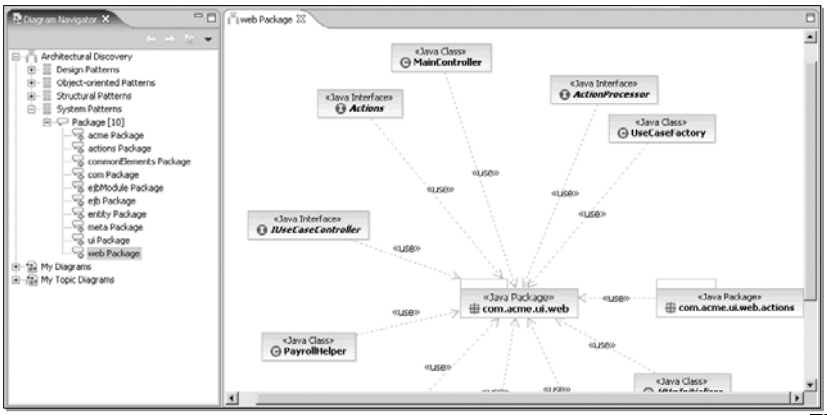
The Diagram Navigator shows the structure of all the diagrams in the workspace. As projects and model structures evolve and become more elaborate, the Diagram Navigator becomes useful for finding and opening the appropriate diagram quickly.

If you were new to an ongoing development project and had to examine an existing set of highly elaborated projects in Software Architect, you might use the Diagram Navigator to examine the high-level diagrams for each model to get familiar with key abstractions and their relationships.

Architectural Discovery in the Diagram Navigator

Architectural Discovery in the Diagram Navigator

- Architectural discovery results are topic diagrams, grouped under folders in the Diagram Navigator.
- Topic diagrams can be converted to design diagrams to be edited.



28

IBM

To inspect the architecture, including all open projects in the workspace, using one of the predefined types under Architectural Discovery, right-click the appropriate folder and then click **Discover Architecture**. A set of topic diagrams will appear as results under the folder you have selected.

Double-click the topic diagram to view it. The topic diagram can then be converted to a class diagram for editing.

Demo: Structural Review

Demo: Structural Review

The instructor will now show you how to:

- Discover architecture
- Perform structural analysis
- Create an architectural control rule
- Verify the architecture

29

IBM

Watch your instructor demonstrate how to perform a structural review.

Lab: Perform an Architectural Review

Lab: Perform an Architectural Review

- Given:
  - Acme Enterprise Application
- Complete the following tasks:
  - Discover the architecture
  - Perform structural analysis
  - Create an architectural control rule
  - Verify the architecture

30

IBM

Complete the Architecture Review lab.

Review

Review

- What three levels of severity are possible for code review rules?
- What are structural antipatterns?
- What is the goal of architectural control?

31

IBM

Model Analysis

Objectives

After completing this lab, you will be able to:

- Describe model structure problems and their effects on the system
- Identify and correct model problems using Rational Software Architect

Given

The following lab artifacts can found in the `DEV396` folder:

- UML2 design model with a qualified but simplistic set of class diagrams for the Payroll Application. (`ACMEModelAnalysis.zip`)

Scenario

In this lab, you will assume the role of senior application architect for the ACME Payroll System. A new hire has submitted a diagram (below) for a simple MVC component of a new system. As architect, you will analyze the model using Rational Software Architect and make any needed corrections.

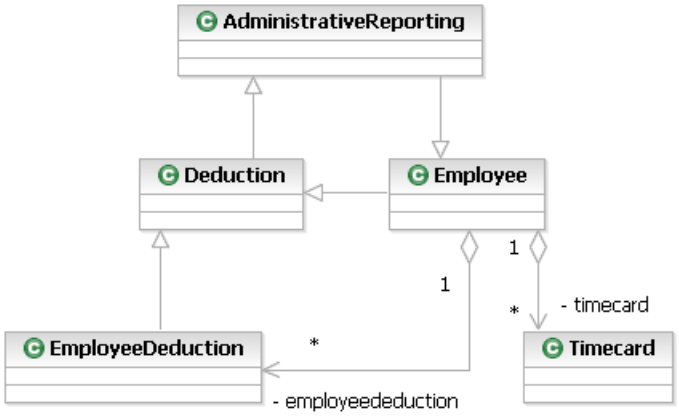


Figure 1: *Class diagram*

Task 1: Import the Model Analysis Model

In this task, you will import and open the model to be analyzed.

1. With the **Modeling** Perspective open, import the `ACMEModelAnalysis` project:
   - On the **File** menu, click **Import**.
   - In the list box, select **Project Interchange** as the import source and then click **Next**.
   - Next to the **From zip file** field, click the **Browse** button and find `ACMEModelAnalysis.zip` in the `DEV396` folder. Select the file and click **Open.**
   - Select `ACMEModelAnalysis` in the `Import Projects` list.
   - Click **Finish.**
2. Open the model file, `ACMEModelAnalysis.emx`.

   After you have opened the model, examine Model Explorer as shown below:

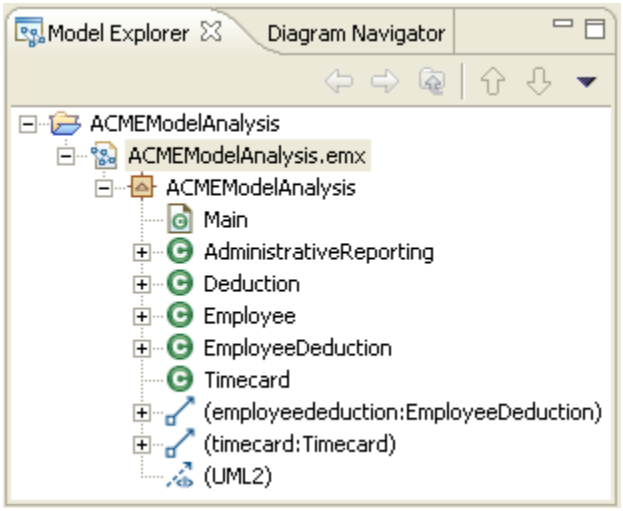


***Figure 1**: Diagram Navigator after project import*

Task 2: Analyze Model Circular Dependencies

In this task, you will use Rational Software Architect to analyze and correct circular dependencies in the model. A circular dependency occurs when a child object has dependencies upon the parent object, and the parent object also has dependencies on the child. You will use Software Architect to determine which objects attempt to inherit their children:

1. Make sure the **Modeling** Perspective and the `ACMEModelAnalysis` project are open.

2. Analyze the model for circular dependencies.

   a. On the **Modeling** menu, click **Model Analysis**.

   b. On the **Model Analysis** dialog box, cancel all selections except **Circular trace relationships** and then click **OK**.

      Software Architect will describe any problems in the Problems view:

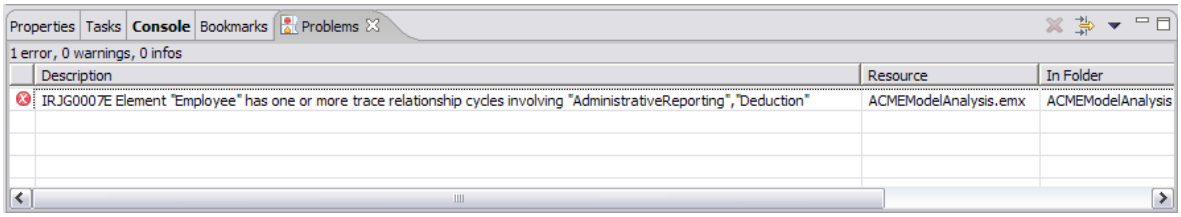


Figure 3: *Payroll model analysis problems*

   c. In the Problems view, right-click the problem and then click **Go To.**

      The classes involved in the circular dependency are selected in the Model Explorer view:

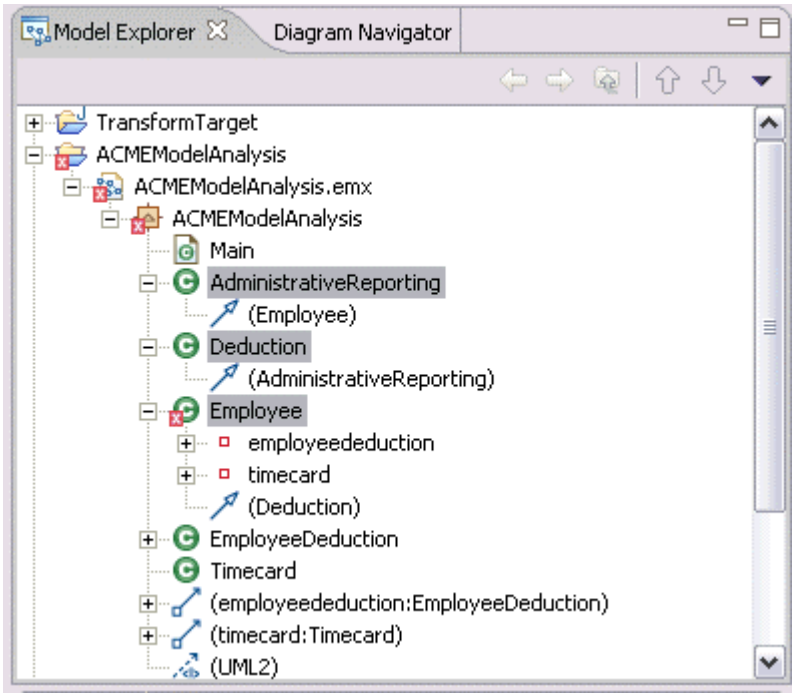


Figure 4: *Model explorer problems*

Task 3: Correct the Model

In this task, you will remove the generalization causing the circular dependency. A close look at the class diagram reveals that dependency relationships are more appropriate here, so you will also add new dependency relationships.

1. In the `Main` diagram, remove the generalizations from the `AdministrativeReporting` class to the `Employee` class and from the `Deduction` class to `AdministrativeReporting` by right-clicking the generalizations and then clicking **Delete from model**.

   TIP: When working on the Diagram editor, selecting **Delete from diagram** or pressing the **Delete** key removes the object from the current diagram. **Delete from model** removes the object from the model.

   TIP: To select more than one shape or connector at a time, click the first relationship, then hold down the `CRTL` key, and click the second.

2. Repeat the model analysis for circular trace relationships and notice that Software Architect finds no problems.

3. Create a usage dependency relationship (which will appear with the «use» stereotype) from `AdministrativeReporting` to the `Deduction` and `Employee` classes.

   TIP: To draw a usage dependency in the **Class Diagram** drawer of the Toolbox, open the **Dependency** tool's drop-down menu and select the **Usage** tool.

4. In the diagram, delete the generalization from the `Employee` class to the `Deduction` class.

   When you are finished, the diagram should contain the following classes and relationships:

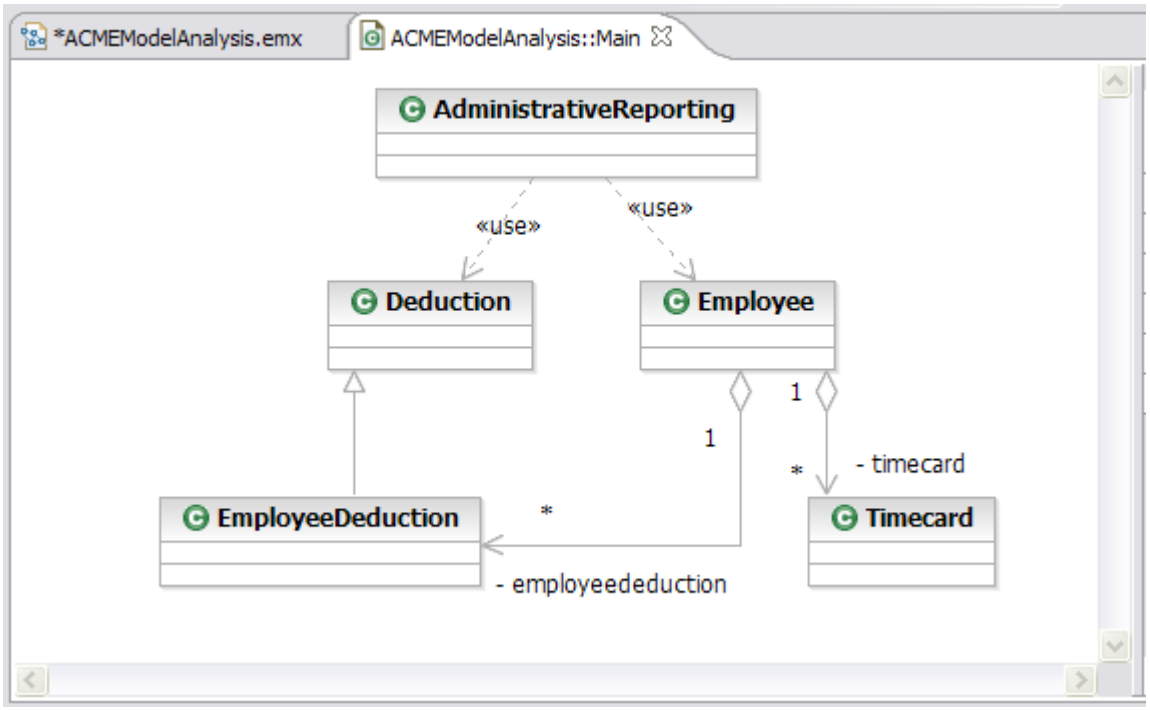


***Figure 5**: Corrected class diagram*

5. Repeat the circular trace analysis and notice that Software Architect finds no problems.

Architecture Review

Objectives

After completing this lab, you will be able to:

- Perform architectural discovery to detect design patterns and important OO structures
- Perform structural review for anti-patterns, such as hubs, and butterflies
- Use architectural control features to create new user defined rules that represent architectural constraints
- Use architectural control features to validate user-defined rules

Given

The following lab artifacts can be found in the `DEV396` folder:

- Payroll Enterprise Application (`ACMEPayrollEnterpriseApplication.zip`)

Scenario

In this lab, you assume the role of senior application architect for the ACME Payroll System. A new architect and a team of developers have begun to implement the project. You have been asked to review the project, make any needed changes, and put in place architectural control features using Rational Software Architect.

Task 1: Import Enterprise Application

In this task, you will import an existing `EAR` file containing the J2EE projects for the ACME payroll enterprise application you will analyze.

1. Close all open projects.

   TIP: Right-click each project in Model Explorer and click **Close Project**.

2. Open the **J2EE** Perspective.

   TIP: Click **Window > Open Perspective > J2EE** to open the **J2EE** Perspective or click on the **J2EE** Perspective icon on the tool bar. If asked to enable J2EE development capabilities, click **OK**.

3. With the **J2EE** Perspective open, import the `ACMEPayrollEnterpriseApplication.zip` file:

   a. On the **File** menu, click **Import**.

   b. In the list box, select **Project Interchange** as the import source and then click **Next**.

   c. Next to the **From zip file** field, click the **Browse** button and find `ACMEPayrollEnterpriseApplication.zip` in the `DEV396` folder. Select the file and click **Open.**

   d. Click the **Select All** button to select all the projects.

   e. Click **Finish.**

   TIP: Double-click the progress bar in bottom right corner of the workbench to monitor progress. The bar will contain a red square until it is complete.

After you have imported the zip file, the `Project Explorer` will display the projects as shown in Figure 1.

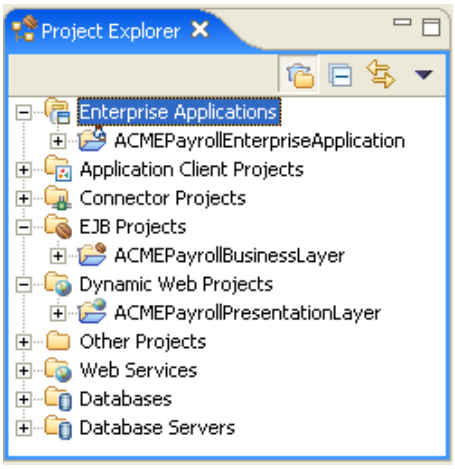
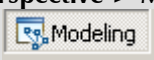


Figure 1: *Project Explorer*

Task 2: Discover Architecture

In this task, you will discover the architecture of the payroll application. You will perform a visual inspection of the architecture by performing a `Package` rule analysis and viewing Browse diagrams generated by the process.

1. Open the **Modeling** Perspective.

   TIP: Click **Window > Open Perspective > Modeling** to open the **Modeling** Perspective or click on the Modeling icon 

on the tool bar.

2. Open the Diagram Navigator view.

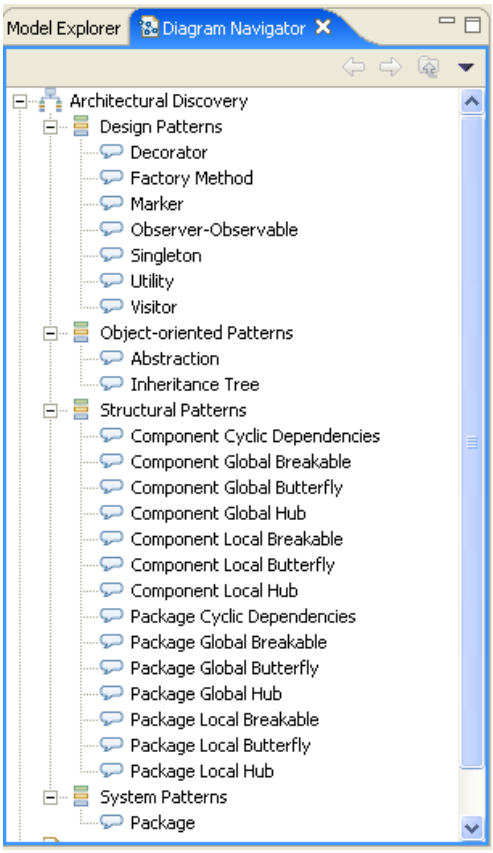


***Figure 2**: **Modeling** Perspective with Diagram Navigator*

3. In the Diagram Navigator, expand **Architectural Discovery > System Patterns.** Right-click **Package** and then click **Discover Architecture**.

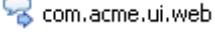
4. Under `Package` in `System Patterns`, open the `com.acme.ui.web Package` browse diagram by right-clicking the 

diagram and then clicking **Show Diagram**. If a topic diagram does not exist for the package, Software Architect automatically creates the diagram.

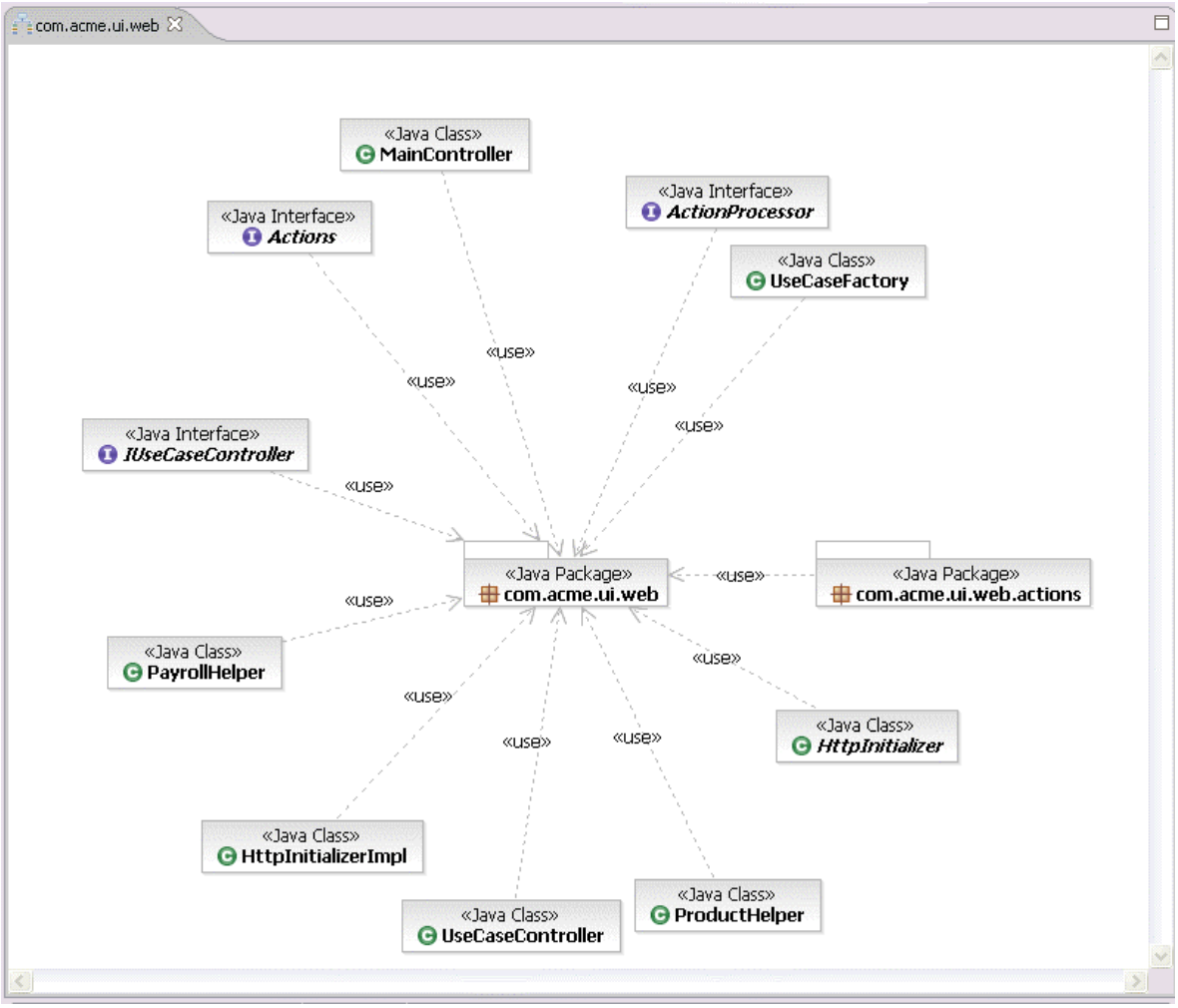


***Figure 3**: Browse Diagram created by Architectural Discovery*

5. Create a topic diagram:

a. In the Browse diagram, right-click on the `web Package` «Java Package» com.acme.ui.web package and select **Make Topic Diagram**.

b. Complete the Topic diagram location dialog using the following instruction:

- **Parent folder:** `ACMEPayrollPresentationLayer`
- **File name**: `WebPackageTopicDiagram`
- Click **Next**.

c. Complete the Topic dialog using the following instructions:

- Select **Related Visualizer Elements**.
- Click **Finish**.

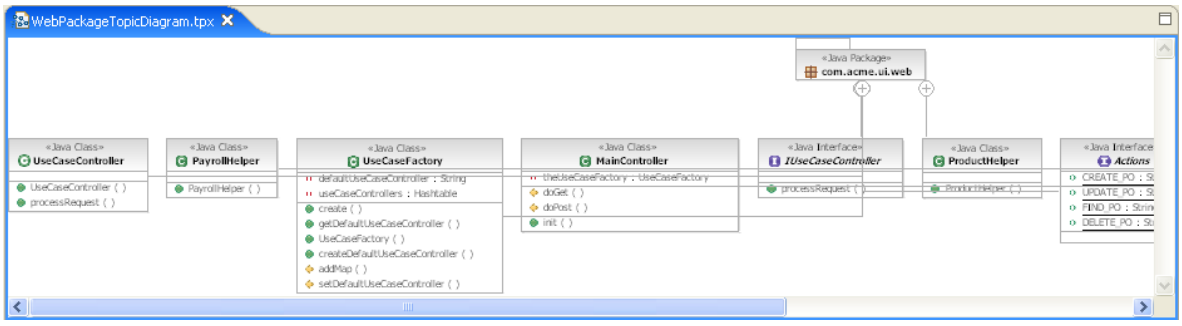


***Figure 4**: Web Package Topic diagram*

6. Create an Inheritance Tree browse diagram:

   a. In the Diagram Navigator, expand **Architectural Discovery > Object-oriented Patterns**, right-click **Inheritance Tree**, and then click **Discover Architecture**.

   b. Display the `DefaultMaintainPayrollAction` browse diagram by double-clicking its icon.

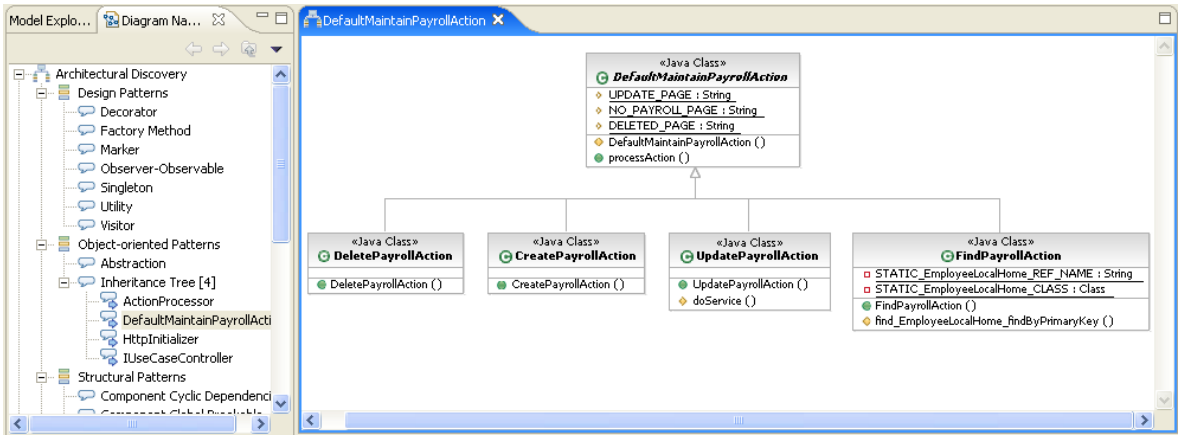


***Figure 5**: DefaultMaintainPayrollAction Inheritance Tree*

Task 3: Structural Analysis of AntiPatterns

In this task, you will discover some of the weak points that exist in the architecture of the payroll application. You will analyze the application for a type of structural antipattern called Global Butterfly.

1. Open the **Modeling** Perspective and Diagram Navigator view if they are not already open.
2. In the Diagram Navigator, expand **Architectural Discovery > Structural Patterns > Component Global Butterfly** and right-click **Discover Architecture**.

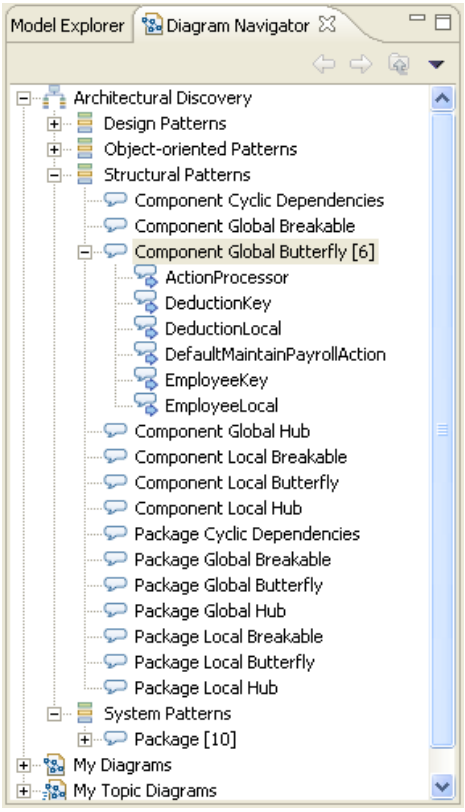


***Figure 6**: Global Butterfly pattern discovery*

3. Open the browse diagram for the `ActionProcessor` component by double-clicking its icon.

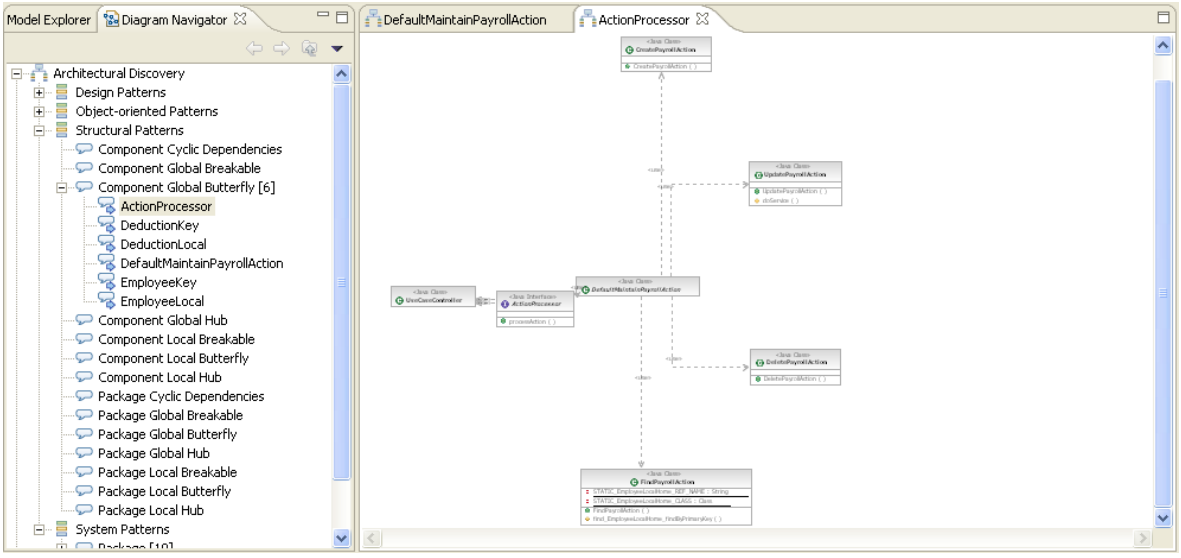


***Figure 7**: Global Butterfly browse diagram*

TIP: **Global butterflies** are objects with a large number of global dependents. Generally, they are basic interfaces, abstract base classes, and utilities. Global butterflies are very important even though they are not necessarily problematic. If you change a global butterfly, the overall global impact will be significant.

4. Review the `ActionProcessor` component global butterfly diagram and answer the following questions:
   - Which classes could be impacted by changes to the ActionProcessor interface?
   - How could the architecture be refactored to minimize global butterflies?

Task 4: Create Architectural Control Rule

In this task, you will learn to create a new code review control rule. You are assigned the task of writing a code review rule that will recognize when presentation layer code calls entity beans directly instead of using session bean façades.

1. Make sure the **Java** Perspective and the `ACMEPayrollEnterpriseApplication` project are open.

   TIP: On the **Window** menu, click **Open Perspective > Java** to open the **Java** Perspective or click on the Java icon  on the tool bar.

2. Open Code Review Preferences by clicking **Window > Preferences > Java > Code Review**.
3. Select **Structural Analysis Code Review** from the **Select Code Review** list box in the Code Review dialog and then click **New Rule**.
4. In the **Select rule template** list box, expand **Architectural Control** and select **Package [dependency] introduced** and then click **Next**.
5. Complete the Configure Package dialog with the following settings and then click **Finish**:
   - **Severity:** Problem
   - **From Independent Package:** `com.acme.ui.web.actions`
   - **To Dependent Package:** `com.acme.ejb.entity`

When you have successfully created the architectural rule, it will appear in the code review dialog as follows:

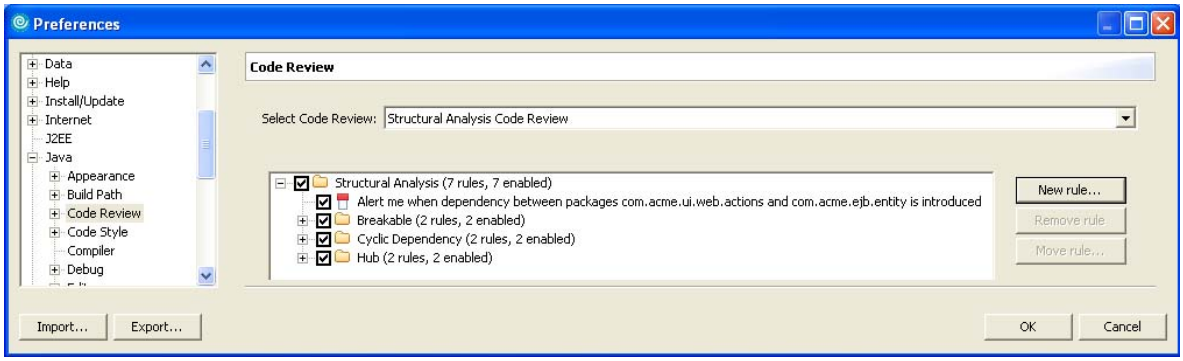


***Figure 8**: User-defined Code Review rule*

6. Click **OK**.

Task 5: Verify the Architecture

In this task, you will verify that the Structural Analysis rules, including your new rule, have not been violated.

1. Make sure the **Java** Perspective and the `ACMEPayrollEnterpriseApplication` project are open.
2. Run the code review:
   a. Open the Code Review view.

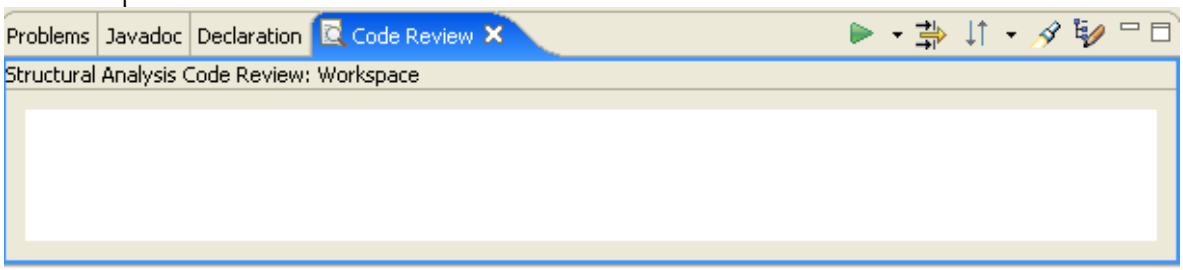


***Figure 9**: Code Review view*

   b. In the Code Review view, select the code review type by clicking the Manage Rules icon.
   c. Ensure that **Structural Analysis Code Review** is selected and then press **OK**.
   d. In the Code Review view, review the workspace by clicking the **Run Review** button.

After the code review is successfully completed, the results of the analysis are displayed in the Code Review view.

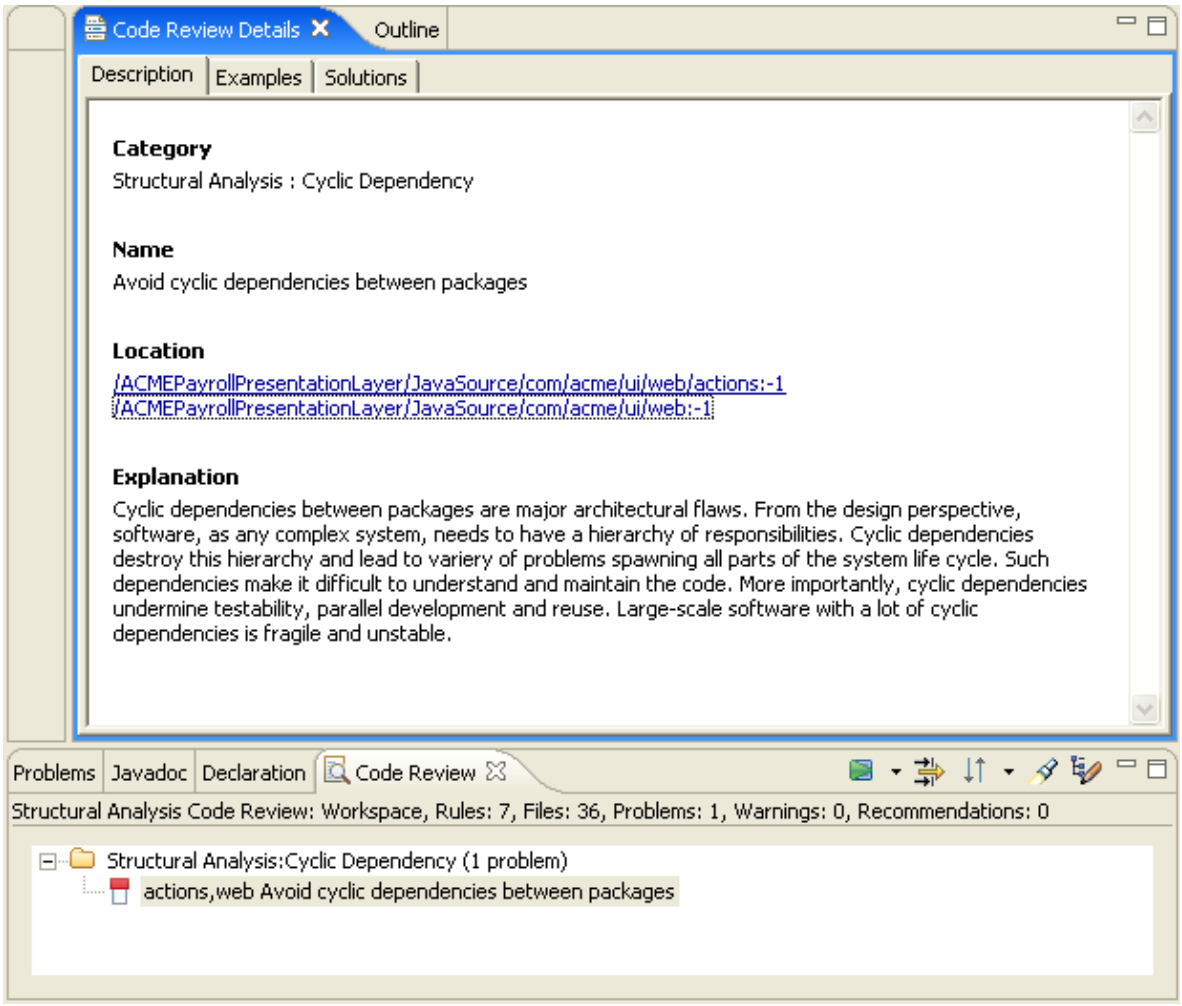


***Figure 10**: Structural Analysis Code Review violations*

3. Review the violations and double-click the problem to examine the code review details.
4. Answer the following questions regarding the cyclic dependency problem:
   - Which packages have cyclic dependencies between them?
   - What is the impact of a lot of cyclic dependencies in large scale software?
   - What steps should you consider to solve the problem?

Task 6: Create Architectural Control Rule Export

In this task, you will export the rule created in task 4 for distribution to the rest of the team.

1. If Code Review Preferences is not already open, click **Preferences > Java > Code Review** on the **Window** menu, and then click the **Export** button.
2. In **Export Code Review configuration** dialog, make the following settings and then press **Finish**:
   - Select **Export user-defined rules**.
   - Name the file `UserDefinedCodeReviewRules.zip` in the `DEV396` and click **Finish**.

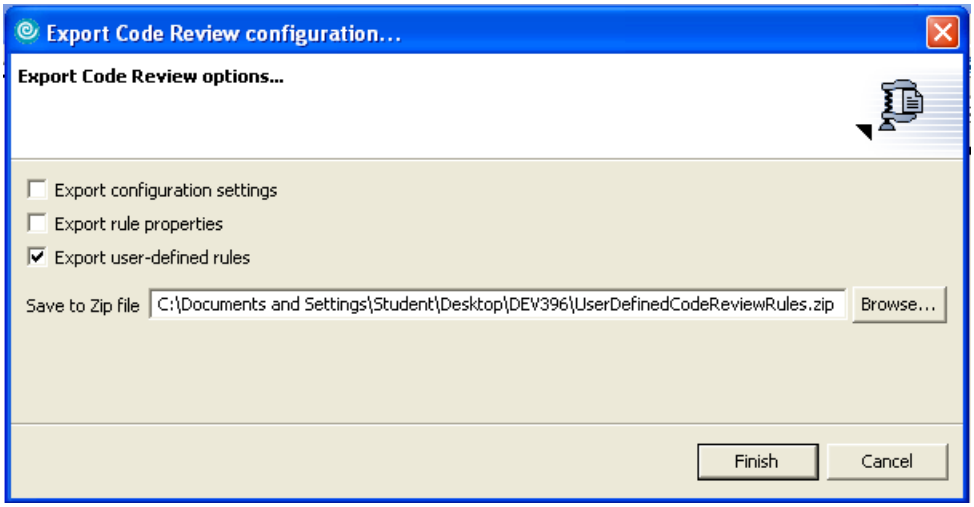


***Figure 11**: Code review export dialog*