哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Chapter 2: Process and Tools of Software Construction

2.2 Process, Systems, and Tools of Software Construction

Ming Liu

24 February 2019

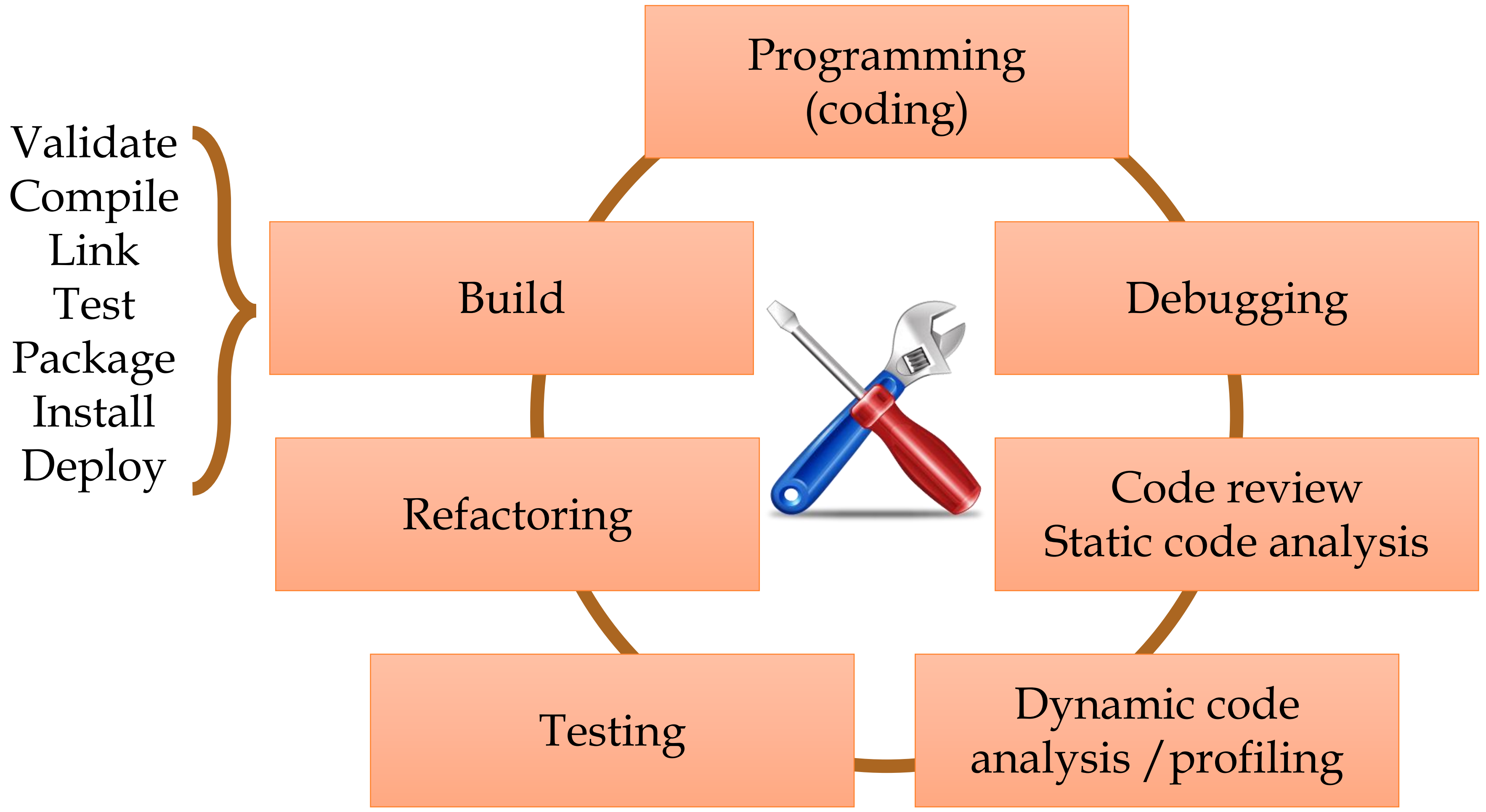
Outline

- **General process of software construction:** Design ⇒ Programming /refactoring ⇒ Debugging ⇒ Testing ⇒ Build ⇒ Release
  - Programming / refactoring
  - Review and static code analysis
  - Debugging (dumping and logging) and Testing
  - Dynamic code analysis /profiling
- **Narrow-sense process of software construction (Build):** Validate ⇒ Compile ⇒ Link ⇒ Test ⇒ Package ⇒ Install ⇒ Deploy
  - Build system: components and process
  - Build variants and build language
  - Build tools: Make, Ant, Maven, Gradle, Eclipse
- **Summary**

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

1 General process of software construction

General process of software construction



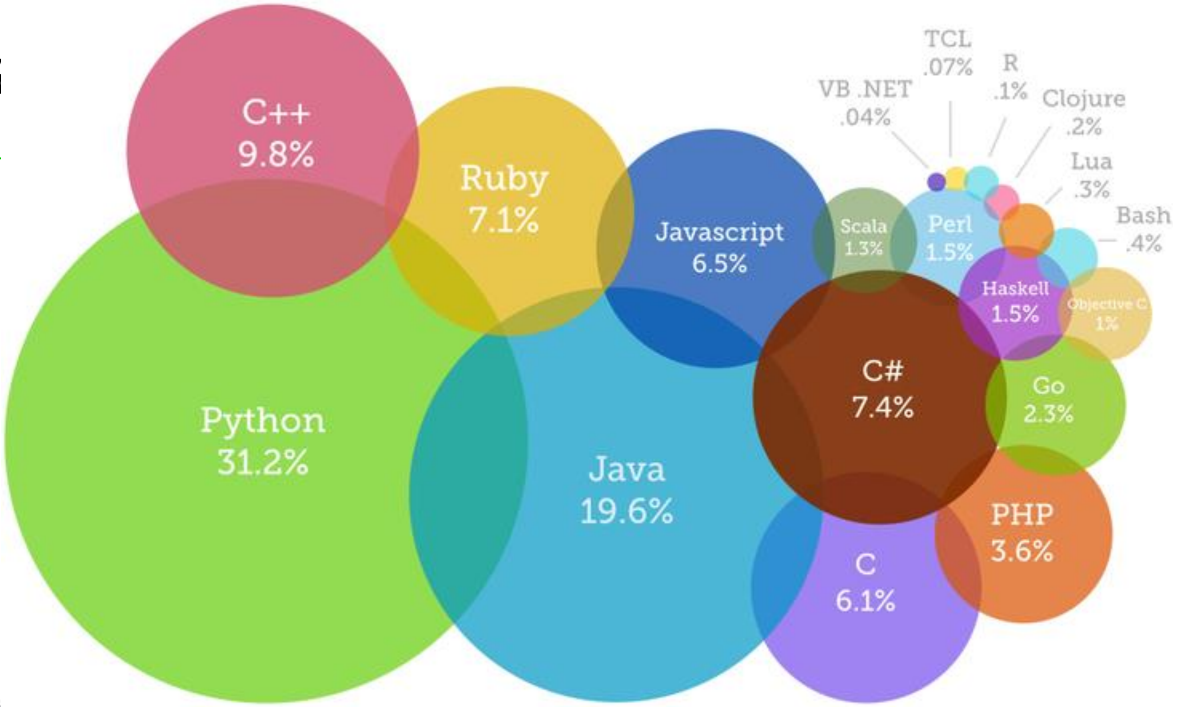
(1) Programming

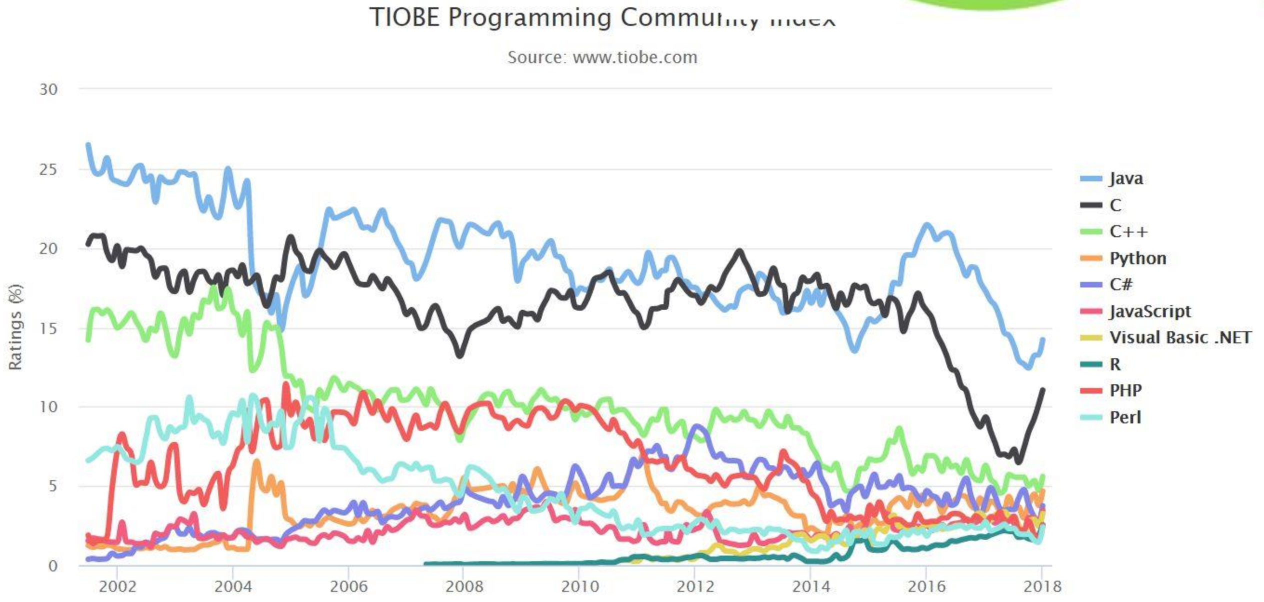
Construction languages

- **Programming languages (e.g., C, C++, Java, Python)**
- **Modeling languages (e.g., UML)**
- **Configuration languages (e.g., XML)**

- **Linguistic-based**
- **Mathematics-based (formal)**
- **Graphics-based (visual)**

(1) Programming La



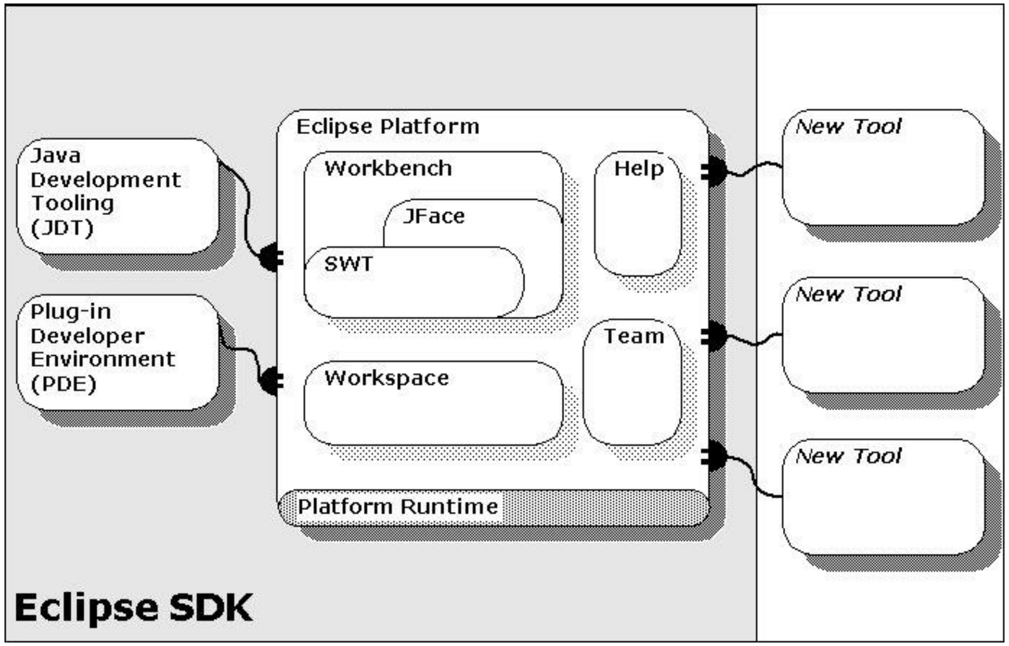


Programming tools

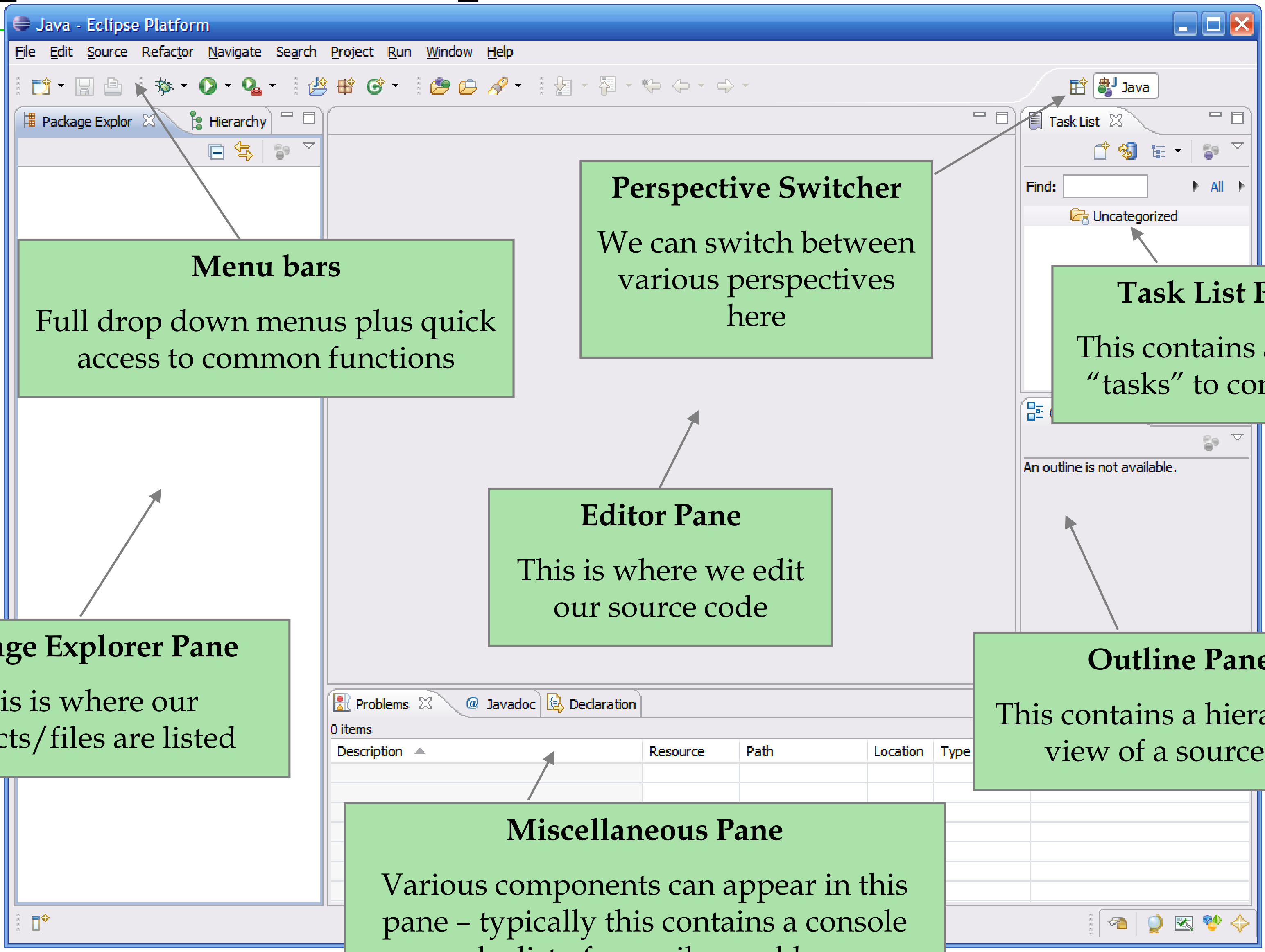
- **Integrated development environment (IDE): comprehensive facilities to programmers for software development.**
- **An IDE normally consists of:**
  - Source code editor with intelligent code completion, code refactoring tool
  - File management tool
  - Library management tool
  - Class browser, object browser, class hierarchy diagram for OOP
  - Graphical User Interface (GUI) builder
  - Compiler, interpreter
  - Build automation tools
  - Version control system
  - …
- **IDE should be extensible by more external third-party tools.**

Eclipse as an IDE example

- **Eclipse IDE: an open source IDE for Java, but not limited to, C/C++, PHP, Python, etc, started as a proprietary IBM product (Visual age for Smalltalk/Java)**
  - It contains a base workspace with tools for coding, building, running and debugging applications, and an extensible plug-in system for customizing the environment.
  - Plug-ins are structured bundles of code and/or data that contribute functionality to the system. Functionality can be contributed in the form of code libraries, platform extensions, or even documentation.
  - Plug-ins can define extension points, well-defined places where other plug-ins can add functionality.



Eclipse IDE Components



Menu bars

Full drop down menus plus quick access to common functions

Perspective Switcher

We can switch between various perspectives here

Task List Pane

This contains a list of "tasks" to complete

Editor Pane

This is where we edit our source code

Package Explorer Pane

This is where our projects/files are listed

Outline Pane

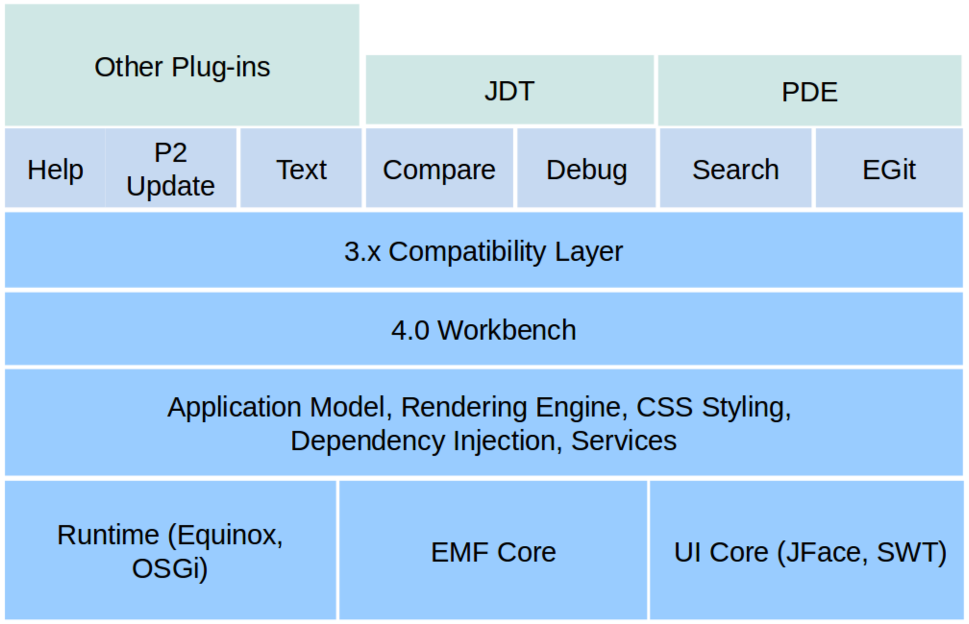
This contains a hierarchical view of a source file

Miscellaneous Pane

Various components can appear in this pane – typically this contains a console and a list of compiler problems

Core components of Eclipse IDE

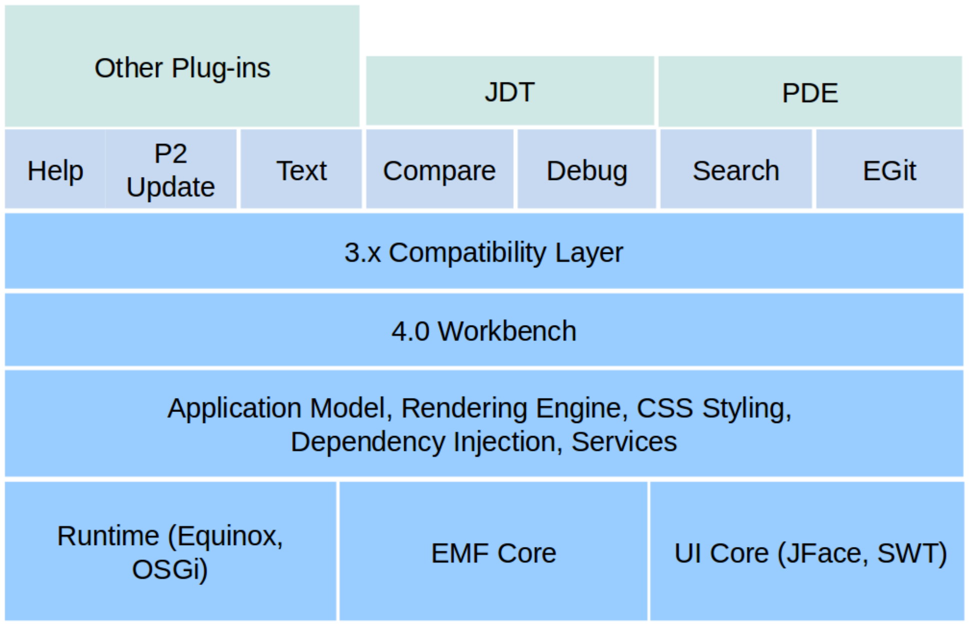
- **Runtime core**
  - The platform runtime core implements the runtime engine that starts the platform base and dynamically discovers and runs plug-ins.
  - A plug-in is a structured component that describes itself to the system using an OSGi manifest (MANIFEST.MF) file and a plug-in manifest (plugin.xml) file. The platform maintains a registry of installed plug-ins and the functionality they provide.
- **Resource management**
  - The resource management plug-in defines a common resource model for managing the artifacts of tool plug-ins. Plug-ins can create and modify projects, folders, and files for organizing and storing development artifacts on disk.



Core components of Eclipse IDE

http://help.eclipse.org/mars/index.jsp

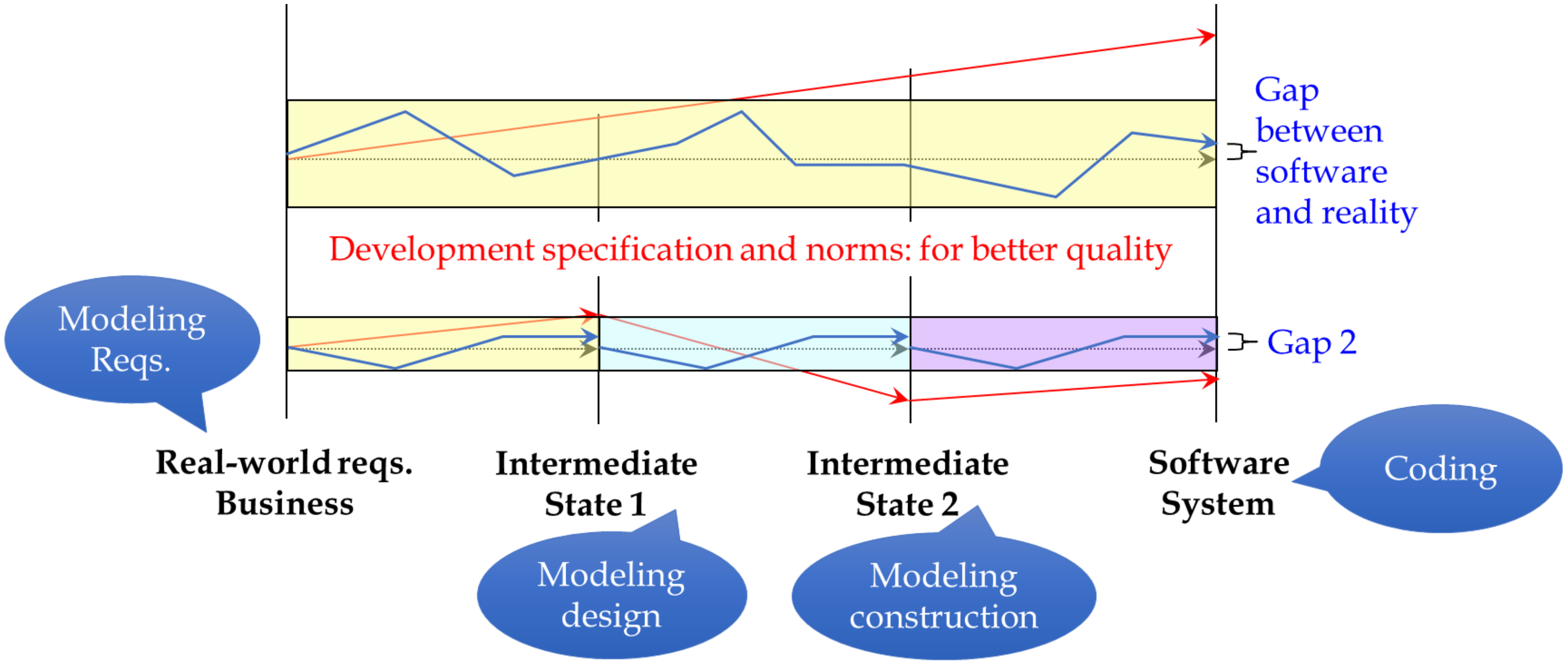
- **Workbench UI Core**
  - This plug-in implements the workbench UI and defines a number of extension points that allow other plug-ins to contribute menu and toolbar actions, drag and drop operations, dialogs, wizards, and custom views and editors.
  - Standard Widget Toolkit (SWT) and JFace framework
- **Java development tools (JDT)**
  - It extend platform workbench by providing features for editing, viewing, compiling, debugging, and running Java code.
- **Plug-in Dev. Env. (PDE)**
  - Tools that automate creation, manipulation, debugging, and deploying of plug-ins.



(2) Modeling languages and tools

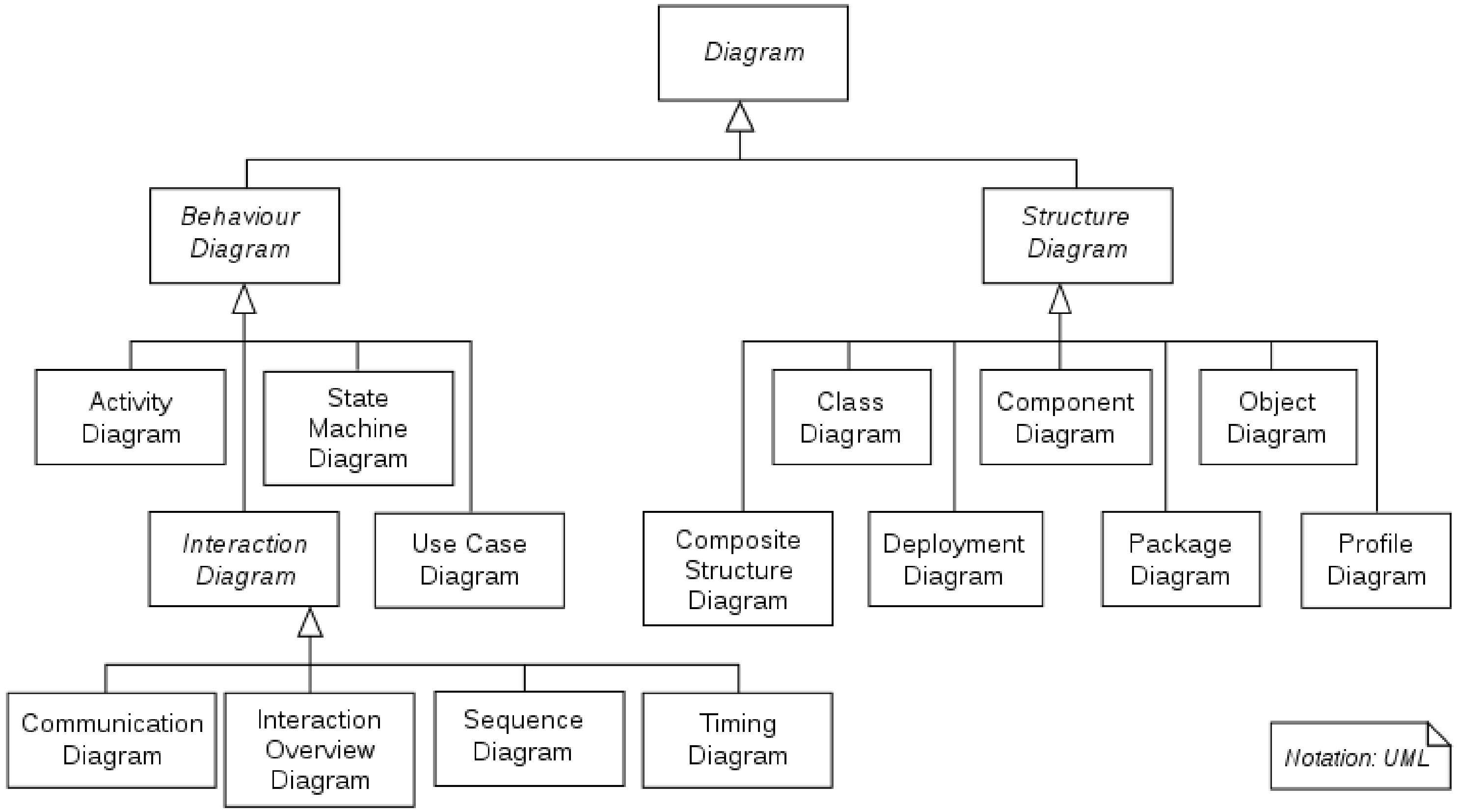
http://modeling-languages.com

- **A modeling language** is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules, with the objective of **visualizing, reasoning, verifying and communicating the design of a system**.



UML as a modeling language and tool example

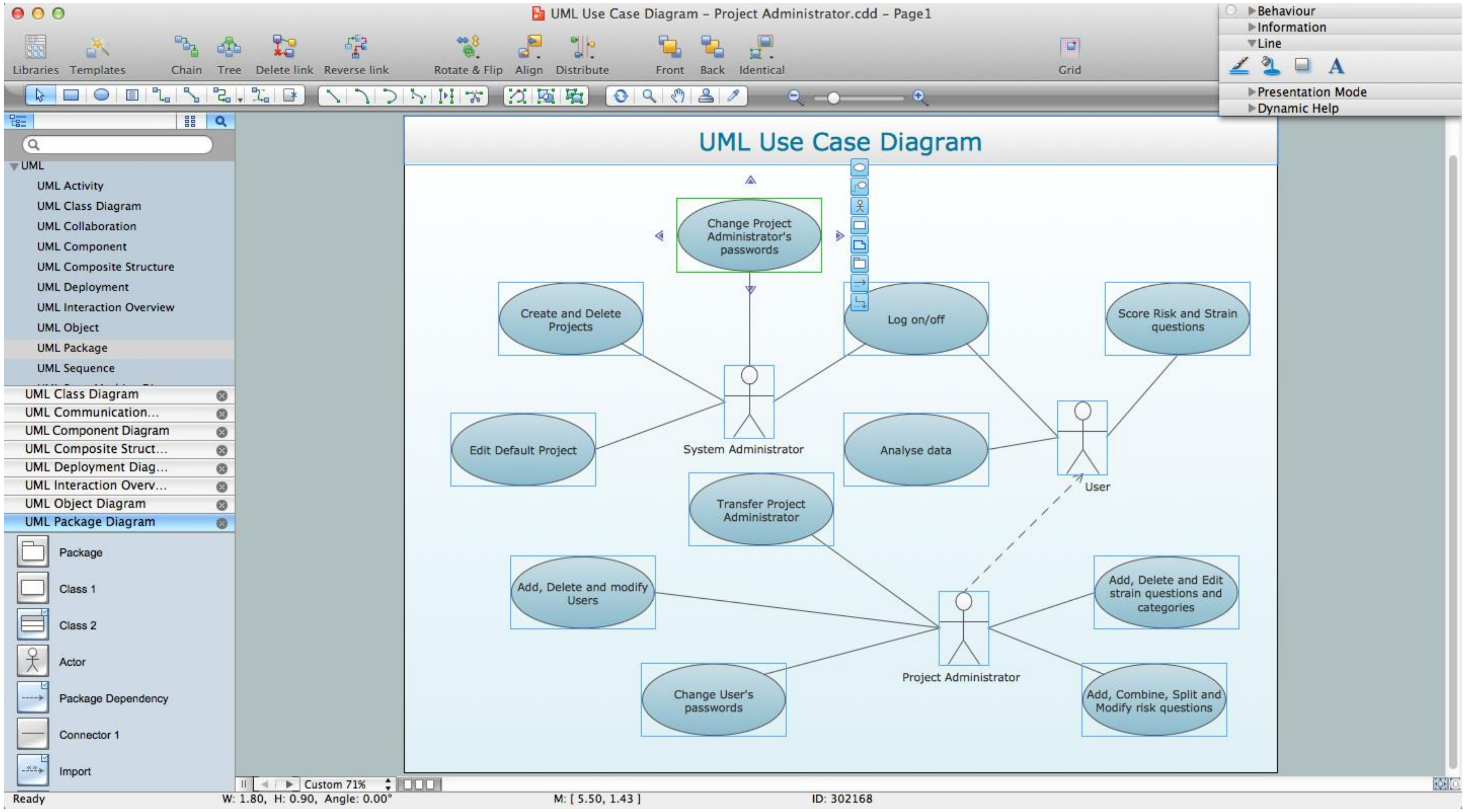
- **UML: Unified Modeling Language**



to learn UML in "Software Process and Tools" in the 3rd year

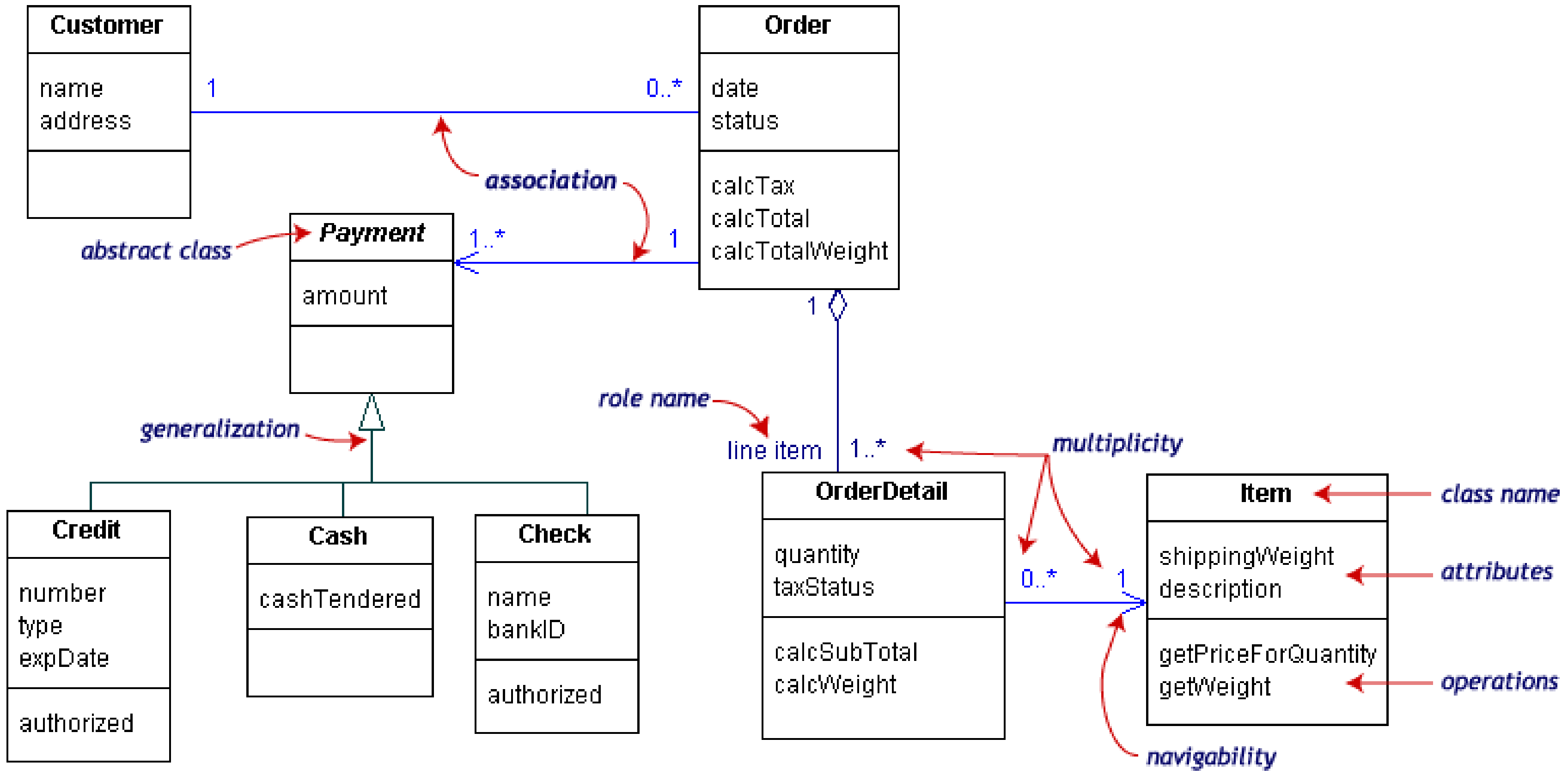
UML as a modeling language and tool example

- **UML Use Case Diagram**



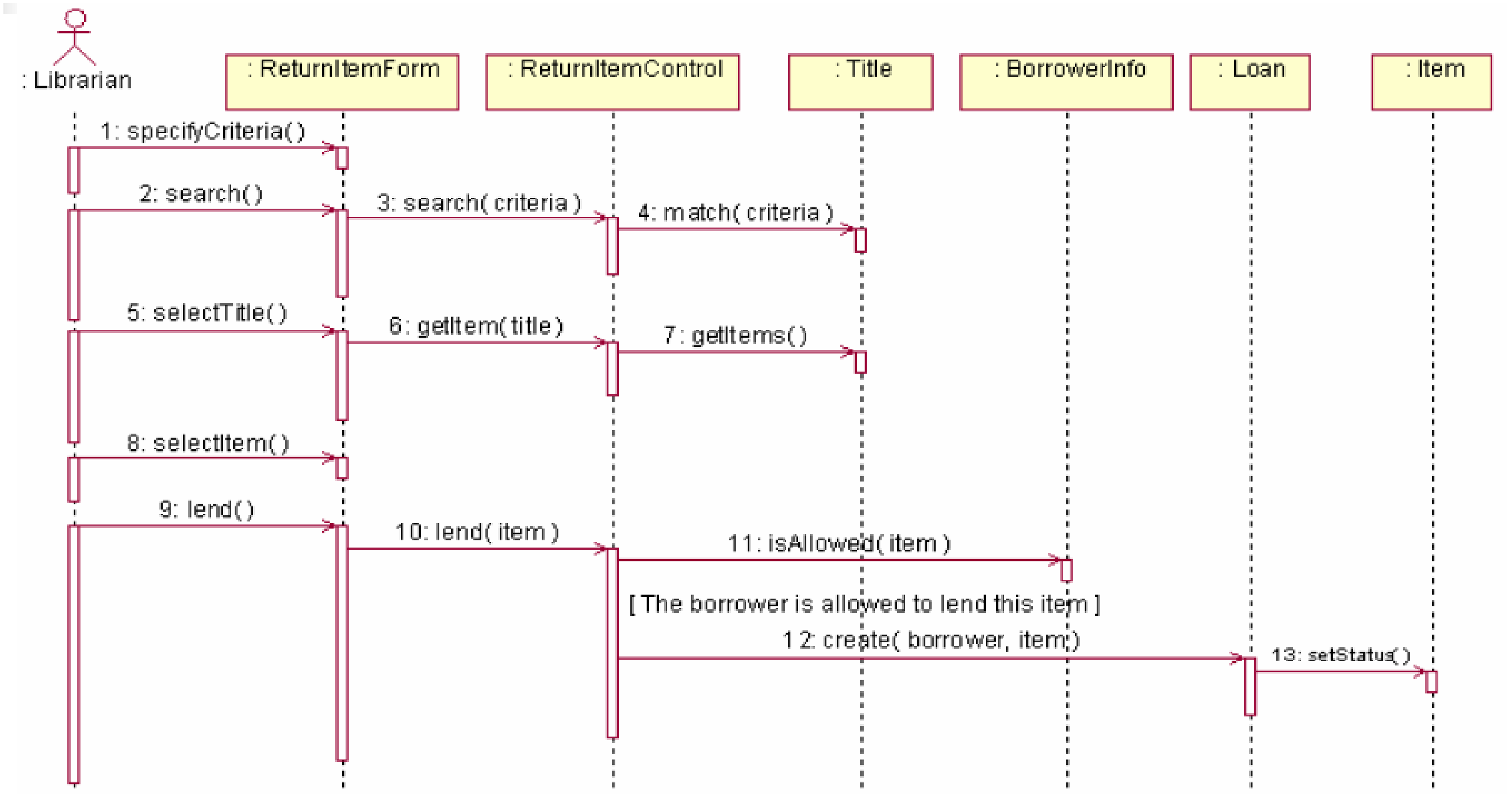
UML as a modeling language and tool example

- **UML Class Diagram**



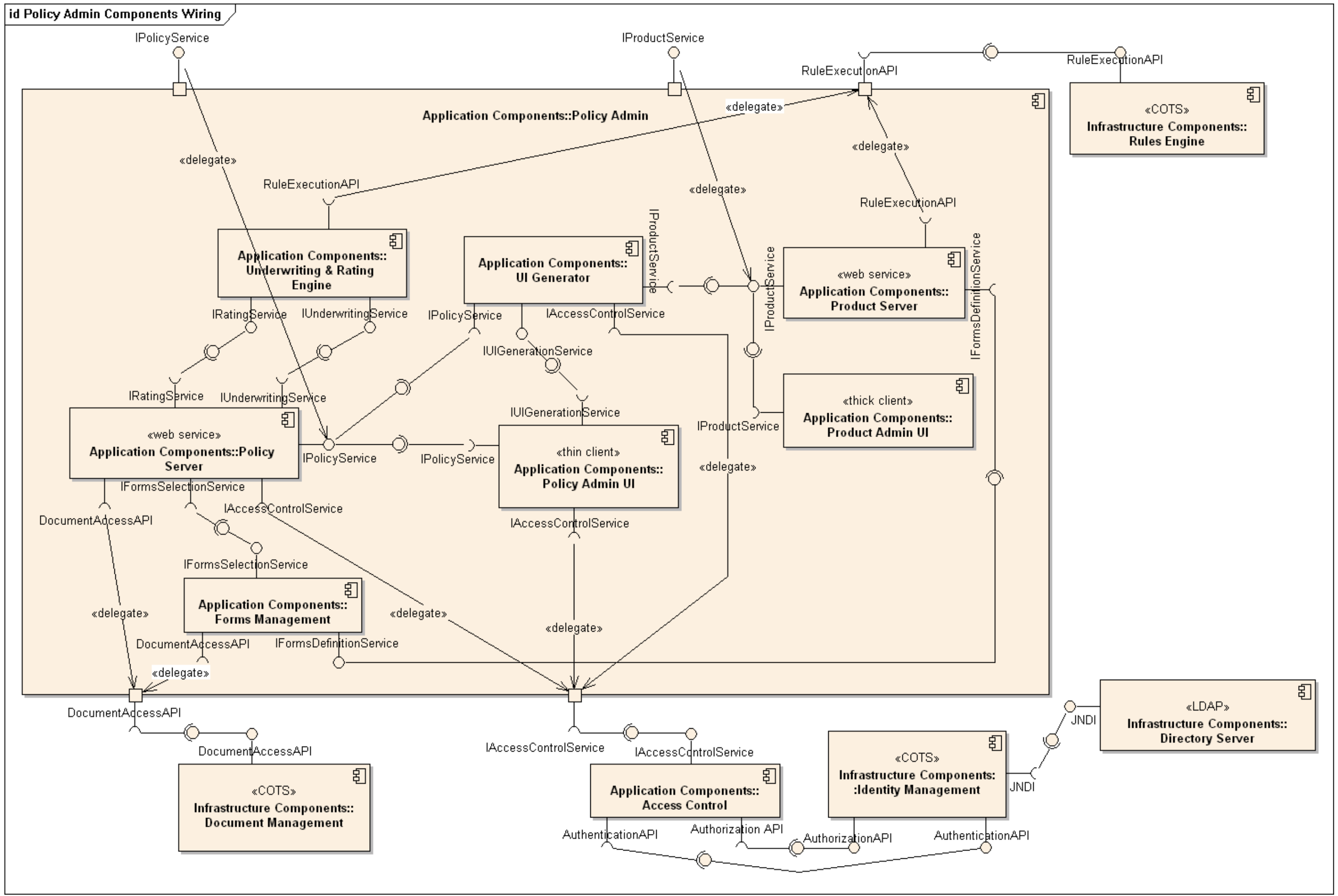
UML as a modeling language and tool example

- **UML Sequence Diagram**



UML as a modeling language and tool example

- **UML Component Diagram**

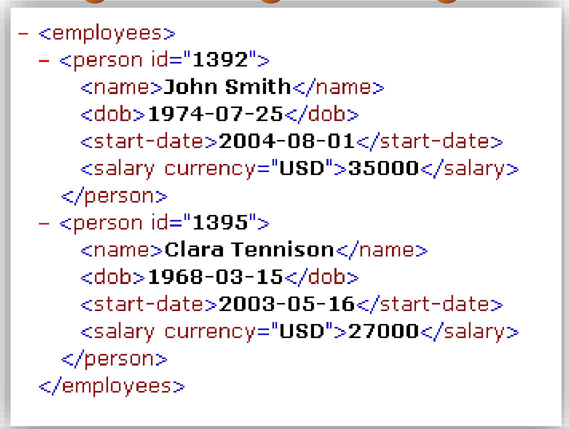

id Policy Admin Components Wiring
IPolicyService
IProductService
RuleExecutionAPI
«delegate»
Application Components::Policy Admin
«COTS» Infrastructure Components:: Rules Engine
RuleExecutionAPI
Application Components:: Underwriting & Rating Engine
Application Components:: UI Generator
IProductService
«web service» Application Components:: Product Server
IFormsDefinitionService
IRatingService
IUnderwritingService
IPolicyService
IAccessControlService
IUIGenerationService
«thick client» Application Components:: Product Admin UI
«web service» Application Components::Policy Server
«thin client» Application Components:: Policy Admin UI
IFormsSelectionService
DocumentAccessAPI
Application Components:: Forms Management
IFormsDefinitionService
«COTS» Infrastructure Components:: Document Management
Application Components:: Access Control
«COTS» Infrastructure Components: :Identity Management
«LDAP» Infrastructure Components:: Directory Server
JNDI
AuthenticationAPI
Authorization API
AuthorizationAPI

(3) Configuration languages

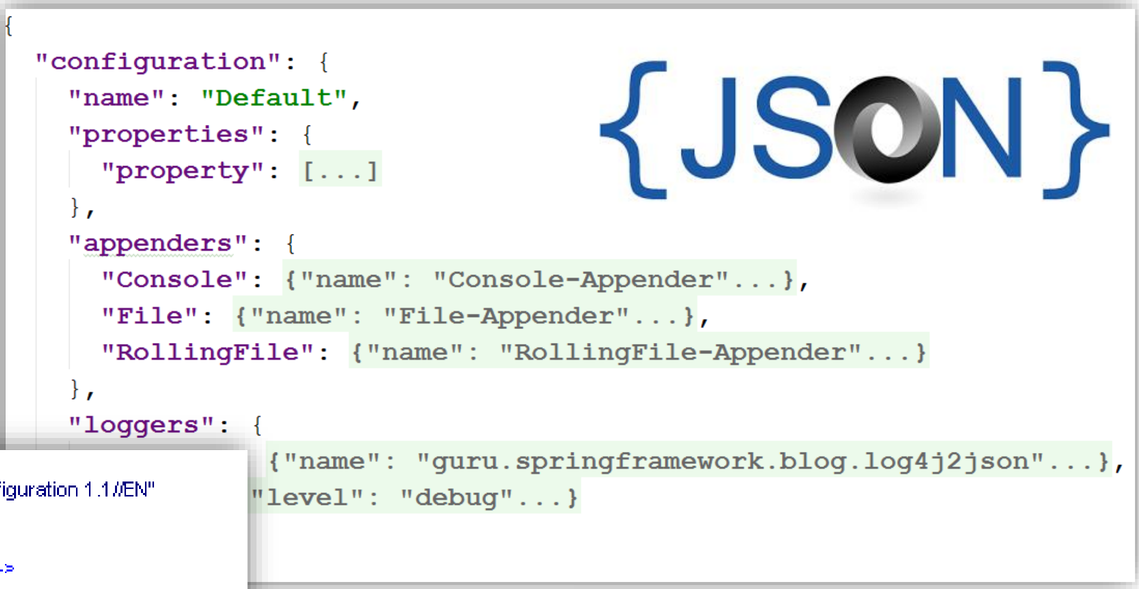
- **Configuration files configure the parameters and initial settings for programs.**
  - Applications should provide tools to create, modify, and verify the syntax of their configuration files;
  - Some computer programs only read their configuration files at startup. Others periodically check the configuration files for changes.
- **Purpose example:**
  - Deployment environment settings
  - Variants of application features
  - Variants of connections between components
- **Configuration language examples:**
  - Key-Value texts (.ini, .properties, .rc, etc)
  - XML, YAML, JSON

To separate stable and unstable parts

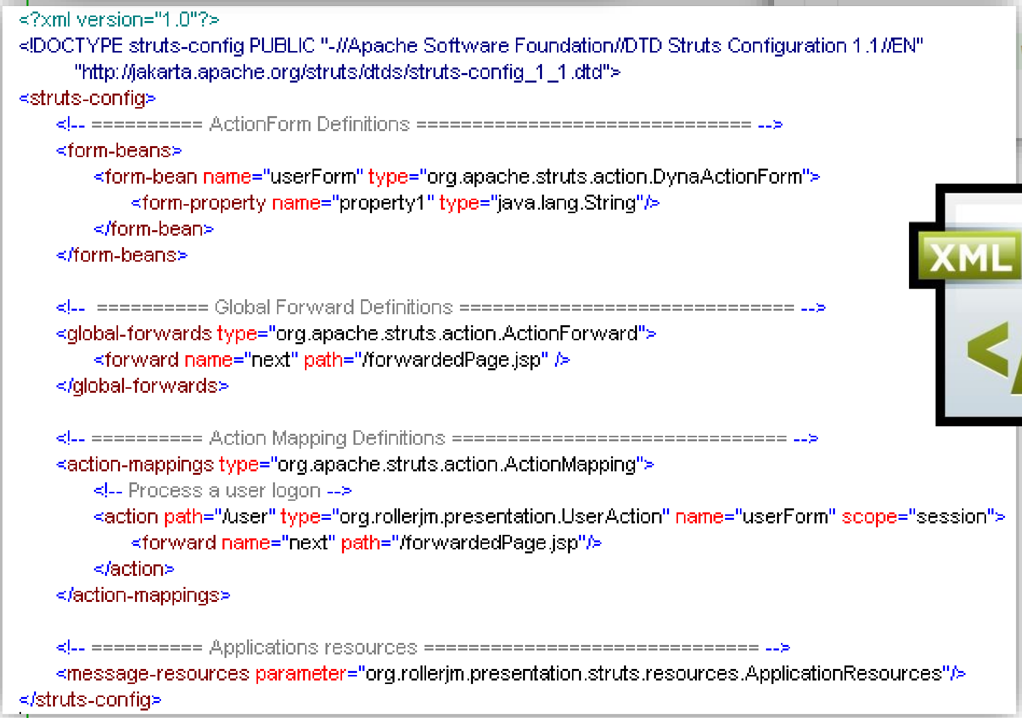
Configuration languages



```
- <employees>
  - <person id="1392">
      <name>John Smith</name>
      <dob>1974-07-25</dob>
      <start-date>2004-08-01</start-date>
      <salary currency="USD">35000</salary>
    </person>
  - <person id="1395">
      <name>Clara Tennison</name>
      <dob>1968-03-15</dob>
      <start-date>2003-05-16</start-date>
      <salary currency="USD">27000</salary>
    </person>
  </employees>
```



```
{
  "configuration": {
    "name": "Default",
    "properties": {
      "property": [...]
    },
    "appenders": {
      "Console": {"name": "Console-Appender"...},
      "File": {"name": "File-Appender"...},
      "RollingFile": {"name": "RollingFile-Appender"...}
    },
    "loggers": {
      {"name": "guru.springframework.blog.log4j2json"...},
     "level": "debug"...}
```

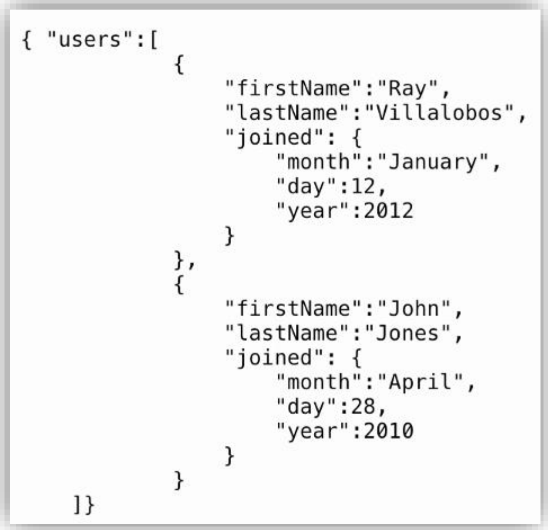


```
<?xml version="1.0"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
        "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
    <!-- ========== ActionForm Definitions ============================== -->
    <form-beans>
        <form-bean name="userForm" type="org.apache.struts.action.DynaActionForm">
            <form-property name="property1" type="java.lang.String"/>
        </form-bean>
    </form-beans>

    <!--  ========== Global Forward Definitions ============================== -->
    <global-forwards type="org.apache.struts.action.ActionForward">
        <forward name="next" path="/forwardedPage.jsp" />
    </global-forwards>

    <!-- ========== Action Mapping Definitions ============================== -->
    <action-mappings type="org.apache.struts.action.ActionMapping">
        <!-- Process a user logon -->
        <action path="/user" type="org.rollerjm.presentation.UserAction" name="userForm" scope="session">
            <forward name="next" path="/forwardedPage.jsp"/>
        </action>
    </action-mappings>

    <!-- ========== Applications resources ============================== -->
    <message-resources parameter="org.rollerjm.presentation.struts.resources.ApplicationResources"/>
</struts-config>
```

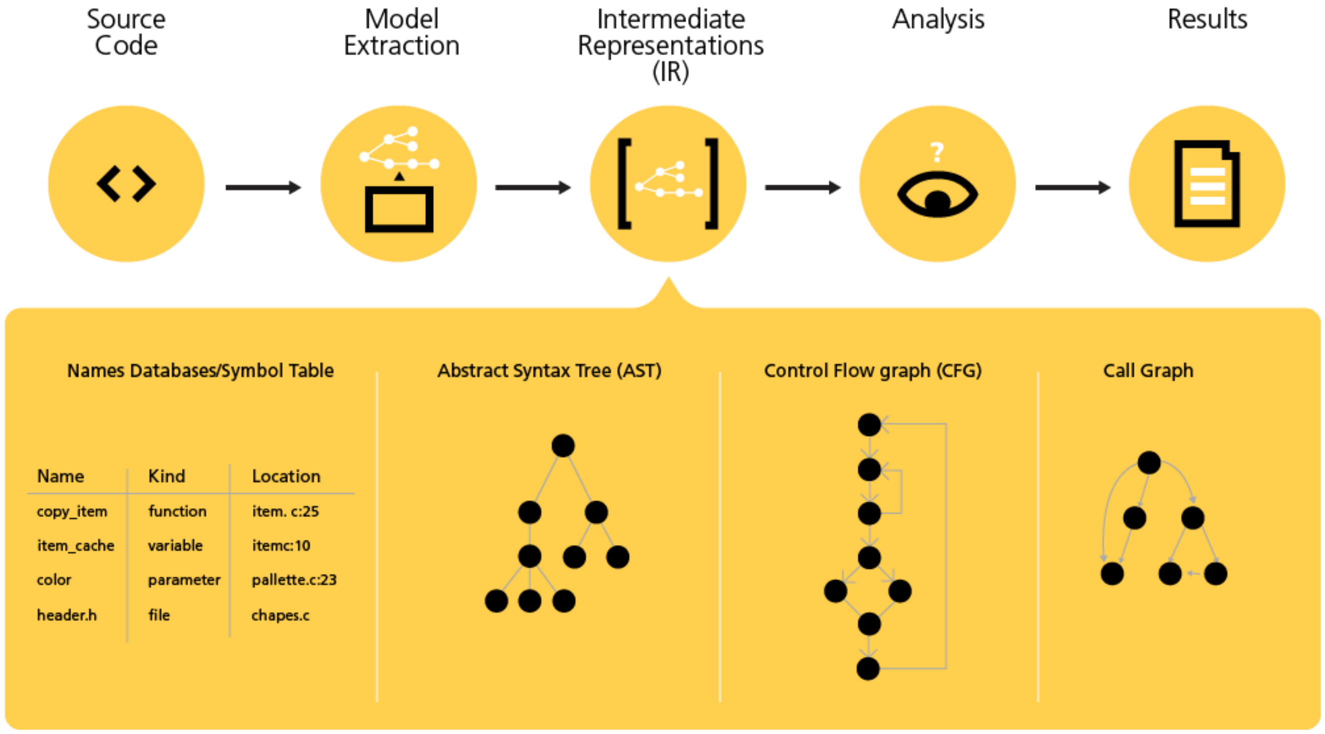


```
{ "users":[
            {
                "firstName":"Ray",
                "lastName":"Villalobos",
                "joined": {
                    "month":"January",
                    "day":12,
                    "year":2012
                }
            },
            {
                "firstName":"John",
                "lastName":"Jones",
                "joined": {
                    "month":"April",
                    "day":28,
                    "year":2010
                }
            }
     ]}
```

(2) Review and static code analysis

Review and static analysis/checking

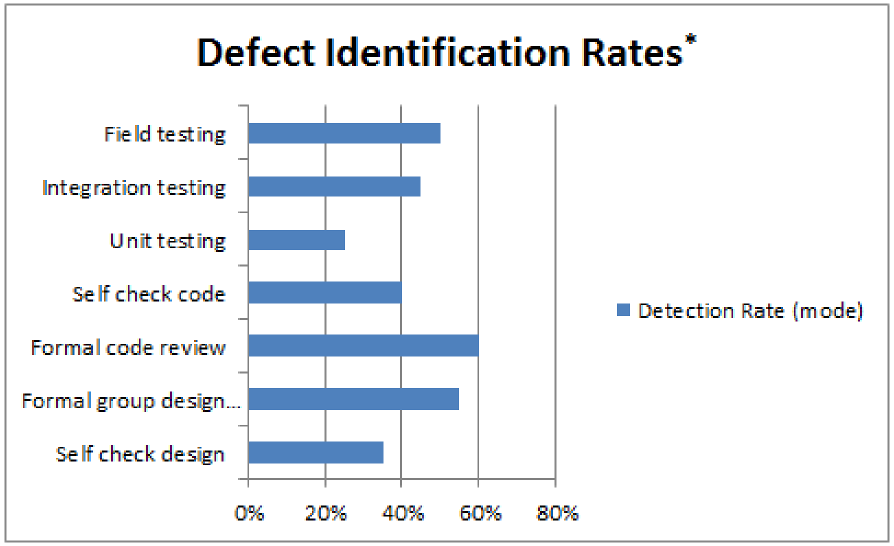
- **Code review is systematic examination (peer review) of source code.**
  - It is intended to find mistakes overlooked in the initial development phase, improving the overall quality of software.
  - Reviews are done in various forms such as pair programming, informal walkthroughs, and formal inspections.

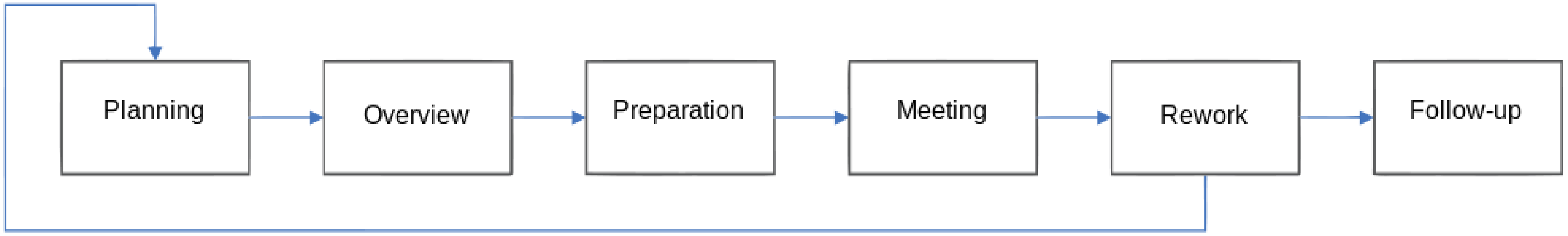

Source Code
Model Extraction
Intermediate Representations (IR)
Analysis
Results
Names Databases/Symbol Table
Abstract Syntax Tree (AST)
Control Flow graph (CFG)
Call Graph

Name	Kind	Location
copy_item	function	item. c:25
item_cache	variable	itemc:10
color	parameter	pallette.c:23
header.h	file	chapes.c

Formal code review

- **Formal code review, such as a Fagan inspection, involves a careful and detailed process with multiple participants and multiple phases.**
  - Formal code reviews are the traditional method of review, in which software developers attend a series of meetings and review code line by line, usually using printed copies of the material.
  - Formal inspections are extremely thorough and have been proven effective at finding defects in the code under review.





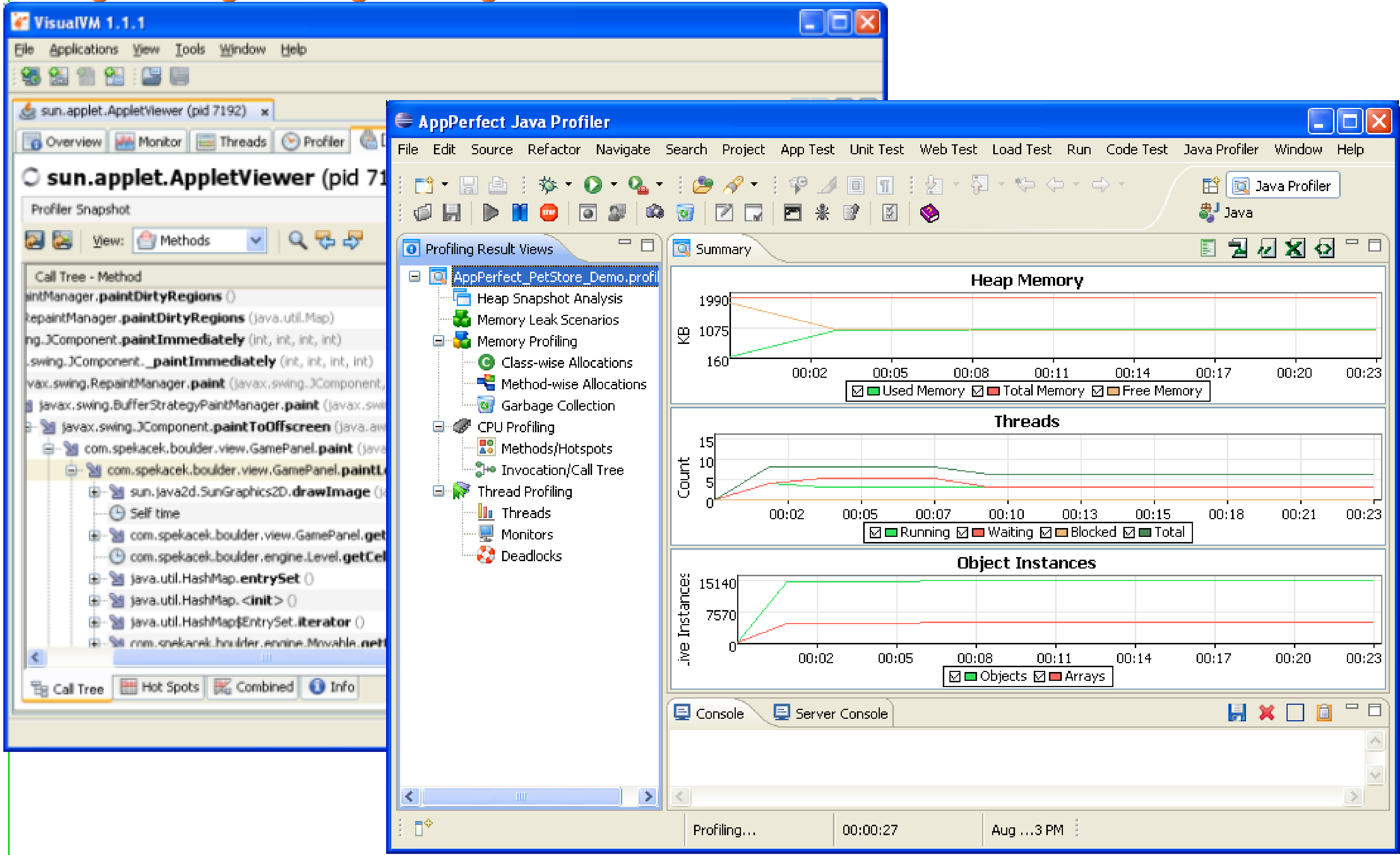
(3) Dynamic code analysis / profiling

Dynamic code analysis / profiling

- **Dynamic program analysis is the analysis of software that is performed by executing programs.**
- **The target program must be executed with sufficient test inputs to produce interesting behavior.**
- **Use of software testing measures such as code coverage helps ensure that an adequate slice of the program's set of possible behaviors has been observed.**

- **Profiling ("program profiling", "software profiling") is a form of dynamic program analysis that measures the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls.**

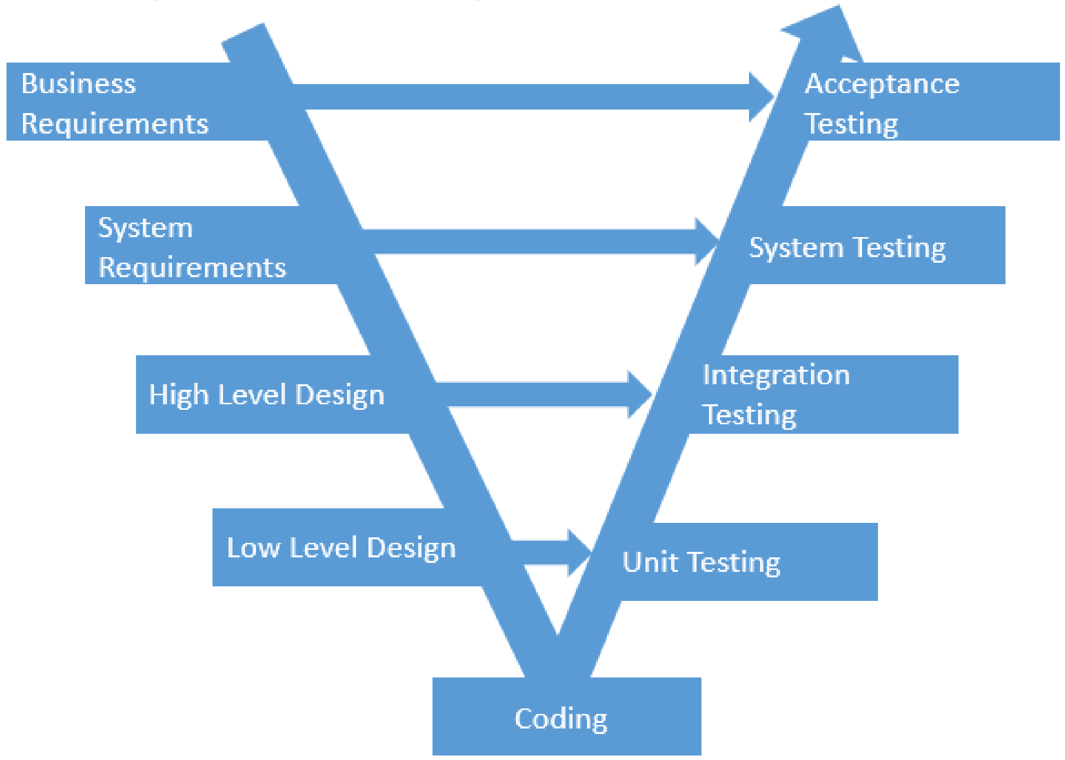
Dynamic code analysis / profiling


VisualVM 1.1.1
File Applications View Tools Window Help
sun.applet.AppletViewer (pid 7192)
Overview Monitor Threads Profiler
sun.applet.AppletViewer (pid 71
Profiler Snapshot
View: Methods
Call Tree - Method
intManager.paintDirtyRegions ()
RepaintManager.paintDirtyRegions (java.util.Map)
ng.JComponent.paintImmediately (int, int, int, int)
.swing.JComponent._paintImmediately (int, int, int, int)
vax.swing.RepaintManager.paint (javax.swing.JComponent,
javax.swing.BufferStrategyPaintManager.paint (javax.sw
javax.swing.JComponent.paintToOffscreen (java.aw
com.spekacek.boulder.view.GamePanel.paint (java
com.spekacek.boulder.view.GamePanel.paintL
sun.java2d.SunGraphics2D.drawImage (j
Self time
com.spekacek.boulder.view.GamePanel.get
com.spekacek.boulder.engine.Level.getCel
java.util.HashMap.entrySet ()
java.util.HashMap.<init> ()
java.util.HashMap$EntrySet.iterator ()
com.spekacek.boulder.engine.Movable.get
Call Tree Hot Spots Combined Info
AppPerfect Java Profiler
File Edit Source Refactor Navigate Search Project App Test Unit Test Web Test Load Test Run Code Test Java Profiler Window Help
Java Profiler
Java
Profiling Result Views
AppPerfect_PetStore_Demo.profil
Heap Snapshot Analysis
Memory Leak Scenarios
Memory Profiling
Class-wise Allocations
Method-wise Allocations
Garbage Collection
CPU Profiling
Methods/Hotspots
Invocation/Call Tree
Thread Profiling
Threads
Monitors
Deadlocks
Summary
Heap Memory
KB
1990
1075
160
00:02 00:05 00:08 00:11 00:14 00:17 00:20 00:23
☑ 🟩 Used Memory ☑ 🟥 Total Memory ☑ 🟧 Free Memory
Threads
Count
15
10
5
0
00:02 00:05 00:07 00:10 00:13 00:15 00:18 00:21 00:23
☑ 🟩 Running ☑ 🟥 Waiting ☑ 🟧 Blocked ☑ 🟩 Total
Object Instances
ive Instances
15140
7570
0
00:02 00:05 00:08 00:11 00:14 00:17 00:20 00:23
☑ 🟩 Objects ☑ 🟥 Arrays
Console Server Console
Profiling...
00:00:27
Aug ...3 PM

(4) Debugging and Testing

What is testing?

- **Software testing** is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.
- **Test techniques** include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use.
- **Software testing** involves the execution of a software component or system component to evaluate one or more properties of interest.



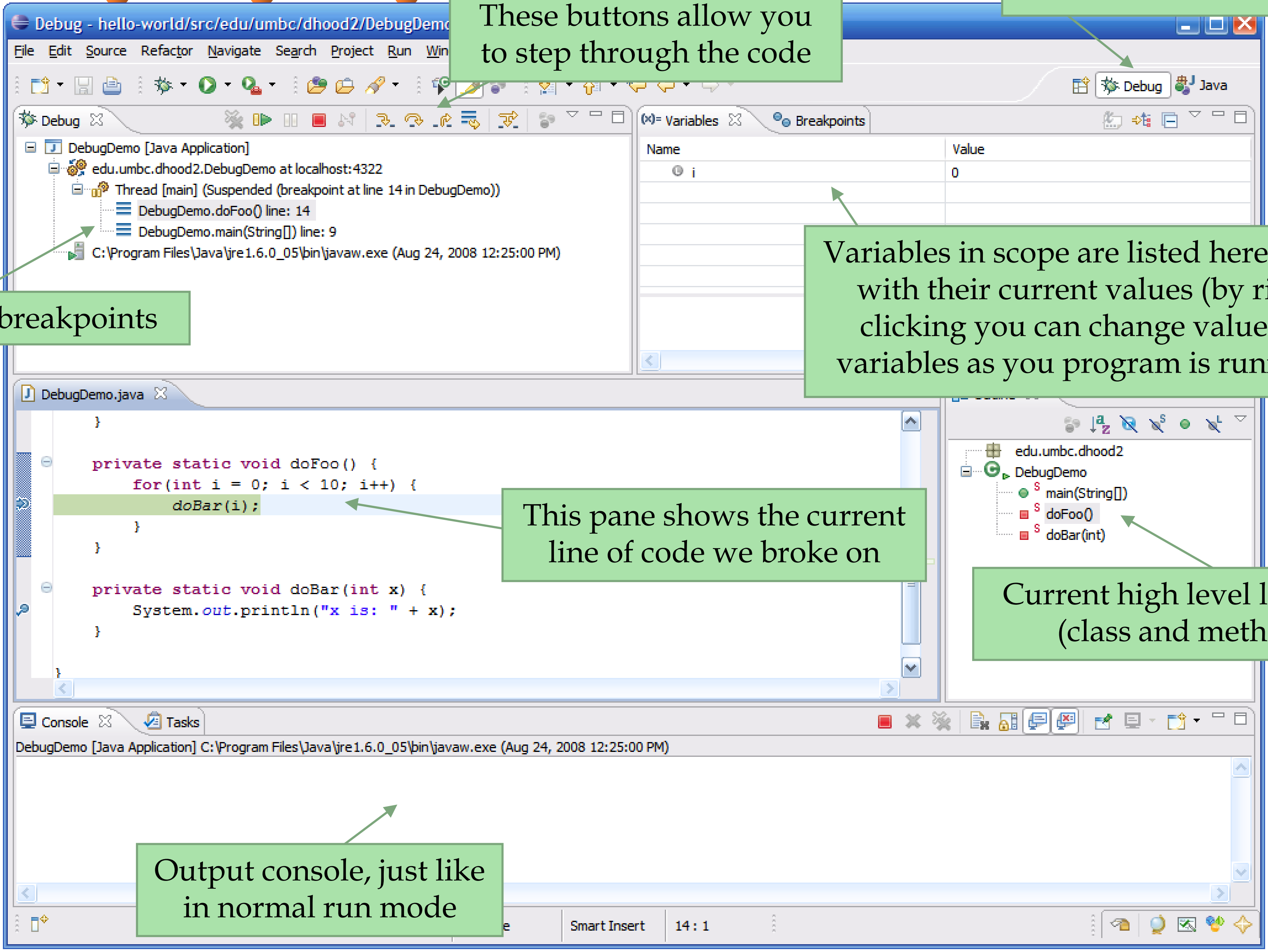
What is Debugging?

- **Debugging is the process of identifying the root cause of an error and correcting it.**
- **It contrasts with testing, which is the process of detecting the error initially, debugging occurs as a consequence of successful testing.**
  - On some projects, debugging occupies as much as 50 percent of the total development time.
  - For many programmers, debugging is the hardest part of programming.
- **Like testing, debugging isn't a way to improve the quality of your software, but it's a way to diagnose defects.**
  - Software quality must be built in from the start. The best way to build a quality product is to develop requirements carefully, design well, and use high-quality coding practices.
  - Debugging is a last resort.

Debug perspective in Eclipse

Note new Debug perspective – click Java to return to normal

These buttons allow you to step through the code


Debug - hello-world/src/edu/umbc/dhood2/DebugDemo
File Edit Source Refactor Navigate Search Project Run Win
Debug
Java
Debug
Variables
Breakpoints
DebugDemo.java
Console
Tasks
Smart Insert
14 : 1

```
DebugDemo [Java Application]
  edu.umbc.dhood2.DebugDemo at localhost:4322
    Thread [main] (Suspended (breakpoint at line 14 in DebugDemo))
      DebugDemo.doFoo() line: 14
      DebugDemo.main(String[]) line: 9
  C:\Program Files\Java\jre1.6.0_05\bin\javaw.exe (Aug 24, 2008 12:25:00 PM)
```

Name	Value
i	0

```
    }

    private static void doFoo() {
        for(int i = 0; i < 10; i++) {
            doBar(i);
        }
    }

    private static void doBar(int x) {
        System.out.println("x is: " + x);
    }

}
```

```
edu.umbc.dhood2
  DebugDemo
    main(String[])
    doFoo()
    doBar(int)
```

```
DebugDemo [Java Application] C:\Program Files\Java\jre1.6.0_05\bin\javaw.exe (Aug 24, 2008 12:25:00 PM)
```

List of breakpoints

Variables in scope are listed here along with their current values (by right clicking you can change values of variables as you program is running)

This pane shows the current line of code we broke on

Current high level location (class and method)

Output console, just like in normal run mode

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

(5) Refactoring

Refactoring

- **Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.**
  - Incurs a short-term time/work cost to reap long-term benefits, and a long-term investment in the overall quality of your system.
- **Refactoring is:**
  - restructuring (rearranging) code...
  - ...in a series of small, semantics-preserving transformations (i.e. the code keeps working)...
  - ...in order to make the code easier to maintain and modify
- **Refactoring is not just any old restructuring**
  - You need to keep the code working
  - You need small steps that preserve semantics
  - You need to have unit tests to prove the code works

哈爾濱工業大學 HARBIN INSTITUTE OF TECHNOLOGY

2 Narrow-sense process of software construction (Build)

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

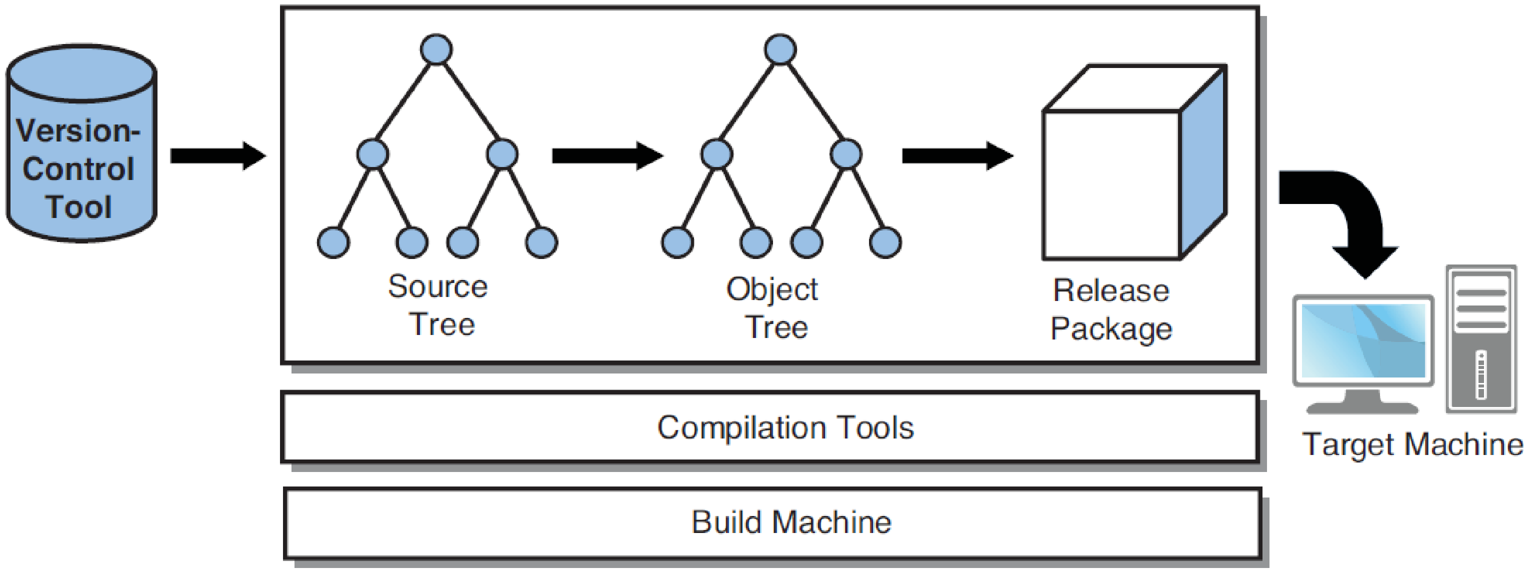
(1) Build System

Typical BUILD scenarios

- **The compilation of software written in traditional compiled languages, such as C, C++, Java and C#.**
- **The packaging and testing of software written in interpreted languages such as Perl and Python.**
- **The compilation and packaging of web-based applications.**
  - These include static HTML pages, source code written in Java or C#, hybrid files written using JSP (JavaServer Pages), ASP (Active Server Pages), or PHP (Hypertext Preprocessor) syntax, along with numerous types of configuration file.

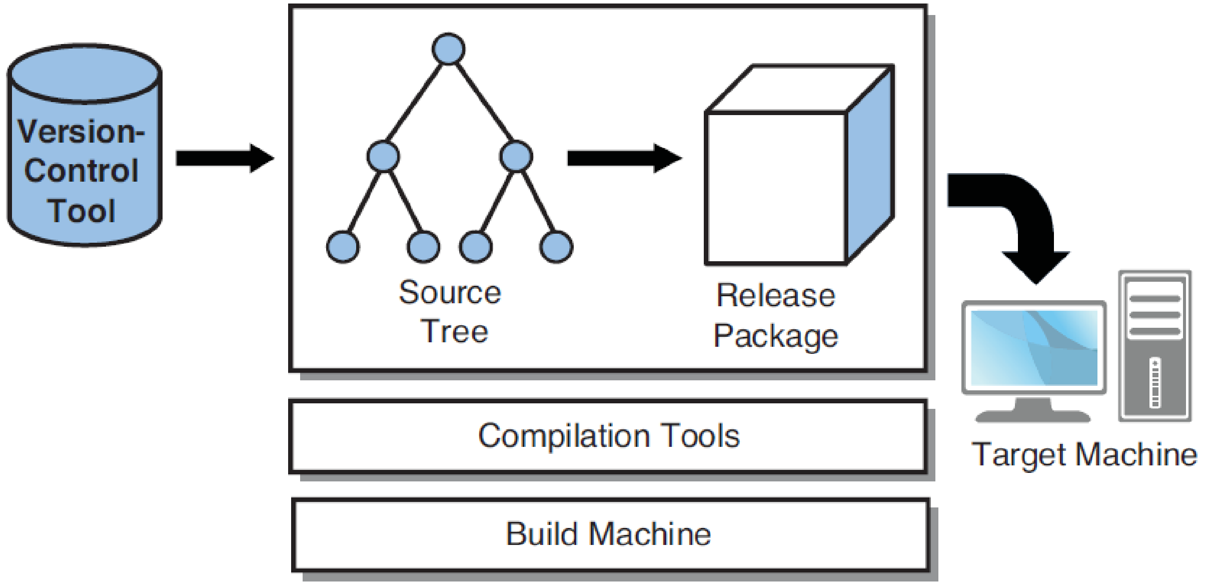
Compiled Languages

- **Compiled languages such as C, C++, Java, and C#. In this model, source files are compiled into object files, which are then linked into code libraries or executable programs.**
- **The resulting files are collected into a release package that can be installed on a target machine.**



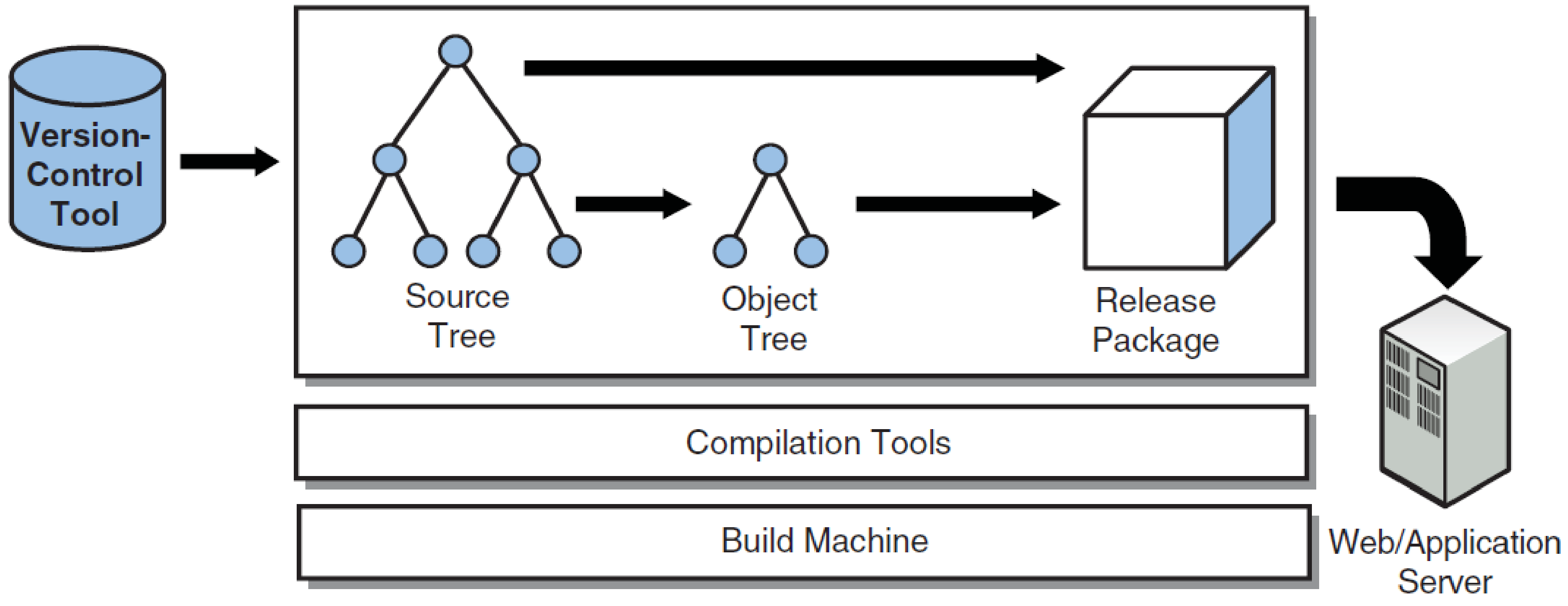
Interpreted Languages

- Interpreted source code isn't compiled into object code, so there's no need for an object tree. The source files themselves are collected into a release package, ready to be installed on the target machine.
- Compilation tools focus on transforming source files and storing them in the release package.
- Compilation into machine code is not performed at build time, even though it may happen at runtime.



Web-Based Applications

- The build system for a web-based application is a mix of compiled code, interpreted code, and configuration or data files. As Figure 1.3 shows, some files (such as HTML files) are copied directly from the source tree to the release package, whereas others (such as Java source files) are first compiled into object code.



Web-Based Applications

- **Static HTML files, containing nothing more than marked-up data to be displayed in a web browser. These files are copied directly to the release package.**
- **JavaScript files containing code to be interpreted by an end user's web browser. These files are also copied directly to the release package.**
- **JSP, ASP, or PHP pages, containing a mix of HTML and program code. These files are compiled and executed by the web application server rather than by the build system. These files are also copied to the release package, ready for installation on the web server.**
- **Java source files to be compiled into object code and packaged as part of the web application. The build system performs this transformation before packaging the Java class files. The Java classes are executed on the web application server or even within the web browser (using a Java applet).**

哈爾濱工業大學 HARBIN INSTITUTE OF TECHNOLOGY

(2) Components of a Build System

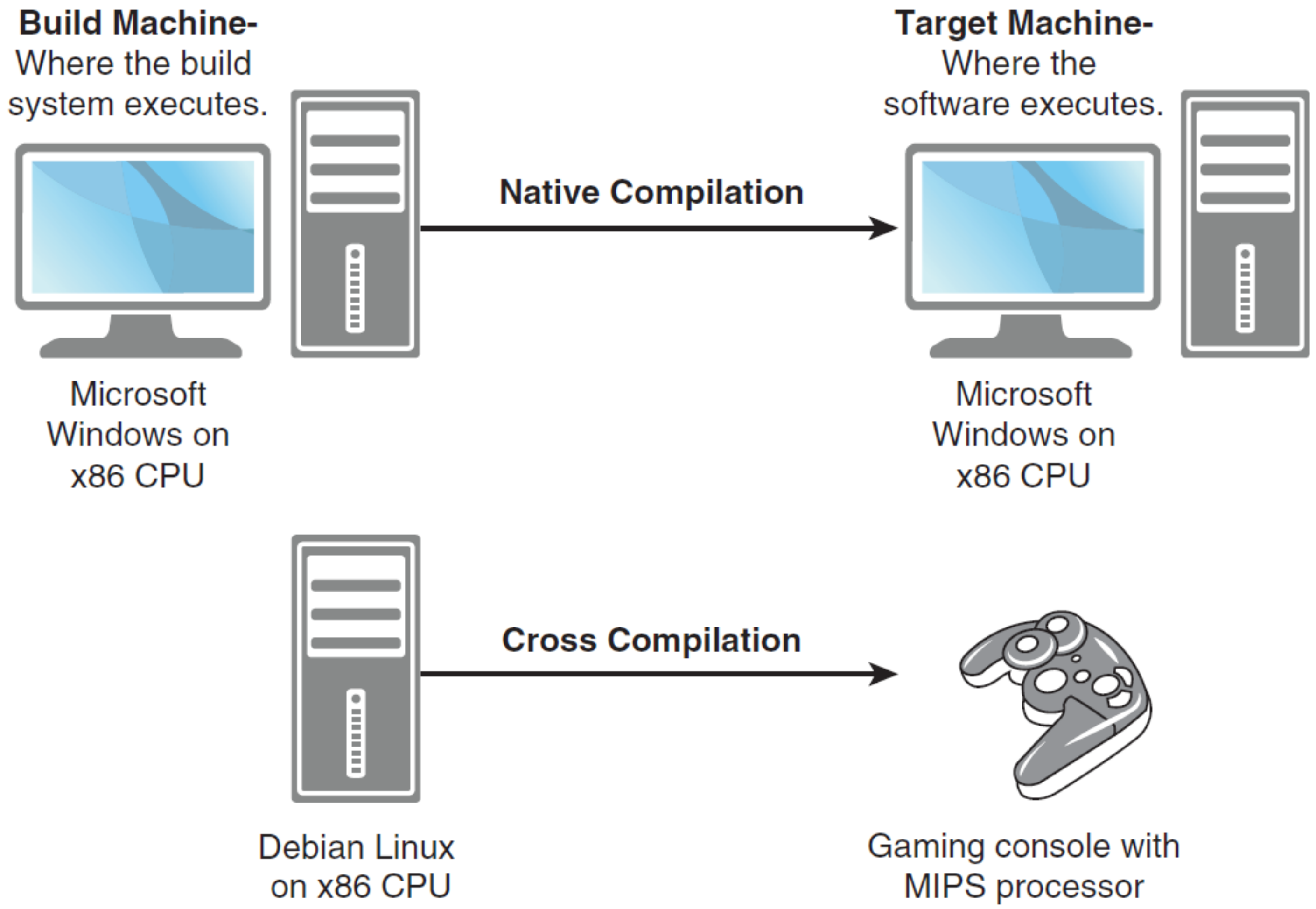
Components of a Build System

- **Version-Control Tools**
- **Source Tree:** a program's source code is stored as a number of disk files. This arrangement of the files into different is known as the **source tree**. The structure of the source tree often reflects the architecture of the software.
- **Object Trees:** a separate tree hierarchy that stores any object files or executable programs constructed by the build process.
- **Compilation Tools:** a program that translate the human-readable source files into the machine-readable executable program files.
  - Compiler: source files $\Rightarrow$ object files
  - Linker: multiple related object files $\Rightarrow$ executable program image
  - UML-based code generator: models $\Rightarrow$ source code files
  - Documentation generator: scripts $\Rightarrow$ documents

Components of a Build System

- **Build Tools:** a program that functions at a level above compilation tools. It must have sufficient knowledge of the relationship between source files and object files that it can orchestrate the entire build process. The build tool calls upon the necessary compilation tools to produce the final build output.
- **Build Machines:** the machine on which the compilation and build tools execute.
  - Native compilation environment: the software is executed on a target machine that's identical to the build machine;
  - Cross-compilation environment: requires two different machines, with a different operating system or CPU on the target machine.

Native compilation vs. cross-compilation


Build Machine- Where the build system executes.
Target Machine- Where the software executes.
Native Compilation
Microsoft Windows on x86 CPU
Microsoft Windows on x86 CPU
Cross Compilation
Debian Linux on x86 CPU
Gaming console with MIPS processor

Components of a Build System

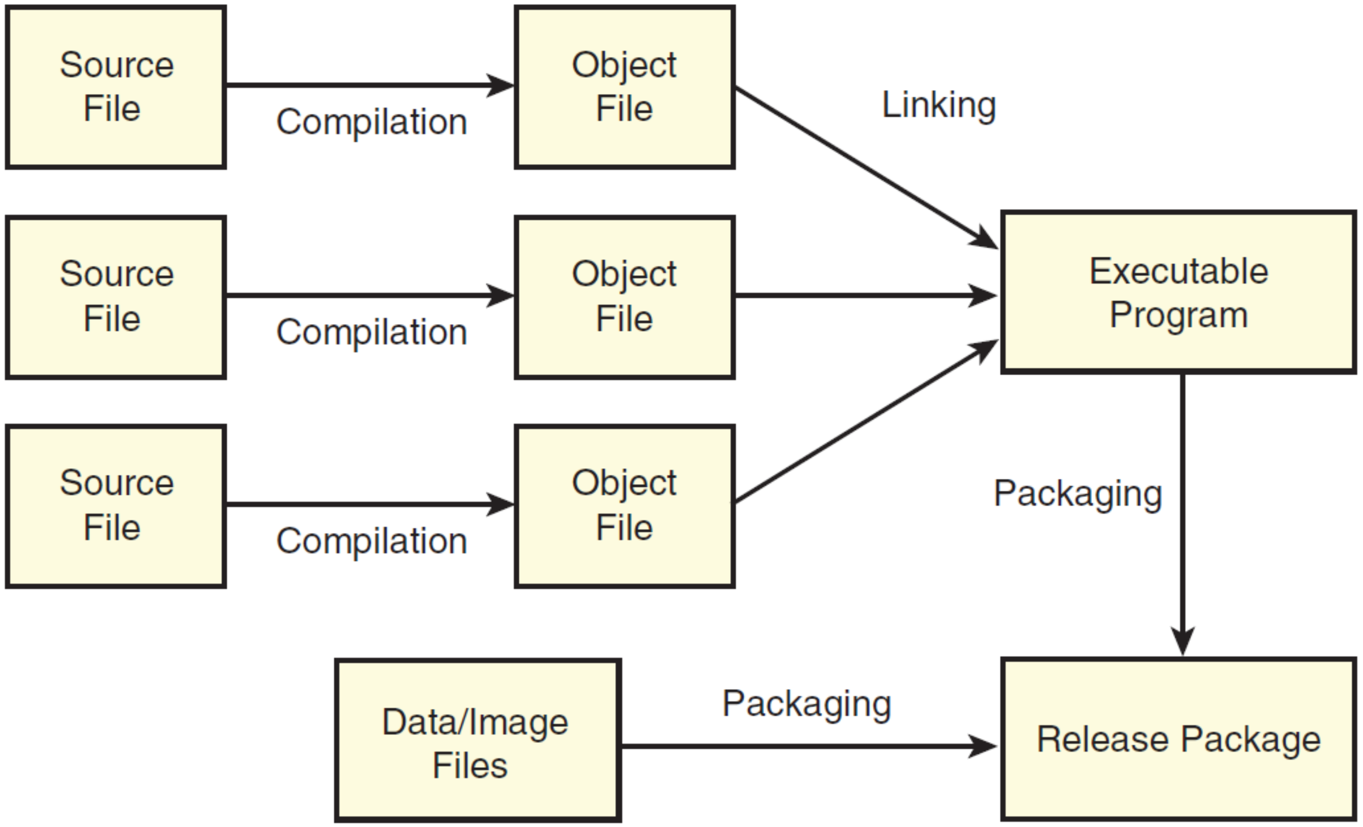
- **Release Packaging and Target Machines:** produces something that you can actually install on a user's machine.
  - To extract the relevant files from the source and object trees and store them in a release package.
  - The release package should be a single disk file and should be compressed to reduce the amount of time it takes to download.
  - Any nonessential debug information should be removed so that it doesn't clutter the software's installation.
- **Types of packaging:**
  - Archive files: zip and unzip
  - Package-management tools: UNIX-style such as .rpm and .deb
  - Custom-built GUI installation tools: Windows-style

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

(3) The Build Process and Build Description

Build process

- **Build process:** the build tool invokes each of the compilation tools to get the job done, which is an end-to-end sequence of events.



Build description

- A build tool needs the **build description** to be written in a text-based format.
- For example, when using `Make`, the interfile dependency information is specified in the form of **rules**, which are stored in a file named `Makefile`.

How a build system is used

- **Developer (or private) build:** The developer has checked out the source code from VCS and is building the software in a private workspace. The resulting release package will be used for the developer's private development.
- **Release build:** to provide a complete software package for the test group to validate. When the testers are convinced that the software is of high enough quality, that same package is made available to customers. The source tree used for a release build is compiled only once, and the source tree is never modified.
- **Sanity build:** This is similar to a release build, except that the software package isn't destined for a customer. Instead, the build process determines whether the current source code is free of errors and passes a basic set of sanity tests. This type of build can occur many times per day and tends to be fully automated.
  - Daily build / nightly build 每日构建

(4) Compilation Tools in Java

What about Java

- One big selling point of the Java language has been its “write once, run anywhere” philosophy.
- That is, it should be possible to compile a Java program on a Linux machine, yet run it on a Windows or Solaris machine without any modification.
- This is achieved by using a standard set of byte codes that are interpreted by the **Java Virtual Machine (JVM)**.
- Because of Java’s security features, it’s possible to restrict the environment in which a Java program executes, therefore allowing untrusted programs to be executed without fear of harming the host computer.

Compilation Tools in Java

- The Java Development Kit (JDK)
- GNU Java Compiler
- Eclipse Java Compiler (ECJ)

Source files in Java

- **Hello.java**

```java
package com.arapiki.examples;

public class Hello {

    private String words;

    public Hello(String message) {
       words = message;
    }

    public void speak() {
        System.out.println("Hello " + words);
    }
}
```

- **Main.java**

```java
package com.arapiki.examples;

import com.arapiki.examples.Hello;

public class Main {

    public static void main(String args[]) {
       Hello speaker = new Hello("World");
       speaker.speak();
    }
}
```

Object files in Java

- The object file format for a Java class is known as a **class file** and has the suffix of .class.
- Machine-independent byte codes to describe the flow of the program instead of compiling directly into native machine code.
- A Java Virtual Machine (JVM) is required to load and interpret these byte codes, although the JVM likely first translates them into native machine code before actually executing the program.
- Use **javac** command to translate Java source files into class files.

Executable Programs of Java

- Java programming is **dynamic class loading**. No build-time link step is required to produce an executable program. Instead, Java classes are individually loaded into memory when a running program needs them. There's no single executable program image to be loaded.
- Java programs are simply a collection of dynamic libraries, although individual classes are loaded one at a time instead of as part of much larger shared libraries.
- **A Java program requires two things to execute:**
  - The JVM must be provided with the name of a class that contains a **main** method. This is used as the starting point for execution.
  - The JVM must also be provided with a **class path**, which is used to identify where additional classes can be located.

Executable Programs of Java

```
C:\Work> javac -verbose:class com.arapiki.examples.Main
[Loaded java.lang.Object from shared objects file]
[Loaded java.io.Serializable from shared objects file]
[Loaded java.lang.Comparable from shared objects file]
[Loaded java.lang.CharSequence from shared objects file]
[Loaded java.lang.String from shared objects file]
...
[... lots of output removed ...]
...
[Loaded java.security.Principal from shared objects file]
[Loaded java.security.cert.Certificate from shared objects file]
[Loaded com.arapiki.examples.Main from file:/C:/Work/]
[Loaded com.arapiki.examples.Hello from file:/C:/Work/]
Hello World
```

Libraries in Java

- In addition to specifying a list of directories in which .class files can be found, Java classes can be placed into larger archive files, known as **JAR files**.
- Most Java applications prefer the JAR file format (with a .jar suffix), simply because it's easier to manipulate JAR files than package and distribute a large number of .class files.
- To create a JAR file:

```
C:\Work> jar -tf example.jar
com/
com/arapiki/
com/arapiki/examples/
com/arapiki/examples/Hello.class
com/arapiki/examples/Main.class
```

- To use a JAR file:

```
C:\Work> java -cp example.jar com.arapiki.examples.Main
Hello World
```

Libraries in Java

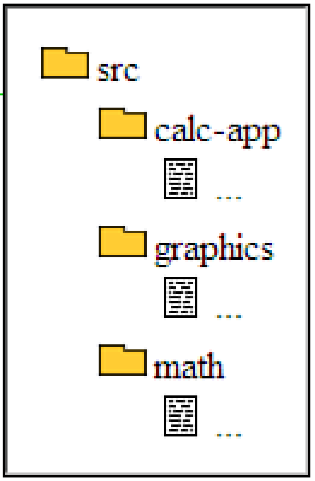
- JAR file is commonly used as a means of distributing programs. Not only do you package your own software in JAR files, but you can incorporate third-party packages by obtaining other people's JAR files and adding them to your own class path.
- Because of the dynamic loading system, you can replace and upgrade JAR files whenever you want.

(5) Subtargets and Build Variants

Three different ways of build

- **Building subtargets:** Developers who are making only incremental changes to one part of the build tree prefer to rebuild only the portion of the tree they're actively working on.
- **Building different editions of the software:** The output is customized to vary the software's behavior. These variations might include support for natural languages or support for different combinations of product features, such as a Home or Professional edition.
- **Building different target architectures:** To support a software product on different target machines, you must compile the same set of source files for a variety of different CPU types and operating systems. This includes CPUs such as x86, MIPS, and PowerPC, as well as operating systems such as Linux, Windows, and Mac OS X.

Building subtargets



- Any large piece of software can be divided into a number of subcomponents, often in the form of a static or dynamic library. Each component provides only a portion of the program's full functionality and is developed somewhat independently from other components.
- To avoid time-consuming in building the entire source tree to create the final executable program, it's better to choose **to limit the number of subcomponents** they build instead of always rebuilding the whole source tree.

Building Different Editions of the Software

- **Building different editions**
  - Language and culture, localization
  - Hardware variations
  - Pricing options
- **Specifying the Build Variant:**

	Home	Professional
English	Valid	Valid
French	Valid	Valid
German	Valid	Not Supported

```
$ make all LANGUAGE=French EDITION=Home
... build output will be shown ...

$ make all LANGUAGE=French EDITION=Professional
... build output will be shown ...
```

- **Varying the Code:**
  - Line-by-line variation
  - Per-variant files
  - Per-variant directories
  - Per-variant build description files

Building Different Target Architectures

- This type of variant is relevant only when programming in languages such as C and C++, which compile into native code.
- It’s not relevant for Java and C#, which use machine-independent virtual machines.

(6) The Build Tools

Build tools

- **For Java:**
  - Make
  - Ant
  - Maven
  - Gradle
  - Eclipse

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Make

Java make

- **Use Make and makefile to build Java projects**
  - https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
  - https://www.cs.swarthmore.edu/~newhall/unixhelp/javamakefiles.html
  - http://www.cnblogs.com/jiqingwu/archive/2012/06/13/java_makefile.html
- **Commands:**

– `# make new`	To generate sub-directories (`src`, `bin`, `res`)
– `# make build`	To compile and generate java classes in `bin`
– `# make clean`	Clean the compilation results
– `# make rebuild`	Clean the compilation results and re-complie (`clean + build`)
– `# make run`	To check the execution results
– `# make jar`	To generate executable `jar` files

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Apache Ant

Apache Ant

- **Ant** is a build tool developed by **Apache Software Foundation**.
- **Ant follows is to encapsulate each activity in the build system into a high-level task.**
  - `ant compile`: For compiling all the Java source files into class files
  - `ant jar`: For packaging the class files into a single Jar file
  - `ant package`: For creating a full software release package, complete with a version number
  - `ant clean`: For removing all generated files from the build tree
  - `ant javadoc`: For generating API documentation using the Javadoc tool
  - `ant`: For executing the default target, which is most likely the same as the package target
- **Buildfile**: `build.xml`

Built-in and optional tasks of Ant

- **Basic file operations** such as `mkdir`, `copy`, `move`, and `delete`
- **The creation of file archives** using an array of different formats (such as `.tar`, `.gz`, `.zip`, `.jar`, and `.rpm`)
- **The compilation of Java code**, including special tools for RMI and `JSP` compilation
- **The automatic generation of API documentation**, using the `Javadoc` tool
- **Direct access to version-control tools** such as `CVS`, `Perforce`, and `ClearCase`
- **Build lifecycle features**, such as updating build version numbers, sending email messages, and playing sounds to indicate the completion of the build process
- Check complete task list: `http://ant.apache.org/manual/tasksoverview.html`

An Ant example

- **Detailed help can be found from** http://ant.apache.org/manual/index.html

```
<project>

    <target name="clean">
        <delete dir="build"/>
    </target>

    <target name="compile">
        <mkdir dir="build/classes"/>
        <javac srcdir="src" destdir="build/classes"/>
    </target>

    <target name="jar">
        <mkdir dir="build/jar"/>
        <jar destfile="build/jar/HelloWorld.jar" basedir="build/classes">
            <manifest>
                <attribute name="Main-Class" value="oata.HelloWorld"/>
            </manifest>
        </jar>
    </target>

    <target name="run">
        <java jar="build/jar/HelloWorld.jar" fork="true"/>
    </target>

</project>
```

```
ant compile
ant jar
ant run
```

```
Buildfile: build.xml

clean:

compile:
    [mkdir] Created dir: C:\...\build\classes
    [javac] Compiling 1 source file to C:\...\build\classes

jar:
    [mkdir] Created dir: C:\...\build\jar
      [jar] Building jar: C:\...\build\jar\HelloWorld.jar

run:
     [java] Hello World

main:

BUILD SUCCESSFUL
```

Apache Ant

- **Read Chapter 7 Ant of the textbook "Software Build Systems: Principles and Experience".**

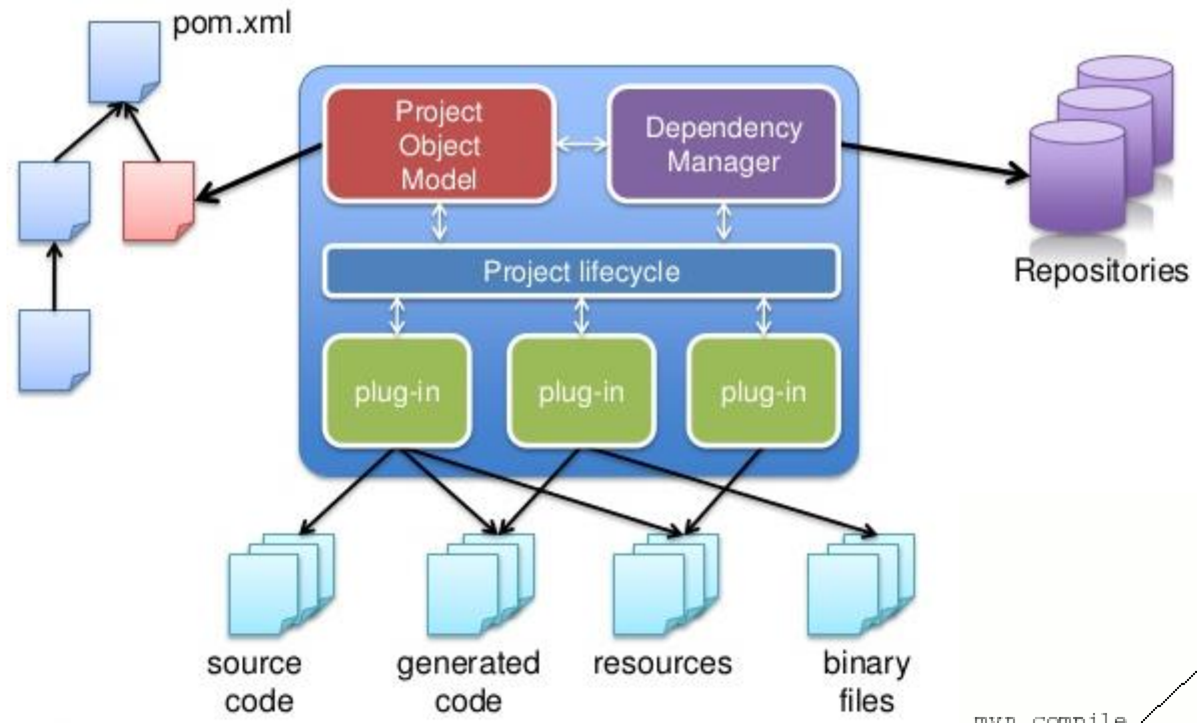
哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

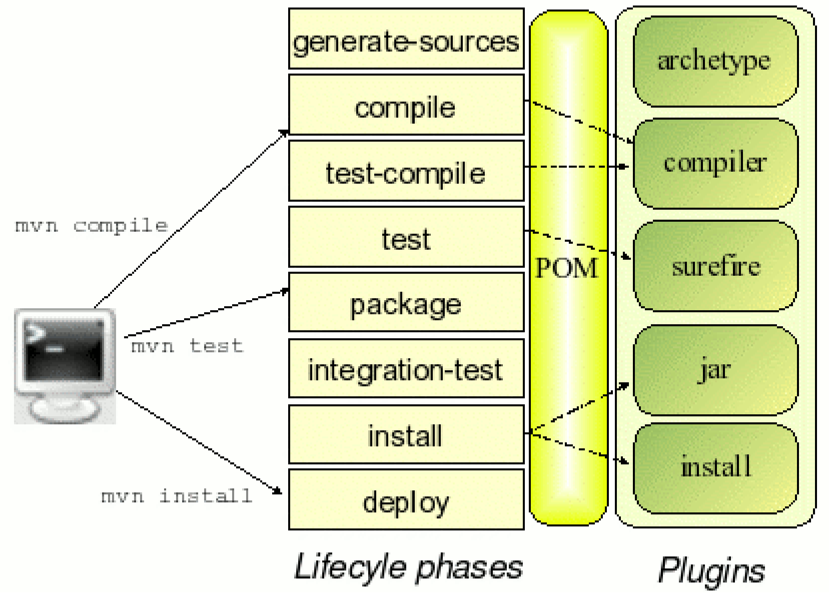
Apache Maven

Apache Maven

- **Apache Maven is a software project management and comprehension tool.**
  - Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.
- **IDE Integration: Eclipse IDE - M2Eclipse**
- **Maven’s primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time.**
  - Making the build process easy
  - Providing a uniform build system
  - Providing quality project information
  - Providing guidelines for best practices development
  - Allowing transparent migration to new features

Apache Maven





Apache Maven

https://maven.apache.org/index.html

- `validate` - **validate the project is correct and all necessary information is available**
- `compile` - **compile the source code of the project**
- `test` - **test the compiled source code using a unit testing framework. These tests should not require the code be packaged or deployed**
- `package` - **take the compiled code and package it in its distributable format, such as a JAR.**
- `verify` - **run any checks on results of integration tests to ensure quality criteria are met**
- `install` - **install the package into the local repository, for use as a dependency in other projects locally**
- `deploy` - **done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.**

Gradle

Gradle

gradle

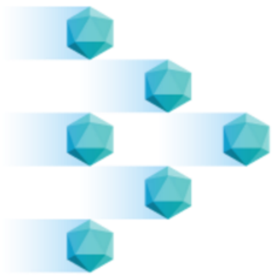
Build Anything

Write in Java, C++, Python or your language of choice. Package for deployment on any platform. Go monorepo or multi-repo. And rely on Gradle's unparalleled versatility to build it all.

Automate Everything

Use Gradle's rich API and mature ecosystem of plugins and integrations to get ambitious about automation. Model, integrate and systematize the delivery of your software from end to end.

Deliver Faster



Scale out development with elegant, blazing-fast builds. From compile avoidance to advanced caching and beyond, we pursue performance relentlessly so your team can deliver continuously.

- Using the Gradle build system in the Eclipse IDE (Tutorial)

http://www.vogella.com/tutorials/EclipseGradle/article.html

Build Java project in Eclipse

Build Java project in Eclipse

- **Eclipse IDE** provides a complete set of development tools for code editing, compilation, version control, testing, and tracking of tasks.
- **The build functionality in Eclipse** is just one part of the wider toolset, the compilation **happens behind the scenes**, and you don't even know it's taking place, Eclipse GUI makes the build work together seamlessly.
- You **don't write a build description file** (such as a makefile): Eclipse already knows enough about the structure of the software.
- Relying on the GUI to provide the build functionality **makes constructing a build system easy** but also **limits the set of available features.**

- **Read Chapter 10 Eclipse of the textbook "Software Build Systems: Principles and Experience".**

Summary

Summary of this lecture

- **General process of software construction:** Design ⇒ Programming /refactoring ⇒ Debugging ⇒ Testing ⇒ Build ⇒ Release
  - Programming / refactoring
  - Review and static code analysis
  - Debugging (dumping and logging) and Testing
  - Dynamic code analysis /profiling
- **Narrow-sense process of software construction (Build):** Validate ⇒ Compile ⇒ Link ⇒ Test ⇒ Package ⇒ Install ⇒ Deploy
  - Build system: components and process
  - Build variants and build language
  - Build tools: Make, Ant, Maven, Gradle, Eclipse

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

The end

February 24, 2019