哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Chapter 1: Views and Quality Objectives of Software Construction

1.1 Multi-Dimensional Views of Software Construction

软件构造过程中的多维度视图

Wang Zhongjie
rainy@hit.edu.cn

February 24, 2019

Objective of this lecture

- **To understand the constituents of a software system in three orthogonal dimensions**
  从三个维度看软件系统的构成
- **To know what models are used to describe the morphology and states of a software system**
  用什么样的模型/视图描述软件系统
- **To treat software construction as the transformations between different views**
  将“软件构造”看作“不同视图之间的转换”

先要搞清楚：软件构造的对象是什么、如何刻画
然后再关注：如何构造

Outline

- **Multi-dimensional software views**
  - By phases: build- and run-time views 按阶段划分：构造时/运行时视图
  - By dynamics: moment and period views 按动态性划分：时刻/阶段视图
  - By levels: code and component views
    按构造对象的层次划分：代码/构件视图
  - Elements, relations, and models of each view
- **Software construction: transformation between views**
  - $\varnothing \Rightarrow$ Code
  - Code $\Rightarrow$ Component
  - Build-time $\Rightarrow$ Run-time
  - Moment $\Rightarrow$ Period
- **Summary**

Reading

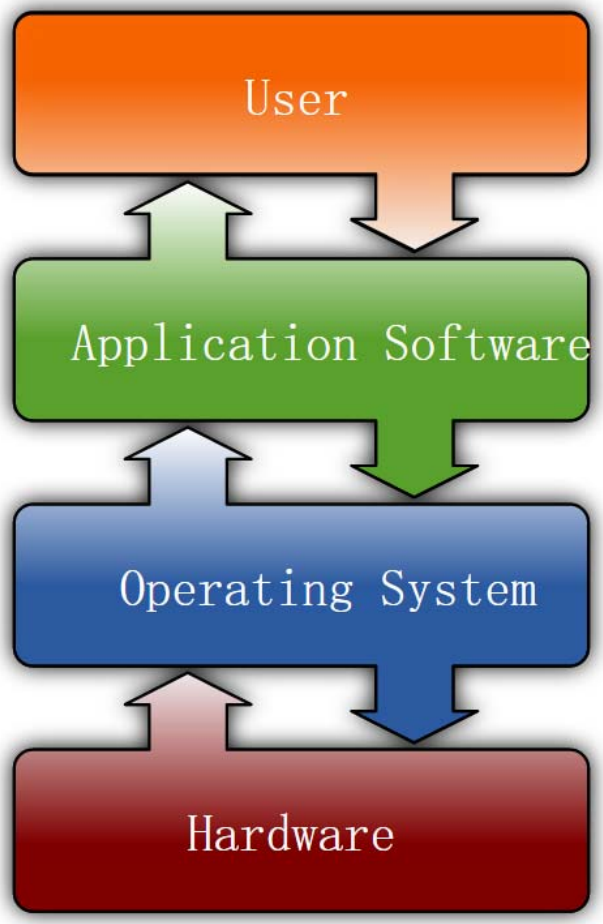
- **MIT 6.031：Getting started, readings 02**
- **CMU 17-214：Aug 30**
- **代码大全：第1-4章**

1 Multi-dimensional software views

What is a Software?

- **The term "software" was firstly proposed by Alan Turing.**
  - System software vs. Application software
  - Desktop/web/mobile/embedded software
  - Business/personal-oriented software
  - Open source vs. proprietary software

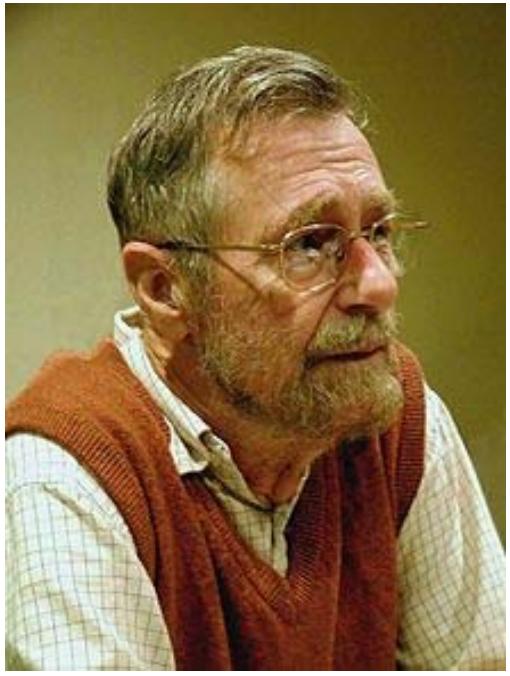
Alan Turing (1912-1954)



Constituents of a software system

- **Software = Program (codes)?**
- **Software = Algorithms + Data Structure?**
- **Software = Program + Data + Documents**
- **Software = Modules (Components) + Data/Control Flows**

Donald E. Knuth (1938-)
Turing Award 1974



Edager Dijkstra (1930-2002)
Turing Award 1972

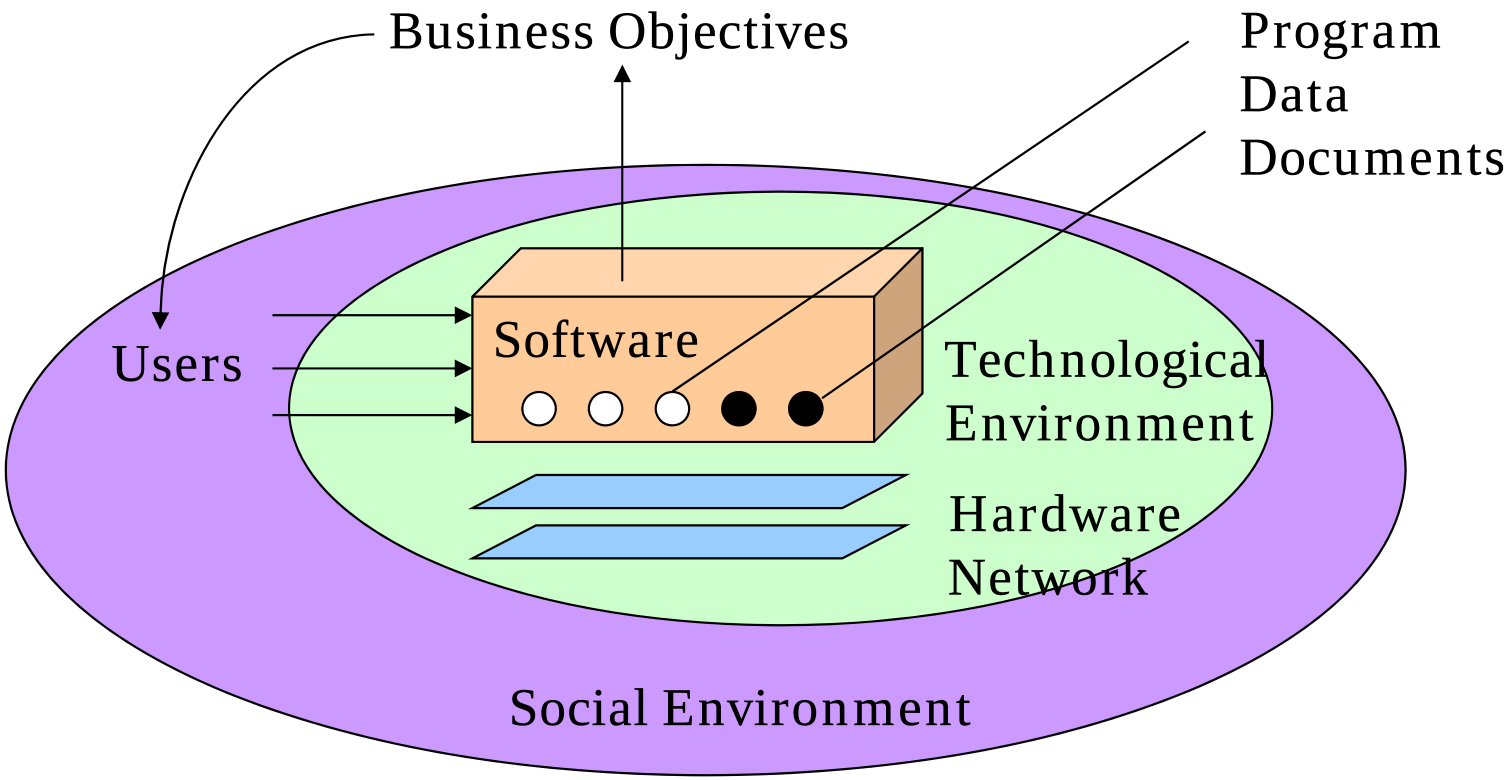
Niklaus Wirth (1934-)
Turing Award 1984

Constituents of a software system: one more step

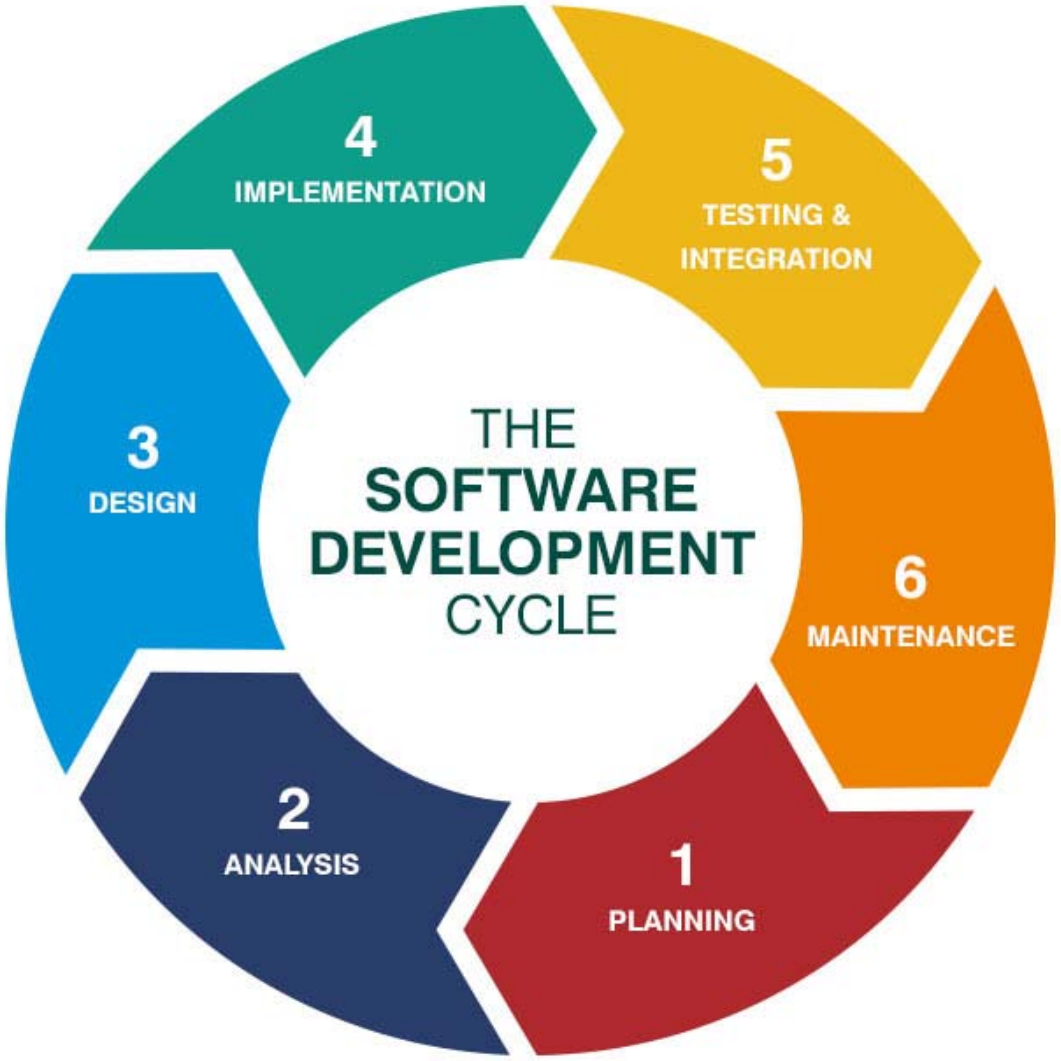
- **Software system =**
  - `Programs` (UI, Algorithms, Utilities, APIs, test cases, etc)
  - `Data`(files, databases, etc)
  - `Documents` (SRS, SDD, user manuals, etc)

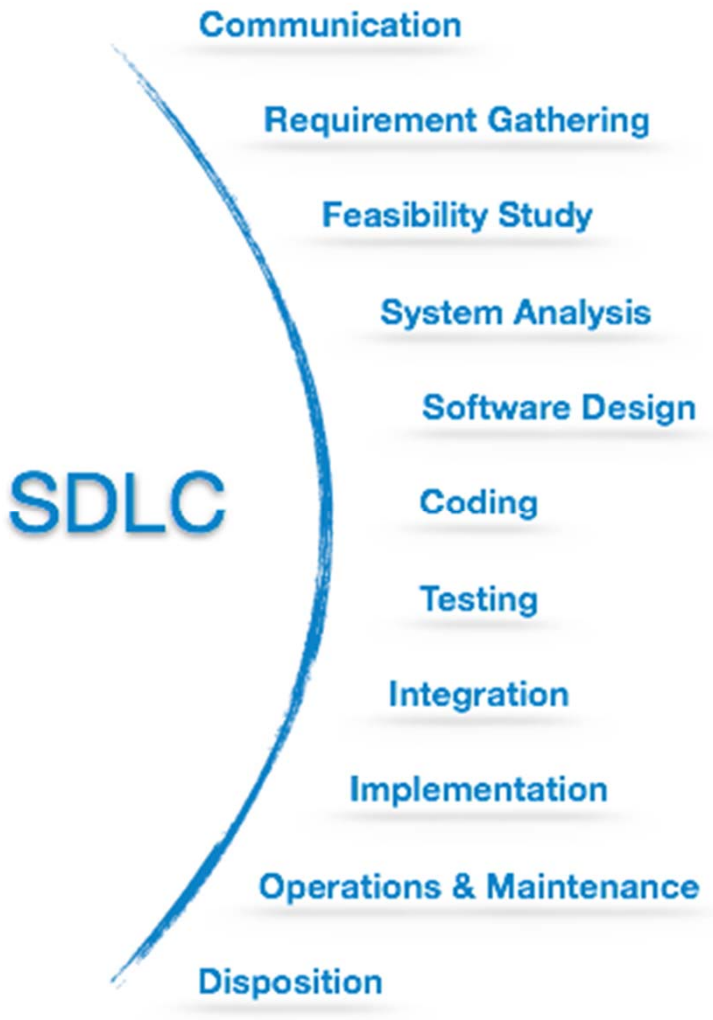
  ++++++++++++++++
  - Users — `WHO` will use it?
  - Business Objectives — `WHY` is it useful?
  - Social Environment — `WHAT` laws/rules should it follow?
  - Technological Environment — `HOW` is it implemented?
  - Hardware / Network — `WHERE` does it run?

Constituents of a software system: one more step



Constituents of a software system: two more steps





Multi-dimensional software views

<table>
<tr><th></th><th colspan="2">Moment</th><th colspan="2">Period</th></tr>
<tr><th></th><th>Code-level</th><th>Component-level</th><th>Code-level</th><th>Component-level</th></tr>
<tr><td>Build-time</td><td>Source code, AST, Interface-Class-Attribute-Method (Class Diagram)</td><td>Package, File, Static Linking, Library, Test Case, Build Script (Component Diagram)</td><td>Code Churn</td><td>Configuration Item, Version</td></tr>
<tr><td rowspan="2">Run-time</td><td rowspan="2">Code Snapshot, Memory dump</td><td rowspan="2">Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)</td><td>Execution stack trace, Concurrent multi-threads</td><td>Event log, Multi-processes, Distributed processes</td></tr>
<tr><td colspan="2">Procedure Call Graph, Message Graph (Sequence Diagram)</td></tr>
</table>

(1) Build-time Views

Build-time views of a software system

- **Build-time (构造阶段): idea ⇒ requirement ⇒ design ⇒ code ⇒ installable / executable package**
  - **Code-level view**: source code ---- how source code are logically organized by basic program blocks such as functions, classes, methods, interfaces, etc, and the dependencies among them 代码的逻辑组织
  - **Component-level view**: architecture ---- how source code are physically organized by files, directories, packages, libraries, and the dependencies among them 代码的物理组织

  - **Moment view**: what do source code and component look like in a specific time 特定时刻的软件形态
  - **Period view**: how do they evolve/change along with time 软件形态随时间的变化

(1) Build-time, moment, and code-level view

- **How source code are logically organized by basic program blocks such as functions, classes, methods, interfaces, etc, and the dependencies among them.**
- **Three inter-related forms:**
  - 词汇层面：Lexical-oriented source code
  - 语法层面：Syntax-oriented program structure: e.g., Abstract Syntax Tree (AST)
  - 语义层面：Semantics-oriented program structure: e.g., Class Diagram

Lexical-based semi-structured source code

- **Source code: the most important assets in software development**

```
01 // First, log in
02 LoginResult loginResult=null;
03 SoapBindingStub sfdc=null;
04 sfdc = (SoapBindingStub) new SforceServiceLocator().getSoap();
05 // login
06 loginResult = sfdc.login("username","password");
07
08 // The set up some security related items
09 // Reset the SOAP endpoint to the returned server URL
10 sfdc._setProperty(SoapBindingStub.ENDPOINT_ADDRESS_PROPERTY,loginResult.getServerUrl());
11 // Create a new session header object
12 // add the session ID returned from the login
13 SessionHeader sh=new SessionHeader();
14 sh.setSessionId(loginResult.getSessionId());
15 sfdc.setHeader(new SforceServiceLocator().getServiceName().getNamespaceURI(),
16   "SessionHeader",sh);
17
18 // now that we're logged in, make some calls
19 GetUserInfoResult userInfo = sfdc.getUs
20
21 // create a new account object lo
22 Account account = new Account()
23 account.setAccountNumber("002D
24 account.setName("My New Accou
25 account.setBillingCity("Glasg
26
27
28 SObject[] sObjects = new SObjec
29 sObjects[0] = account;
30
31 // persist the object
32 SaveResult[] saveResults = sfdc.create(sObjec
```

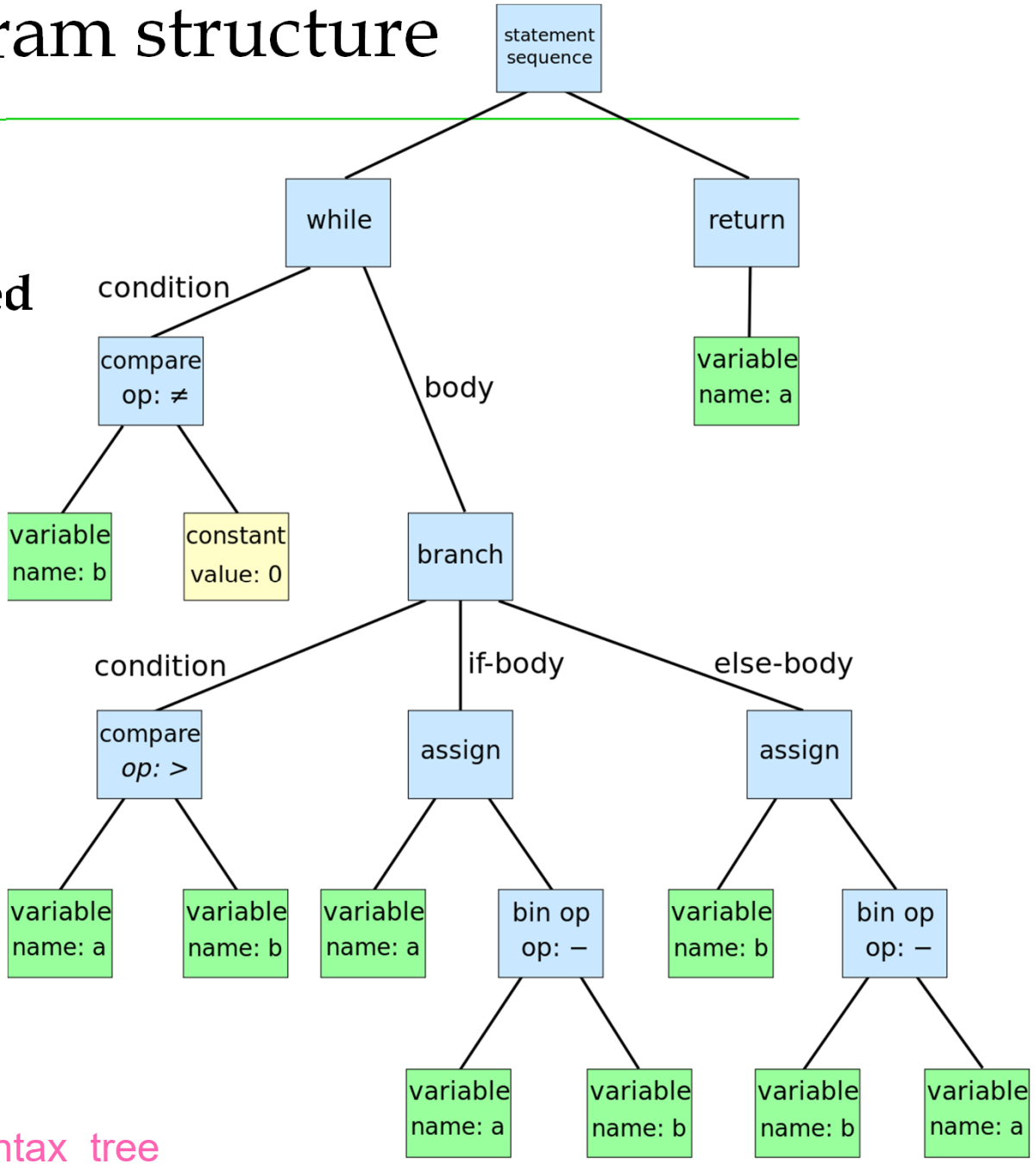
半结构化：近乎自然语言的风格+遵循特定的编程语法

前者：方便程序员
后者：方便编译器

Syntax-oriented program structure

- **Abstract Syntax Tree (AST)**
- **To represent semi-structured source code as a structured tree.**

```
while (a ≠ b) {
    if (a > b)
        a = a – b;
    else
        b = b – a;
}
return a;
```



https://en.wikipedia.org/wiki/Abstract_syntax_tree
http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/index.html

Processing Java Source Files by AST

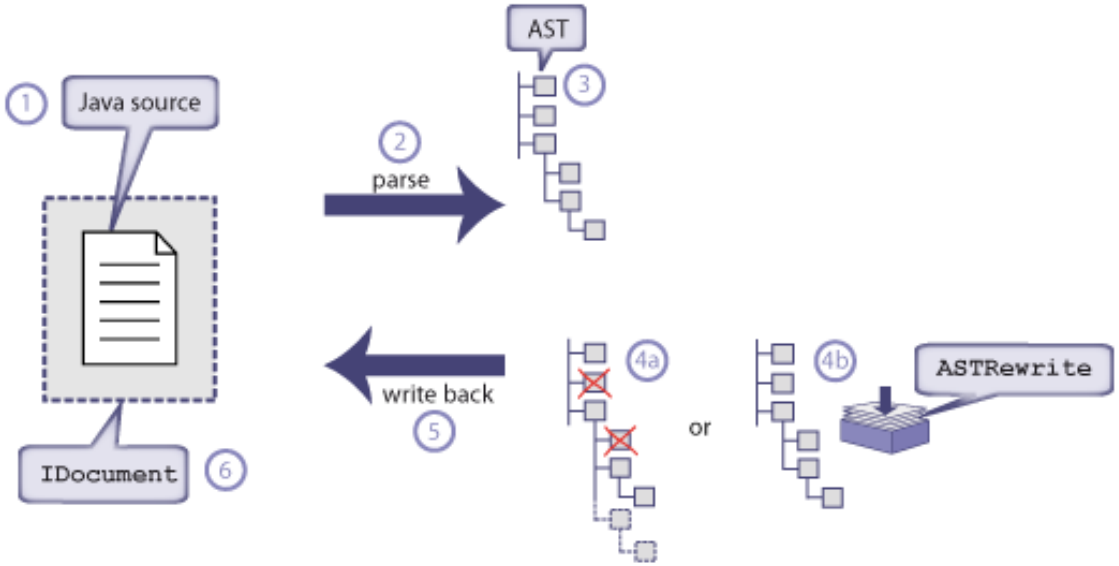
```
import org.eclipse.jdt.core.dom.*;
import org.eclipse.jface.text.Document
import org.eclipse.text.edits.TextEdit

public class AST {

 void processJavaFileByAST(){

    Document doc = new Document(javaFi
    ASTParser parser = ASTParser.newPa
    parser.setResolveBindings(true);
    parser.setSource(doc.get().toCharArray());
    CompilationUnit cu = (CompilationUnit) parser.createAST(null);
    cu.recordModifications();

    AST ast = cu.getAST();
    ImportDeclaration id = ast.newImportDec
    id.setName(ast.newName(new String[] {"
    cu.imports().add(id); // add import dec
    TextEdit edits = cu.rewrite(doc, null);
 }
}
```



AST：彻底结构化，将源代码变为一棵树，对树做各种操作==对源代码的修改

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

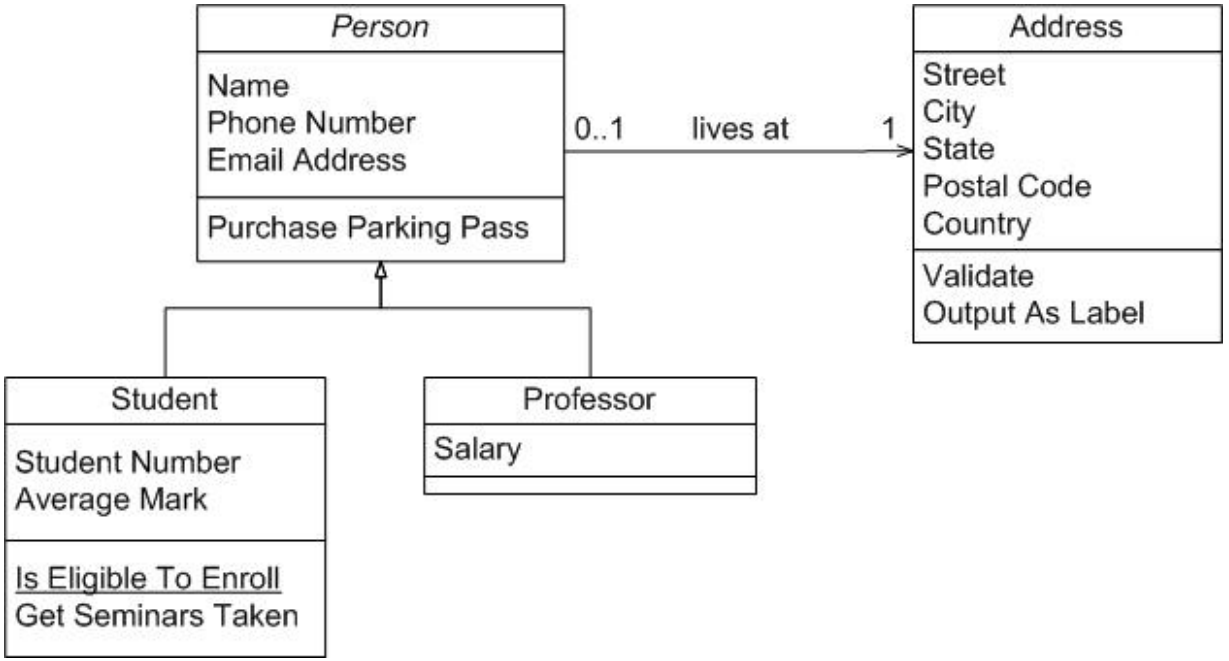
Semantics-oriented program structure

语义：源代码具体想实现什么目标？

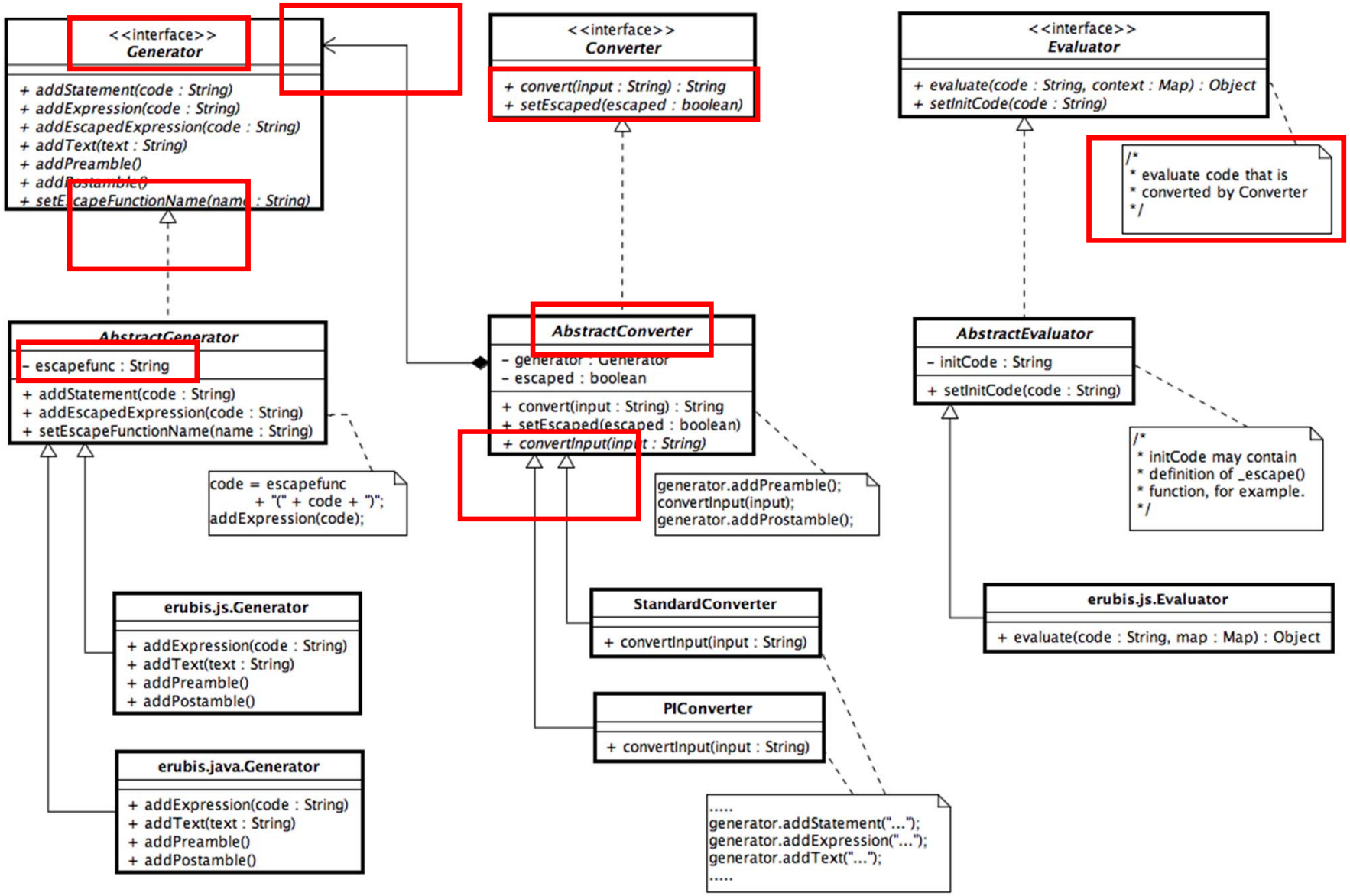
源代码---现实世界

- **E.g., using Class Diagram (UML) to describe interfaces, classes, attributes, methods, and relationships among them.**
- **Graphics-based or formally defined.** 通常是图形化或形式化的
- **Modeled in design phase, and transformed into source code.**
- **It is the result of Object-Oriented Analysis and Design in terms of user requirements.**

用于表达“需求”和“设计”思想，再转化成code



Semantics-oriented program structure



(2) Build-time, period, and code-level view

- **Views describing "changes" along with time.**
- **Code churn 代码变化: Lines added, modified or deleted to a file from one version to another.**

Code Before

```
int i = n;
while (i--)
   printf (" %d", i);
```

Code After

```
//print n integers iff n≥ 0
int i = n;
while (--i > 0)
   printf (" %d", i);
```

two lines added

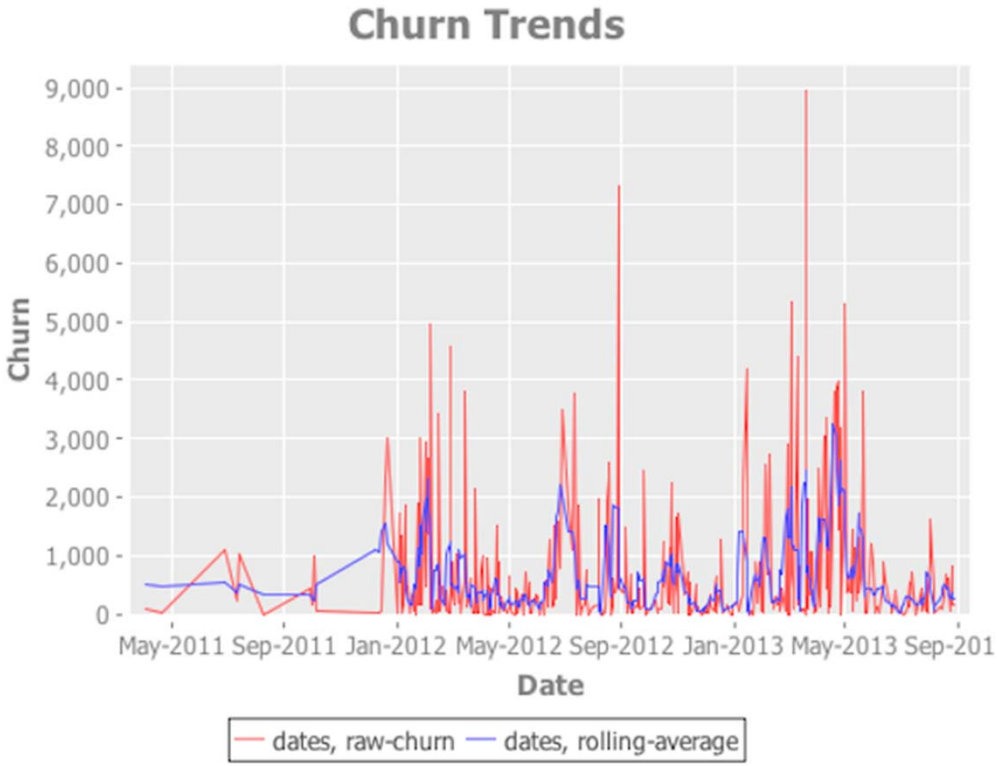
Code Before

```
int i = n;
while (i--)
   printf (" %d", i);
```

Code After

```
//print n integers iff n≥ 0
int i = n;
while (--i > 0)
   printf (" %d", i);
```

one line deleted



Code churn

Code churn is defined as lines added, modified or deleted to a file from one version to another.

Event: Add radio click triggering tests

Browse files

Ref b442aba
Ref gh-3423

master

alexr101 committed with gibson042 13 days ago — 1 parent b442aba — commit 5f35b5b406ae7d504de86a3f0a5647b2fdf4f2af

Showing 1 changed file with 26 additions and 11 deletions.

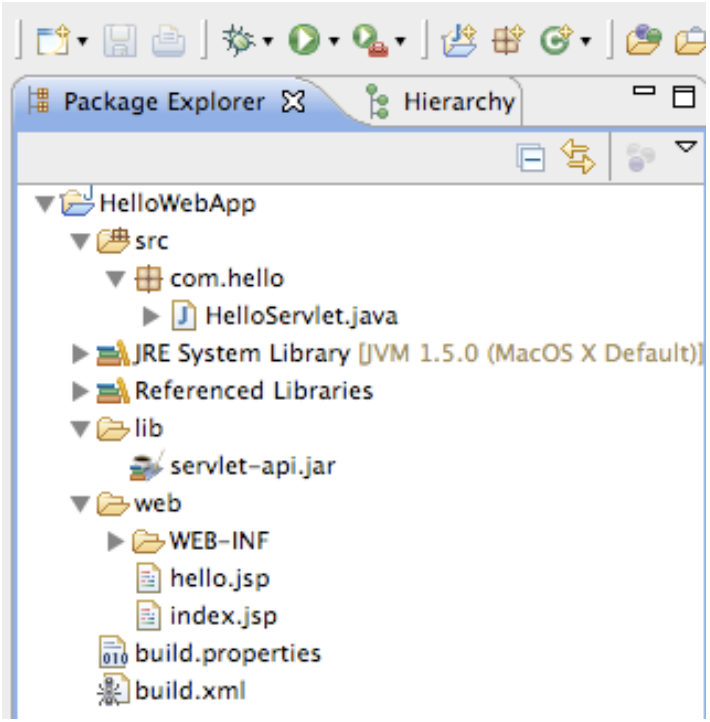
Unified | Split

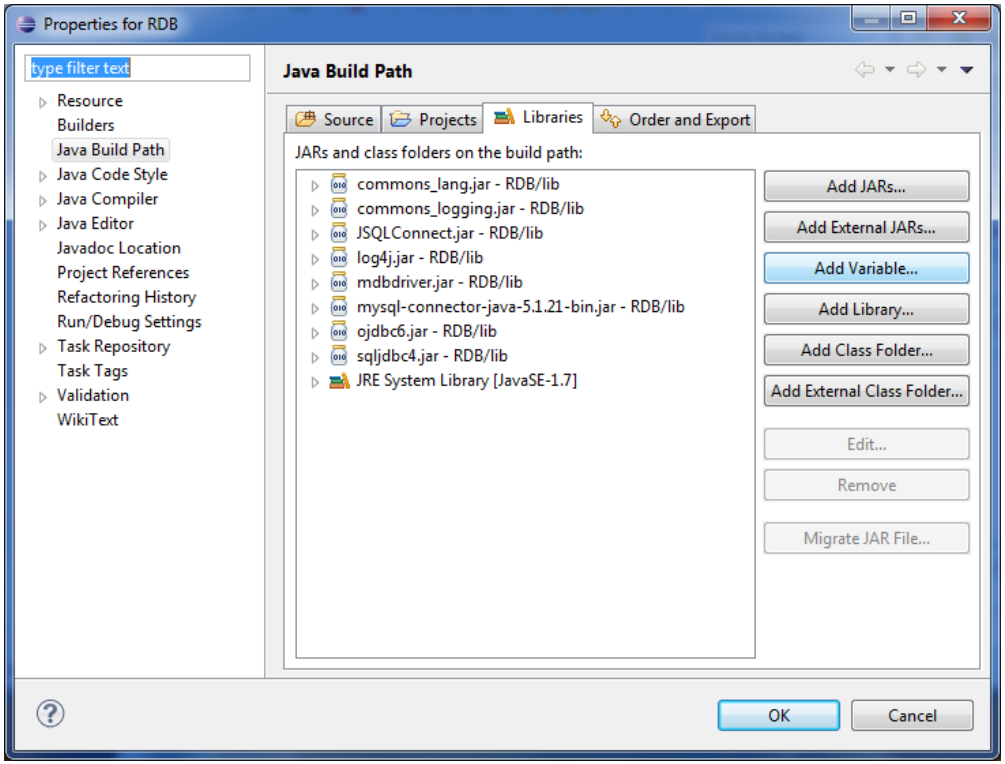
37 test/unit/event.js — View

```
@@ -2364,33 +2364,48 @@ QUnit.test( "clone() delegated events (#11076)", function( assert ) {
2364 2364         clone.remove();
2365 2365  } );
2366 2366
2367      -QUnit.test( "checkbox state (#3827)", function( assert ) {
2368      -       assert.expect( 9 );
     2367 +QUnit.test( "checkbox state (#3827, gh-3423)", function( assert ) {
     2368 +       assert.expect( 17 );
2369 2369
2370      -       var markup = jQuery( "<div><input type=checkbox><div>" ).appendTo( "#qunit-fixture" ),
2371      -               cb = markup.find( "input" )[ 0 ];
     2370 +       var cbParent = jQuery( "<div><input type=checkbox><div>" ).appendTo( "#qunit-fixture" ),
     2371 +               cb = cbParent.find( "input" )[ 0 ],
     2372 +               radioParent = jQuery(
     2373 +                       "<div><input type=radio name=gh3423><input type=radio name=gh3423><div>"
     2374 +               ).appendTo( "#qunit-fixture" ),
     2375 +               radio = radioParent.find( "input" )[ 0 ],
     2376 +               radio2 = radioParent.find( "input" )[ 1 ];
2372 2377
```

(3) Build-time, moment, and component-level view

- **Source code are physically organized into files which further are organized by directories;**
- **Files are encapsulated into packages and, logically, components and sub-systems.**
- **Reusable modules are in the form of libraries.**





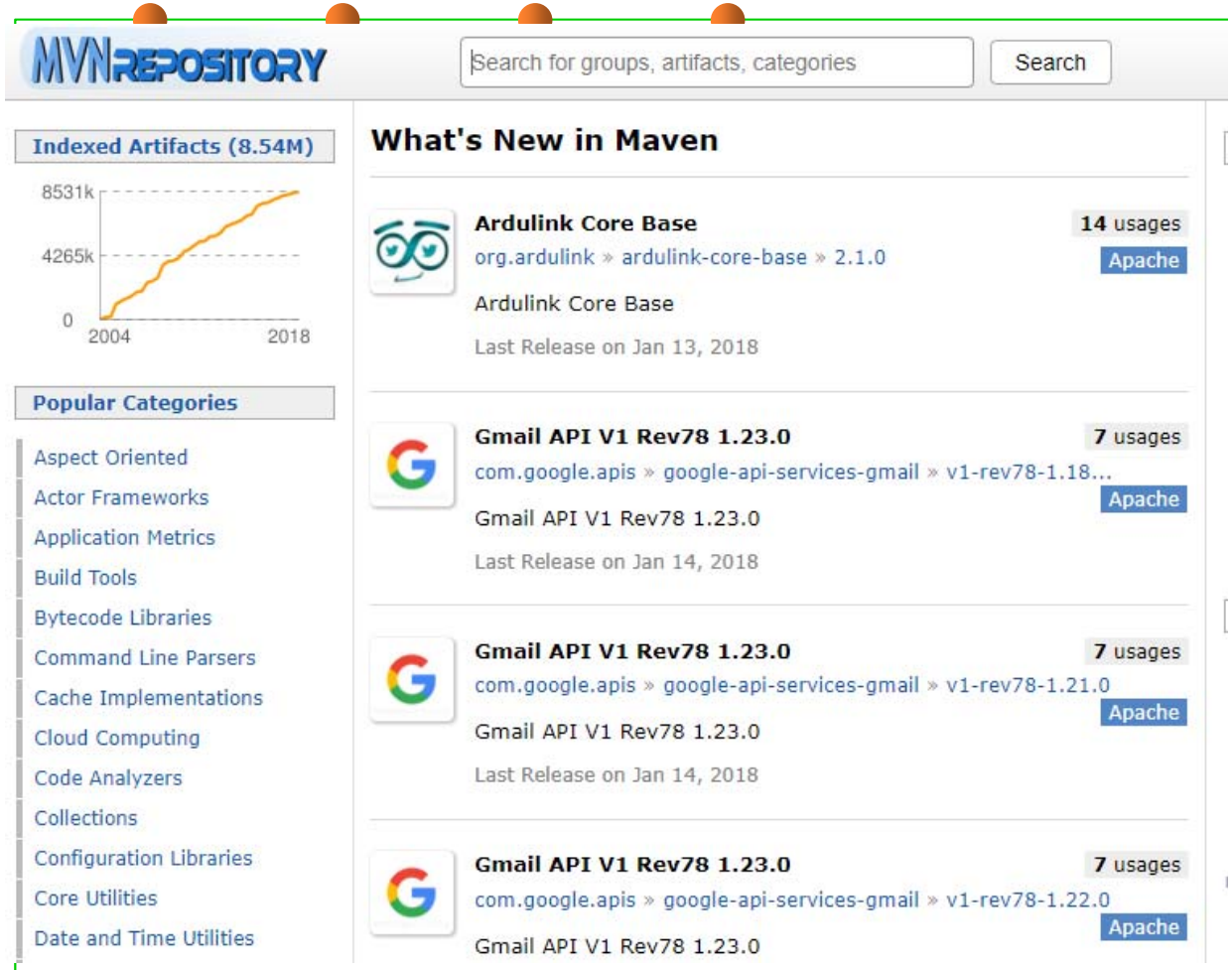
Library

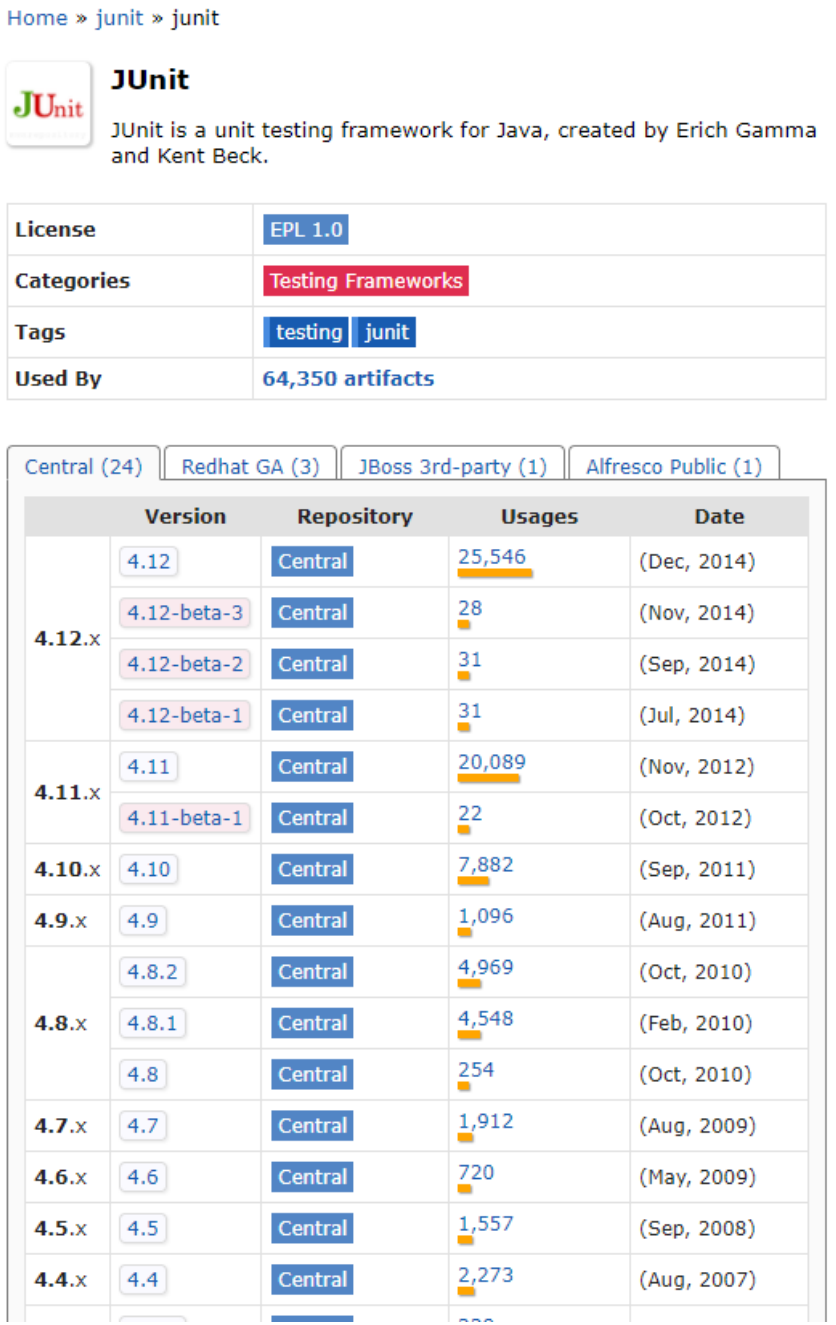
- **Libraries** are stored in disk files of their own, collect a set of code functions that can be reused across a variety of programs.
  - Developers aren't always building a single executable program file, but join custom-developed software and prebuilt libraries into a single program.
- **In build-time**, a library function can be viewed as an extension to the standard language and is used in the same way as functions written by the developers. 开发者像使用编程语言指令一样使用库中的功能

```
System.out.println("Hello World");
```

- **Sources of libraries:**
  - From OS pre-installed set of libraries for operations such as file and network I/O, GUI, mathematics, database assess; 操作系统提供的库
  - From language SDK; 编程语言提供的库
  - From third-party sources; 第三方公司提供的库
  - Developers can also publish their own libraries. 你自己积累的库

Maven Repository






```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

Linking with a library

- **When a program is edited, built and installed, a list of libraries to search must be provided.** 编程时和build时，需告诉IDE和JVM在哪里寻找某些库
- **If a function is referenced in the source code but the developer didn't explicitly write it, the list of libraries is searched to locate the required function.**

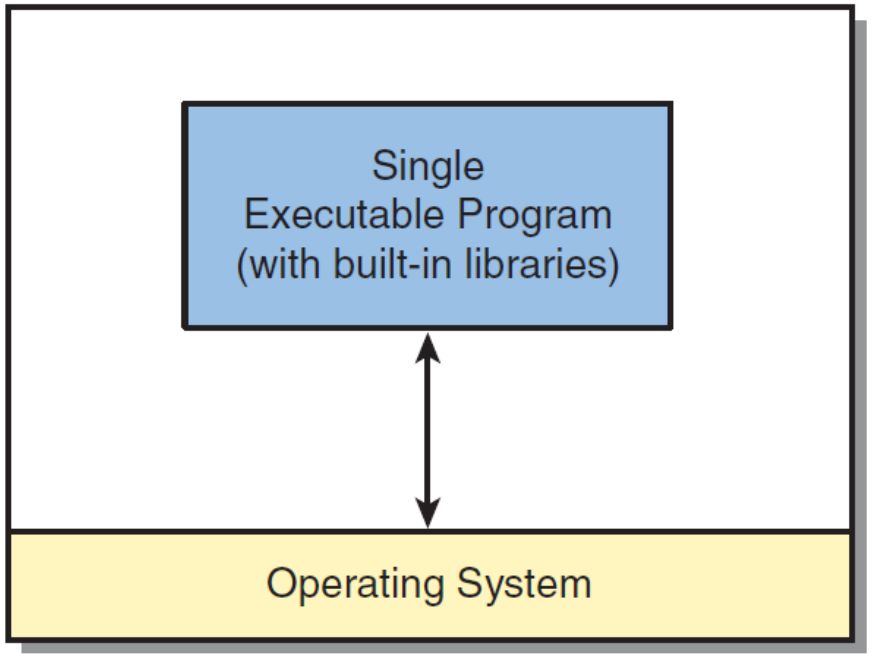
```
javac -classpath  ./lib/*.jar
```

- **When the function is found, the appropriate object file is copied into the executable program.**
- **Two different approaches of integrating a library into an executable program:**
  - Static linking (静态链接)
  - Dynamic linking (动态链接)

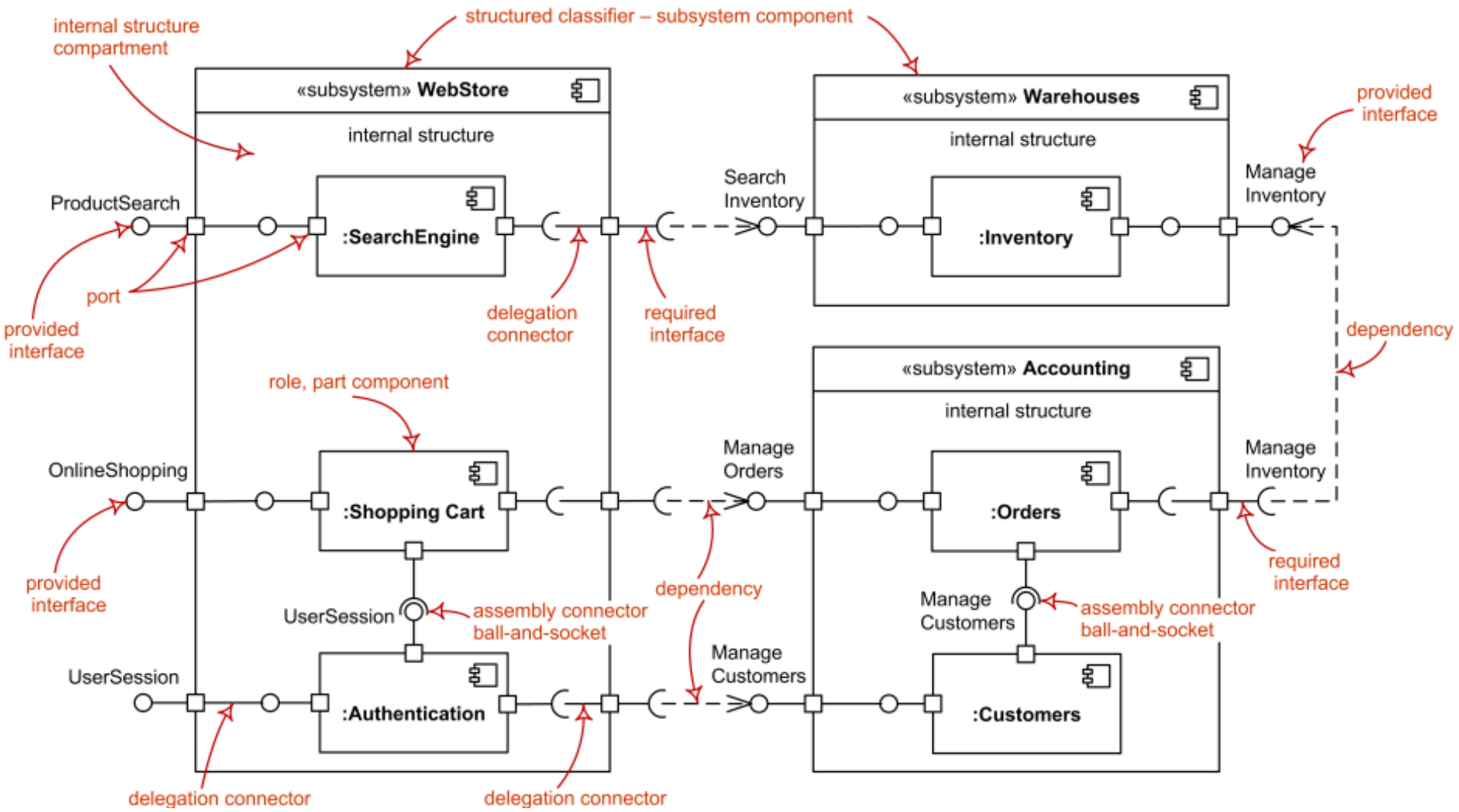
Static linking

- **In static linking, a library is a collection of individual object files.**
- **During the build process**, when the linker tool determines that a function is required, **it extracts the appropriate object file from the library and copies it into the executable program**
  - 库被拷贝进入代码形成整体，执行的时候无需提供库文件
  - The library's object file appears identical to any of the object files the developer created on his or her own.
- **Static linking happens in build time**
  - End up with a single executable program to be loaded onto the target machine.
  - After the final executable program has been created, it's impossible to separate the program from its libraries.

  静态链接发生在构造阶段

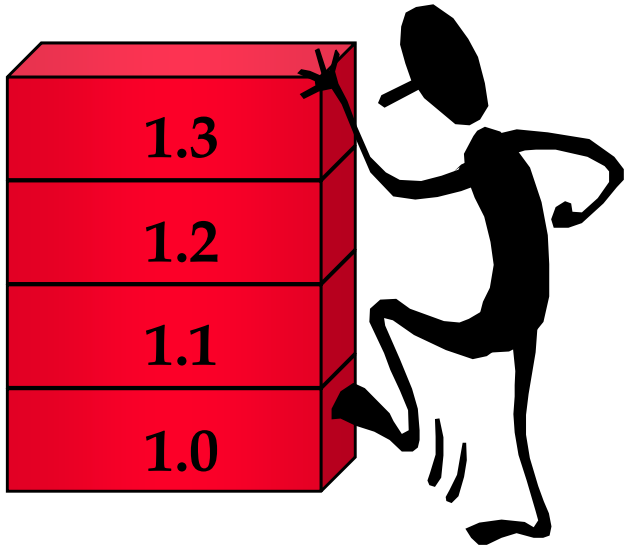


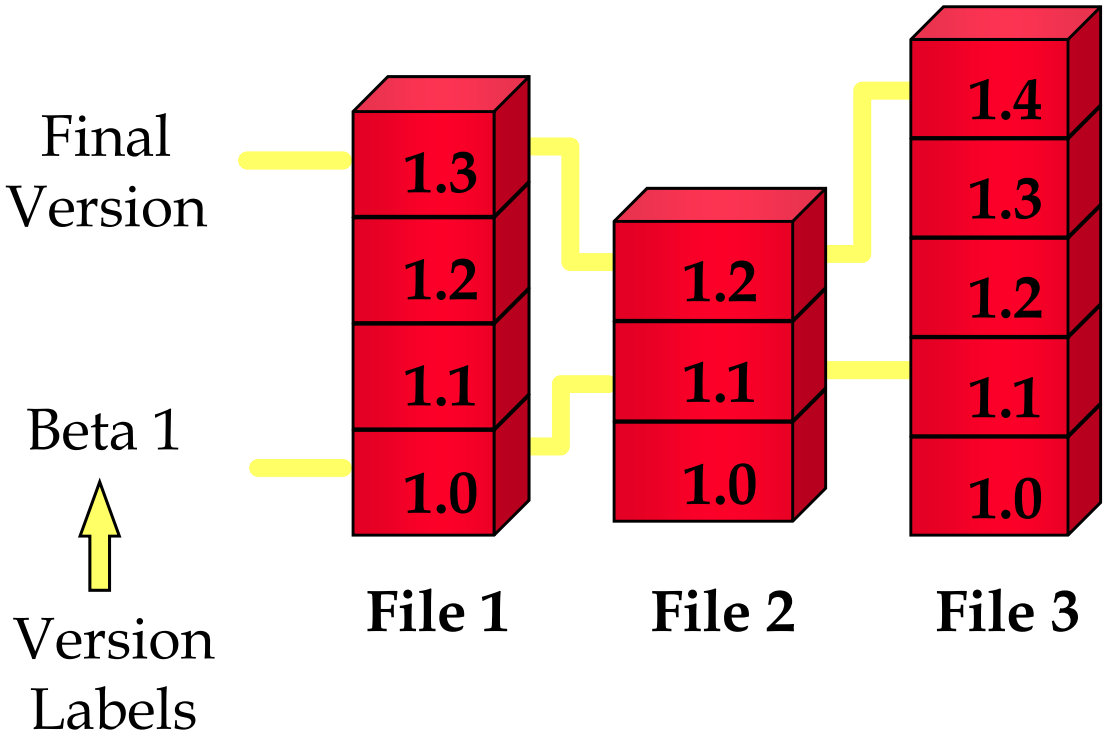
Component diagram in UML



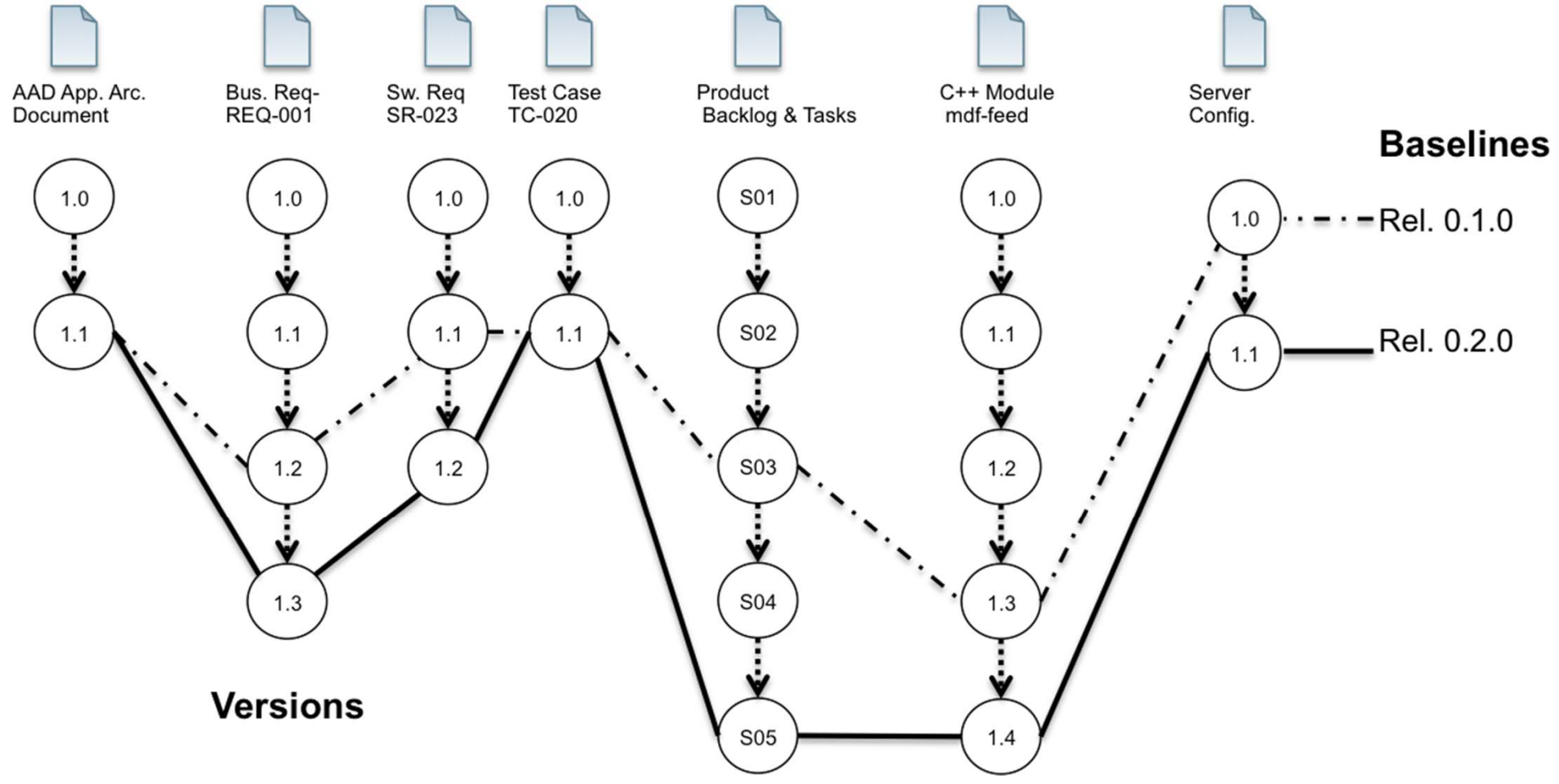
(4) Build-time, period, and component-level view

- **How do all files/packages/components/libraries change in a software system along with time?** 各项软件实体随时间如何变化？
- **Software Configuration Item** (SCI，配置项)
- **Version** (版本)

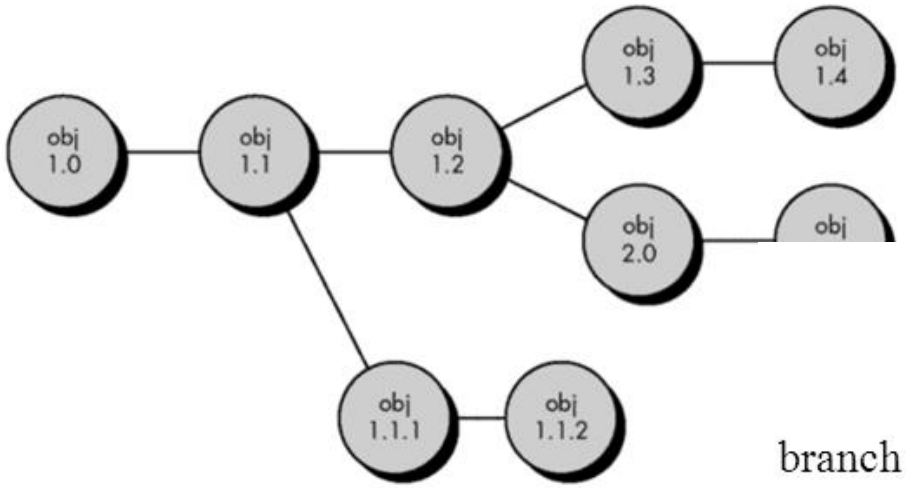




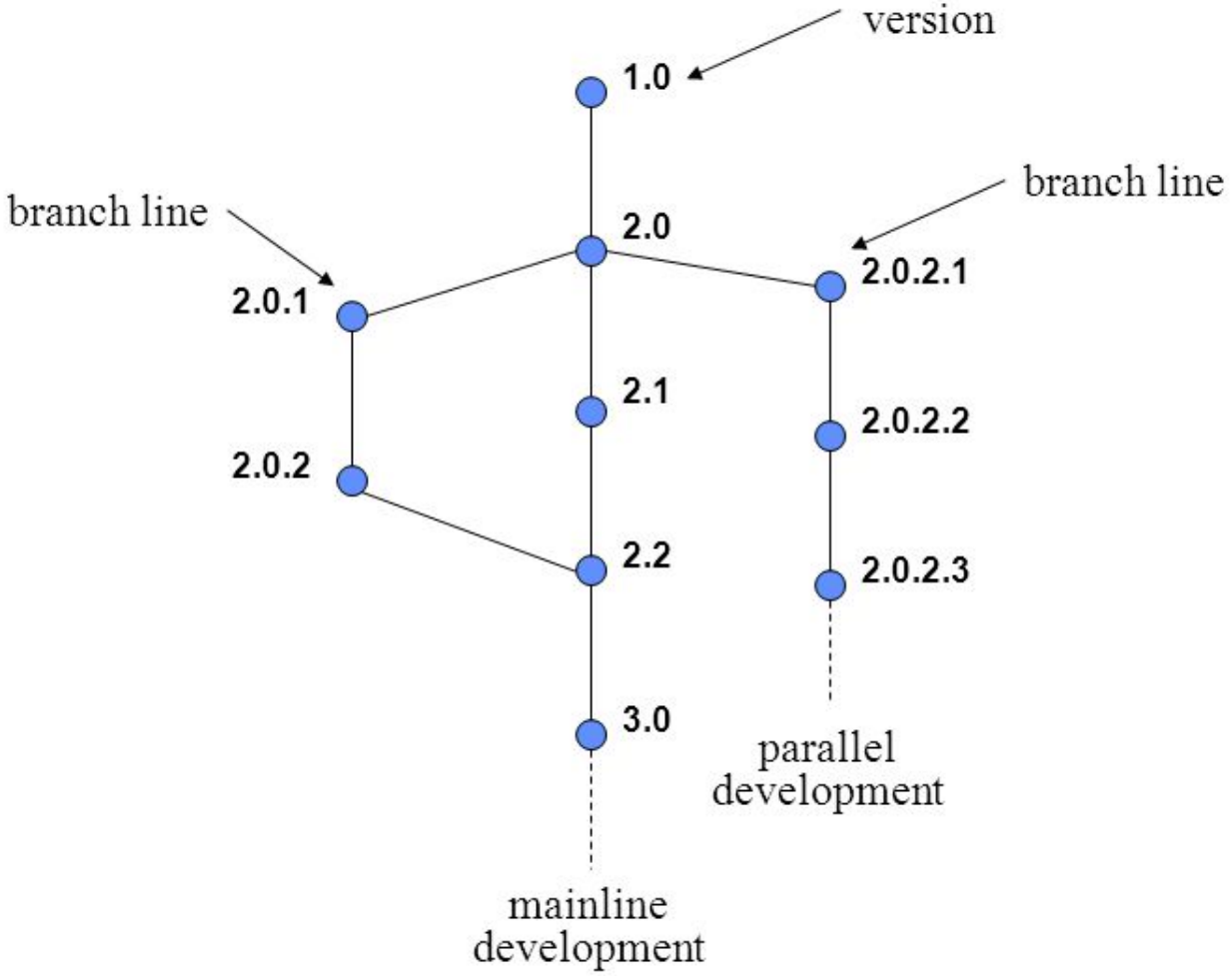
Version Control System (VCS)



Evolution Graph (of a SCI or a Software)

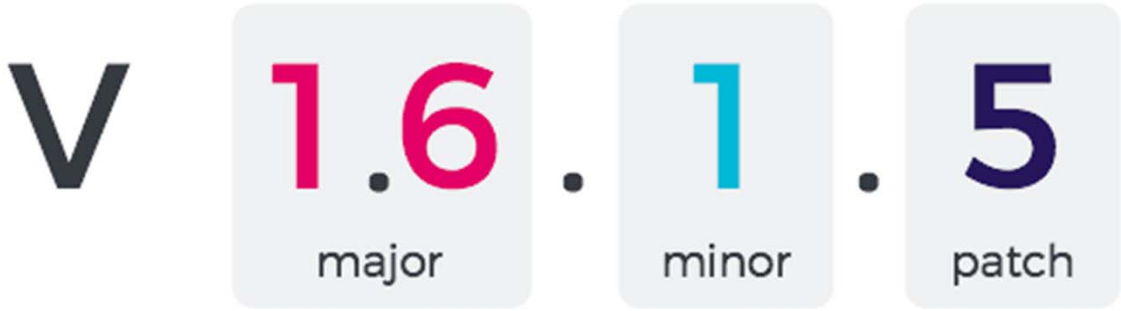


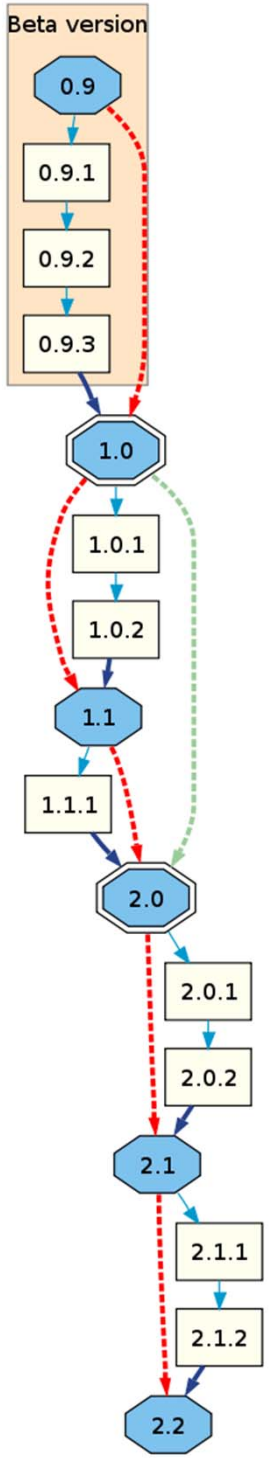
Evaluation graph describes the development process and phases of a software.



Versioning

- **Software versioning** is the process of assigning either unique version names or unique version numbers to unique states of computer software.
  - Within a given version number category (major, minor), these numbers are generally assigned in increasing order and correspond to new developments in the software.
  - At a fine-grained level, revision control is often used for keeping track of incrementally different versions of electronic information, whether or not this information is computer software.





Software Evolution

- **Software evolution is a term used in software maintenance, referring to the process of developing software initially, then repeatedly updating it for various reasons.**
  - Over 90% of the costs of a typical system arise in the maintenance phase, and that any successful piece of software will inevitably be maintained.

Frederick Brooks (1931-)
Turing Award 1999

(2) Runtime Views

Runtime views of a software system

- **Runtime:** what does a program look like when it runs inside the target machine, and what are all the disk files that the target machine needs to load into memory? 运行时：程序被载入目标机器，开始执行
  - **Code-level view**: source code ---- what do the in-memory states of an executable program look like and how do program units (objects, functions, etc) interact with each other? 代码层面：逻辑实体在内存中如何呈现？
  - **Component-level view**: architecture ---- how are software packages deployed into physical environment (OS, network, hardware, etc) and how do they interact? 构件层面：物理实体在物理硬件环境中如何呈现？

  - **Moment view**: how do programs behave in a specific time 逻辑/物理实体在内存/硬件环境中特定时刻的形态如何？
  - **Period view**: how do they behave along with time 逻辑/物理实体在内存/硬件环境中的形态随时间如何变化？

High-level concepts of run-time software

- **Executable programs:** The sequence of machine-readable instructions that the CPU executes, along with associated data values.
  - This is the fully compiled program that's ready to be loaded into the computer's memory and executed.

- **Libraries:** Collections of commonly used object code that can be reused by different programs.
  - Most operating systems include a standard set of libraries that developers can reuse, instead of requiring each program to provide their own.
  - A library can't be directly loaded and executed on the target machine; it must first be linked with an executable program.

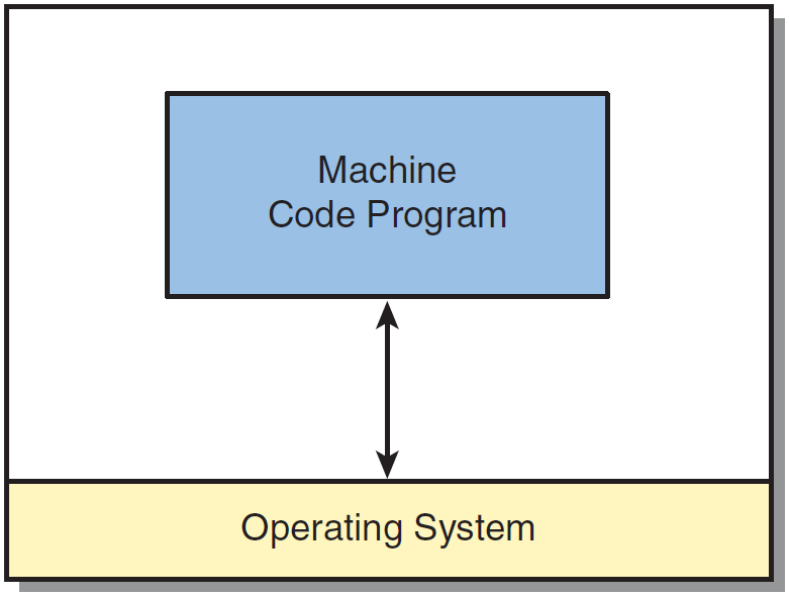
High-level concepts of run-time software

- **Configuration and data files:** These are not executable files; they provide useful data and configuration information that the program can load from disk.

- **Distributed programs:** This type of software consists of multiple executable programs that communicate with each other across a network or simply as multiple processes running on the same machine.
  - This contrasts with more traditional software that has a single monolithic program image.

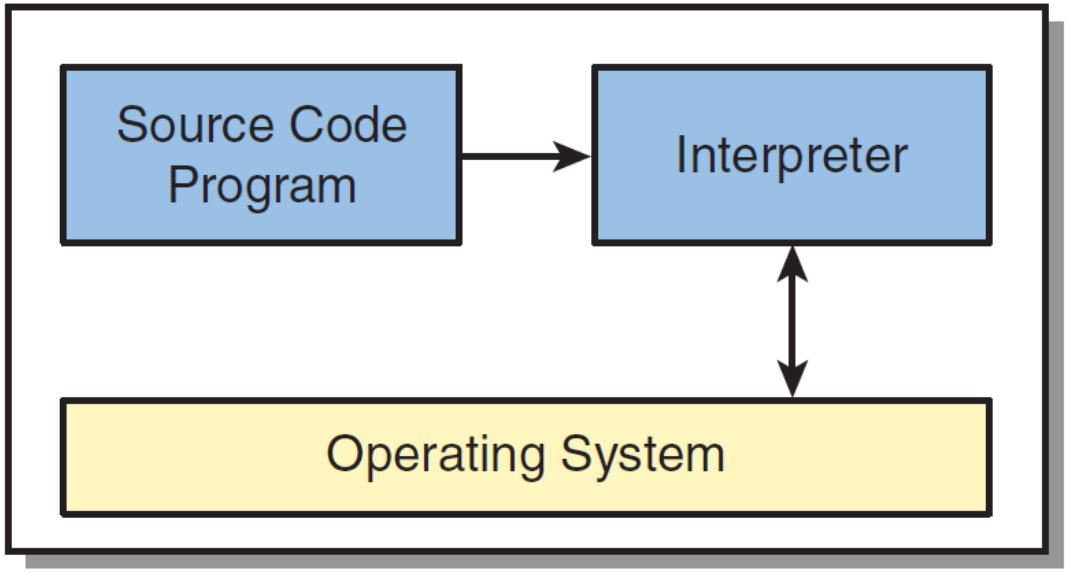
Executable Programs: Native Machine Code

- A program is loaded into memory first, and several mechanisms exist for executing the software, depending on how much compilation took place before the program was loaded and how much OS supports the program requires.
- **Native Machine Code (原生机器码):**
  - Fully converted executable program into the CPU's native machine code.
  - The CPU simply "jumps" to the program's starting location, and all the execution is performed purely using the CPU's hardware.
  - While it's executing, the program optionally makes calls into the operating system to access files and other system resources.
  - This is the fastest way to execute code, because the program full accesses to the CPU's features.



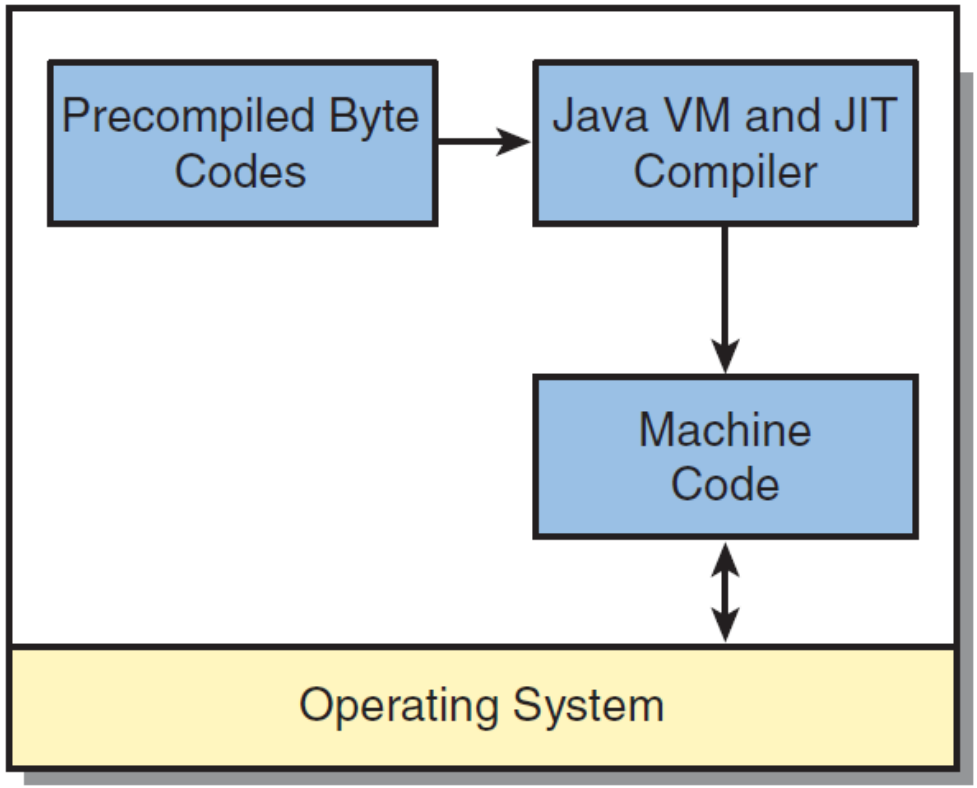
Executable Programs: Full Interpretation

- **Full Program Interpretation (程序完全解释执行):**
  - The runtime system loads the entire source code into memory and interprets it (such as BASIC, UNIX shell, etc)



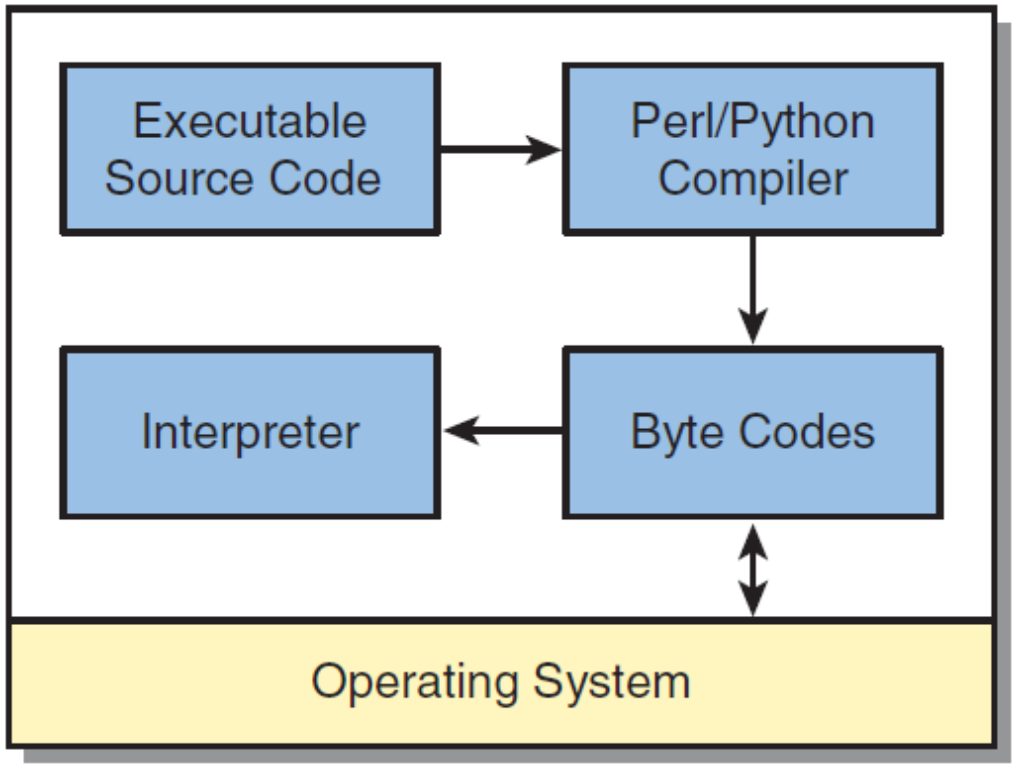
Executable Programs: Interpreted Byte Codes

- **Interpreted Byte Codes (解释型字节码):**
  - Byte codes are similar to native machine code, except that the CPU doesn't directly understand them.
  - It first translates them into native machine code or interprets them as the program executes.
  - A byte code environment therefore requires that an additional interpreter or compiler be loaded alongside the program.
- **Java Virtual Machine (JVM)**



Executable Programs: Interpreted Byte Codes

- **Perl or Python:** they are interpreted rather than compiled, but use byte codes at runtime.
- **The simple act of executing the Perl or Python script automatically triggers the generation of byte codes.**



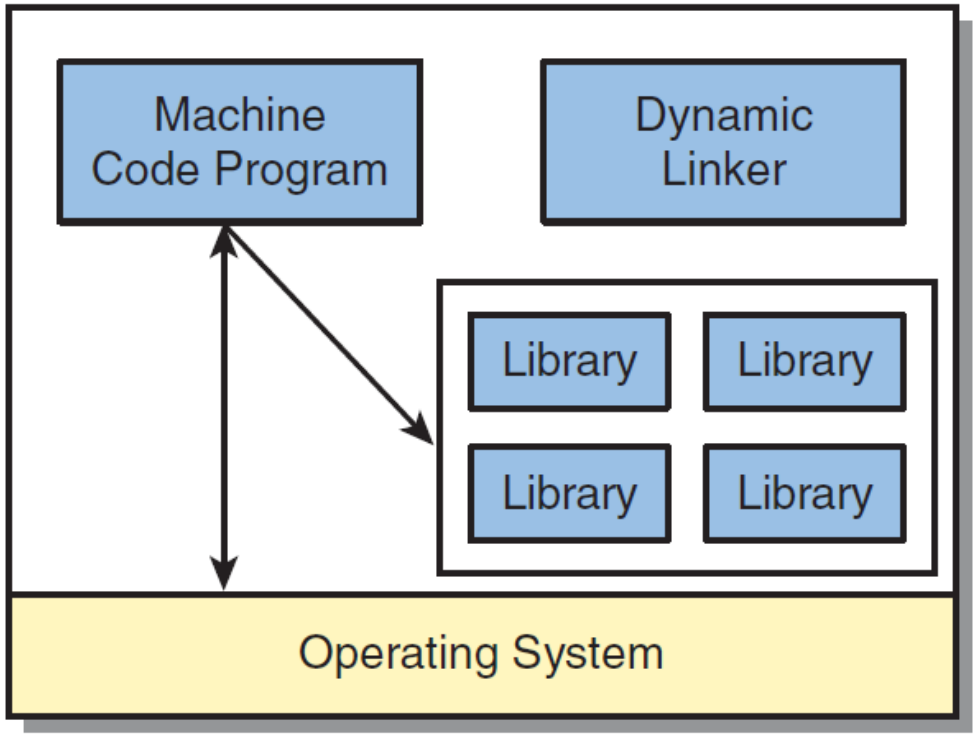
Dynamic linking

- **Dynamic linking** method doesn't copy the object file into the executable image; instead, it notes which libraries are required to successfully execute the program. 库文件不会在**build**阶段被加入可执行软件，仅仅做出标记
- When the program starts running, the **libraries are loaded into memory as separate entities** and then are connected with the main program. 程序运行时，根据标记装载库至内存
- A dynamic library is a disk file that is constructed by joining object files. **The library is then collected into the release package and installed on the target machine**. Only then can it be loaded into the machine's memory. 发布软件时，记得将程序所依赖的所有动态库都复制给用户

你的实验，提交到**GitHub**的时候，切记把各种**lib**都一并提交上来

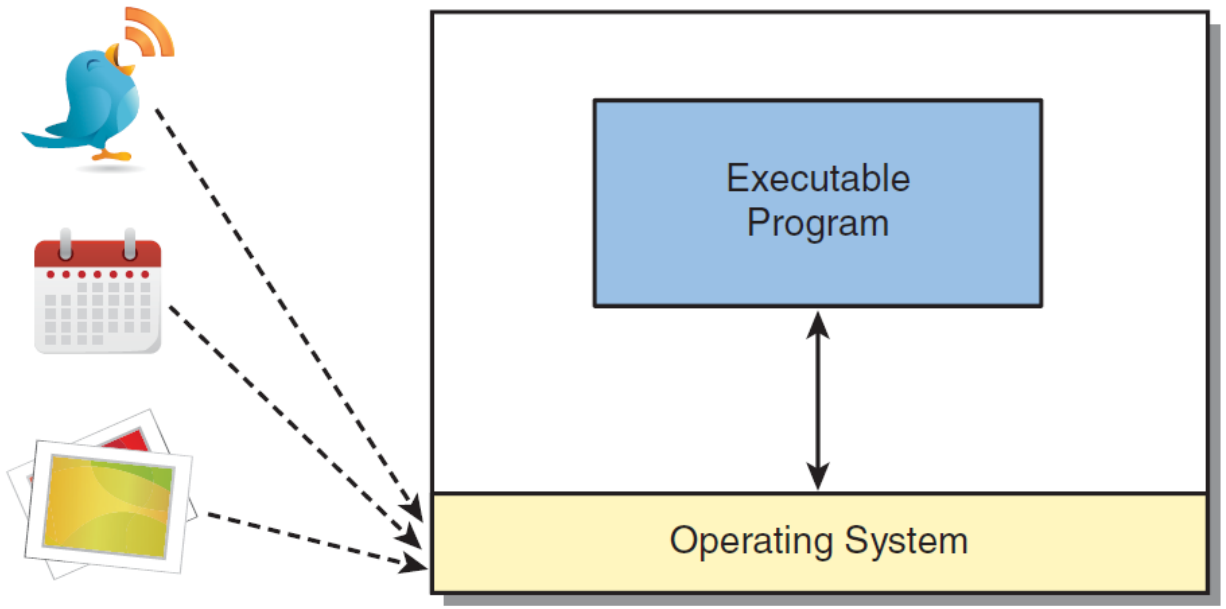
Dynamic linking

- **Advantages:**
  - It's possible to upgrade to a newer version of a library (adding features or fixing bugs), without needing to re-create the executable program.
  - Many operating systems can optimize their memory usage by loading only a single copy of the library into memory, yet sharing it with other programs that require that same library.



Configuration and Data Files

- **Any program of significant size uses external data sources, such as a file on a disk.**
- **Your program makes calls into the operating system to request that data be read into memory.**
  - A bitmap graphic image displayed onscreen
  - A sound stored as a digitized wave form
  - A configuration file that customizes the behavior of a program
  - A set of documents containing online help text
  - A database containing names and addresses



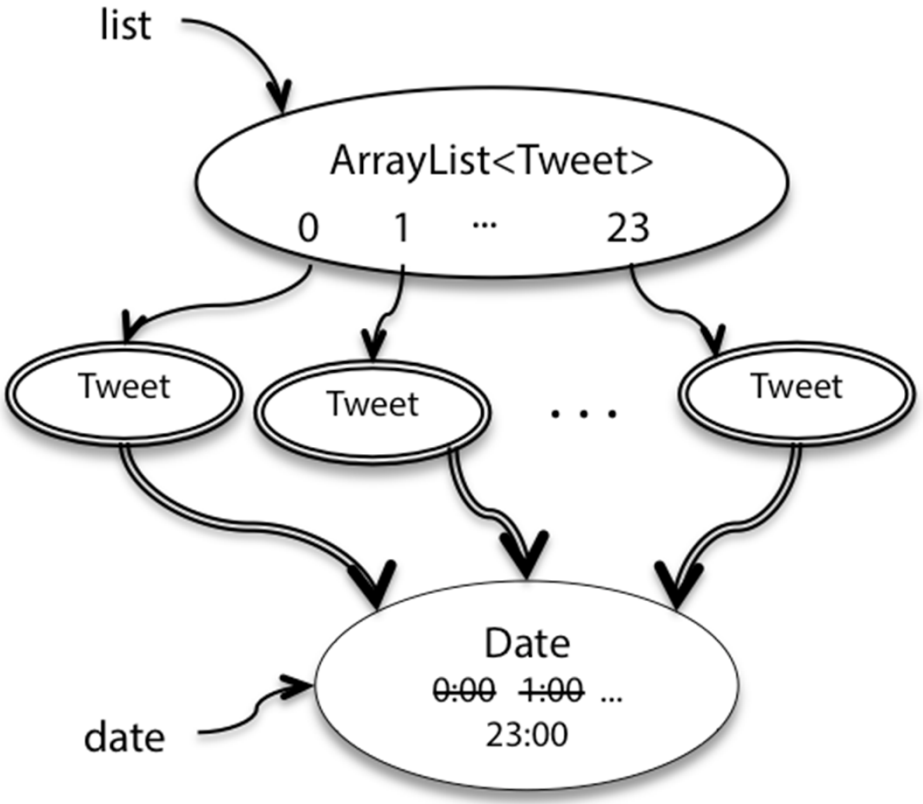
Distributed Programs

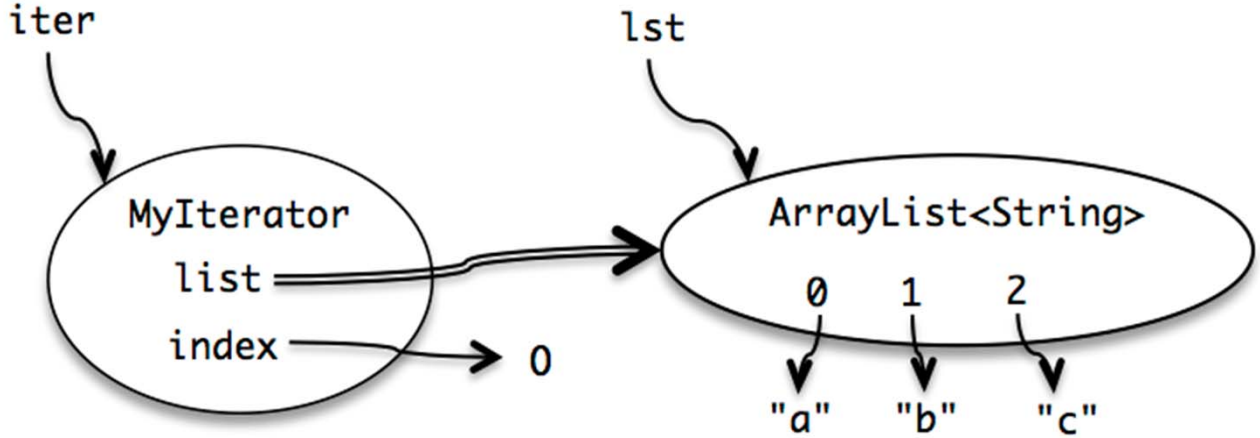
- For example, a software system might use the client/server model, with a single server program running on one computer and a large number of client programs running on many other computers.
- In this scenario, the build system could create two release packages, given that different people will be installing the server program versus the client program.
- Alternatively, the same release package could be used to install the two separate programs.

- 分布式程序的运行态：需要多个运行程序，分别部署于多个计算机物理环境。

(5) Run-time, moment, and code-level view

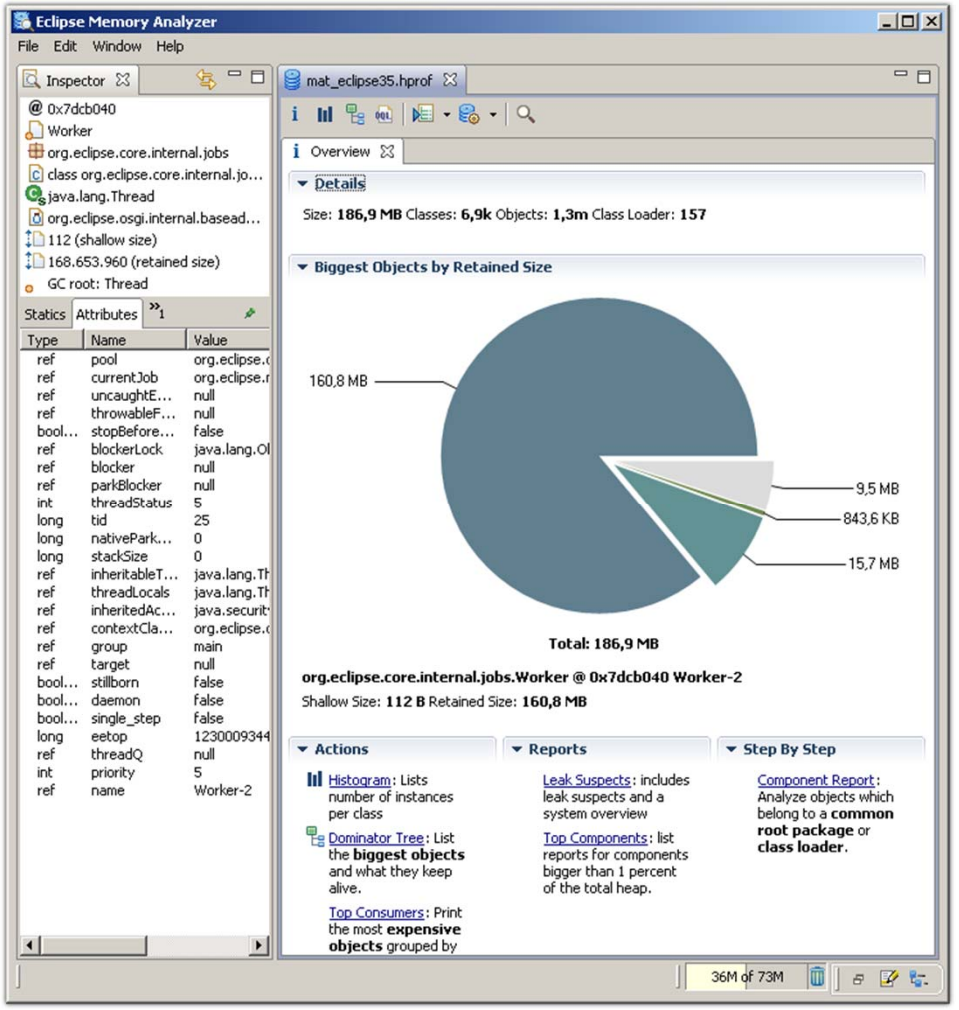
- **Snapshot diagram:** focusing on variable-level execution states in the memory of a target computer.
- **Fine-grained states of a program.**
- 代码快照图：描述程序运行时内存里变量层面的状态



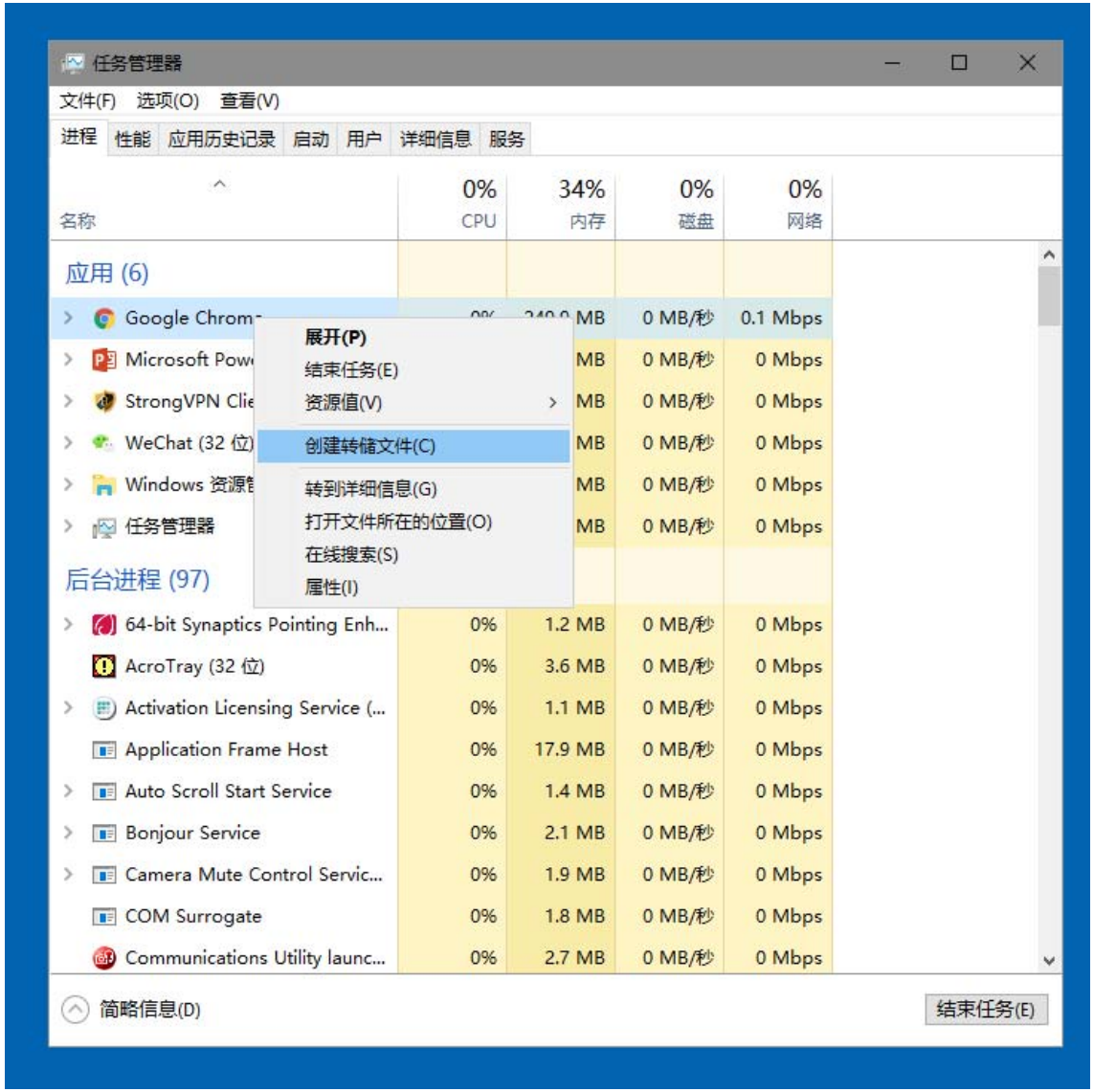


Memory dump (内存信息转储)

- **Memory dump**: a file on hard disk containing a copy of the contents of a process's memory, produced when a process is aborted by certain kinds of internal error or signal.
  - Debuggers can load the dump file and display the information it contains about the state of the running program.
  - Information includes the contents of registers, the call stack and all other program data (counters, variables, switches, flags, etc).
  - It is taken in order to analyze the status of the program, and the programmer looks into the memory buffers to see which data items were being worked on at the time of failure.

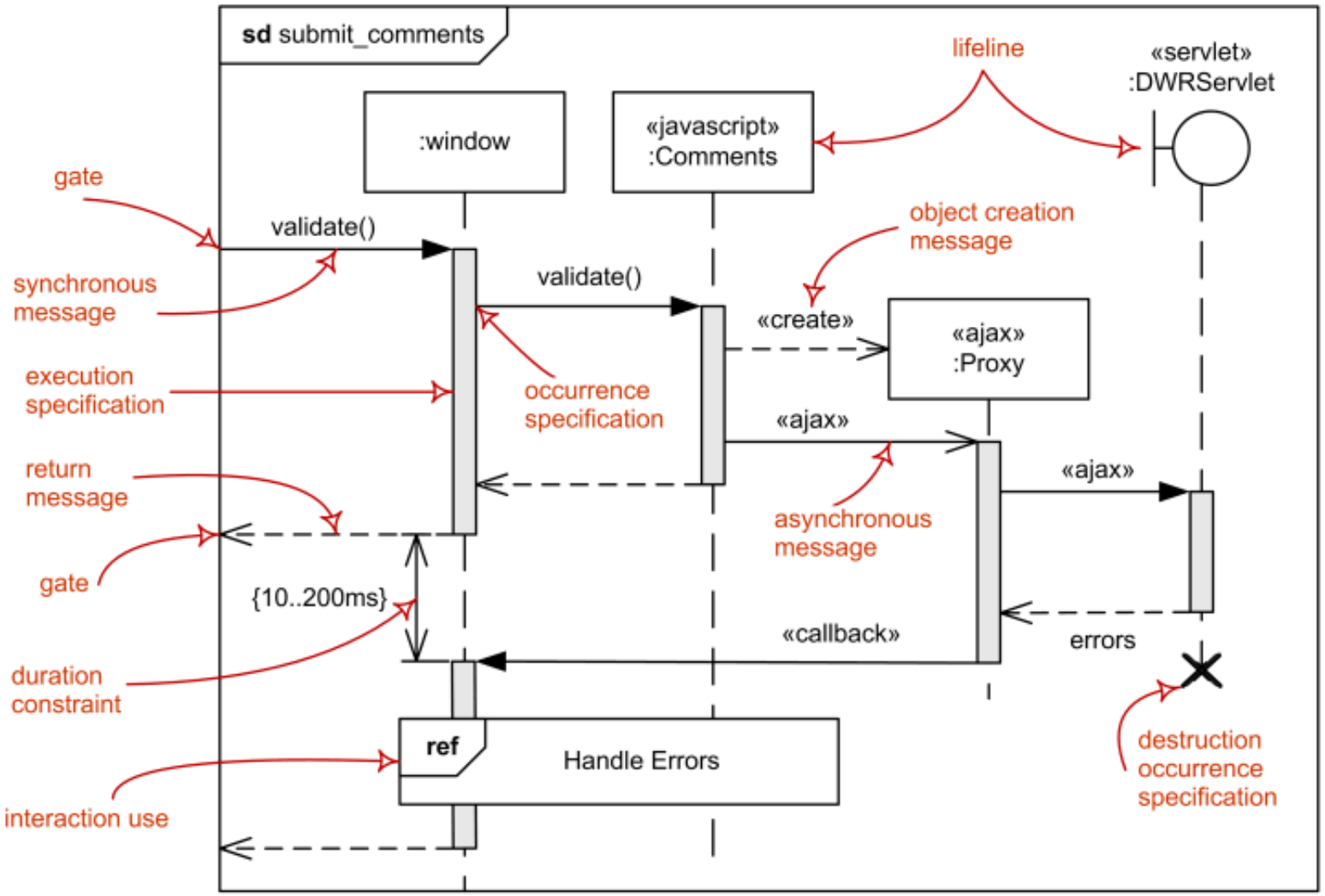


Memory dump


任务管理器
文件(F) 选项(O) 查看(V)
进程 性能 应用历史记录 启动 用户 详细信息 服务
0% CPU
34% 内存
0% 磁盘
0% 网络
名称
应用 (6)
Google Chrom
Microsoft Pow
StrongVPN Clie
WeChat (32 位)
Windows 资源管
任务管理器
展开(P)
结束任务(E)
资源值(V)
创建转储文件(C)
转到详细信息(G)
打开文件所在的位置(O)
在线搜索(S)
属性(I)
后台进程 (97)
64-bit Synaptics Pointing Enh... 0% 1.2 MB 0 MB/秒 0 Mbps
AcroTray (32 位) 0% 3.6 MB 0 MB/秒 0 Mbps
Activation Licensing Service (... 0% 1.1 MB 0 MB/秒 0 Mbps
Application Frame Host 0% 17.9 MB 0 MB/秒 0 Mbps
Auto Scroll Start Service 0% 1.4 MB 0 MB/秒 0 Mbps
Bonjour Service 0% 2.1 MB 0 MB/秒 0 Mbps
Camera Mute Control Servic... 0% 1.9 MB 0 MB/秒 0 Mbps
COM Surrogate 0% 1.8 MB 0 MB/秒 0 Mbps
Communications Utility launc... 0% 2.7 MB 0 MB/秒 0 Mbps
0.1 Mbps
简略信息(D)
结束任务(E)

(6) Run-time, period and code-level view

- **Sequence diagram in UML: interactions among program units (objects)**



Execution tracing 执行跟踪

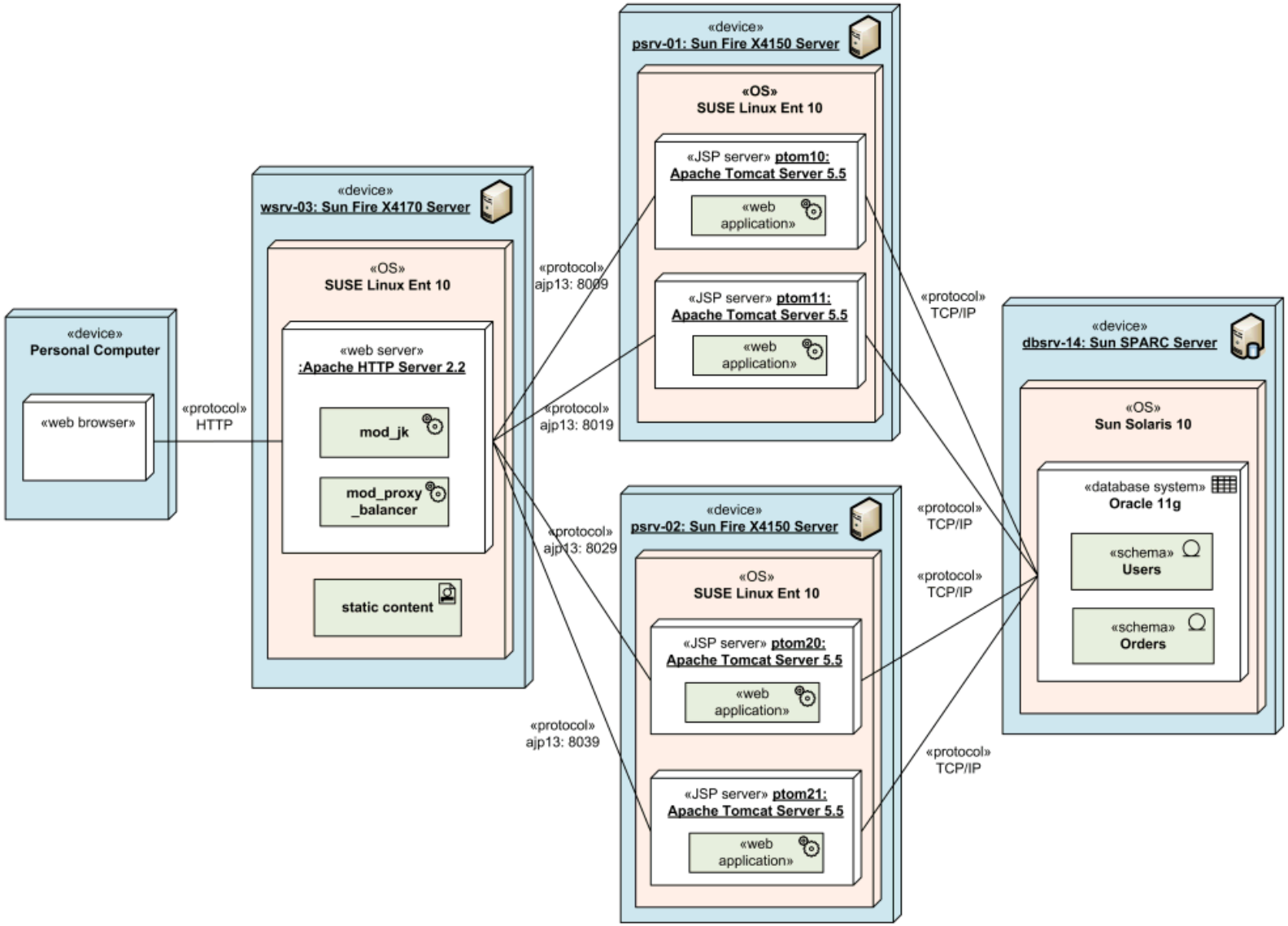
- **Tracing involves a specialized use of logging to record information about a program's execution.** 用日志方式记录程序执行的调用次序
  - This is typically used by programmers for debugging purposes, and depending on the type and detail of information contained in a trace log, by experienced administrators or technical-support personnel and by software monitoring tools to diagnose common problems with software.

```
09-26 10:59:38.056  28584-28584/
        java.lang.NullPointerExc
        at android.content.Conte
        at android.widget.Toast.
        at android.widget.Toast.
        at com.app.app.MainActiv
        at android.os.Handler.ha
        at android.os.Handler.di
        at android.os.Looper.loo
        at android.app.ActivityT
        at java.lang.reflect.Met
        at java.lang.reflect.Met
        at com.android.internal.
        at com.android.internal.
        at dalvik.system.NativeS
```

```
    0  java.lang.RuntimeException
!   1  at com.crittercism.testapp.errors.CustomError.crash(CustomError.java:57)
    2  at com.crittercism.testapp.fragments.ErrorFragment$4.onClick(ErrorFragment.java:150)
    3  at android.view.View.performClick(View.java:4438)
    4  at android.view.View$PerformClick.run(View.java:18422)
    5  at android.os.Handler.handleCallback(Handler.java:733)
    6  at android.os.Handler.dispatchMessage(Handler.java:95)
    7  at android.os.Looper.loop(Looper.java:136)
    8  at android.app.ActivityThread.main(ActivityThread.java:5001)
    9  at java.lang.reflect.Method.invokeNative(Native Method)
   10  at java.lang.reflect.Method.invoke(Method.java:515)
   11  at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
   12  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
   13  at de.robv.android.xposed.XposedBridge.main(XposedBridge.java:132)
   14  at dalvik.system.NativeStart.main(Native Method)
   15  Caused by: java.lang.NullPointerException
   16  at com.crittercism.testapp.errors.NullPointerCustomError.performError(NullPointerCustomError.java:10)
   17  at com.crittercism.testapp.errors.CustomError.initiateError(CustomError.java:83)
```

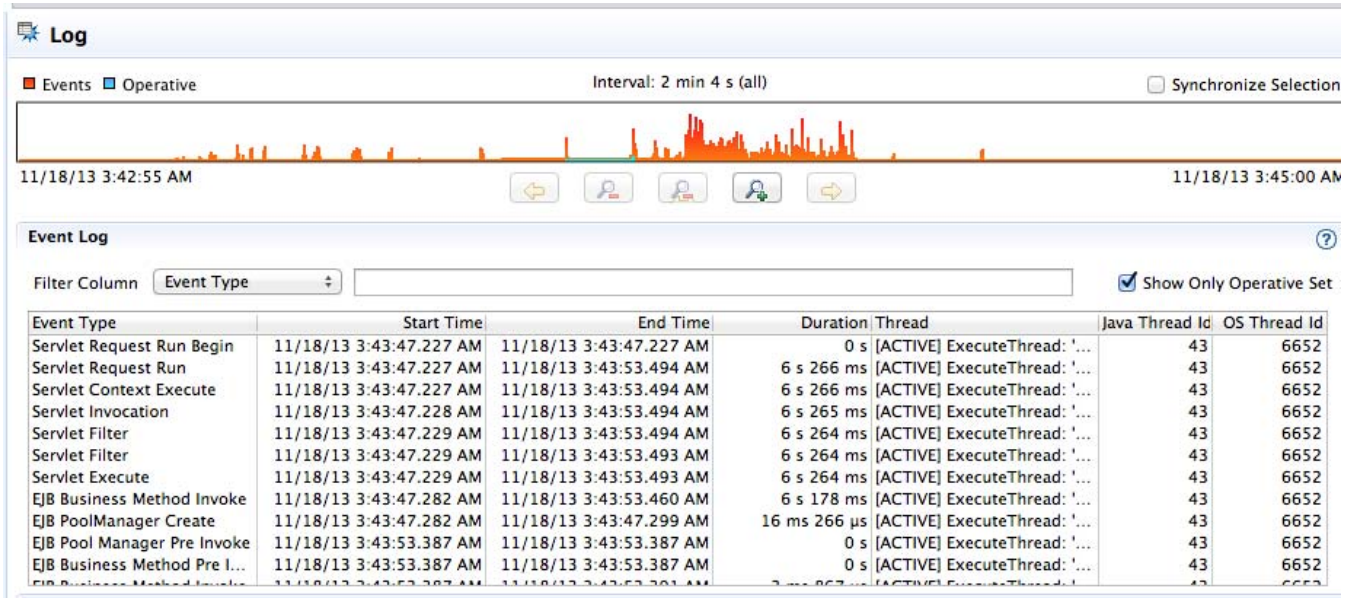
(7) Run-time, moment, and component-level view

- **Deployment diagram in UML**



(8) Run-time, period, and component-level view

- **Event logging provides system administrators with information useful for diagnostics and auditing.** 事件日志：系统层面
  - The different classes of events that will be logged, as well as what details will appear in the event messages, are considered in development cycle.
- **Each class of event to be assigned a unique "code" to format and output a human-readable message.**
  - This facilitates localization and allows system administrators to more easily obtain information on problems that occur.



Event Type	Start Time	End Time	Duration	Thread	Java Thread Id	OS Thread Id
Servlet Request Run Begin	11/18/13 3:43:47.227 AM	11/18/13 3:43:47.227 AM	0 s	[ACTIVE] ExecuteThread: '...	43	6652
Servlet Request Run	11/18/13 3:43:47.227 AM	11/18/13 3:43:53.494 AM	6 s 266 ms	[ACTIVE] ExecuteThread: '...	43	6652
Servlet Context Execute	11/18/13 3:43:47.227 AM	11/18/13 3:43:53.494 AM	6 s 266 ms	[ACTIVE] ExecuteThread: '...	43	6652
Servlet Invocation	11/18/13 3:43:47.228 AM	11/18/13 3:43:53.494 AM	6 s 265 ms	[ACTIVE] ExecuteThread: '...	43	6652
Servlet Filter	11/18/13 3:43:47.229 AM	11/18/13 3:43:53.494 AM	6 s 264 ms	[ACTIVE] ExecuteThread: '...	43	6652
Servlet Filter	11/18/13 3:43:47.229 AM	11/18/13 3:43:53.493 AM	6 s 264 ms	[ACTIVE] ExecuteThread: '...	43	6652
Servlet Execute	11/18/13 3:43:47.229 AM	11/18/13 3:43:53.493 AM	6 s 264 ms	[ACTIVE] ExecuteThread: '...	43	6652
EJB Business Method Invoke	11/18/13 3:43:47.282 AM	11/18/13 3:43:53.460 AM	6 s 178 ms	[ACTIVE] ExecuteThread: '...	43	6652
EJB PoolManager Create	11/18/13 3:43:47.282 AM	11/18/13 3:43:47.299 AM	16 ms 266 µs	[ACTIVE] ExecuteThread: '...	43	6652
EJB Pool Manager Pre Invoke	11/18/13 3:43:53.387 AM	11/18/13 3:43:53.387 AM	0 s	[ACTIVE] ExecuteThread: '...	43	6652
EJB Business Method Pre I...	11/18/13 3:43:53.387 AM	11/18/13 3:43:53.387 AM	0 s	[ACTIVE] ExecuteThread: '...	43	6652

Execution tracing and event logging

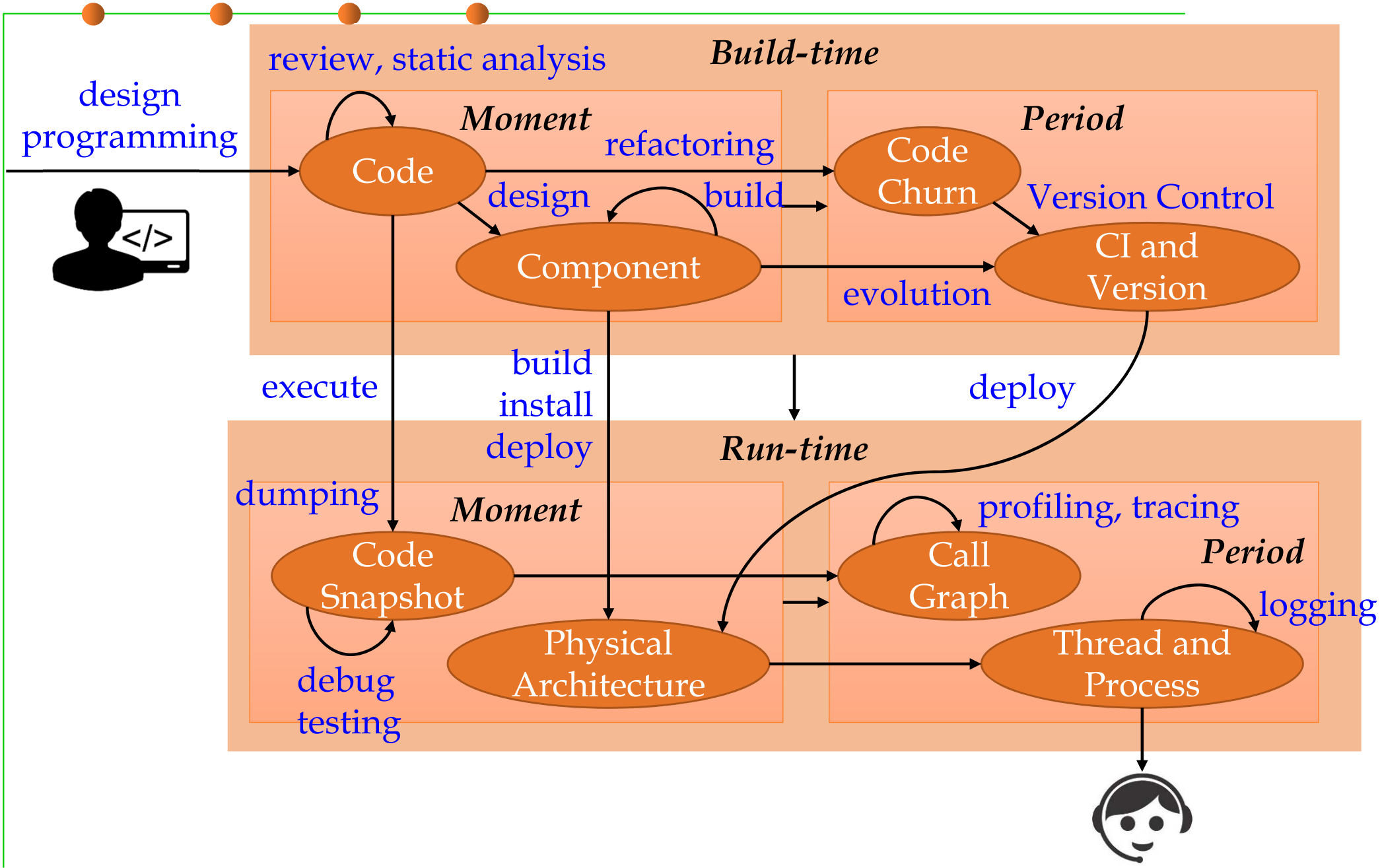
Event logging (构件/系统层面)	Execution tracing (代码层面)
Consumed primarily by system administrators	Consumed primarily by developers
Logs "high level" information (e.g. failed installation of a program)	Logs "low level" information (e.g. a thrown exception)
Must not be too "noisy" (contain many duplicate events or information not helpful to its intended audience)	Can be noisy
A standards-based output format is often desirable, sometimes even required	Few limitations on output format
Event log messages are often localized	Localization is rarely a concern
Addition of new types of events, as well as new event messages, need not be agile	Addition of new tracing messages *must* be agile

What we are focused on in this semester

	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST, Interface-Class-Attribute-Method (Class Diagram)	Package, File, Static Linking, Library, Test Case, Build Script (Component Diagram)	Code Churn	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution stack trace, Concurrent multi-threads	Event log, Multi-processes, Distributed processes
			Procedure Call Graph, Message Graph (Sequence Diagram)	

2 Software construction: Transformation between views

Software construction: transformation btw views



Types of Transformations in Software Construction

- **∅ ⇒ Code**
  - Programming / Coding (Chapter 3 ADT/OOP)
  - Review, static analysis/checking (Chapter 4 Understandability)
- **Code ⇒ Component**
  - Design (Chapter 3 ADT/OOP; Chapter 5 Reusability; Chapter 6 Maintainability)
  - Build: compile, static link, package, install, etc (Chapter 2 Construction process)
- **Build-time ⇒ Run-time**
  - Install / deploy (Course in the 3rd year)
  - Debug, unit/integration testing (Chapter 7 Robustness)
- **Moment ⇒ Period**
  - Refactoring (Chapter 9 Refactoring)
  - Version control (Chapter 2 Construction process)
  - Loading, dynamic linking, interpreting, execution (dumping, profiling, logging) (Chapter 8 Performance)

Summary

Summary of this lecture

- **Three dimensions of describing a software system:**
  - By phases: build- and run-time views
  - By dynamics: moment and period views
  - By levels: code and component views
- **Elements, relations, and models of each view**
- **Software construction: transformation between views**
  - $\varnothing \Rightarrow$ Code
  - Code $\Rightarrow$ Component
  - Build-time $\Rightarrow$ Run-time
  - Moment $\Rightarrow$ Period

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

The end

February 24, 2019