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

Xu Hanchuan
xhc@hit.edu.cn

February 24, 2019

Outline

- **Multi-dimensional software views**
  - 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
- **Summary**

Objectives of this lecture

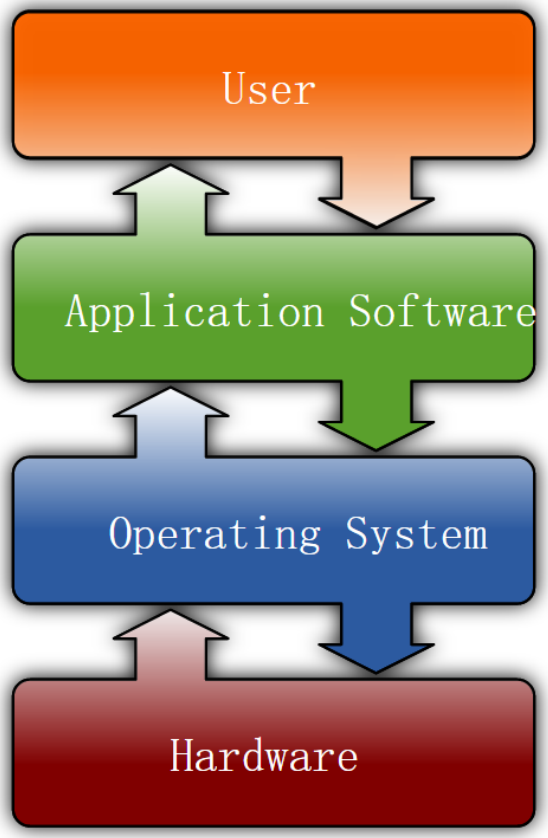
- **To understand the constituents(要素) of a software system in three orthogonal(正交的) dimensions;** 掌握软件系统三个维度中的各要素
- **To know what models are used to describe the morphology(形态) and states of a software system;** 掌握各类描述软件系统的模型
- **To treat software construction as the transformations between different views;** 理解软件的构造就是在不同视图之间的变换

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(专有的) 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

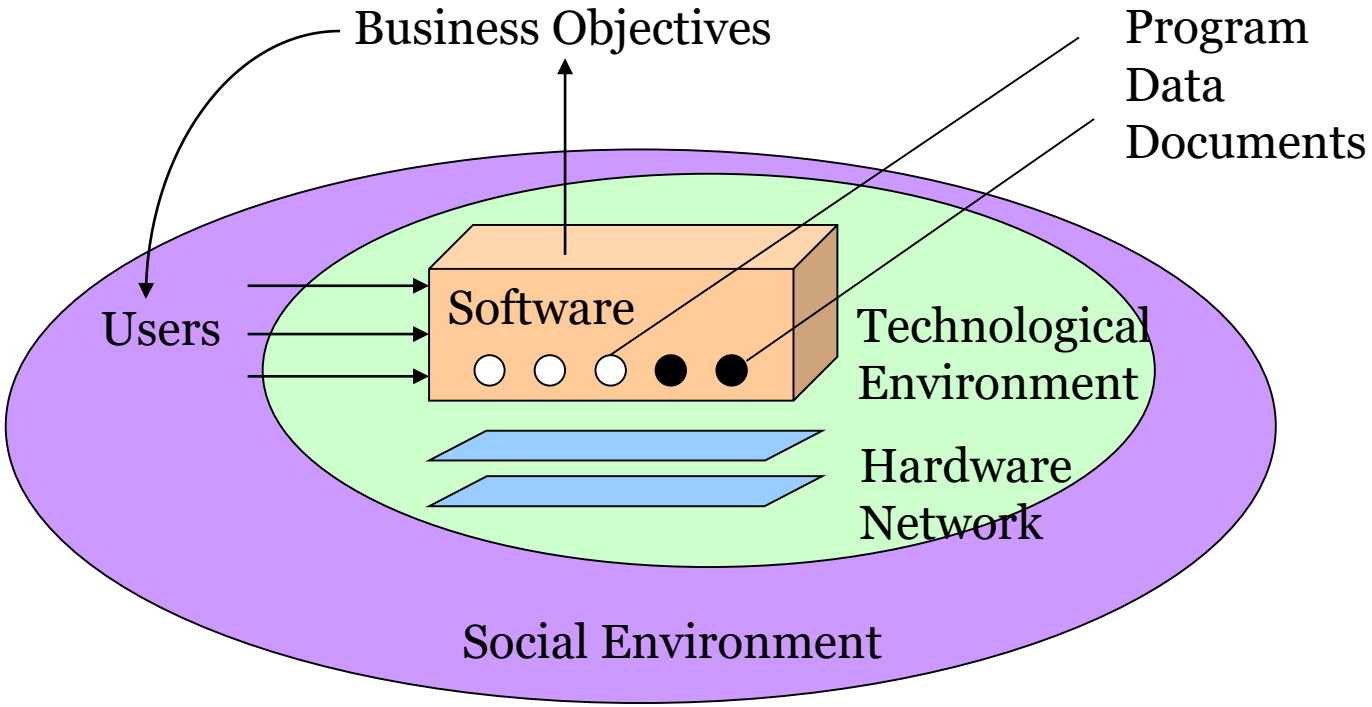
Constituents of a software system: one more step

- **Software system =**
  - `Programs` (UI, Algorithms, Utilities, APIs, test cases, etc)
  - `Data`(files, databases, etc)
  - `Documents` (需求规格说明SRS, 设计规格说明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



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(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)</td><td>Package, Source File, Static linking, Library, Test Case (Component Diagram)</td><td>Code Churn (代码变化)</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 trace</td><td>Event log</td></tr>
<tr><td colspan="2">Procedure Call Graph, Message Graph (Sequence Diagram)

Parallel and multi-threads/processes
Distributed processes</td></tr>
</table>

(1) Build-time Views

Multi-dimensional software views

	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static linking, Library, Test Case (Component Diagram)	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/processes Distributed processes	

Build-time views of a software system

- **Build-time: idea ⇒ requirement ⇒ design ⇒ code ⇒ installable / executable package** 软件构建的核心过程和环节
  - **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
  - **Component-level view**: **architecture** ---- how source code are physically organized by files, directories, packages, libraries, and the dependencies among them

    ---
  - **Moment view**: what do source code and component look like in a specific time
  - **Period view**: how do they evolve/change along with time

Multi-dimensional software views

	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static linking, Library, Test Case (Component Diagram)	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/processes Distributed processes	

(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. 在逻辑上代码是如何通过基本程序块组织的**
- **Three inter-related forms:**
  - Lexical-oriented source code(面向词法)
  - Syntax-oriented program structure: e.g., Abstract Syntax Tree (AST)(面向语法)
  - Semantics-oriented program structure: e.g., Class Diagram (面向语义)

Lexical-based semi-structured source code

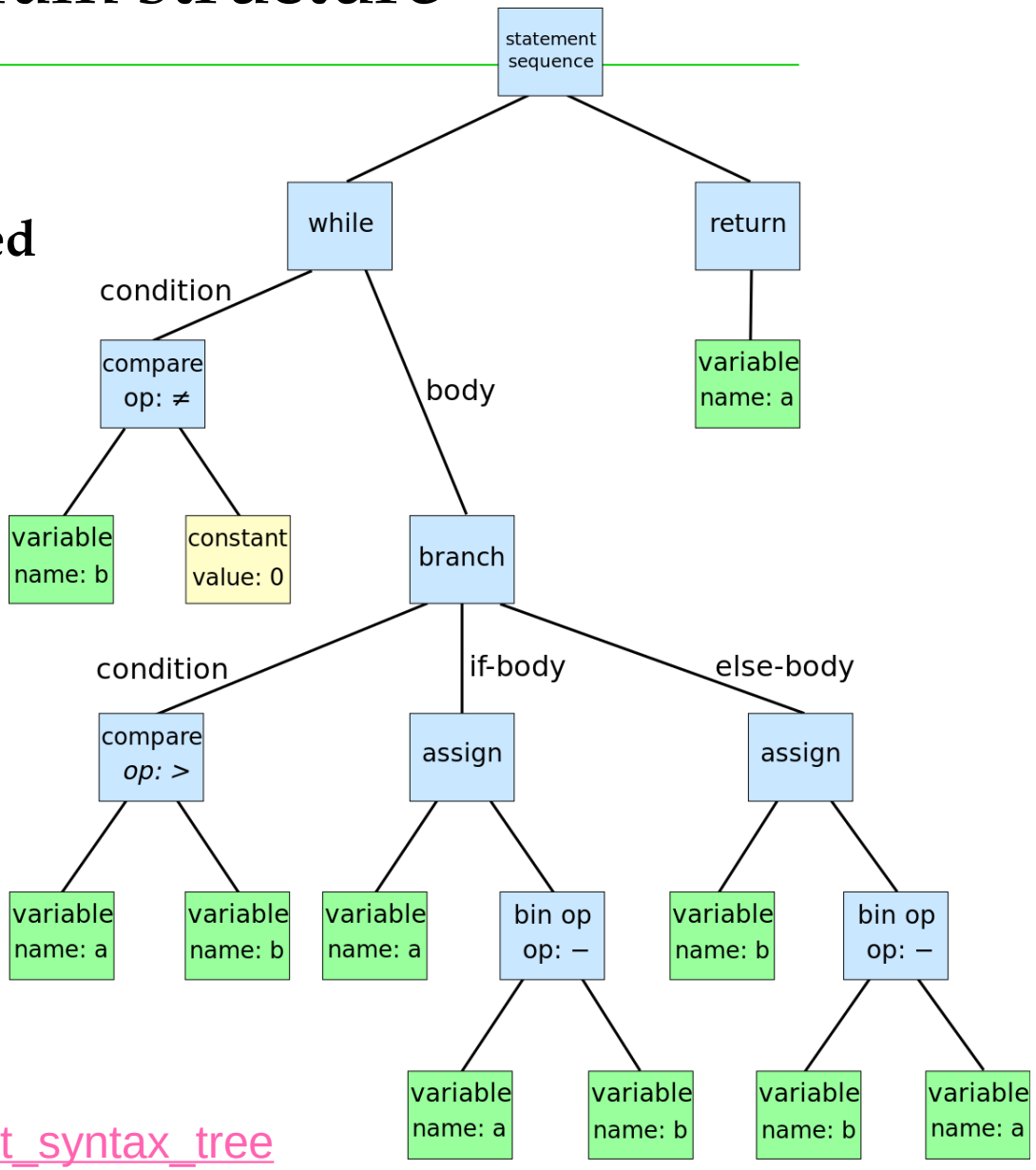
- **Source code: the most important assets in software development**

```
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 - retrieve information about the user
19 GetUserInfoResult userInfo = sfdc.getUserInfo();
20
21 // create a new account object locally
22 Account account = new Account();
23 account.setAccountNumber("002DF99ELK9");
24 account.setName("My New Account");
25 account.setBillingCity("Glasgow")
26
27
28 SObject[] sObjects = new SObject[2];
29 sObjects[0] = account;
30
31 // persist the object
32 SaveResult[] saveResults = sfdc.create(sObjects);
```

Syntax-oriented program structure

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

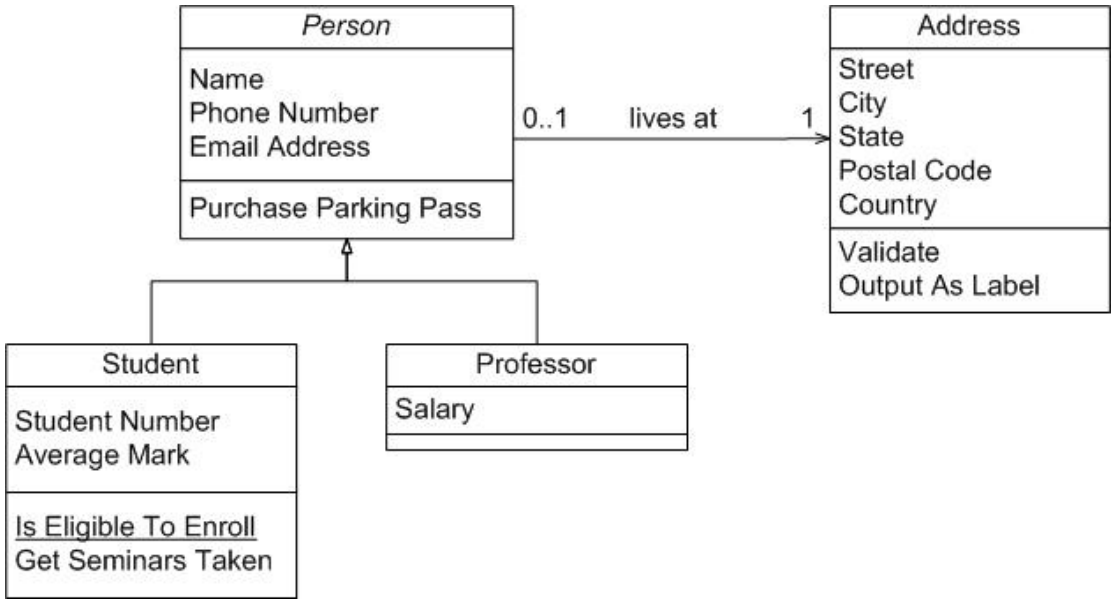
```
while (b ≠ 0) {
    if (a > b)
        a = a – b;
    else
        b = b – a;
}
return a;
```



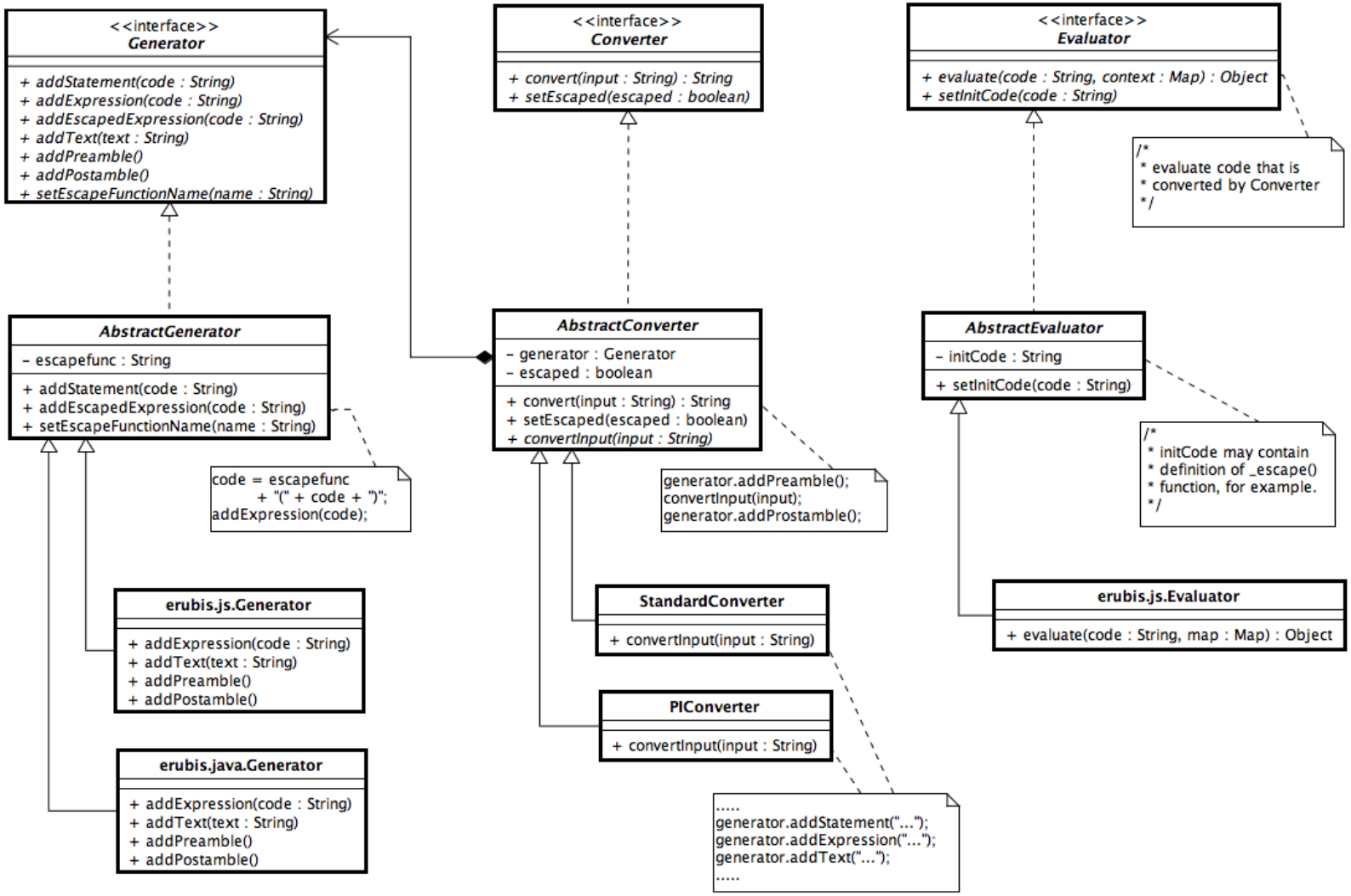
https://en.wikipedia.org/wiki/Abstract_syntax_tree

Semantics-oriented program structure

- **E.g., using Class Diagram (UML) to describe the interfaces, classes, attributes, methods, and relationships among them.**
- **Graphics-based or formally defined.**
- **Usually modeled in design phase, and transformed into source code.**
- **It is the result of Object-Oriented Analysis and Design in terms of user requirements.**



Semantics-oriented program structure



Multi-dimensional software views

	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static linking, Library, Test Case (Component Diagram)	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/processes Distributed processes	

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

- **Views describing "changes" along with time.**
- **Code churn: 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);
```

one line added and one line modified

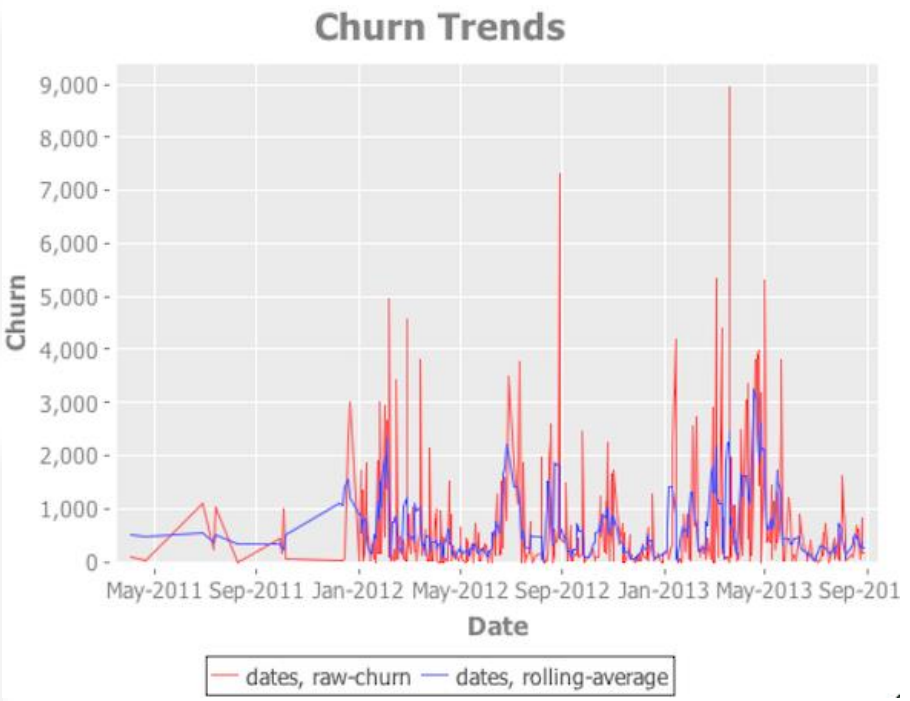
Code Before

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

Code After

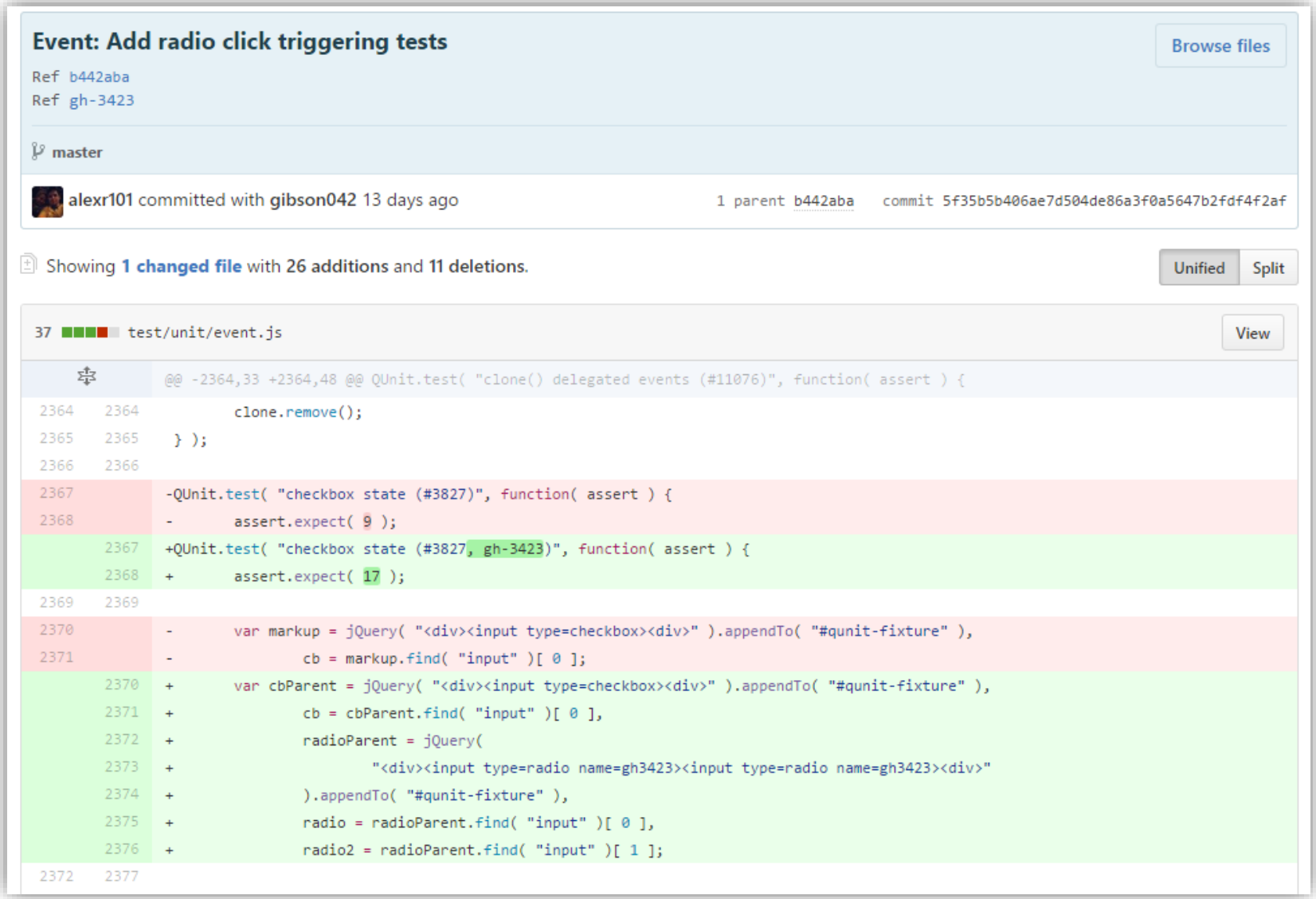
```
//print n integers iff n≥ 0
int i = n;
while (--i > 0)
   printf (" %d", i);
```

one line deleted and one line modified



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
```

Multi-dimensional software views

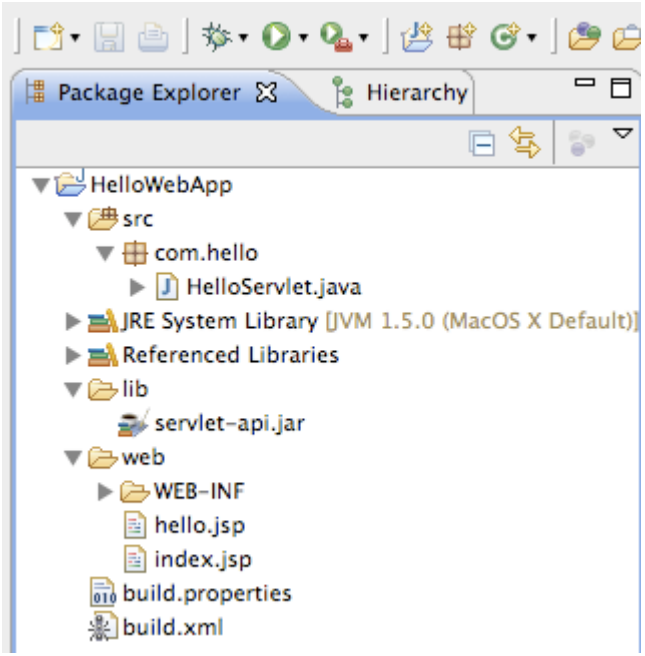
	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static linking Library, Test Case (Component Diagram)	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/processes Distributed processes	

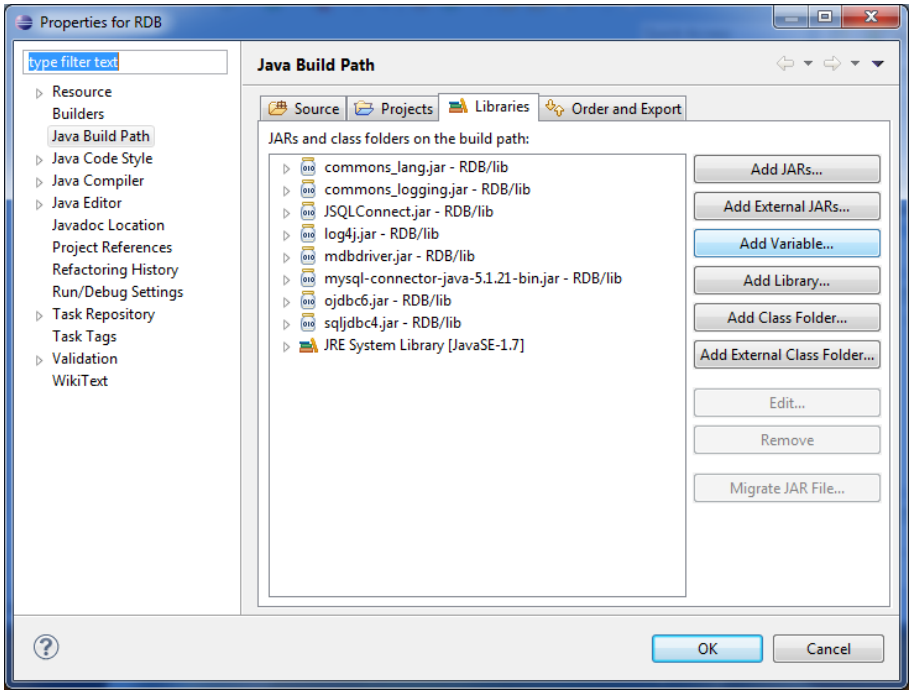
(3) Build-time, moment, and component-level view

- **Source code are physically organized into files which further are organized by directories;**

Static linking

- **Files are encapsulated into packages and, logically, components and sub-systems. 文件被封装成包、组件和子系统**
- **Reusable modules are in the form of libraries. 可复用模块形成类库**





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.

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

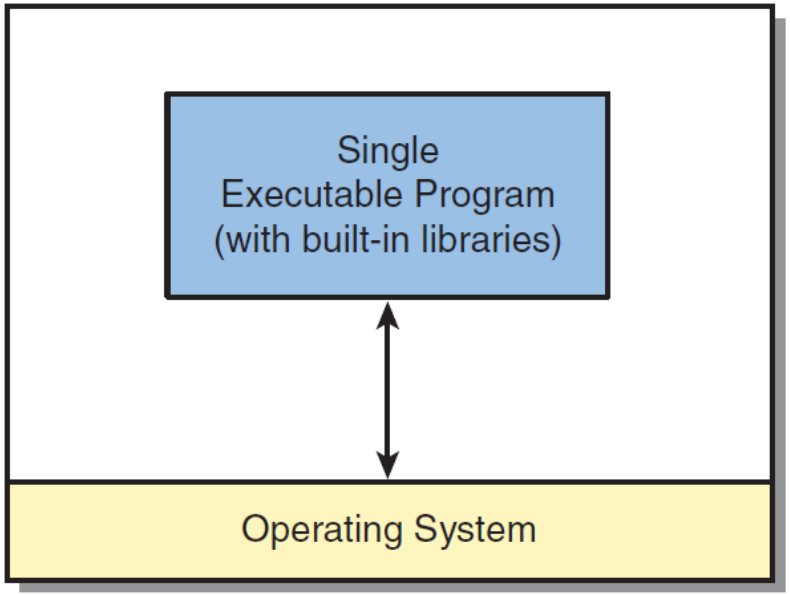
- **Sources of libraries:**
  - From OS pre-installed set of libraries for operations such as file and network I/O, GUI, mathematics, database assess; OS预装
  - From language SDK; 语言自带
  - From third-party sources, such as downloading them from the Internet. 第三方
  - Developers can also publish their own libraries. 自行开发

Linking with a library

- **When a program is edited, built and installed, a list of libraries to search must be provided.** 用到的类库列表
- **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.**
- **When the function is found, the appropriate object file is copied into the executable program.** 复制类库到可执行程序中
- **Two different approaches of integrating a library into an executable program:**
  - Static linking 静态链接
  - Dynamic linking 动态链接

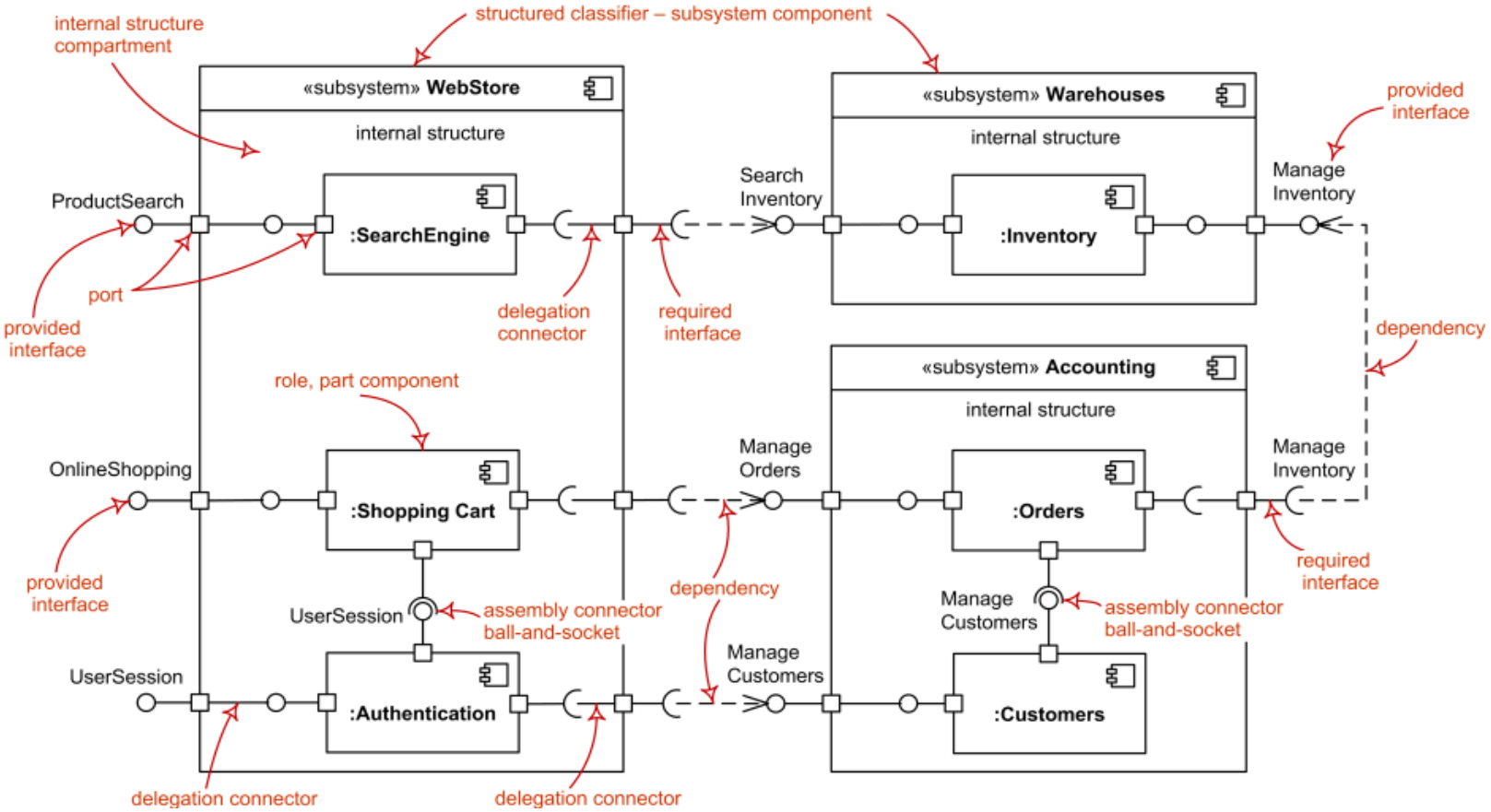
Static linking

- **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.** 将类库文件复制到可执行文件中，称为可执行文件的一部分。
  - 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** --- The act of linking a library with the developer's own software happens during the build process.
  - **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.



Component diagram in UML

- **In Unified Modeling Language (UML), a component diagram depicts how components are wired together to form larger components or software systems.**

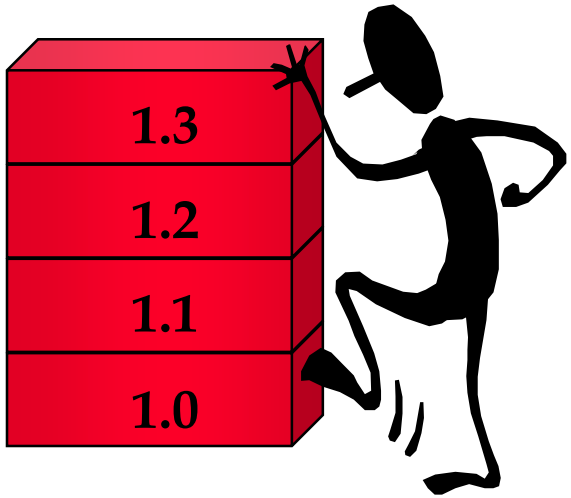


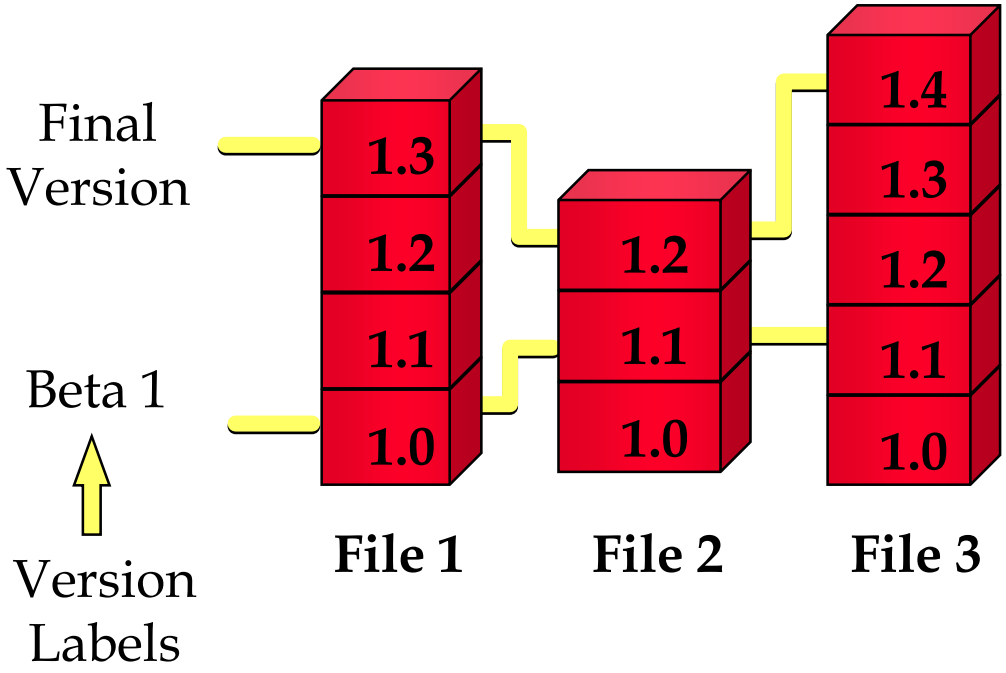
Multi-dimensional software views

	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static linking, Library, Test Case (Component Diagram)	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/processes Distributed processes	

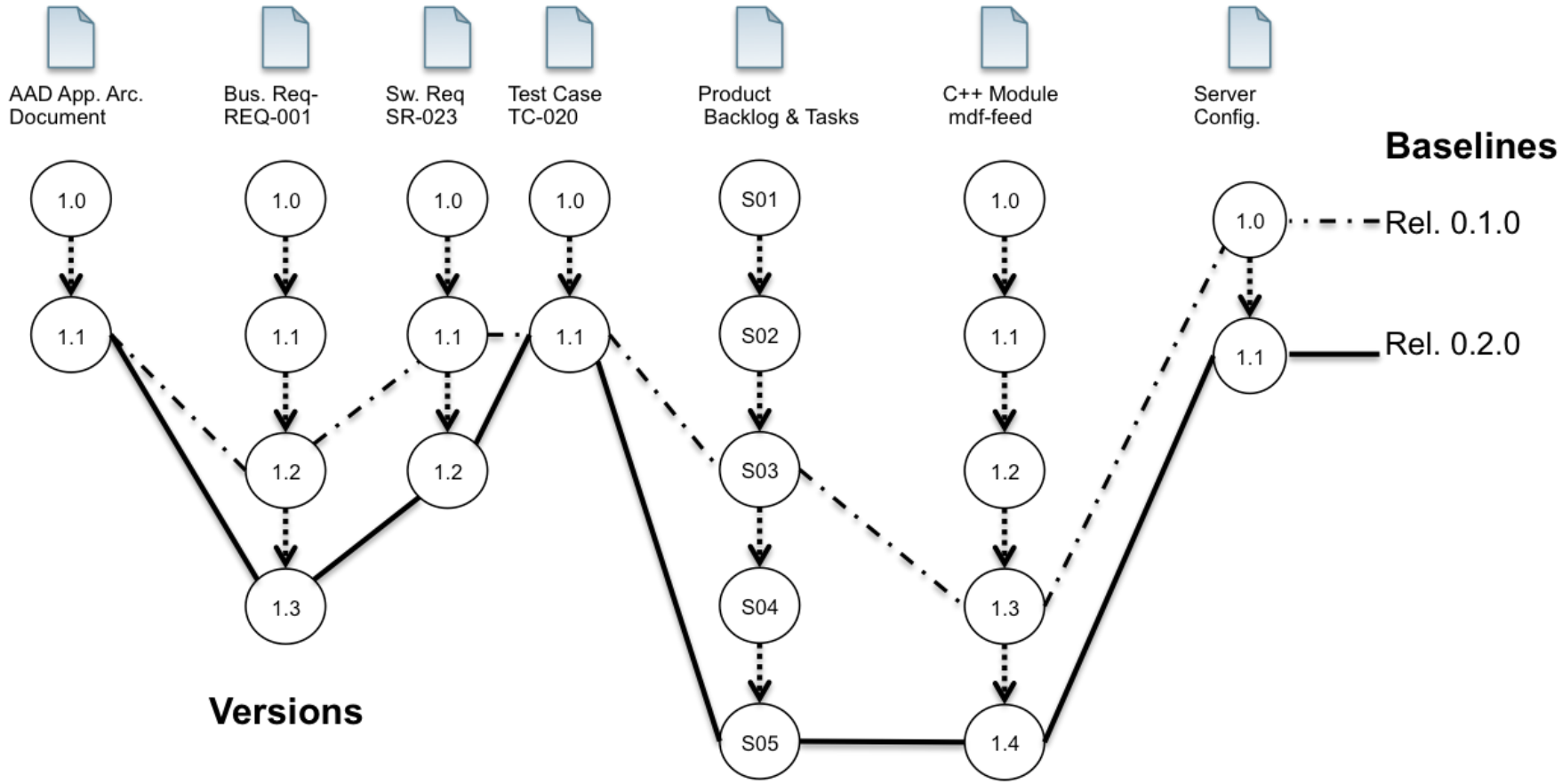
(4) Build-time, period, and component-level view

- **How do all files/packages/components/libraries change in a software system along with time?**
- **Software Configuration Item (SCI)**
- **Version**

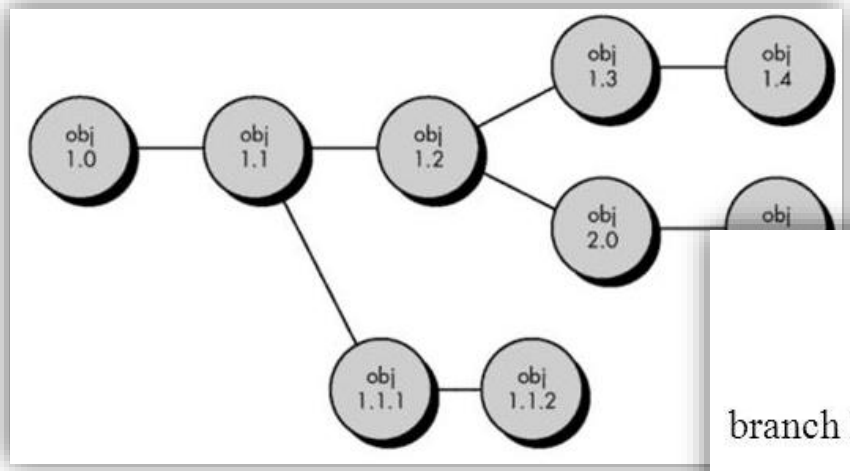


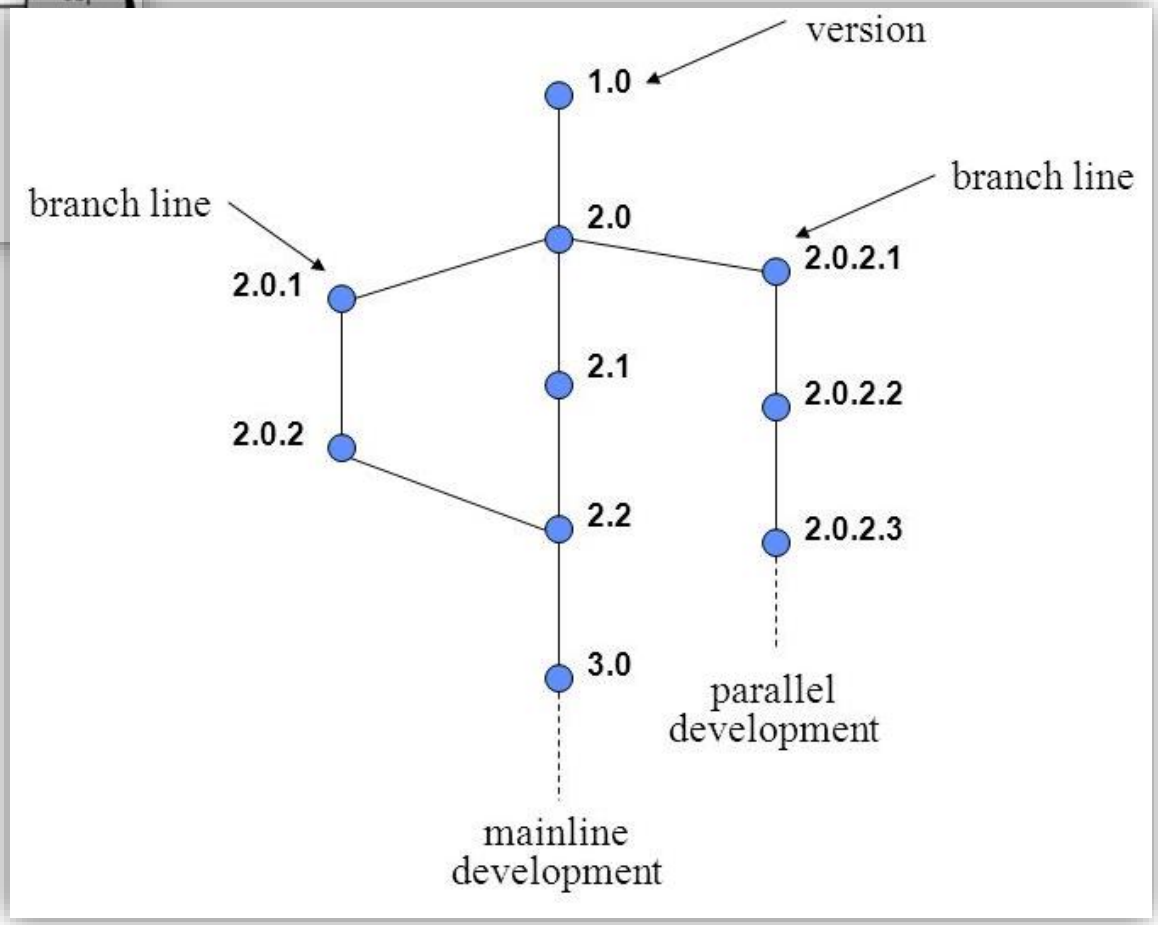


Version Control System (VCS)



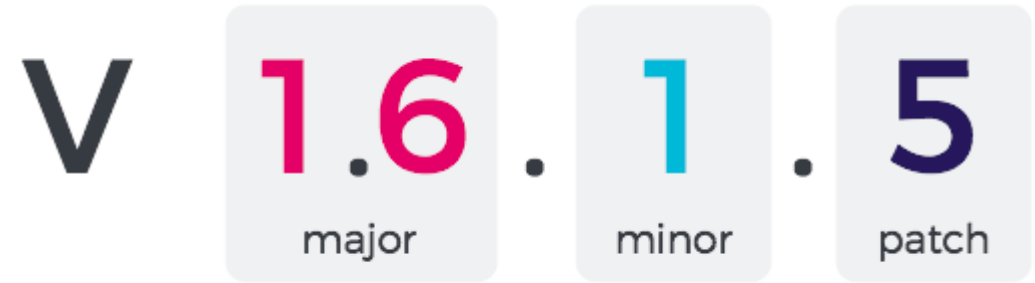
Evolution Graph (of a SCI or a Software)

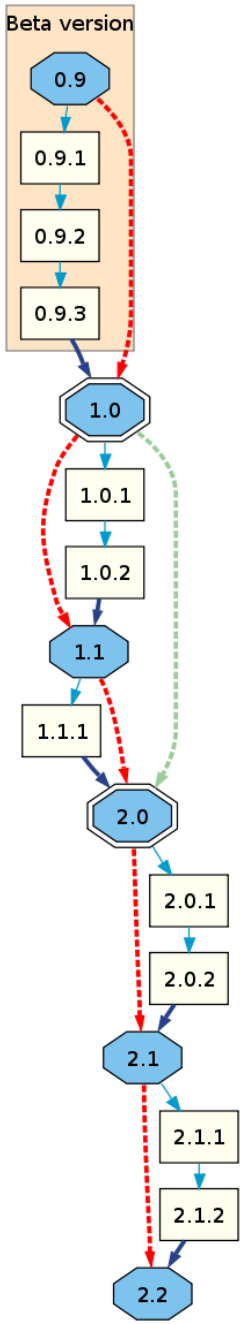




Versioning

- **Software versioning is the process of assigning either unique version names or unique version numbers to unique states of computer software. 版本控制是给计算机软件的不同状态分配唯一的名字或者编号的过程**
  - 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.软件演化是指软件产生之后，由于不同的原因对其进行持续地升级过程。**
  - 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

Multi-dimensional software views

	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static linking, Library, Test Case (Component Diagram)	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/ processes Distributed processes	

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?
  - **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?
  - **Component-level view**: architecture ---- how are software packages deployed into physical environment (OS, network, hardware, etc) and how do they interact?

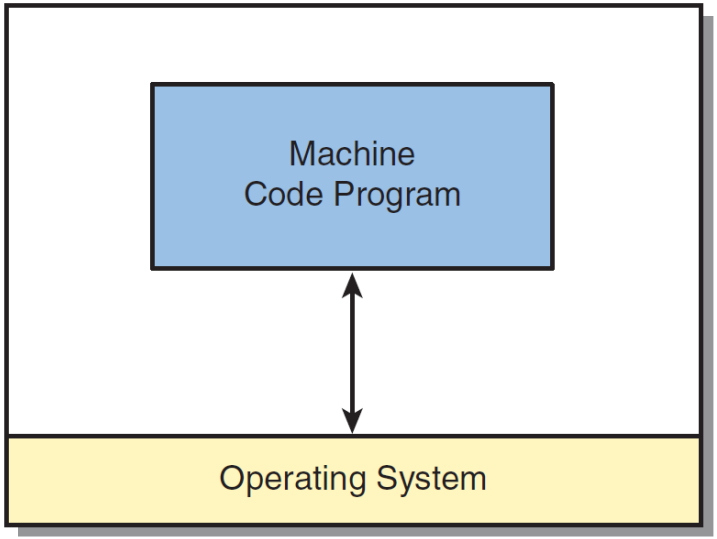
  - **Moment view**: how do programs behave in a specific time
  - **Period view**: how do they behave along with time

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.

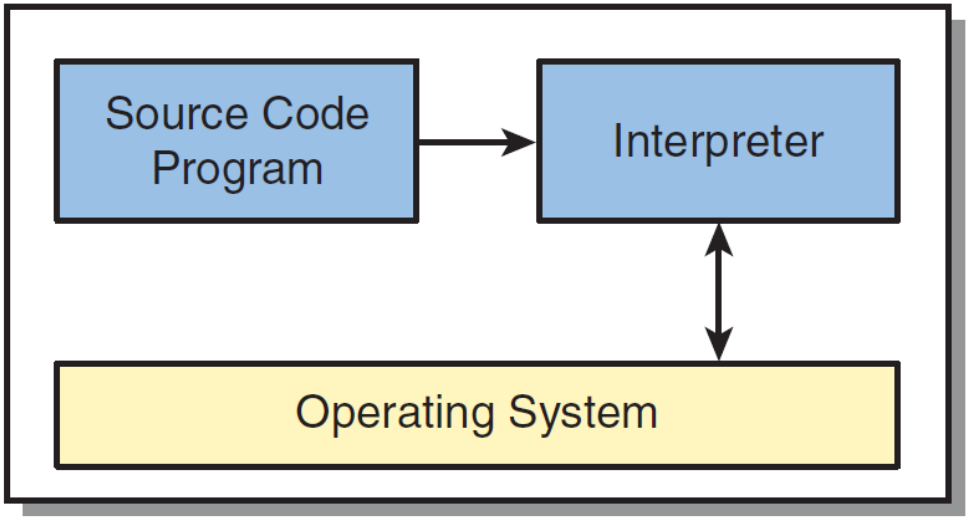
Executable Programs: Native Machine Code

- 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:** 完全转换为CPU能识别的机器码
  - 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.



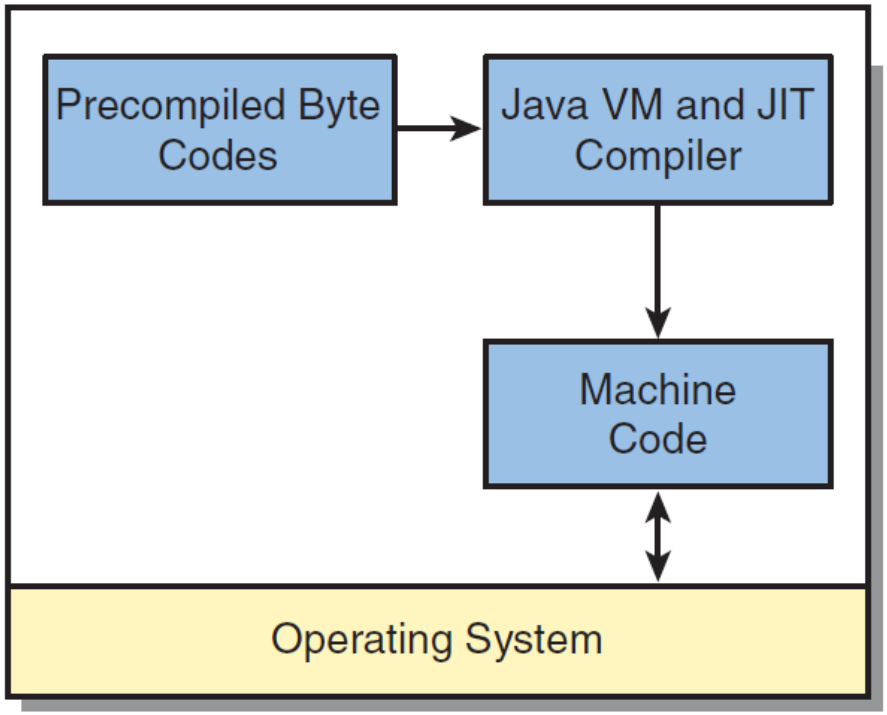
Executable Programs: Full Interpretation

- **Full Program Interpretation:** the runtime system loads the entire source code into memory and interprets it (such as BASIC, UNIX shell, etc) 源代码被加载入内存进行解释执行



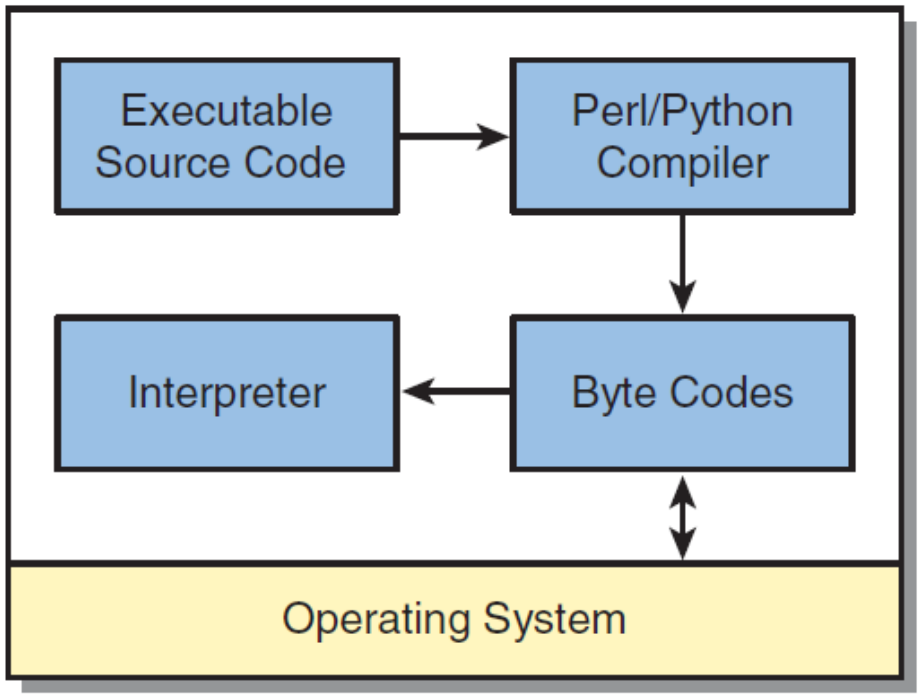
Executable Programs: Interpreted Byte Codes

- **Interpreted Byte Codes:**
  - Byte codes are similar to native machine code, except that the CPU doesn't directly understand them. 程序被编译成字节码形式(如 java的class文件)
  - It first translates them into native machine code or interprets them as the program executes. 运行时需要由解析器转换成机器码或者解释执行
  - 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. (在执行时编译为字节码解释执行)
- **The simple act of executing the Perl or Python script automatically triggers the generation of byte codes.**

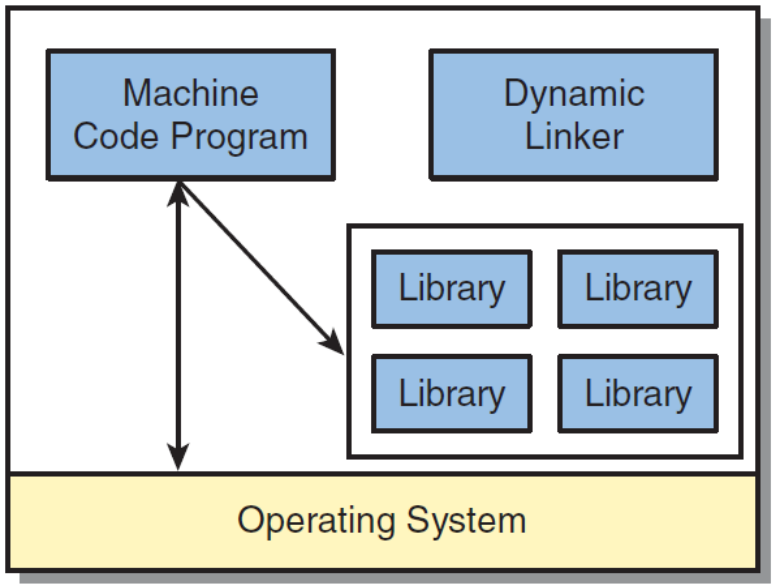


Dynamic linking

- **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.** 不将目标文件复制到可执行程序中，而是会标注用到的类库
- When the program starts running, the **libraries are loaded into memory as separate entities** and then are connected with the main program. 在运行时，加载用到的库到内存中，然后同主程序链接
- 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. 部署时需要将用到的类库同程序一起部署

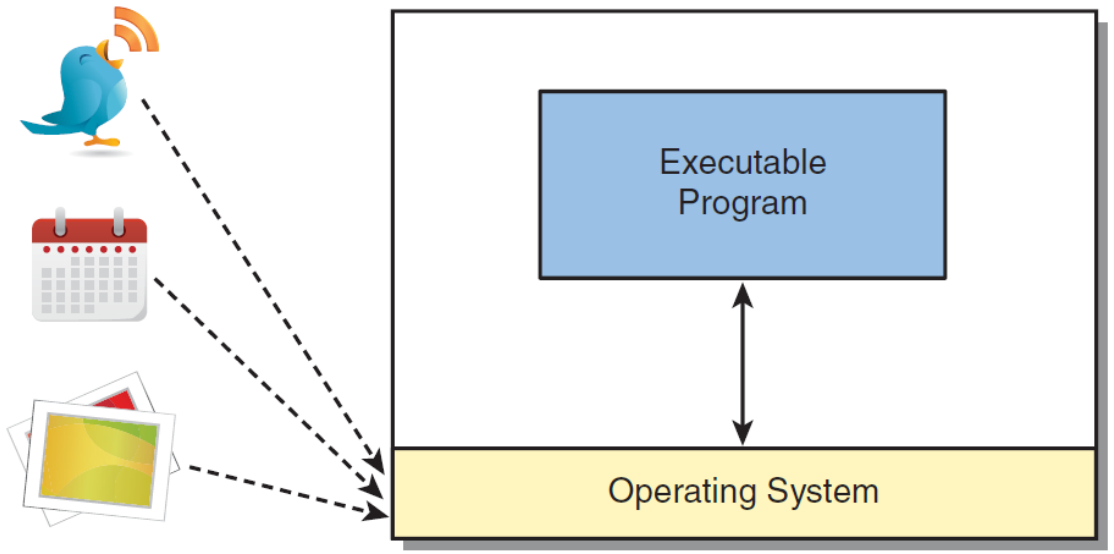
Dynamic linking

- **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. 类库变化时，不需要重新生成可自行程序
  - 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. 多个运行中程序可共享同一类库，优化内存使用



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.

Multi-dimensional software views

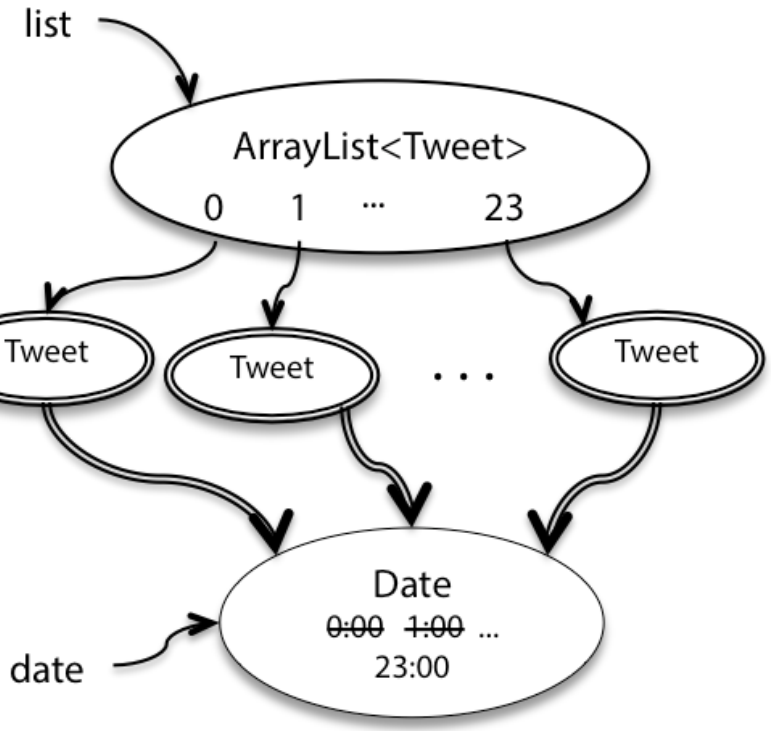
	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static linking, Library, Test Case (Component Diagram)	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/processes Distributed processes	

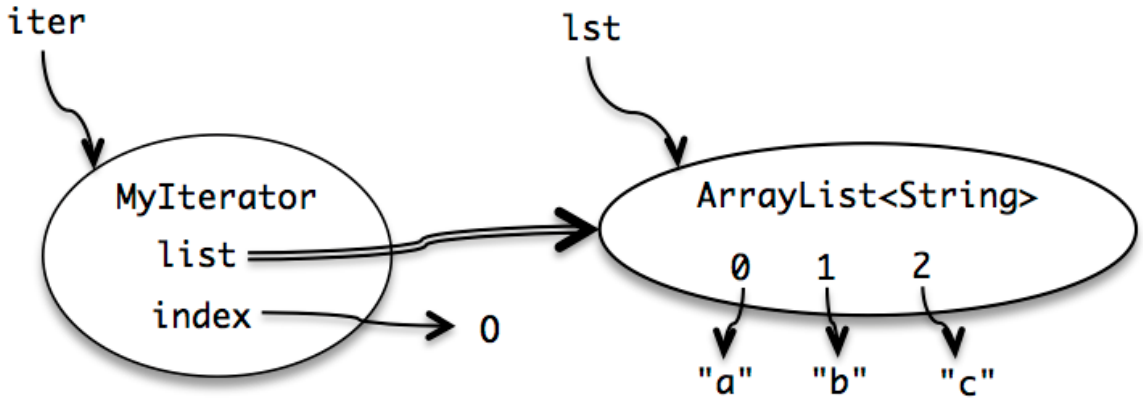
(5) Run-time, moment, and code-level view

- **Snapshot diagram: focusing on variable-level execution states in the memory of a target computer.**

  刻画内存中某时刻变量的状态

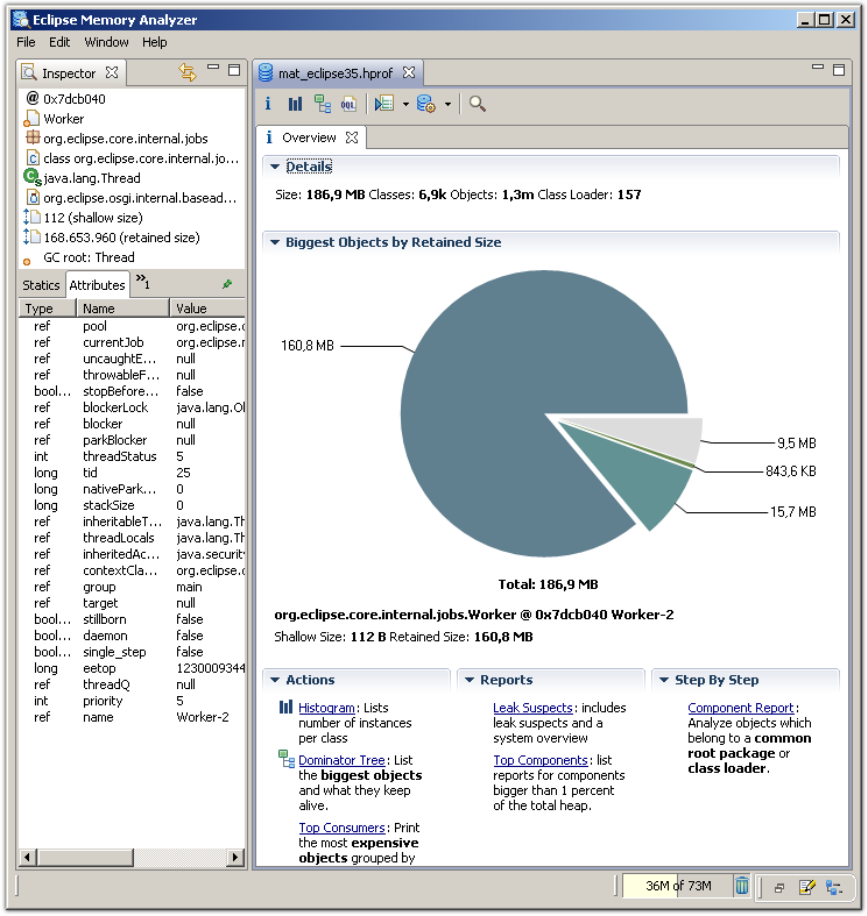
- **Fine-grained states.**



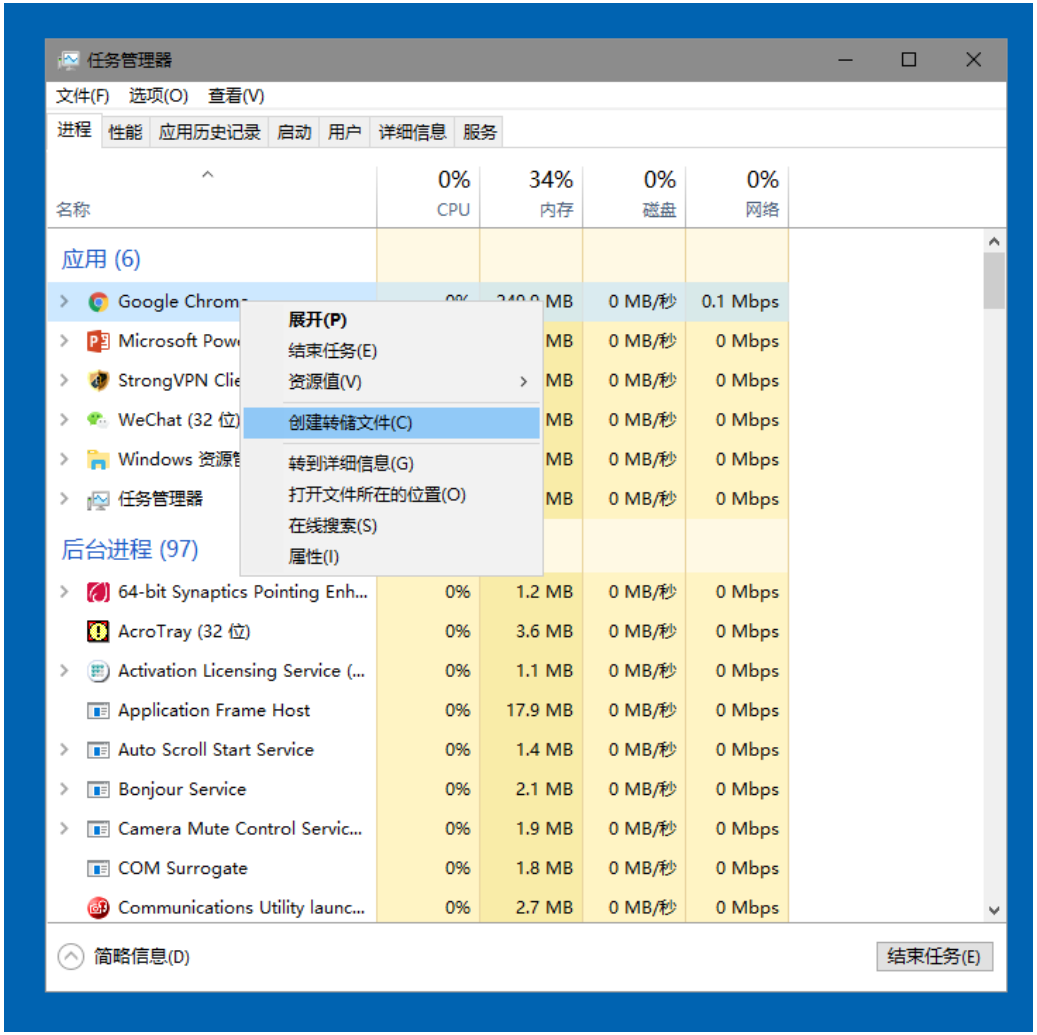


Memory dump

- **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

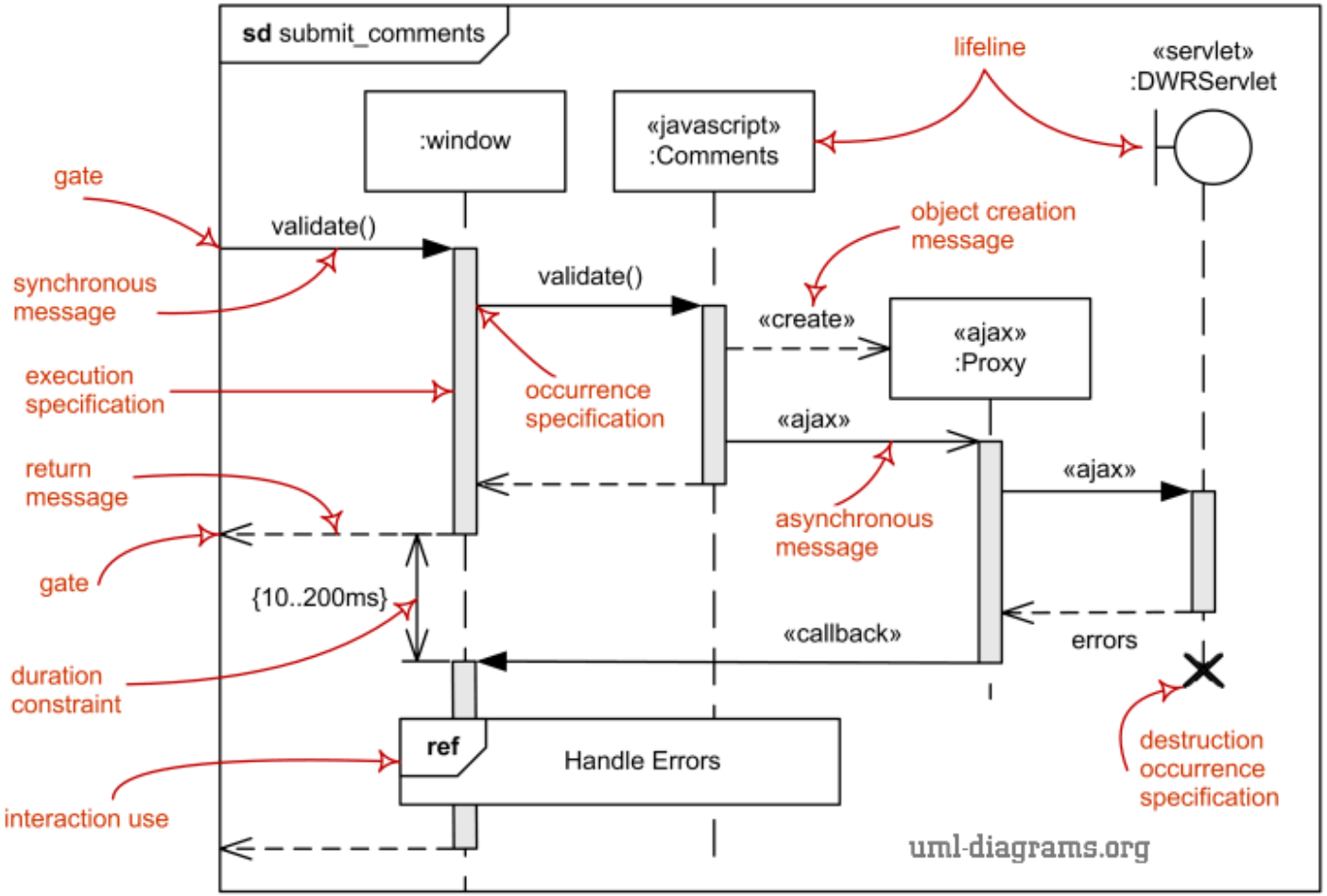

任务管理器
文件(F) 选项(O) 查看(V)
进程 性能 应用历史记录 启动 用户 详细信息 服务
名称
0% CPU
34% 内存
0% 磁盘
0% 网络
应用 (6)
Google Chrom
Microsoft Pow
StrongVPN Clie
WeChat (32 位)
Windows 资源管
任务管理器
展开(P)
结束任务(E)
资源值(V)
创建转储文件(C)
转到详细信息(G)
打开文件所在的位置(O)
在线搜索(S)
属性(I)
后台进程 (97)
64-bit Synaptics Pointing Enh... 0% 1.2 MB 0 MB/秒 0 Mbps
AcroTray (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 MB/秒 0.1 Mbps
0 MB/秒 0 Mbps
简略信息(D)
结束任务(E)

Multi-dimensional software views

	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static linking, Library, Test Case (Component Diagram)	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/processes Distributed processes	

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

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



Execution tracing

- **Tracing involves a specialized use of logging to record information about a program's execution.**
  - This information is typically used by programmers for **debugging purposes**, and additionally, depending on the type and detail of information contained in a trace log, by experienced system administrators or technical-support personnel and by software monitoring tools to **diagnose common problems with software**.

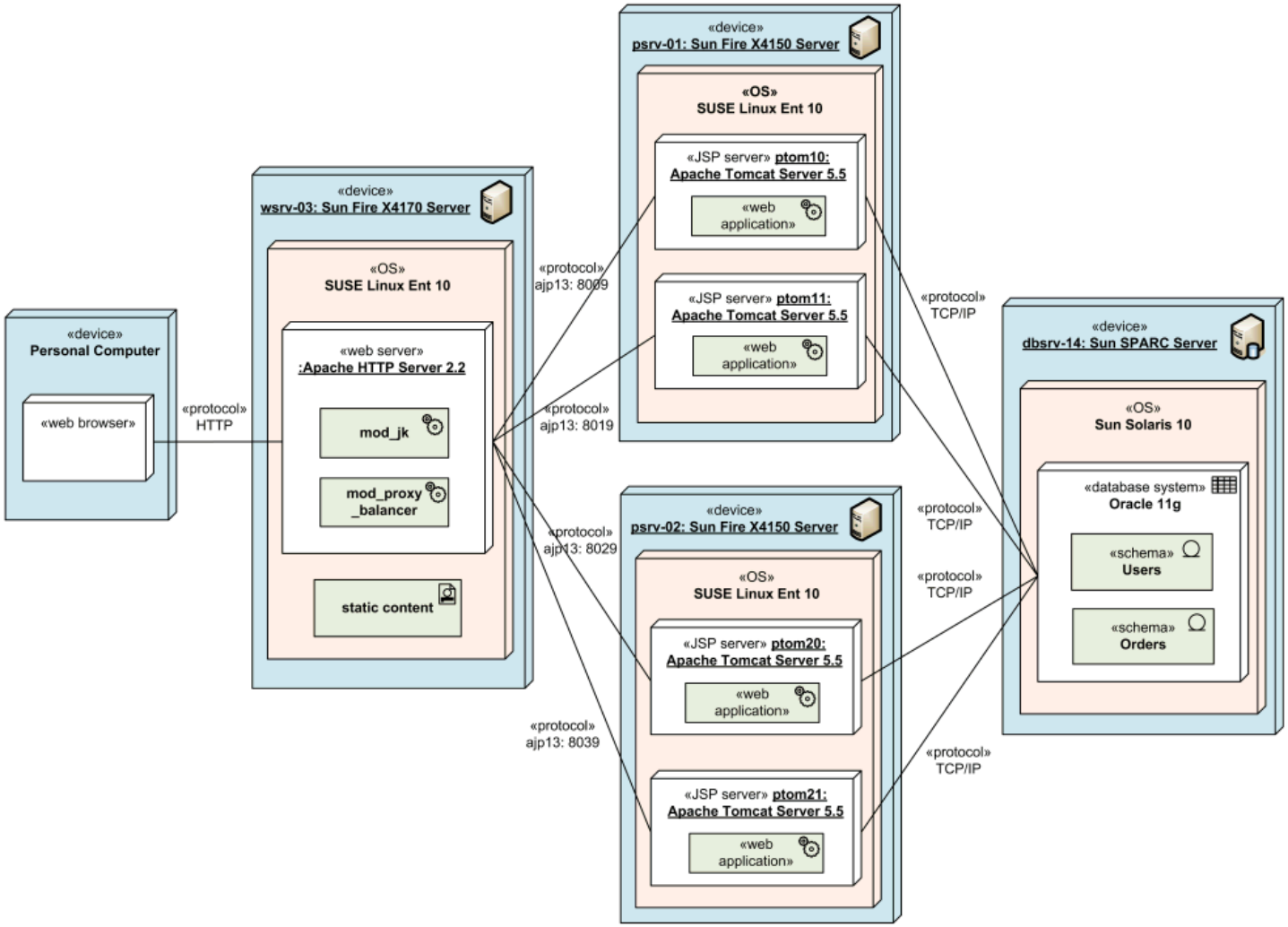
```
09-26 10:59:38.056  28584-28584/? E/AndroidRuntime:  FATAL EXCEPTION: main
        java.lang.NullPointerException
        at android.content.ContextWrapper.getResources(ContextWrapper.java:81)
        at android.widget.Toast.<init>(Toast.java:92)
        at android.widget.Toast.makeText(Toast.java:233)
        at com.app.app.MainActivity$3.run(MainActivity.java:132)
        at android.os.Handler.handleCallback(Handler.java:605)
        at android.os.Handler.dispatchMessage(Handler.java:92)
        at android.os.Looper.loop(Looper.java:137)
        at android.app.ActivityThread.main(ActivityThread.java:4428)
        at java.lang.reflect.Method.invokeNative(Native Method)
        at java.lang.reflect.Method.invoke(Method.java:511)
        at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:787)
        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:554)
        at dalvik.system.NativeStart.main(Native Method)
```

Multi-dimensional software views

	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static linking, Library, Test Case (Component Diagram)	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/processes Distributed processes	

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

- **Deployment diagram in UML**

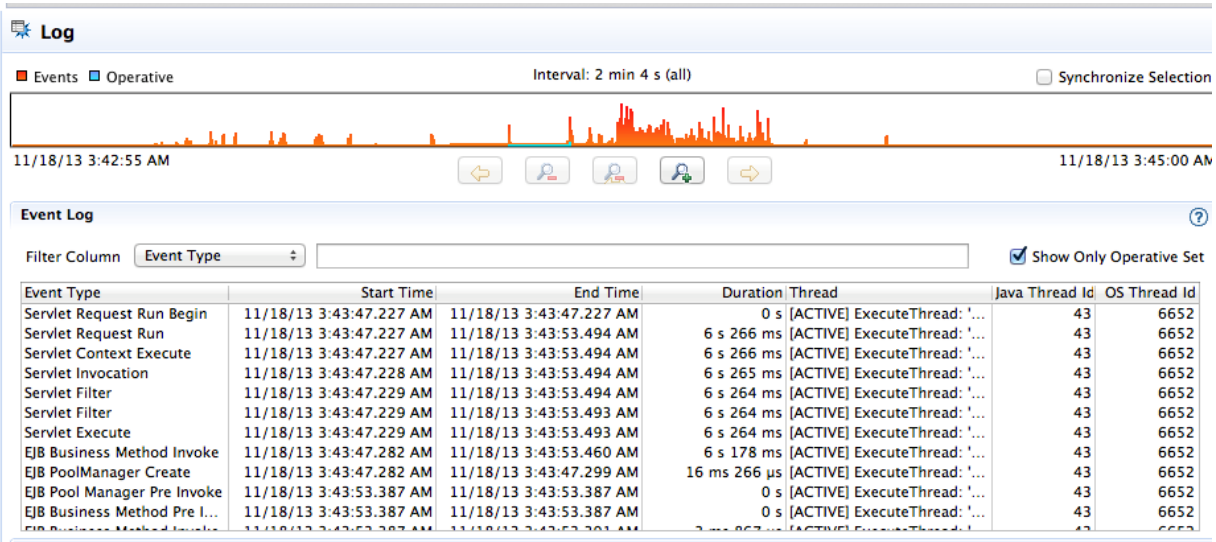


Multi-dimensional software views

	Moment		Period	
	Code-level	**Component-level**	**Code-level**	**Component-level**
Build-time	Source code, AST(抽象语法树), Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static linking, Library, Test Case (Component Diagram)	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/processes Distributed processes	

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

- **Event logging provides system administrators with information useful for diagnostics and auditing. 事件日志**
  - 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.**



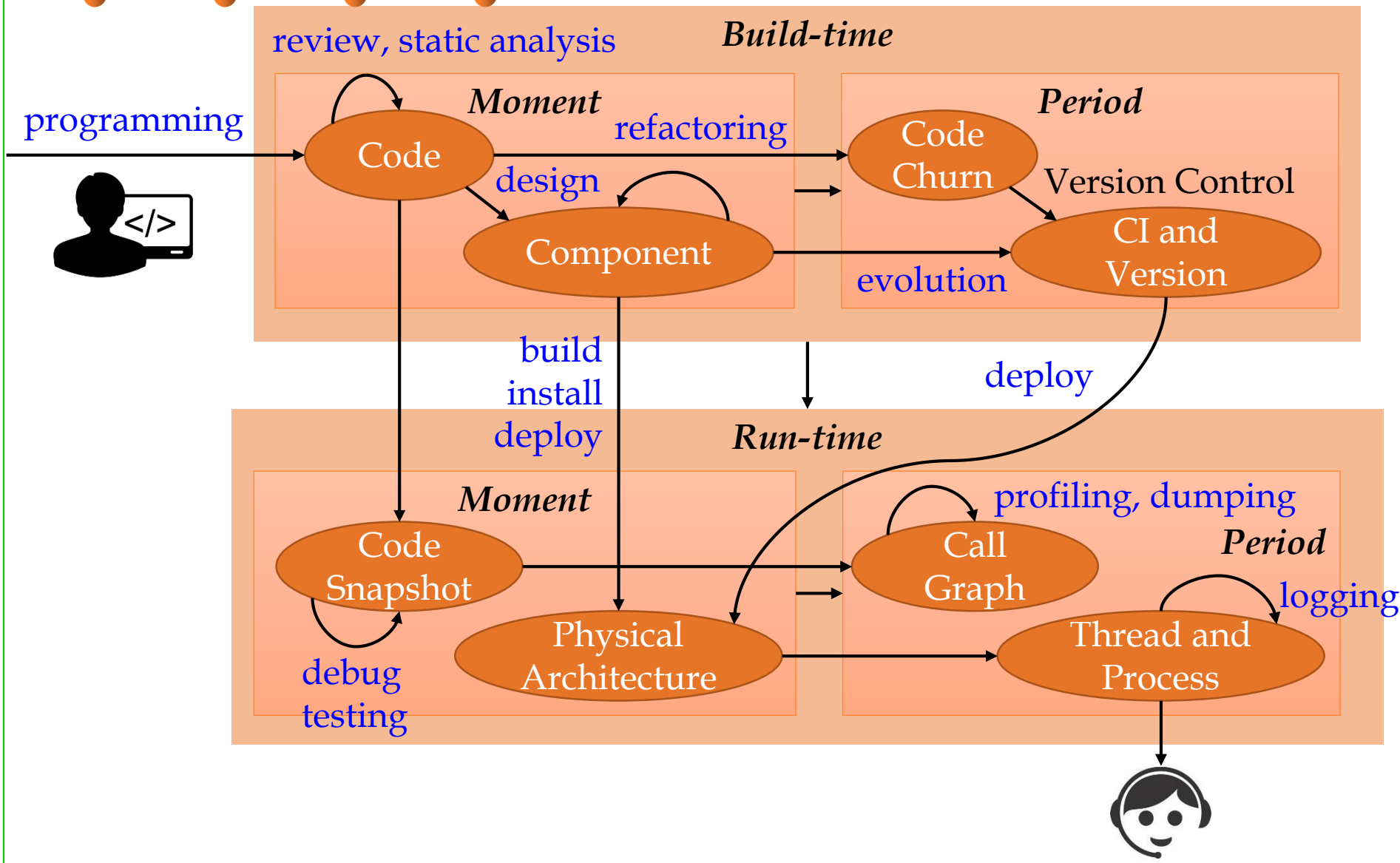
Event Type	Start Time	End Time	Duration	Thread	Java Thread Id	OS Thread Id
Servlet Request Run Begin	11/18/13 3:43:47.227 AM	11/18/13 3:43:47.227 AM	0 s	[ACTIVE] ExecuteThread: '...	43	6652
Servlet Request Run	11/18/13 3:43:47.227 AM	11/18/13 3:43:53.494 AM	6 s 266 ms	[ACTIVE] ExecuteThread: '...	43	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	Execution tracing
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**

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, clean (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 SCM)
  - 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

The end

February 24, 2019