Software Metrics

Lecture 1

Introduction

Yuming Zhou

Contents

- Overview
- Outline of the course
- What is measurement?
- What are software metrics?
- Scope of software engineering metrics: a chronological review

软件度量背景简介

- 在计算机学科中的位置？
- 我们能用它来干什么？

百年来对软件最有影响的10人

① Charles Babbage

② Lady Ada Byron King

③ Alan Kay (Xerox …)

④ Alan Turing

⑤ Capers Jones (IBM)

⑥ Watts S. Humphrey (IBM→CMU)

⑦ Barry Boehm

⑧ Tim Berners-Lee

⑨ Grace Murray Hopper

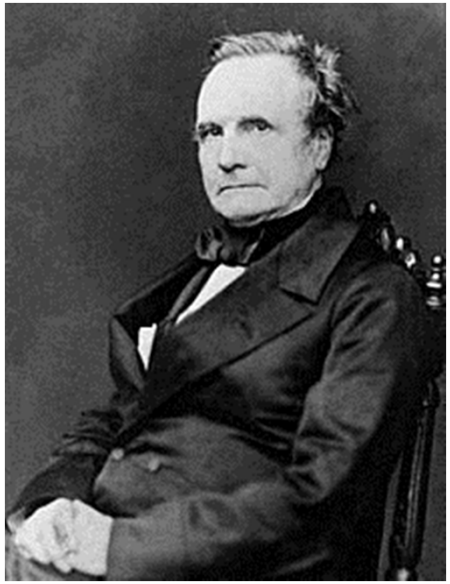
⑩ Ed Yourdon

百年来对软件最有影响的10人

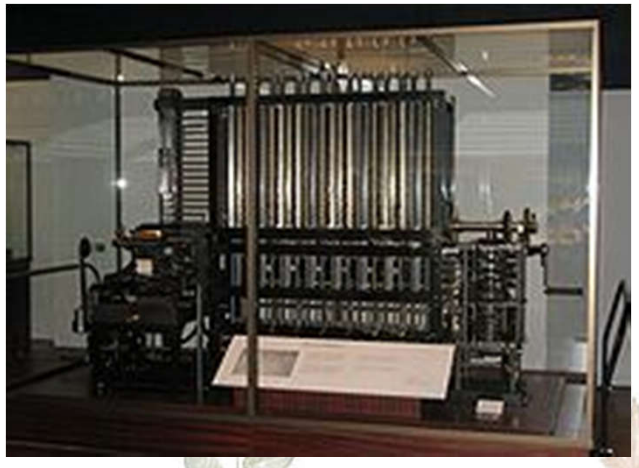
① Charles Babbage (1791-1871)

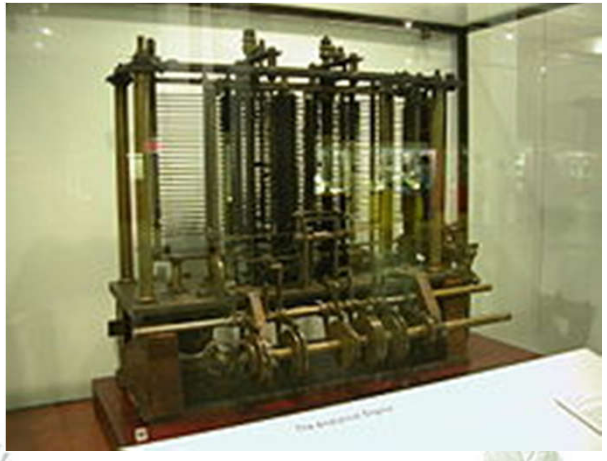
剑桥大学毕业

“father of computer” 可编程计算机



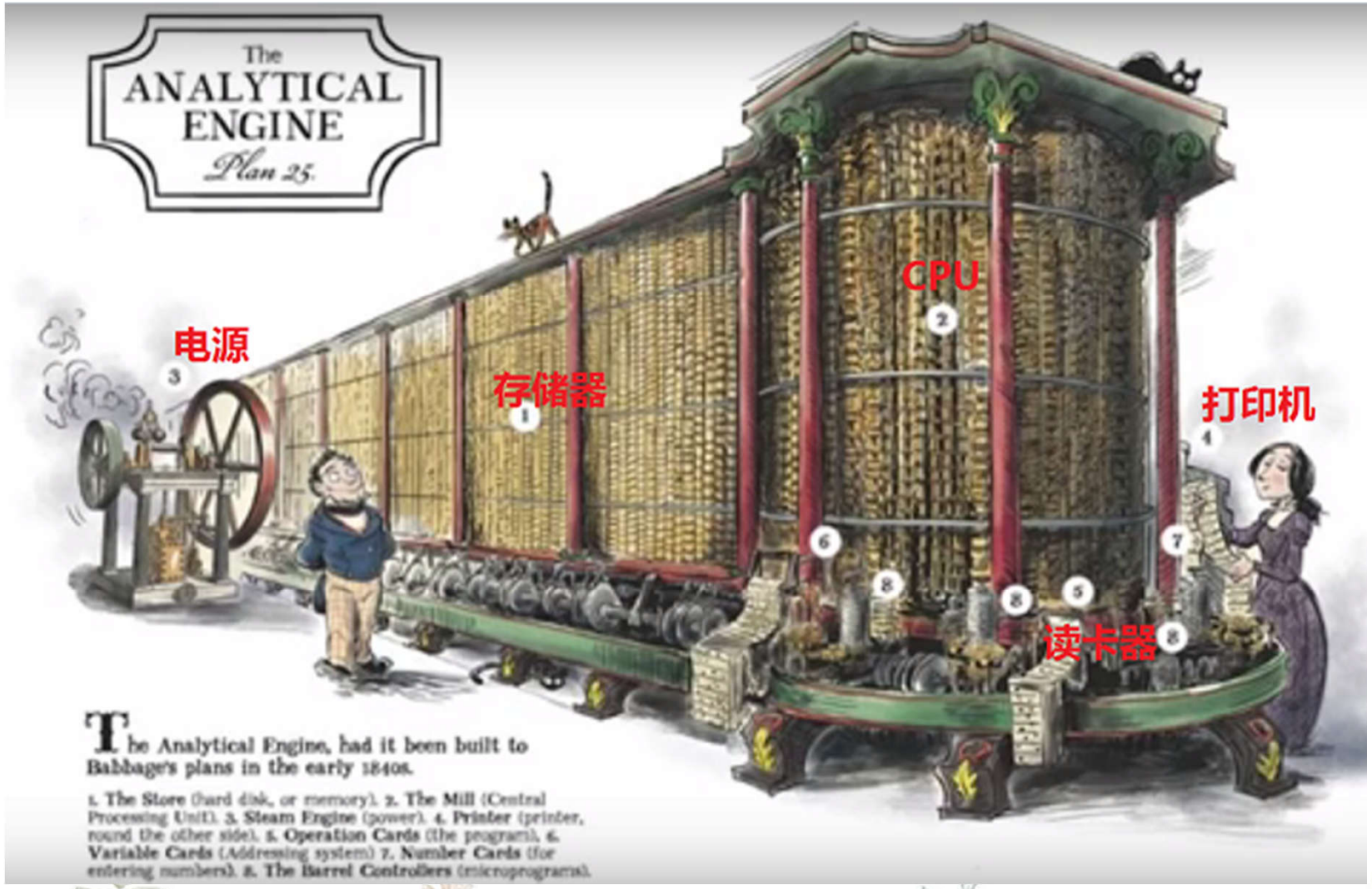
差分机





分析机

分析机

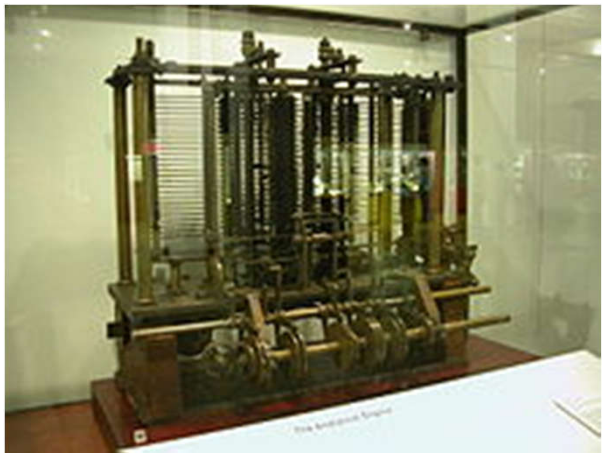

The
ANALYTICAL
ENGINE
Plan 25.
电源
存储器
CPU
打印机
读卡器
The Analytical Engine, had it been built to Babbage's plans in the early 1840s.
1. The Store (hard disk, or memory). 2. The Mill (Central Processing Unit). 3. Steam Engine (power). 4. Printer (printer, round the other side). 5. Operation Cards (the program). 6. Variable Cards (Addressing system) 7. Number Cards (for entering numbers). 8. The Barrel Controllers (microprograms).

百年来对软件最有影响的10人

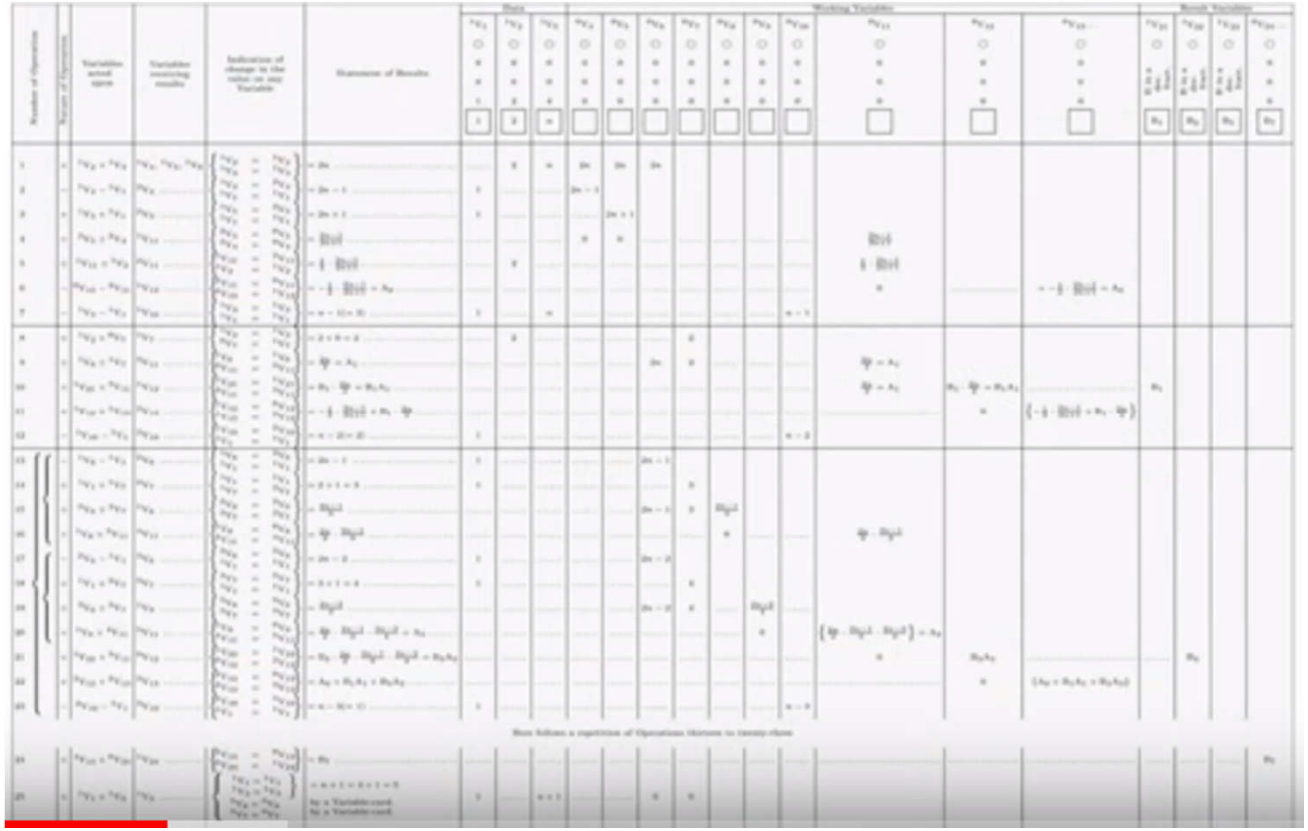
② Lady Ada Byron King (1815-1852)

“World's First Computer Programmer”

世界上第一个程序员



第一个程序



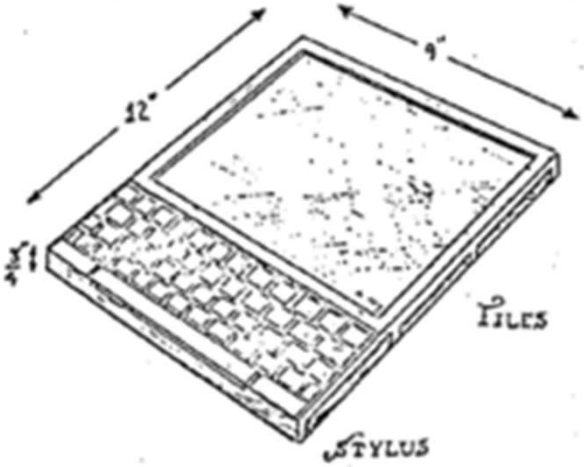
百年来对软件最有影响的10人

③ Alan Kay (1940-) 施乐公司

“one of the fathers of the idea of object-oriented programming”

面向对象编程之父

Dynabook, Smalltalk, GUI,… (1968~)

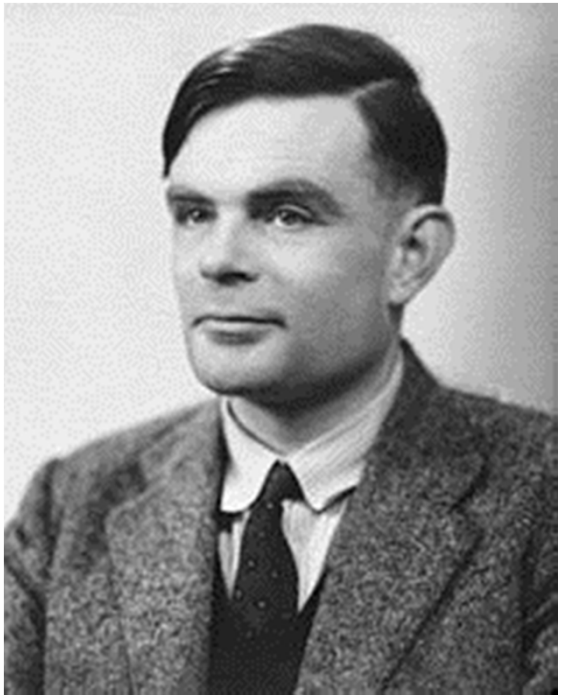


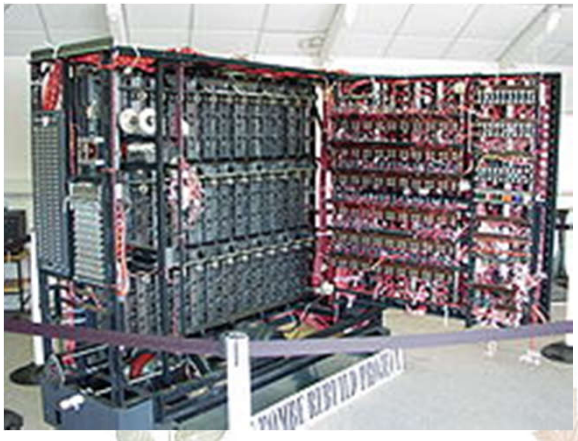
百年来对软件最有影响的10人

④ Alan Turing (1912-)

剑桥大学

“ father of computer science and artificial intelligence”





百年来对软件最有影响的10人

⑤ Capers Jones

“function point model of cost estimation”

⑥ Watts S. Humphrey

“father of Software quality”

CMM, PSP, TSP

百年来对软件最有影响的10人

⑦ Barry Boehm

“the Constructive Cost Model (COCOMO),

the Spiral Model of the software process”

⑧ Tim Berners-Lee

“father of the Web”

万维网的发明人

百年来对软件最有影响的10人

⑨ Grace Murray Hopper

“the first compiler”

⑩ Ed Yourdon

“structured analysis techniques ”

百年来对软件最有影响的10人

① Charles Babbage

② Lady Ada Byron King

③ Alan Kay　面向对象

④ Alan Turing

⑤ Capers Jones　功能点估算

⑥ Watts S. Humphrey　CMM/CMMI

⑦ Barry Boehm　COCOMO成本估算

⑧ Tim Berners-Lee

⑨ Grace Murray Hopper

⑩ Ed Yourdon　结构化分析

ACM Computing Classification System (January 1998)

- A. General Literature
- B. Hardware
- C. Computer Systems Organization
- D. Software
- E. Data
- F. Theory of Computation
- G. Mathematics of Computing
- H. Information Systems
- I. Computing Methodologies
- J. Computer Applications
- K. Computing Milieux

ACM Computing Classification System (January 1998)

- A. General Literature
- B. Hardware
- C. Computer Systems Organization
- **D. Software**
- E. Data
- F. Theory of Computation
- G. Mathematics of Computing
- H. Information Systems
- I. Computing Methodologies
- J. Computer Applications
- K. Computing Milieux

ACM Computing Classification System (January 1998)

- D. Software
  - D.0 General
  - D.1 Programming Techniques
  - D.2 Software Engineering
  - D.3 Programming languages
  - D.4 Operating systems
  - D.m MISCELLANEOUS

ACM Computing Classification System (January 1998)

- D. Software
  - D.0 General
  - D.1 Programming Techniques
  - **D.2 Software Engineering**
  - D.3 Programming languages
  - D.4 Operating systems
  - D.m MISCELLANEOUS

- D.2 Software Engineering
  - D.2.0 General
  - D.2.1 Requirements/Specifications
  - D.2.2 Design Tools and Techniques
  - D.2.3 Coding Tools and Techniques
  - D.2.4 Software/Program Verification
  - D.2.5 Testing and Debugging
  - D.2.6 Programming Environments
  - D.2.7 Distribution, Maintenance, and Enhancement
  - **D.2.8 Metrics**
  - D.2.9 Management
  - D.2.10 Design
  - D.2.11 Software Architectures
  - D.2.12 Interoperability
  - D.2.13 Reusable Software

ACM Computing Classification System (January 1998)

- D.2.8 Metrics
  - Complexity measures
  - Performance measures
  - Process metrics
  - Product metrics
  - Software science

软件度量背景简介

- 在计算机学科中的位置？
- 我们能用它来干什么？

场景1

单元测试中的一个问题

(1) 所测试的系统规模大，模块数目很多

(2) 时间非常紧，人手不够用

不可能测试所有模块，那么应该优先测试哪些模块？

建议：

(1) 优先测试“关键”模块

(2) 优先测试“最有可能包含bug”的模块

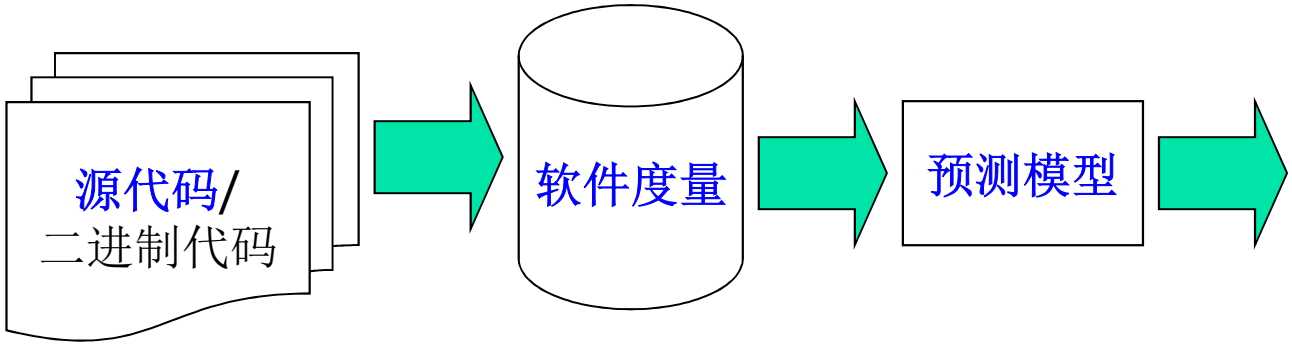
场景1

引发两个新的问题

(1) 如何识别系统中的“关键”模块？

(2) 如何识别系统中“最有可能包含bug”的模块？

基本思路：

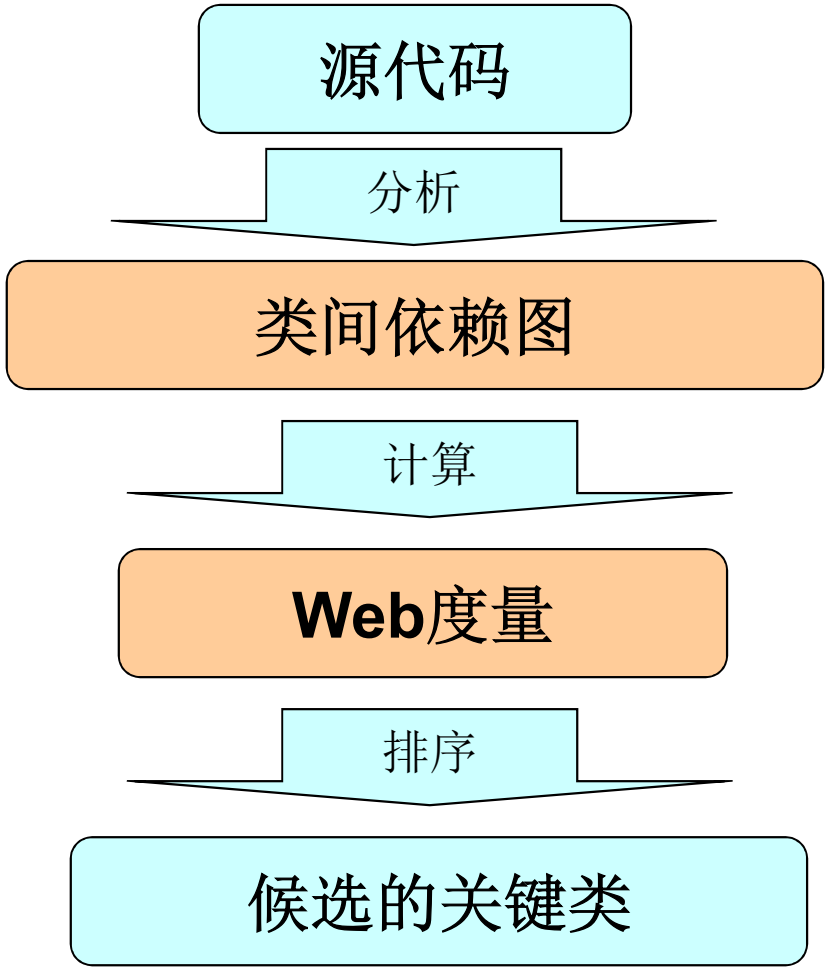


关键模块序列

缺陷模块序列

场景1

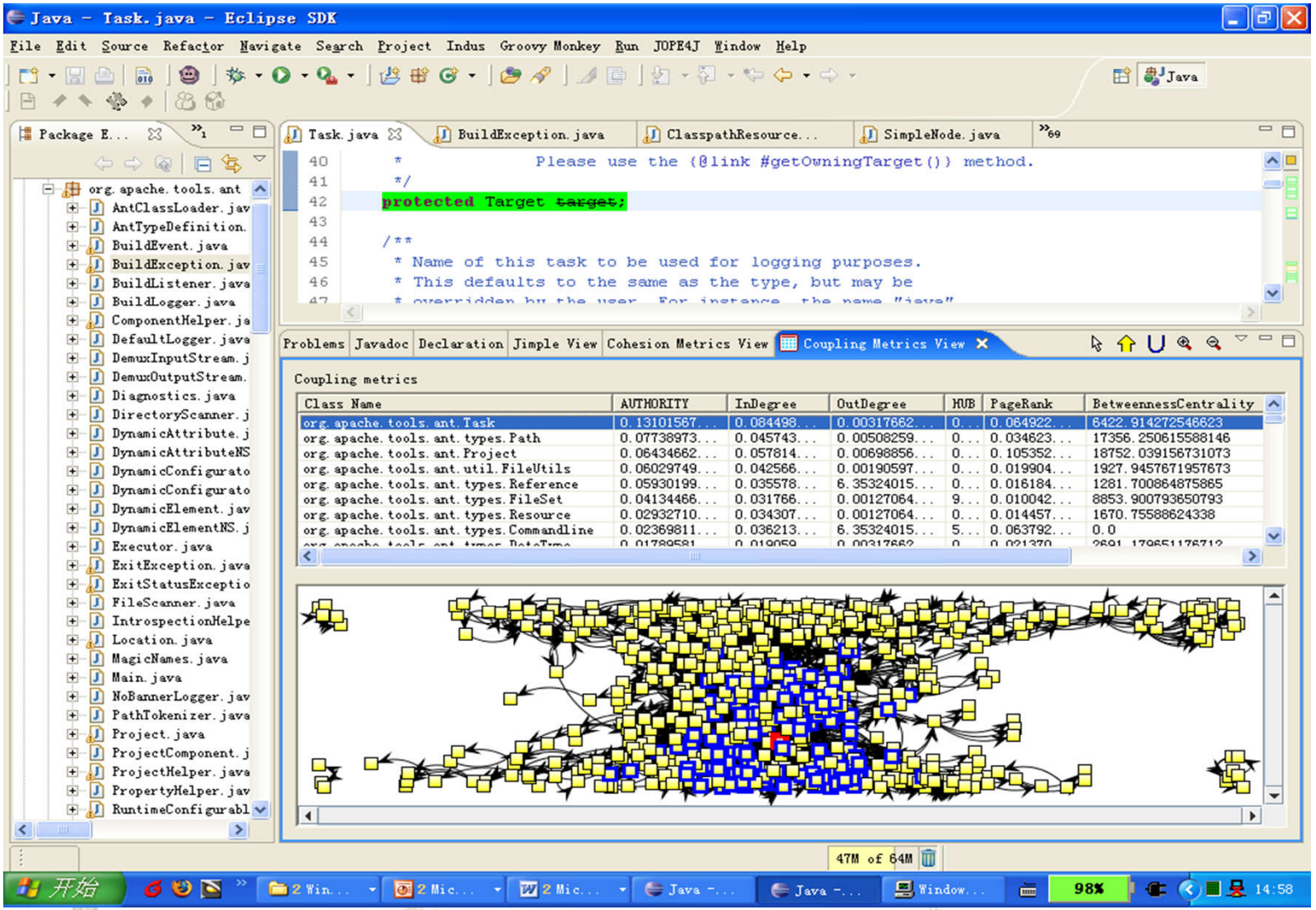
Step 1:



Step 2:

Step 3:

场景1


Java - Task.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Indus Groovy Monkey Run JOPE4J Window Help
Package E...
org.apache.tools.ant
AntClassLoader.jav
AntTypeDefinition.
BuildEvent.java
BuildException.jav
BuildListener.java
BuildLogger.java
ComponentHelper.ja
DefaultLogger.java
DemuxInputStream.j
DemuxOutputStream.
Diagnostics.java
DirectoryScanner.j
DynamicAttribute.j
DynamicAttributeNS
DynamicConfigurato
DynamicConfigurato
DynamicElement.jav
DynamicElementNS.j
Executor.java
ExitException.java
ExitStatusExceptio
FileScanner.java
IntrospectionHelpe
Location.java
MagicNames.java
Main.java
NoBannerLogger.jav
PathTokenizer.java
Project.java
ProjectComponent.j
ProjectHelper.java
PropertyHelper.jav
RuntimeConfigurabl
Task.java
BuildException.java
ClasspathResource...
SimpleNode.java
* Please use the {@link #getOwningTarget()} method.
*/
protected Target target;
/**
* Name of this task to be used for logging purposes.
* This defaults to the same as the type, but may be
Problems Javadoc Declaration Jimple View Cohesion Metrics View Coupling Metrics View
Coupling metrics
Class Name | AUTHORITY | InDegree | OutDegree | HUB | PageRank | BetweennessCentrality
org.apache.tools.ant.Task | 0.13101567... | 0.084498... | 0.00317662... | 0... | 0.064922... | 6422.914272546623
org.apache.tools.ant.types.Path | 0.07738973... | 0.045743... | 0.00508259... | 0... | 0.034623... | 17356.250615588146
org.apache.tools.ant.Project | 0.06434662... | 0.057814... | 0.00698856... | 0... | 0.105352... | 18752.039156731073
org.apache.tools.ant.util.FileUtils | 0.06029749... | 0.042566... | 0.00190597... | 0... | 0.019904... | 1927.9457671957673
org.apache.tools.ant.types.Reference | 0.05930199... | 0.035578... | 6.35324015... | 0... | 0.016184... | 1281.700864875865
org.apache.tools.ant.types.FileSet | 0.04134466... | 0.031766... | 0.00127064... | 9... | 0.010042... | 8853.900793650793
org.apache.tools.ant.types.Resource | 0.02932710... | 0.034307... | 0.00127064... | 0... | 0.014457... | 1670.75588624338
org.apache.tools.ant.types.Commandline | 0.02369811... | 0.036213... | 6.35324015... | 5... | 0.063792... | 0.0
47M of 64M
开始
Java -...
Java -...
Window...
98%
14:58

场景1

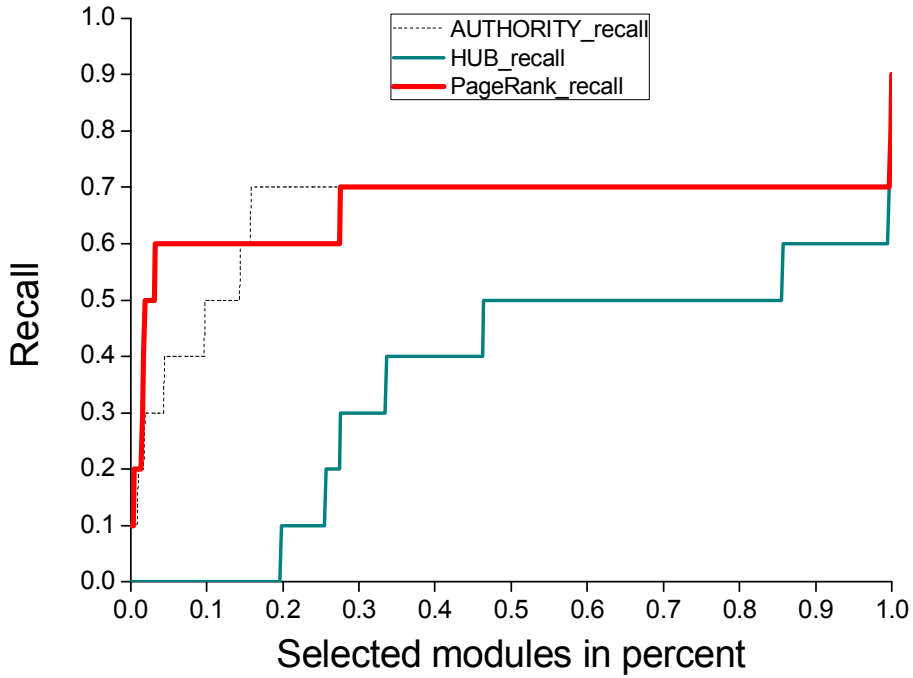
实验分析

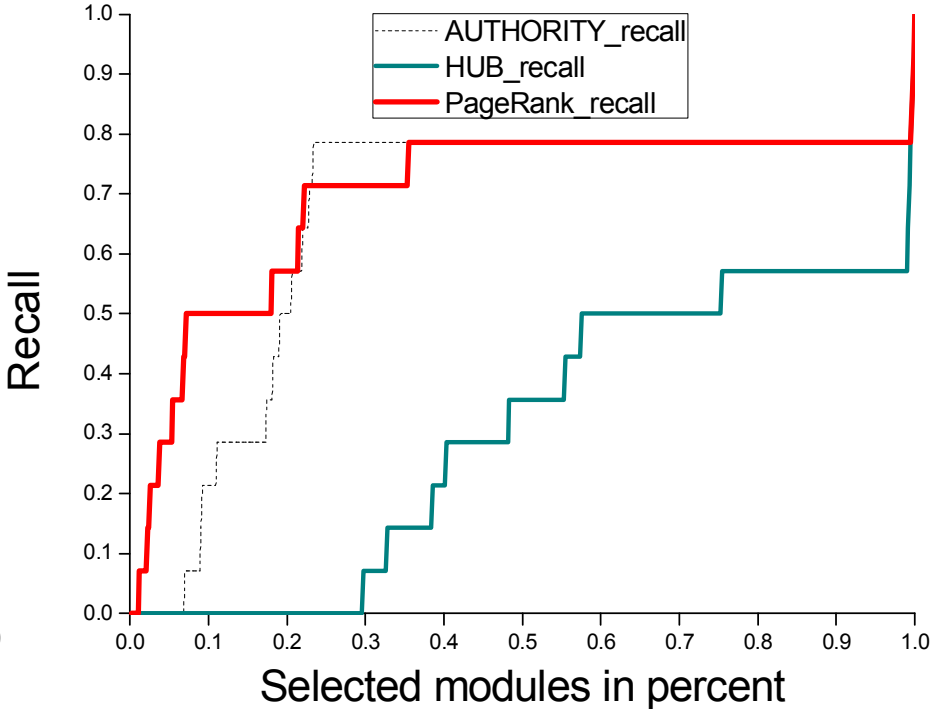
系统	类的数目	总代码行数	真实的关键类
Ant 1.6.1	664	76K	10
JMeter 2.0.1	706	73K	14

Identified from documents by Zaidman et al. (JSME 2008)

场景1

Recall





Ant 1.6.1

Jmeter 2.0.1

场景2

程序调试中的一个问题

给定一个有bug的程序和一批测试用例。当这些测试用例在程序上执行时，有些测试用例passed，有些测试用例failed。如何仅通过程序结构信息和测试用例的执行信息自动识别出有bug的语句？

mid() { int x,y,z,m;	3, 3, 5					
1: read(x, y, z);						
2: m = z;						
3: if (y<z)						
4: if (x<y)						
5: m = y;						
6: else if (x<z)						
7: m = y;						
8: else						
9: if (x>y)						
10: m = y;						
11: else if (x>z)						
12: m = x;						
13: print(m);						
}						

mid() { int x,y,z,m;	3, 3, 5					
1: read(x, y, z);	•					
2: m = z;						
3: if (y<z)						
4: if (x<y)						
5: m = y;						
6: else if (x<z)						
7: m = y;						
8: else						
9: if (x>y)						
10: m = y;						
11: else if (x>z)						
12: m = x;						
13: print(m);						
}						

mid() { int x,y,z,m;	3, 3, 5					
1: read(x, y, z);	•					
2: m = z;	•					
3: if (y<z)						
4: if (x<y)						
5: m = y;						
6: else if (x<z)						
7: m = y;						
8: else						
9: if (x>y)						
10: m = y;						
11: else if (x>z)						
12: m = x;						
13: print(m);						
}						

mid() { int x,y,z,m;	3, 3, 5					
1: read(x, y, z);	•					
2: m = z;	•					
3: if (y<z)	•					
4: if (x<y)						
5: m = y;						
6: else if (x<z)						
7: m = y;						
8: else						
9: if (x>y)						
10: m = y;						
11: else if (x>z)						
12: m = x;						
13: print(m);						
}						

mid() { int x,y,z,m;	3, 3, 5					
1: read(x, y, z);	•					
2: m = z;	•					
3: if (y<z)	•					
4: if (x<y)	•					
5: m = y;						
6: else if (x<z)						
7: m = y;						
8: else						
9: if (x>y)						
10: m = y;						
11: else if (x>z)						
12: m = x;						
13: print(m);						
}						

mid() { int x,y,z,m;	3, 3, 5					
1: read(x, y, z);	•					
2: m = z;	•					
3: if (y<z)	•					
4: if (x<y)	•					
5: m = y;						
6: else if (x<z)	•					
7: m = y;						
8: else						
9: if (x>y)						
10: m = y;						
11: else if (x>z)						
12: m = x;						
13: print(m);						
}						

mid() { int x,y,z,m;	3, 3, 5					
1: read(x, y, z);	•					
2: m = z;	•					
3: if (y<z)	•					
4: if (x<y)	•					
5: m = y;						
6: else if (x<z)	•					
7: m = y;	•					
8: else						
9: if (x>y)						
10: m = y;						
11: else if (x>z)						
12: m = x;						
13: print(m);						
}						

mid() { int x,y,z,m;	3, 3, 5					
1: read(x, y, z);	•					
2: m = z;	•					
3: if (y<z)	•					
4: if (x<y)	•					
5: m = y;						
6: else if (x<z)	•					
7: m = y;	•					
8: else						
9: if (x>y)						
10: m = y;						
11: else if (x>z)						
12: m = x;						
13: print(m);	•(3)					
}						

mid() { int x,y,z,m;	3, 3, 5					
1: read(x, y, z);	•					
2: m = z;	•					
3: if (y<z)	•					
4: if (x<y)	•					
5: m = y;						
6: else if (x<z)	•					
7: m = y;	•					
8: else						
9: if (x>y)						
10: m = y;						
11: else if (x>z)						
12: m = x;						
13: print(m);	•(3)					
}	P					

mid() { int x,y,z,m;	3, 3, 5	1, 2, 3	3, 2, 1	5, 5, 5	5, 3, 4	2, 1, 3
1: read(x, y, z);	•	•	•	•	•	•
2: m = z;	•	•	•	•	•	•
3: if (y<z)	•	•	•	•	•	•
4: if (x<y)	•	•			•	•
5: m = y;		•				
6: else if (x<z)	•				•	•
7: m = y;	•					•
8: else			•	•		
9: if (x>y)			•	•		
10: m = y;			•			
11: else if (x>z)				•		
12: m = x;						
13: print(m);	•(3)	•(2)	•(2)	•(5)	•(4)	•(1)
}	P	P	P	P	P	F

场景2

基本方法

- 构造执行频谱矩阵
  - 行: 语句
  - 列: 测试用例
  - 值: 测试用例执行时是否覆盖到了该语句
- 利用各种方法计算语句的可疑程度
  - 被越多的passing测试用例执行，越不可能有bug
  - 被越多的failing测试用例执行，越有可能包含bug
  - …

场景2

语句可疑度度量示例

$$suspiciousness_S(s) = \frac{failed(s)}{passed(s) + failed(s)}$$

$$suspiciousness_J(s) = \frac{failed(s)}{totalfailed + passed(s)}$$

$$suspiciousness_O(s) = \frac{failed(s)}{\sqrt{totalfailed * (failed(s) + passed(s))}}$$

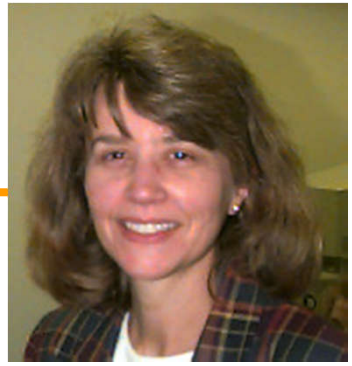
$$suspiciousness_T(s) = \frac{\%\text{failed}(s)}{\%\text{failed}(s) + \%\text{passed}(s)}$$

场景2

语句可疑度度量示例

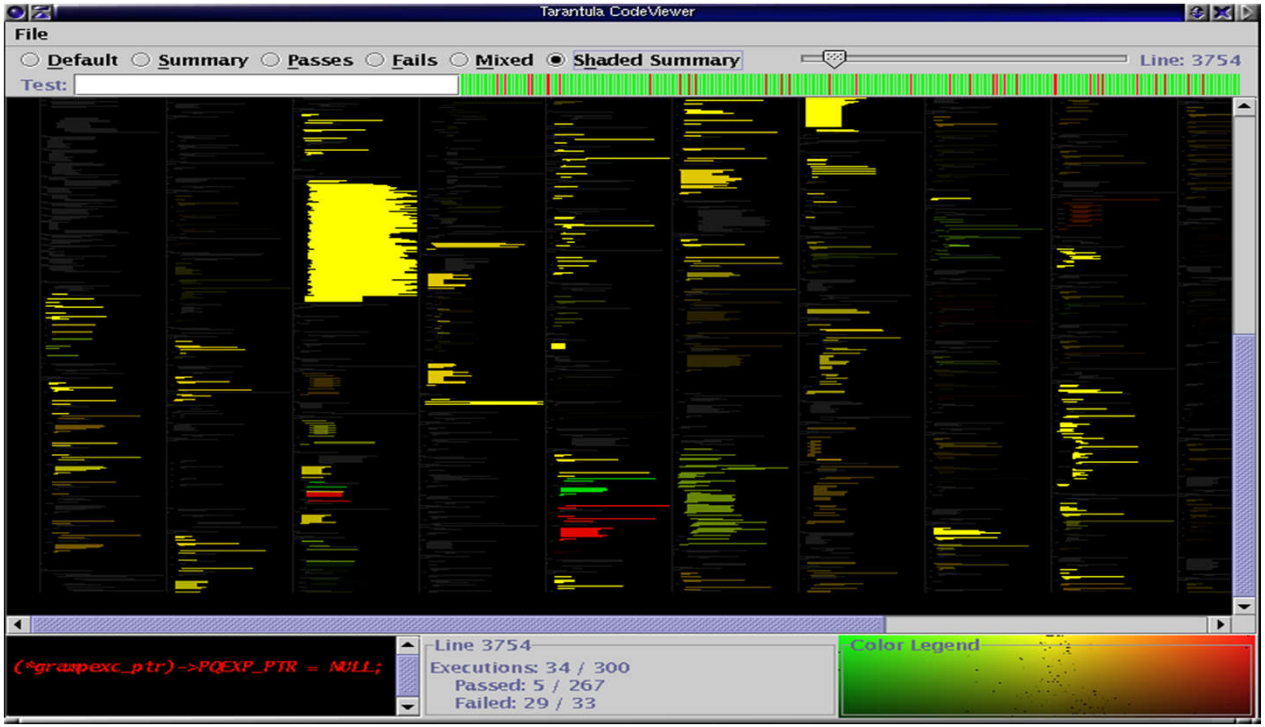
	Test Cases								Tarantula			SBI		Jaccard		Ochiai	
	t1	t2	t3	t4	t5	t6	t7	t8	suspiciousness $_T$	confidence	rank	suspiciousness $_S$	rank	suspiciousness $_J$	rank	suspiciousness $_O$	rank
`mid() {` `int x,y,z,m;`	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	7,5,4	2,1,3	4,3,5									
1: `read("Enter 3 numbers:",x,y,z);`	●	●	●	●	●	●	●	●	0.5	1.0	7	0.25	7	0.25	7	0.5	7
2: `m = z;`	●	●	●	●	●	●	●	●	0.5	1.0	7	0.25	7	0.25	7	0.5	7
3: `if (y<z)`	●	●	●	●	●	●	●	●	0.5	1.0	7	0.25	7	0.25	7	0.5	7
4: `if (x<y)`	●	●			●		●	●	0.67	1.0	3	0.4	3	0.4	3	0.63	3
5: `m = y;`		●							0.0	0.17	12	0.0	13	0.0	13	0.0	13
6: `else if (x<z)`	●				●		●	●	0.75	1.0	2	0.5	2	0.5	2	0.71	2
7: `m = y;  // *** bug ***`	●						●	●	0.86	1.0	1	0.67	1	0.67	1	0.82	1
8: `else`			●	●		●			0.0	0.5	9	0.0	13	0.0	13	0.0	13
9: `if (x>y)`			●	●		●			0.0	0.5	9	0.0	13	0.0	13	0.0	13
10: `m = y;`			●			●			0.0	0.33	10	0.0	13	0.0	13	0.0	13
11: `else if (x>z)`				●					0.0	0.17	12	0.0	13	0.0	13	0.0	13
12: `m = x;`									0.0	0.0	13	0.0	13	0.0	13	0.0	13
13: `print("Middle number is:",m);`	●	●	●	●	●	●	●	●	0.5	1.0	7	0.25	7	0.25	7	0.5	7
`}` Pass/Fail Status	P	P	P	P	P	P	F	F									

场景2



Mary Jean Harrold

工具支持



场景2

mid() { int x,y,z,m;	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3
1: read(x,y,z);	●	●	●	●	●	●
2: m=z;	●	●	●	●	●	●
3: if(y<z)	●	●	●	●	●	●
4: if(x<y)		●				
5: m=y;		●				
6: elseif(x<z)	●				●	●
7: m=y;	●					●
8: else	●		●	●		
9: if(x>y)			●			
10: m=y;			●			
11: elseif(x>z)						
12: m=x;						
13: print(m);	●	●	●	●	●	●
}	P	P	P	P	P	F

场景2

实验结果

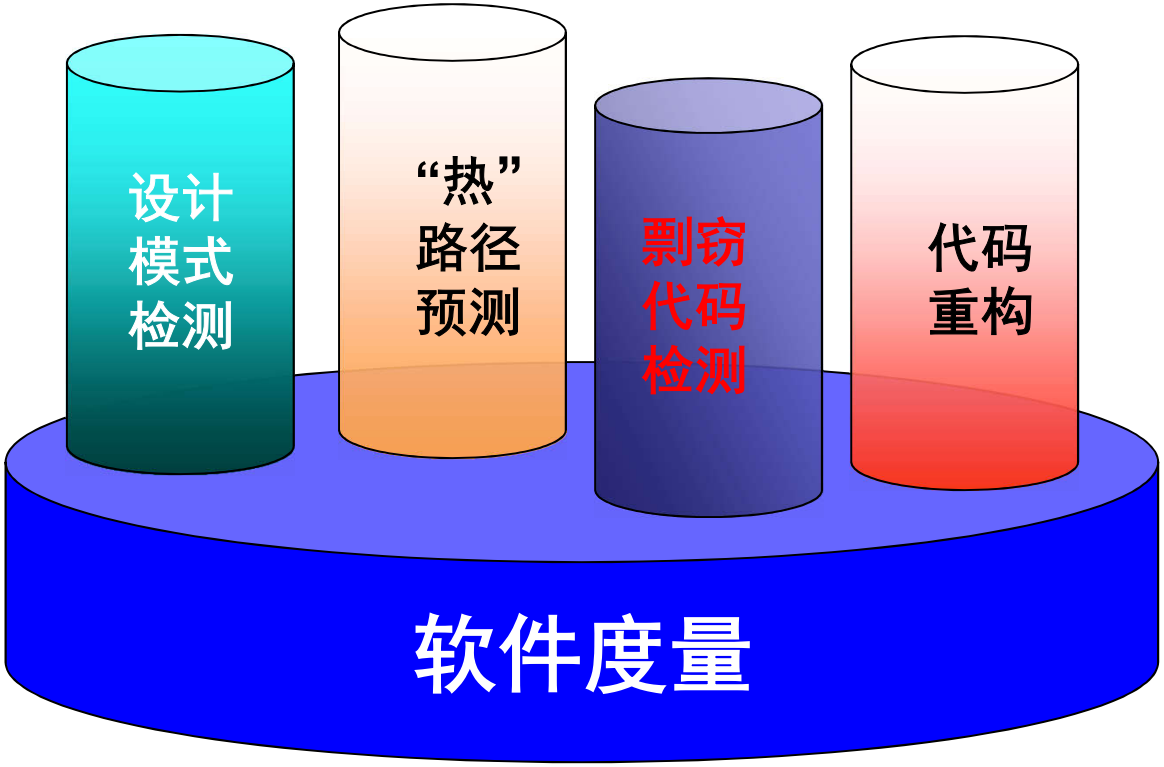
R. Santelices et al.
ICSE 2009

subject	LOC	tests	faults
Tcas	131	1608	10
Tot_info	283	1052	10
Schedule1	290	2650	9
Schedule2	317	2710	7
Print_tokens1	478	4130	5
Print_tokens2	410	4115	10
NanoXML v1	3497	214	7
NanoXML v2	4009	214	7
NanoXML v3	4608	216	8
NanoXML v5	4782	216	7
XML-security v1	21613	92	7
XML-security v2	22318	94	7
XML-security v3	19895	84	2
JABA	37966	677	11
TOTAL FAULTS			107

For all 107 faults:

measure	statement	branch	du-pair
average cost	11.49%	10.24%	9.02%
standard dev.	*16.25%*	*15.40%*	*12.04%*

其他场景



Section 1

Overview

- Allocation of Marks
- Teaching Assistants
- Course Goals
- Position
- Related universities

Allocation of Marks

- 平时到课+练习：70%
- 大作业：　30%

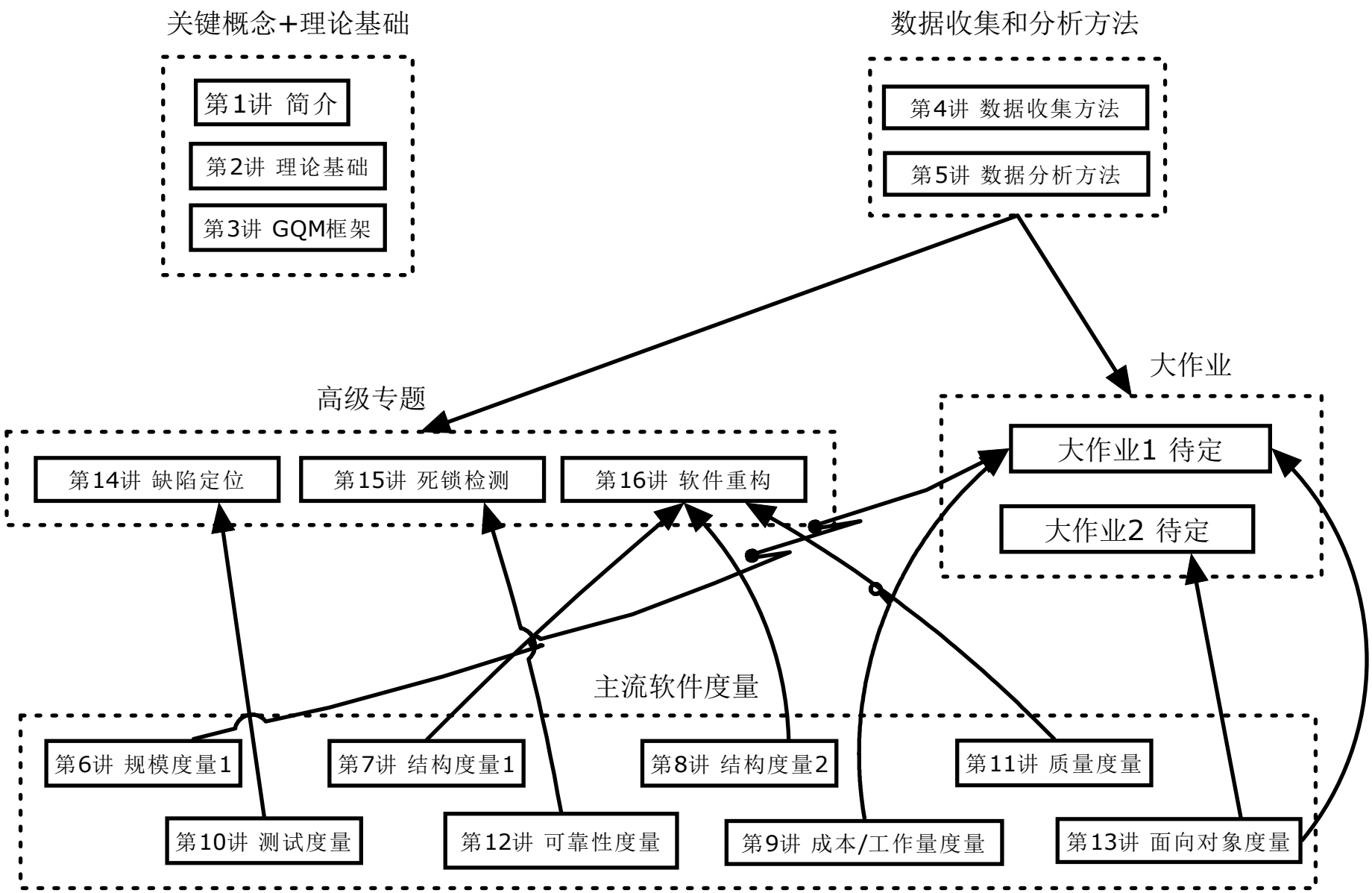
Reference Books

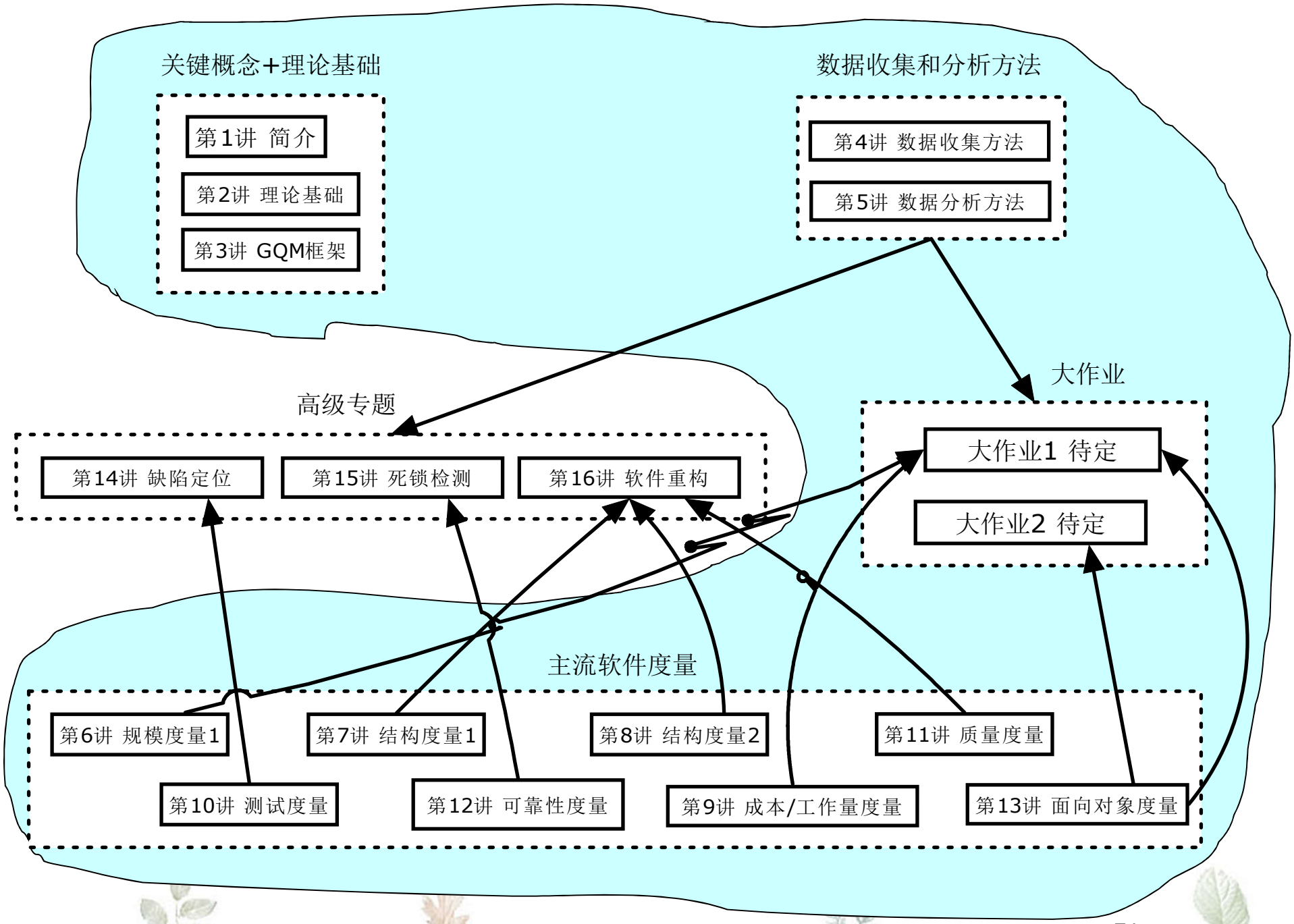
- ***Software Metrics: A Rigorous and Practical Approach***, (3rd ed.), N.E. Fenton, J. Bieman, CRC Press, 2015, (Required)

- ***Metrics and Models in Software Quality Engineering*** (344 pages), Stephen H. Kan, Addison-Wesley, 1995, ISBN: 0201633396.
- ***Software Metrics: Measurement for Software Process Improvement***, B.A. Kitchenham, Blackwell Pub, 1996,
- ***Software Metrics: Establishing a Company-wide Program*** (288 pages), R.B. Grady, D.L. Caswell, Prentice Hall, 1998,
- ***Applied Software Measurement: Assuring Productivity and Quality***, C. Jones, McGraw-Hill, 1996.

Reference Books

- ***Basics of Software Engineering Experimentation.*** Natalia Juristo, Ana M. Moreno, Kluwer Academic Publishers, 2001, (very good)


关键概念+理论基础
第1讲 简介
第2讲 理论基础
第3讲 GQM框架
数据收集和分析方法
第4讲 数据收集方法
第5讲 数据分析方法
大作业
大作业1 待定
大作业2 待定
高级专题
第14讲 缺陷定位
第15讲 死锁检测
第16讲 软件重构
主流软件度量
第6讲 规模度量1
第7讲 结构度量1
第8讲 结构度量2
第11讲 质量度量
第10讲 测试度量
第12讲 可靠性度量
第9讲 成本/工作量度量
第13讲 面向对象度量


关键概念+理论基础
第1讲 简介
第2讲 理论基础
第3讲 GQM框架
数据收集和分析方法
第4讲 数据收集方法
第5讲 数据分析方法
大作业
大作业1 待定
大作业2 待定
高级专题
第14讲 缺陷定位
第15讲 死锁检测
第16讲 软件重构
主流软件度量
第6讲 规模度量1
第7讲 结构度量1
第8讲 结构度量2
第11讲 质量度量
第10讲 测试度量
第12讲 可靠性度量
第9讲 成本/工作量度量
第13讲 面向对象度量

重要信息

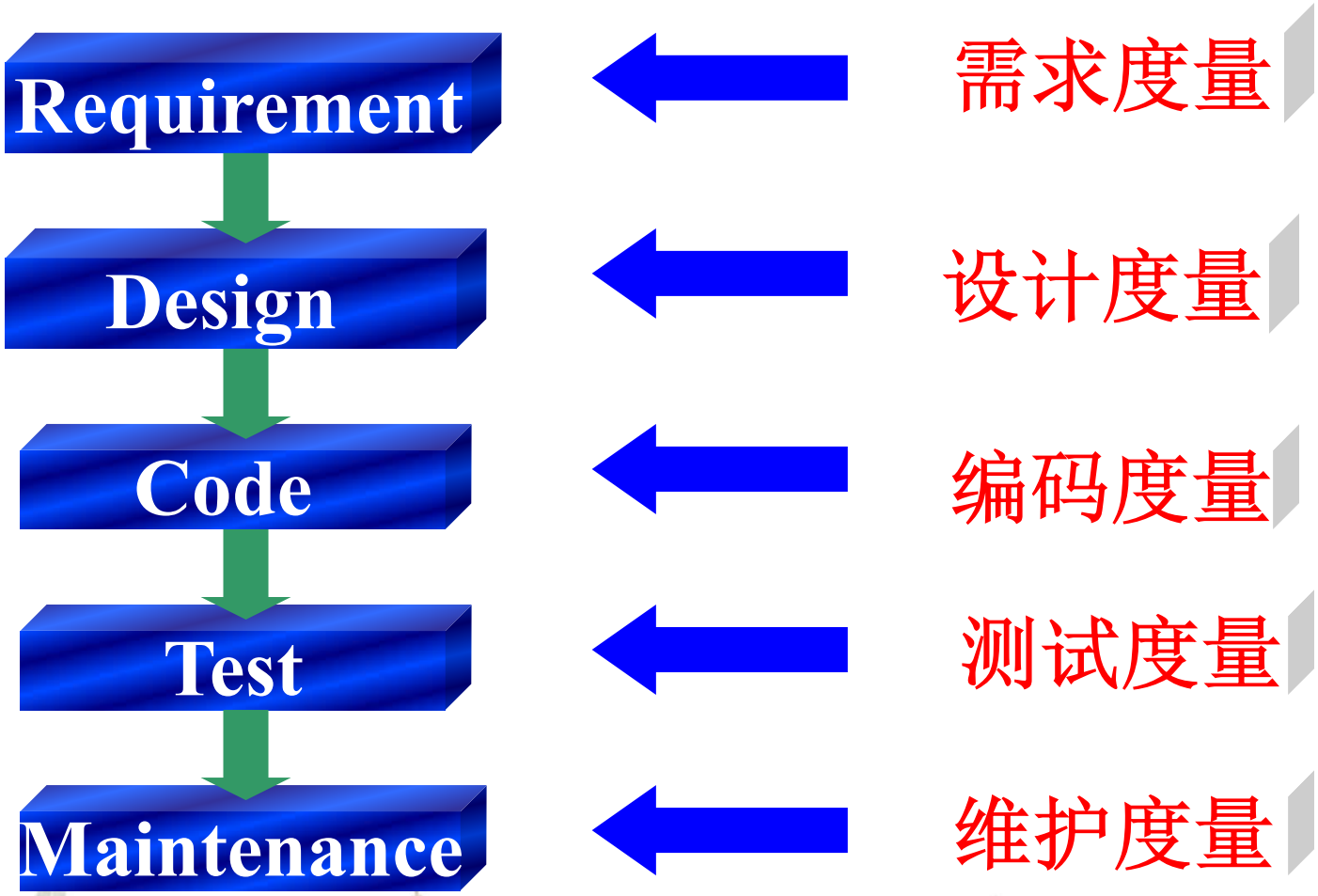
- 课程公共邮箱：课件下载+作业下载+作业提交
  用户名:cs2017metrics@21cn.com
  密码:xianlinyic302

- 我的联系方式
  电话：89682450
  办公室：计算机系楼703
  邮箱:zhouyuming@nju.edu.cn

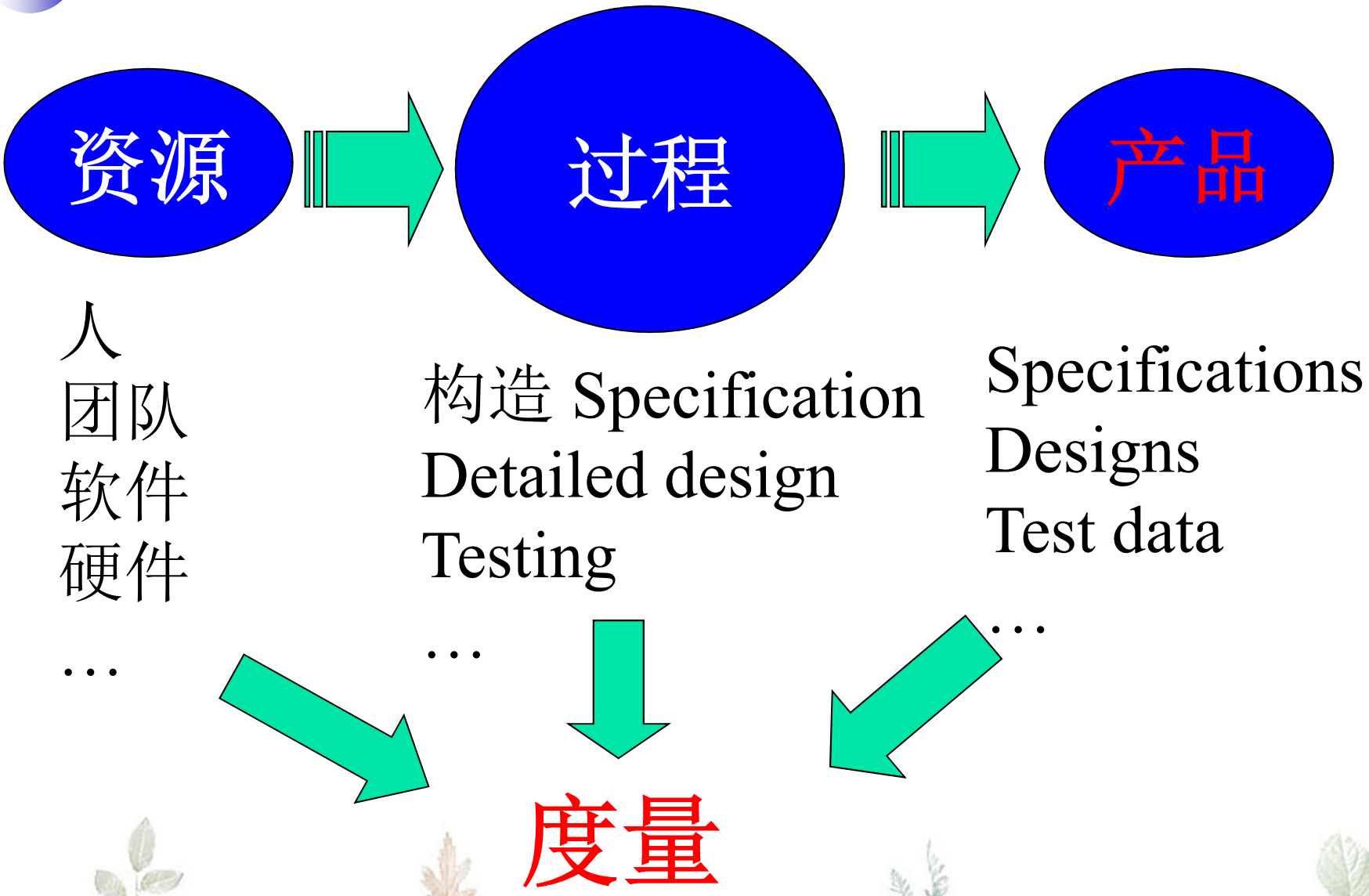
Course Goals

- 介绍软件度量的理论基础
- 分析当前各种主要的度量，掌握软件度量的基本方法
- 学会运用度量开发框架，能够根据实际需要开发新的度量
- 熟悉典型度量工具的使用

Position


Requirement
Design
Code
Test
Maintenance
需求度量
设计度量
编码度量
测试度量
维护度量

Course Position (cont'd)



Section 2

Outline of The Course

- Measurement theory
- Measurement in Software Engineering
- Measurement & management

1: Measurement Theory

- The basics of measurement
- Framework for software measurement
- Empirical investigation

2: Measurement in Software Engineering

- Software metrics data collection
- Analyzing software measurement data
- Measuring internal product attributes: size
- Measuring internal product attributes: structure
- Measuring external product attributes
- Measuring software reliability
- Making process predictions

3: Measurement & Management

- Planning a measurement program
- Resource measurement: productivity, teams, and tools

Section 3

What is Measurement?

Measurement

abstractions used to reflect perceptions of the real world

- **Measurement** is [illegible] **numbers** or **symbols** are assigned to **attributes** of **entities** in the real world in such a way as to describe them according to defined rules

feature or property of an entity

an object or an event in the real world

$$Entity = \left\{ \begin{array}{ll} a[illegible] & , value_{12}, \ldots) \\ a[illegible] & , value_{22}, \ldots) \\ \ldots & \ldots \\ attribute_n & (value_{n1}, value_{n2}, \ldots) \end{array} \right\}$$

Measurement: Definition /2

- **Metrics** are standards (i.e., commonly accepted scales) that define measurable attributes of entities, their units and their scopes

Measurement: Types

- Measurements are needed as:
  - **Descriptors** of entities already in existence.
  - **Predictors** to estimate properties of entities yet to be designed or implemented.

Measurement : How to

- In order to make entities measurable:
  - What **entities** should be selected?
  - What **attributes** should be selected?
  - What **values** should be assigned to the attributes?
  - What shall be the **rules** ascribed to the attributes and their entities?
- Note: assigned values and/or ascribed rules can be either **quantitative** or **qualitative**

Example1: entities/attributes

- Clinton's heights

- T-shirt's size

Section 4

What are Software Metrics?

Concerns & FAQ

- **for Project Managers**
  - Is the project within the progress?
  - Whether the budget will be exceeded?
  - When the final product is ready for delivery?
- **for Software Developers**
  - Whether the requirements are consistent and complete?
  - Whether the design is of high quality?
  - Does the code still contain defects?
  - How about the software maintainability, reliability, etc.?

Concerns & FAQ (cont'd)

- **for Customers**
  - Whether the delivered product meets our needs defined in requirements?
- **for Maintainers**
  - upgraded or re-developed?

What are Software Metrics?

- Software metrics are metrics that are used to quantify software products, software development resources, and/or the software development process
- This includes items which are directly measurable, such as *lines of code*, as well as items which are calculated from measurements, such as *software quality*

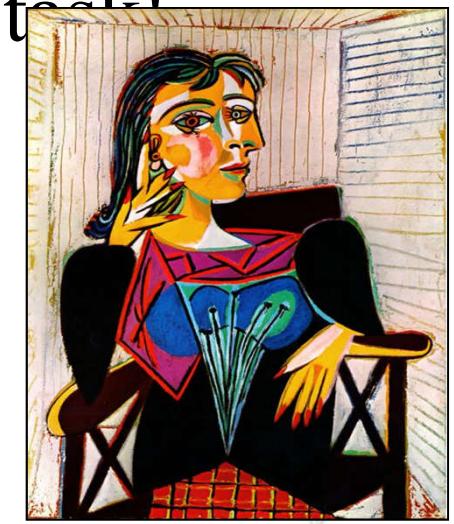
Why Use Metrics?

"You cannot control what you cannot measure" (Demacro, 1982)

"You can neither predict nor control what you cannot measure" (Norman 1992)

Measurement in SE

- Measurement in SE is selecting, measuring and putting together many different attributes of the software, and adding our subjective interpretations in order to get a whole picture of the software
- This is not a trivial task!



Various Types of Metrics

- Project Performance Metrics
- Quality Metrics
- Productivity Metrics
- Business Metric
- User Service Metrics
- …

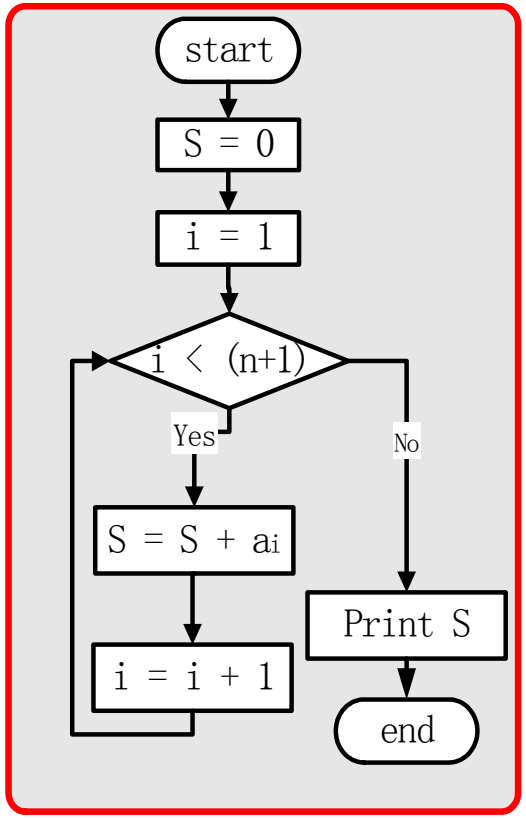
Example

圈复杂性 $v(G) = e - n + 2p$

$= 8 - 8 + 2 \times 1$

$= 2$

```
void sum()
{
    S=0;
    i=1;
    while(i<n+1)
    {
        S=S+a[i];
        i=i+1;
    }
  printf("S
  is %d", s);
}
```



Section 5

Scope of software engineering metrics: a chronological review

Measurement: Definition /1

- **Measurement** is the process by which *numbers* or *symbols* are assigned to *attributes* of *entities* in the real world in such a way as to ascribe them according to *defined rules*

$$entity = \left\{ \begin{array}{ll} attribute_1 & (value_{11}, value_{12}, \ldots) \\ attribute_2 & (value_{21}, value_{22}, \ldots) \\ \ldots & \ldots \\ attribute_n & (value_{n1}, value_{n2}, \ldots) \end{array} \right\}$$

Measurement: Definition /2

- **Metrics** are standards (i.e., commonly accepted scales) that define measurable attributes of entities, their units and their scopes

- **Software metrics** are measures that are used to quantify:
  - Software product
  - Software development resources
  - Software development process

Software Metrics Challenges

- Physical metrics:

entity	attribute	unit	value
Human	Height	cm	178

- Non-physical metrics:

entity	attribute	unit	value
Human	IQ	IQ index	89

- SE metrics are mostly non-physical

  Reliability, maturity, portability, flexibility, maintainability, interoperability, …

Misleading Metrics!

- Fact (1): **Knowledge is power**
- Fact (2): **Time is money**
- Relation (rule) : **power = work / time**

Substituting "power" and "time"

Knowledge = work / money

- As knowledge approaches zero money approaches infinity regardless of the amount of work done!
- Conclusion:

The less you know, the more you make!

Scope of Software Metrics

- Cost and effort estimation
- Productivity measures and models
- Data collection
- Quality models and measures
- Reliability models
- Performance evaluation and models
- Structural and complexity metrics
- Capability maturity assessment
- Management by metrics
- Evaluation of methods and tools

Scope of Software Metrics /1

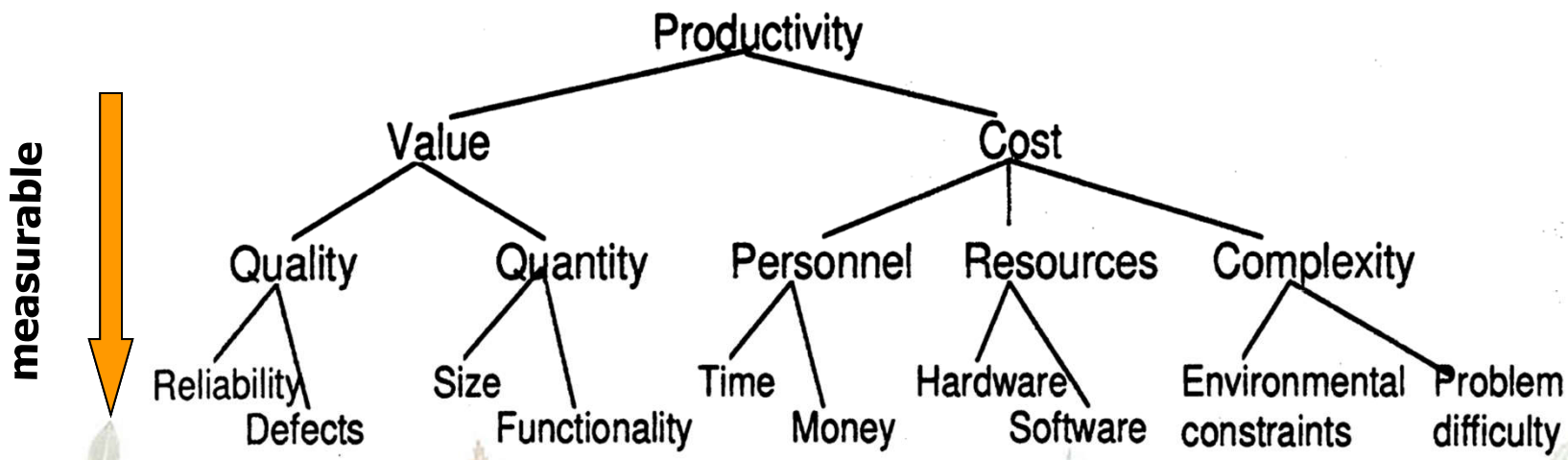
Cost and effort estimation

- Software cost estimation is the process of predicting the amount of effort required to build a software system
- Estimates for project cost and time requirements is derived during the planning stage of a project
- Models used to estimate cost can be categorized as either cost models (e.g., Constructive Cost Model COCOMO) or constraint models (e.g., SLIM)
- Experience is often the only guide used to derive these estimates, but it may be insufficient if the project breaks new ground
- Many models are available as automated tools

Scope of Software Metrics /2

Productivity models and measures

- **Definition**: The rate of output per unit of input
  - Productivity = size / effort
    Productivity = LOC / person-month
- Productivity model based on decomposition to measurable attributes:



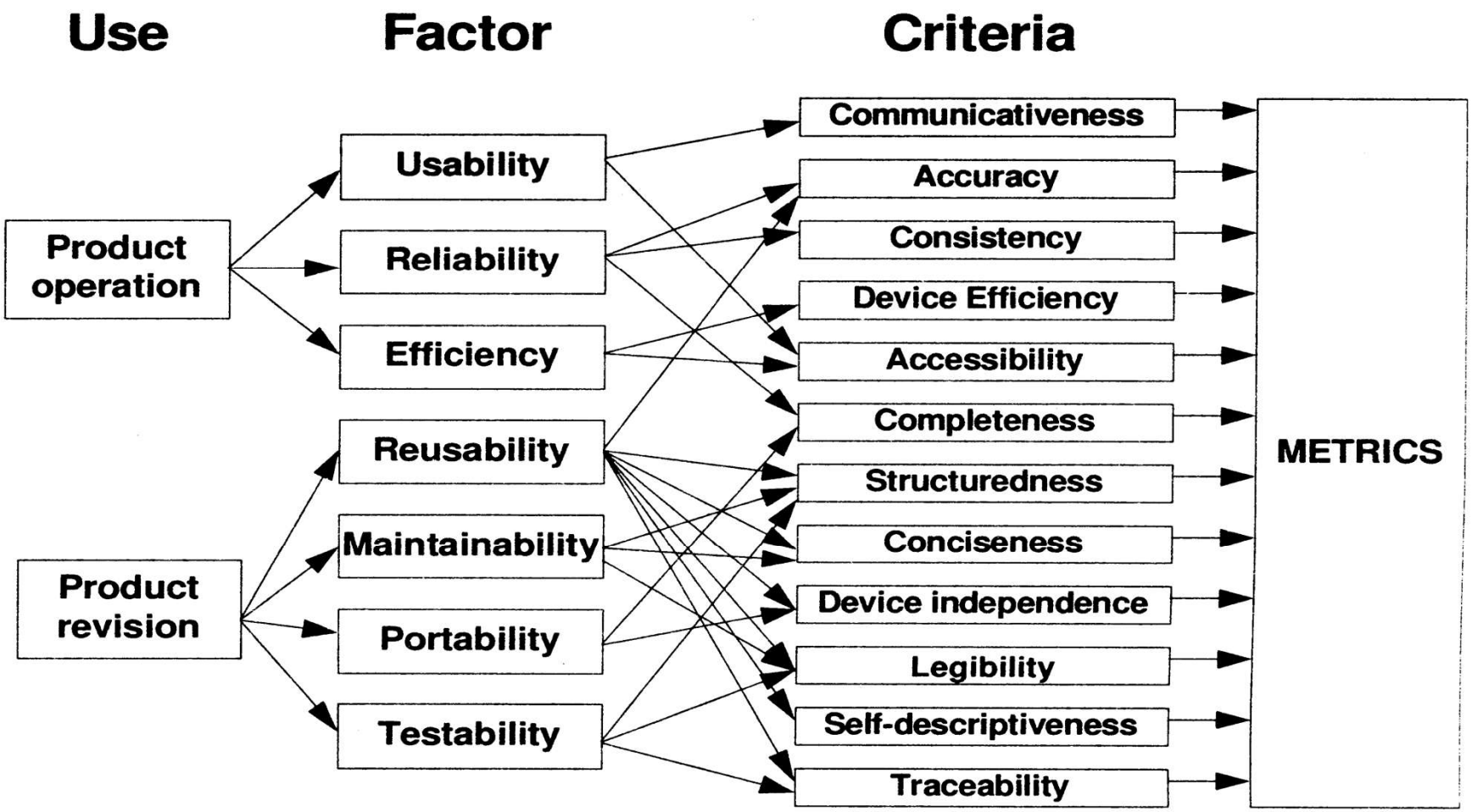
Scope of Software Metrics /3

Data collection

- Very critical and very hard step
  - What data should be collected?
  - How it should be collected?
  - Is collected data reproducible?
- Example: failure data collection
  1) Time of failure
  2) Time interval between failures
  3) Cumulative failure up to a given time
  4) Failures experienced in a time interval

Scope of Software Metrics /4

Quality models and measures



Scope of Software Metrics /5

Reliability models

- Plot the change of failure intensity (λ) against time
- Many models are proposed. The most famous ones are **basic exponential model** and **logarithmic Poisson model**
- The basic exponential model assumes finite failures in infinite time; the logarithmic Poisson model assumes infinite failures
- Automated tools such as CASRE are available

Scope of Software Metrics /6

Performance evaluation and models

- Using externally observable performance characteristics such as response time and completion rate (Ferrari 1978~)
- Efficiency of algorithms (Garey 1979~)

Scope of Software Metrics /7

Structural and complexity metrics

- Control-flow structure
- Data-flow structure
- Data structure
- Information flow attributes

- Complexity metrics (1979~)
  - Cyclomatic complexity (McCabe 1989) defining number of independent paths in execution of a program

Scope of Software Metrics /8

Metrics for Management

- Metrics for project control (1980~)
  - Metrics related to specification quality
  - Metrics for the design model
  - Metrics for documentation
  - Checking and testing metrics
  - Resource metrics
  - Change metrics

Scope of Software Metrics /9

Evaluation of methods and tools

- Efficiency of methods (1991~)
- Efficiency and reliability of tools
- Certification test of acquired tools and components

Scope of Software Metrics /10

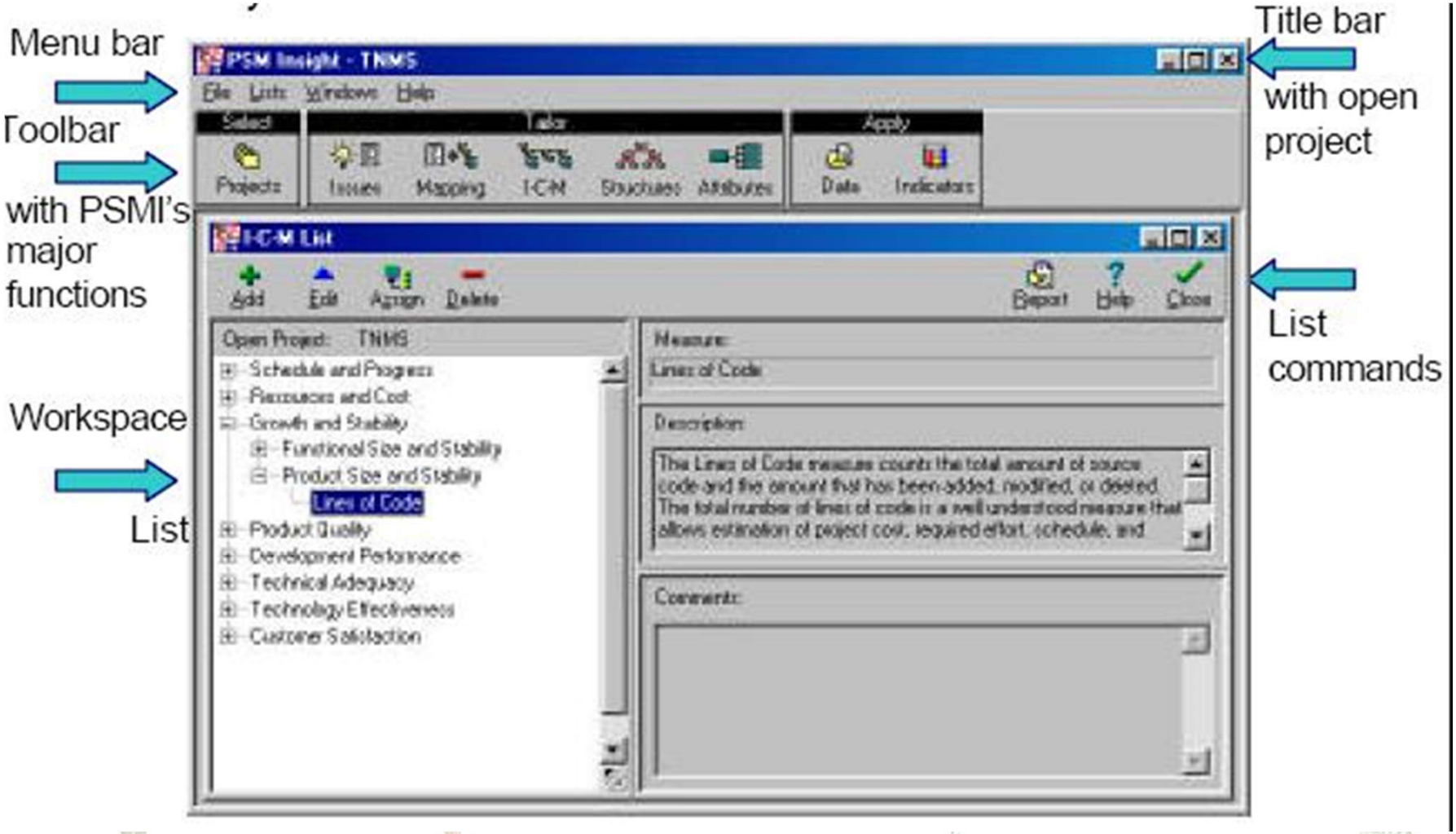
Capability maturity assessment

- US Software Engineering Institute (SEI) model (1989): Grading using five-level scale.
- ISO 9001: Quality systems: models for quality assurance in design/development, production, installation and servicing (1991)
- ISO 9000-3: Guidelines for application of ISO 9001 to the development, supply and maintenance of software (1991)

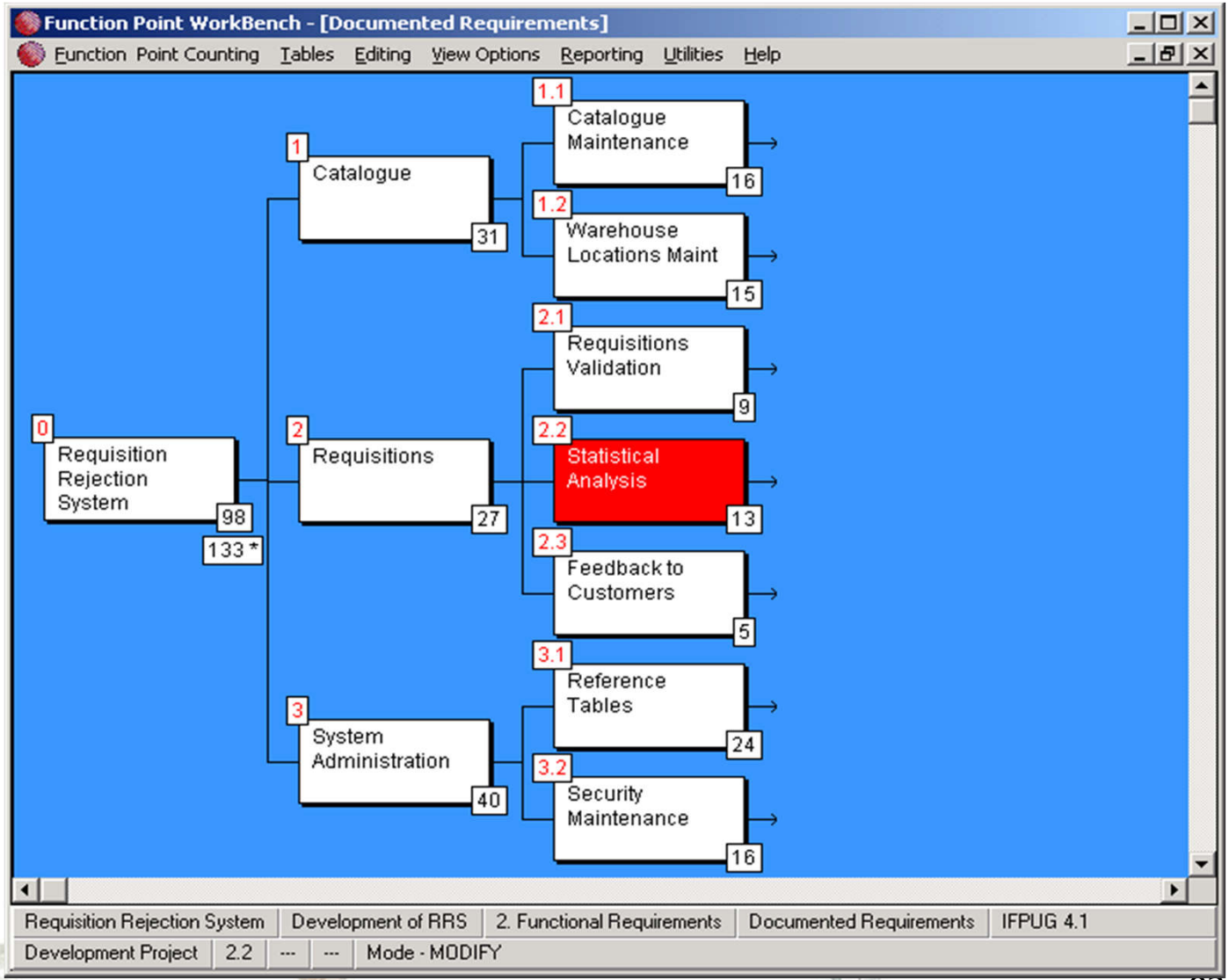
Who benefits from Measurement

- **Managers**
  - What does each process cost?
  - How productive is the staff?
  - How good is the code being developed?
  - Will the user be satisfied with the product?
  - How can we improve?
- **Software Developers**
  - Are the requirements testable?
  - Have we found the defects?
  - Have we met our product or process goals?
  - What can we predict about our software product in the future?

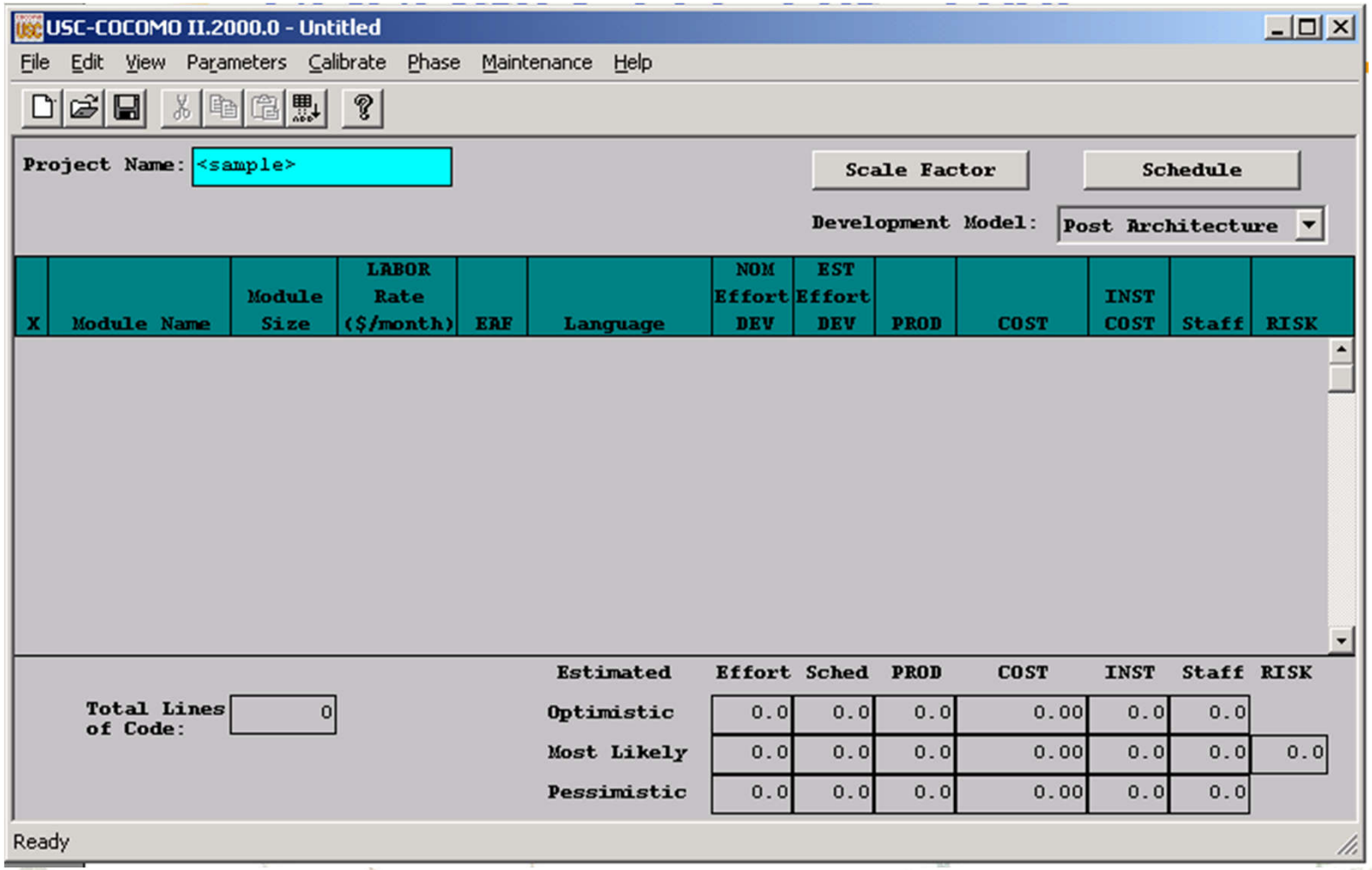
Tools: PSM


Menu bar
Toolbar
with PSMI's major functions
Workspace
List
Title bar
with open project
List commands
PSM Insight - TNMS
File Lists Windows Help
Select
Tailor
Apply
Projects
Issues
Mapping
I-C-M
Structures
Attributes
Data
Indicators
I-C-M List
Add
Edit
Assign
Delete
Report
Help
Close
Open Project: TNMS
Schedule and Progress
Resources and Cost
Growth and Stability
Functional Size and Stability
Product Size and Stability
Lines of Code
Product Quality
Development Performance
Technical Adequacy
Technology Effectiveness
Customer Satisfaction
Measure:
Lines of Code
Description:
Comments:

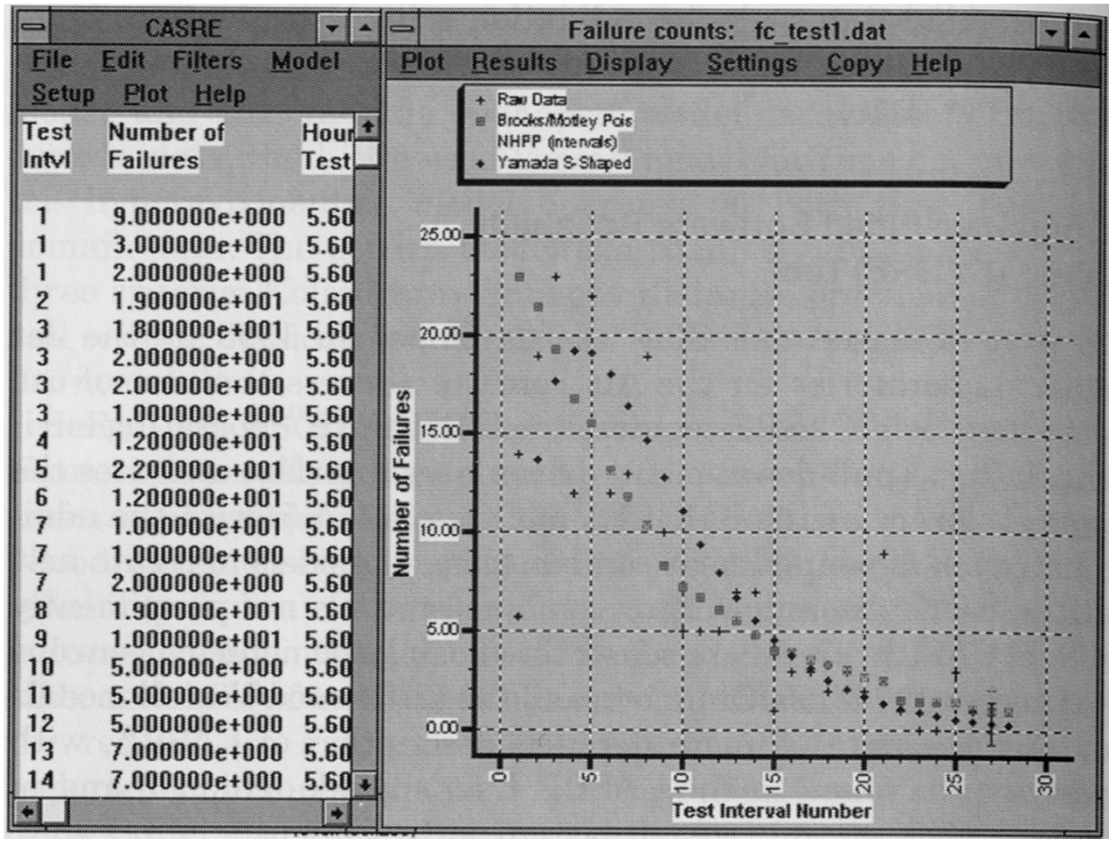
Tools: Function Point WorkBench



Tools: COCOMO II 2000



Tools: CASRE


CASRE
File Edit Filters Model
Setup Plot Help
Test Intvl
Number of Failures
Hour Test
1 9.000000e+000 5.60
1 3.000000e+000 5.60
1 2.000000e+000 5.60
2 1.900000e+001 5.60
3 1.800000e+001 5.60
3 2.000000e+000 5.60
3 2.000000e+000 5.60
3 1.000000e+000 5.60
4 1.200000e+001 5.60
5 2.200000e+001 5.60
6 1.200000e+001 5.60
7 1.000000e+001 5.60
7 1.000000e+000 5.60
7 2.000000e+000 5.60
8 1.900000e+001 5.60
9 1.000000e+001 5.60
10 5.000000e+000 5.60
11 5.000000e+000 5.60
12 5.000000e+000 5.60
13 7.000000e+000 5.60
14 7.000000e+000 5.60
Failure counts: fc_test1.dat
Plot Results Display Settings Copy Help
Raw Data
Brocks/Motley Pois
NHPP (intervals)
Yamada S-Shaped
Number of Failures
25.00
20.00
15.00
10.00
5.00
0.00
0
5
10
15
20
25
30
Test Interval Number

Tools: Together Control Center

Together 6 -- CashSales

File Edit Search View Project Run Deploy Selection Tools Help

Workspace: CashSales

Explorer

Java Metrics

Title	Abbreviation
Basic	
Lines Of Code	LOC
Number Of Attributes	NOA
Number Of Classes	NOC
Number Of Constructors	NOCON
Number Of Members	NOM
Number Of Operations	NOO
Number Of Import Statements	NOIS
Cohesion	
Complexity	
Coupling	
Encapsulation	
Halstead	
Inheritance	
Inheritance-Based Coupling	
Maximum	
Polymorphism	
Ratio	
UI	
Audit Violations	

Select All | Unselect All | Set Defaults | Save Set As... | Load Set...

Limits:	Lower	Upper
Package:	0	0
Class:	0	1000
Operation:	0	0
Aggregation:	Sum	
Granularity:	Class	

Count documentation comments

Count implementation comments

Count blank lines

LOC - Lines Of Code

This is the traditional measure of size. It counts the number of code lines. Documentation and implementation comments as well as blank lines can be optionally interpreted as code.

Documentation comments are Javadoc comments for Java, C++ and C#.

Start | Cancel | Help

Update Inv.

>= threshold]

Reorder Item

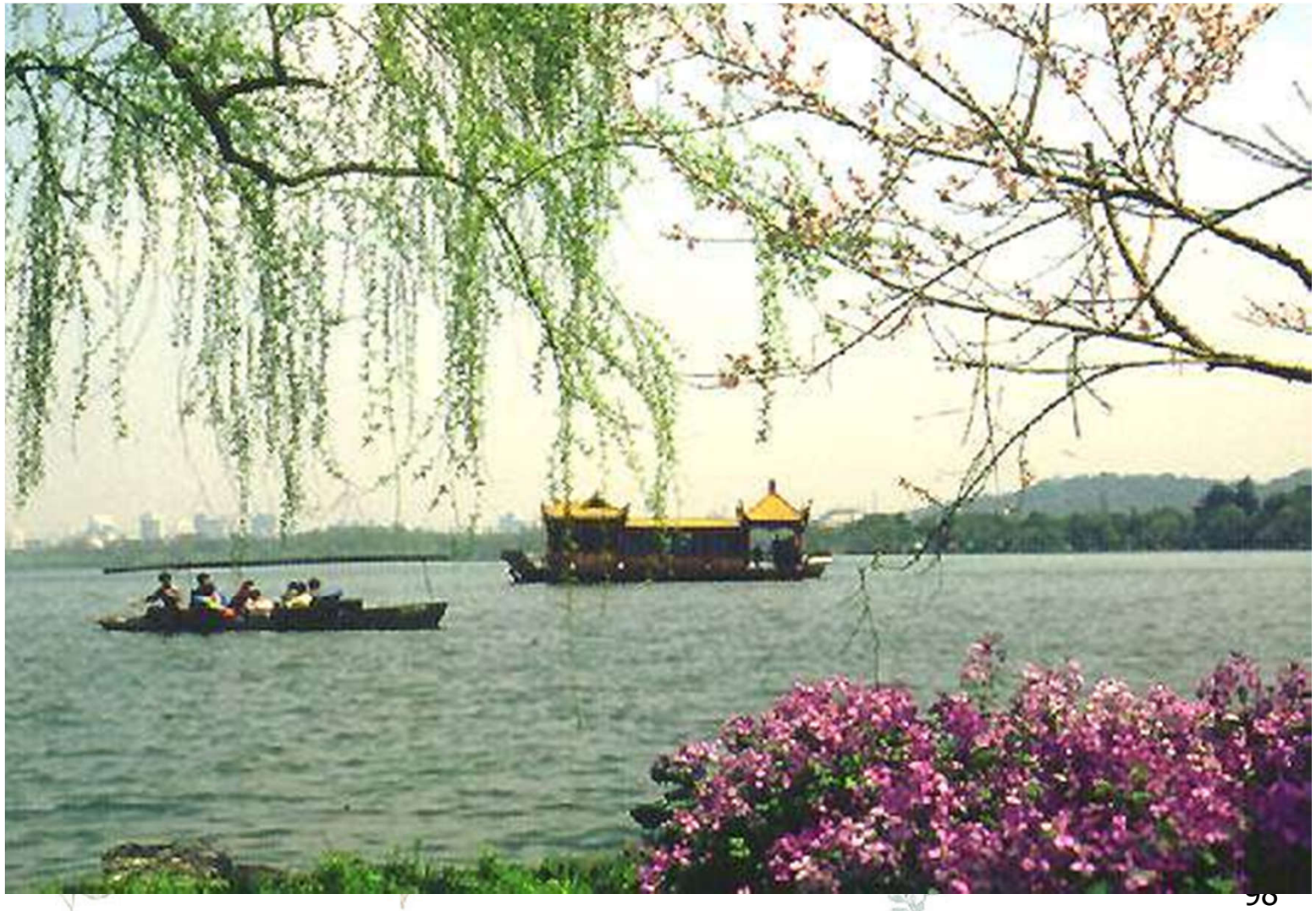
POSFrame_AboutBox.java

Progress | Insert | Ln: 7 | Col: 19

Summary

- What is measurement?
- What are software metrics?
- Scope of software engineering metrics

Further Reading

- **H. Zuse. History of Software Measurement, http://irb.cs.tu-berlin.de/~zuse/3-hist.ps.gz**

Thanks for your time and attention!