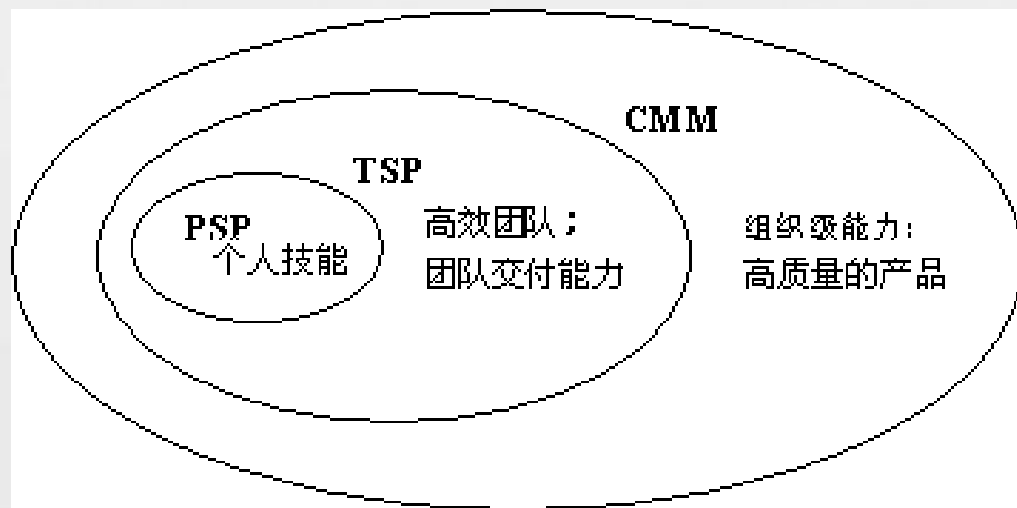


个体软件工程 (PSP)

为什么要学系PSP

从软件工程的角度

- “The ability of engineering teams to develop quality software products efficiently and effectively greatly depends on the ability of the individual engineers.”
– Watts Humphrey

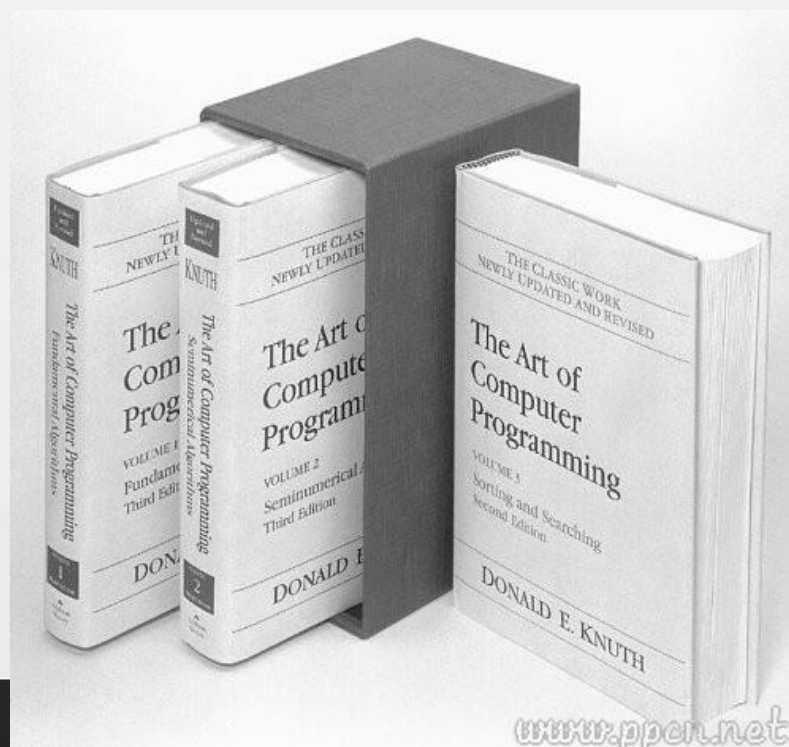
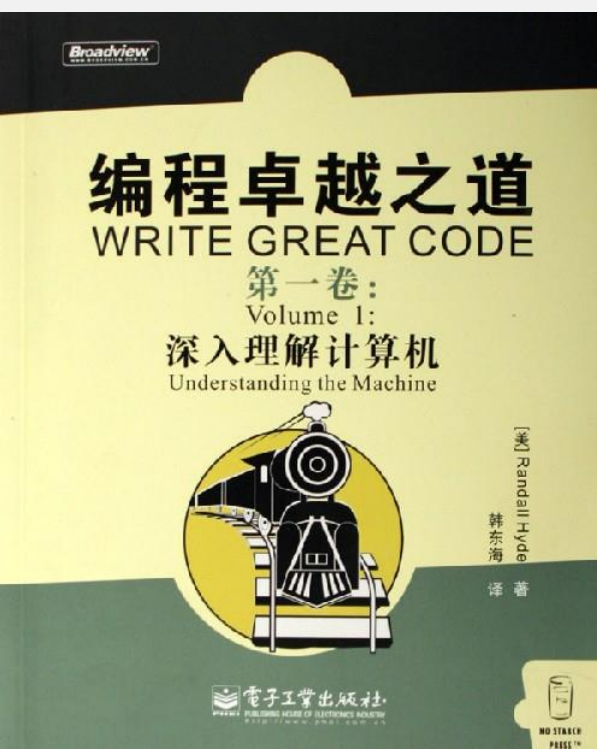


为什么要学系PSP

从程序员成长的角度

➤ 工欲善其事，必先利其器

- 技术：编程语言、算法、数据结构.....
- 方法：软件工程，PSP,TSP.....



PSP提高程序员的技能

“The PSP’s sole purpose is to help you be a better software engineer.”

– Watts Humphrey

PSP注重于个人的技能，能够指导软件工程师如何保证自己的工作质量，估计和规划自身的工作，度量和追踪个人的表现，管理自身的软件过程和产品质量。经过PSP学习和实践的正规训练，软件工程师们能够在他们参与的项目工作之中充分利用PSP，从而保证了项目整体的进度和质量

PSP和TSP

- **PSP**为软件人员进行软件开发提供了一个规范的个人过程框架,PSP过程由一系列方法、表单、脚本等组成,用以指导软件人员计划、度量和管理工作,同时它显示了如何定义过程及如何测量其质量和生产率
- **团队软件过程 (Team Software Process,TSP)** 是为开发软件产品的开发团队提供指导, TSP的早期实践侧重于帮助开发团队改善其质量和生产率, 以使其更好的满足成本及进度的目标。TSP被设计为满足2~20人规模的开发团队, 大型的多团队过程的TSP被设计为大约最多为150人左右的规模

PSP原则(6, 不尽相同)

- 一个软件系统的质量决定了它最差的组件的质量。
- 一个软件组件的质量取决于开发它的个体。
- 一个软件组件的质量取决于开发它所使用过程的质量。
- 质量的关键是个体开发人员的技巧、提交产物和执行什么样的个体过程。
- 作为软件专业人士，你应该积极寻找自己的个体过程。
- 你应该测量、跟踪和分析你的工作。

PSP能够

- 说明个体软件过程的原则；
- 帮助软件工程师作出准确的计划；
- 确定软件工程师为改善产品质量要采取的步骤；
- 建立度量个体软件过程改善的基准；
- 确定过程的改变对软件工程师能力的影响

度量个体的能力

个体能力的衡量-NBA球员

#	Player	Team	GP	MPG	FGM	FGA	FG%	3PM	3PA	3P%	FTM	FTA	FT%	TOV	PF	ORB	DRB	RPG	APG	SPG	BPG	PPG
1	Stephen Curry	GSW	79	34.2	10.2	20.2	.504	5.1	11.2	.454	4.6	5.1	.907	3.3	2.0	0.9	4.6	5.4	6.7	2.1	0.2	30.1
2	James Harden	HOU	82	38.1	8.7	19.7	.439	2.9	8.0	.359	8.8	10.2	.860	4.6	2.8	0.8	5.3	6.1	7.5	1.7	0.6	29.0
3	Kevin Durant	OKC	72	35.8	9.7	19.2	.505	2.6	6.7	.388	6.2	6.9	.898	3.5	1.9	0.6	7.6	8.2	5.0	1.0	1.2	28.2
4	DeMarcus Cousins	SAC	65	34.6	9.2	20.5	.451	1.1	3.2	.333	7.3	10.2	.718	3.8	3.6	2.4	9.1	11.5	3.3	1.6	1.4	26.9
5	LeBron James	CLE	76	35.6	9.7	18.6	.520	1.1	3.7	.309	4.7	6.5	.731	3.3	1.9	1.5	6.0	7.4	6.8	1.4	0.6	25.3
6	Damian Lillard	POR	75	35.7	8.2	19.7	.419	3.1	8.1	.375	5.5	6.2	.892	3.2	2.2	0.6	3.4	4.0	6.8	0.9	0.4	25.1
7	Anthony Davis	NOP	61	35.5	9.2	18.6	.493	0.6	1.8	.324	5.3	7.0	.758	2.0	2.4	2.1	8.1	10.3	1.9	1.3	2.0	24.3
8	Russell Westbrook	OKC	80	34.4	8.2	18.1	.454	1.3	4.3	.296	5.8	7.2	.812	4.3	2.5	1.8	6.0	7.8	10.4	2.0	0.2	23.5
9	DeMar DeRozan	TOR	78	35.9	7.9	17.7	.446	0.6	1.8	.338	7.1	8.4	.850	2.2	2.1	0.8	3.7	4.5	4.0	1.0	0.3	23.5
10	Paul George	IND	81	34.8	7.5	17.9	.418	2.6	7.0	.372	5.6	6.5	.860	3.3	2.8	1.0	6.0	7.0	4.1	1.9	0.4	23.1
11	Isaiah Thomas	BOS	82	32.2	7.2	16.9	.428	2.0	5.7	.359	5.8	6.6	.871	2.7	2.0	0.6	2.4	3.0	6.2	1.1	0.1	22.2
12	Klay Thompson	GSW	80	33.3	8.1	17.3	.470	3.5	8.1	.425	2.4	2.8	.873	1.7	1.9	0.4	3.4	3.8	2.1	0.8	0.6	22.1
13	Carmelo Anthony	NYK	72	35.1	7.9	18.2	.434	1.5	4.3	.339	4.6	5.6	.829	2.4	2.5	1.4	6.4	7.7	4.2	0.9	0.5	21.8
14	Kyle Lowry	TOR	77	37.0	6.6	15.6	.427	2.8	7.1	.388	5.2	6.4	.811	2.9	2.7	0.7	4.0	4.7	6.4	2.1	0.4	21.2
15	Kawhi Leonard	SAS	72	33.0	7.7	15.1	.506	1.8	4.0	.443	4.1	4.6	.874	1.5	1.8	1.3	5.5	6.8	2.6	1.8	1.0	21.2

个体能力的衡量-搬砖工人

工作量

有多少砖

要搬多远

工作质量

多快搬完

搬的过程中损坏了多少块砖

个体能力的衡量-软件工程师

PSP认为有以下4个要素

- a) 项目/任务有多大？（工作量估计）
- b) 花了多少时间？（实现花了多少时间）
- c) 质量如何？交付的代码中有多少缺陷？
- d) 是否按时交付

PSP认为

PSP2.1

Planning

- Estimate

Development

- Analysis
- Design Spec
- Design Review
- Coding Standard
- Design
- Coding
- Code Review
- Test

Record Time Spent

Test Report

Size Measurement

Postmortem

Process Improvement Plan

计划

- 估计这个任务需要多少时间

开发

- 分析需求
- 生成设计文档
- 设计复审 (和同事审核设计文档)
- 代码规范 (为目前的开发制定合适的规范)
- 具体设计
- 具体编码
- 代码复审
- 测试 (包括自我测试, 修改代码, 提交修改)

记录时间花费

测试报告

计算工作量

事后总结

提出过程改进计划

本讲内容

- 工作量估算
- 个体开发技术
- 缺陷管理

工作量估算(规模)的方法

1. 代码行方法
2. 功能点方法(FP)
3. 用例点方法(UCP)
4. 故事点方法(USP)



■ 用软件代码行数(KLOC)来表示软件项目规模

➤ 从软件程序量的角度定义项目规模。要求：

- 功能分解足够详细的
- 有一定的经验数据（类比和经验方法）
- 与具体的编程语言有关

代码行 (KLOC 2/2)

■ 优点

- 简单易行，自然直观

■ 缺点

- 依赖于程序设计语言的表达能力和功能
- 软件开发初期很难估算出最终软件的代码行数
- 对精巧的软件项目不合适
- 只适合于过程式程序设计语言

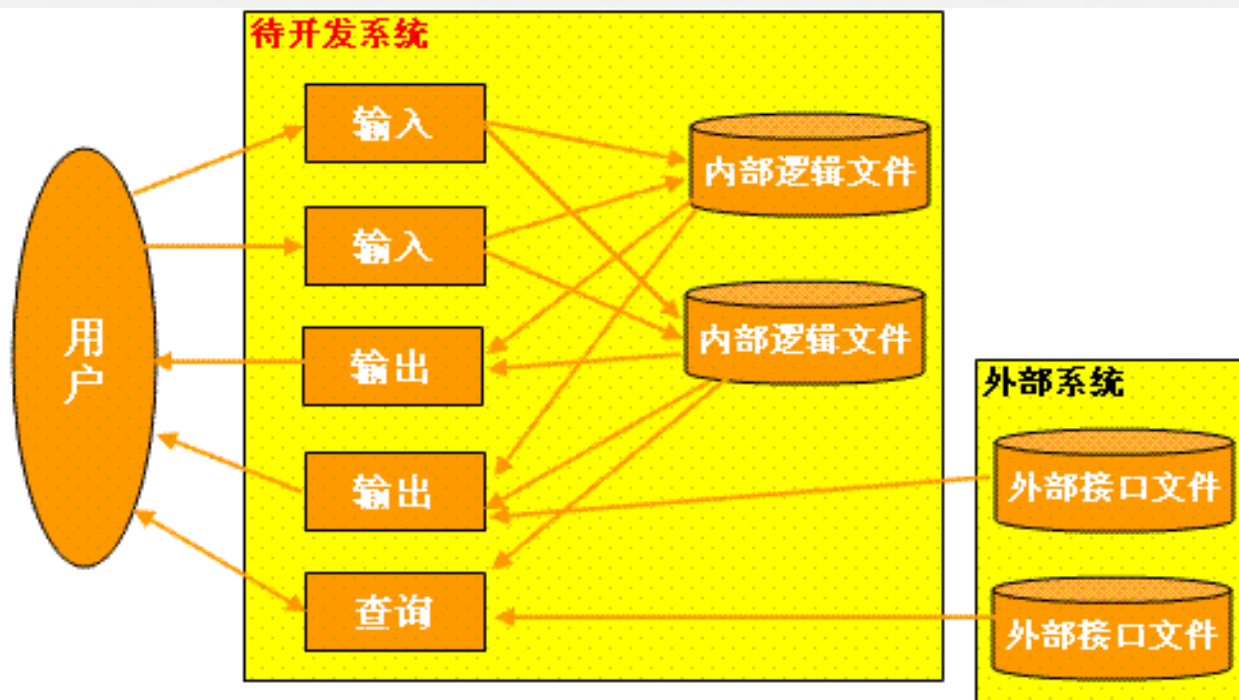
■ 用软件的功能点表示软件的规模(功能点=功能?)

➤ 功能点是一个的度量单位，用于度量工作产品的规模。就像公斤和千米一样，仅仅是一个抽象化的单位。

➤ 功能点不直接度量软件的内部架构和技术复杂度。

■ FP方法有一个组织IFPUG (International Function Point Users Group) 发布了一个文档，该文档已经提交给了ISO

■ FPA将商业信息系统抽象成输入数据，输出数据，查询数据，系统内部文件，系统外部文件



输入(EI): External Input
输出(EO): External Output
查询(EQ): External Inquiries

内部逻辑文件(ILF): Internal Logical File
外部接口文件(EIF): External Interface File

功能点FP

■ 估算原理

- 分为事务功能点(TFP,系统的规模)和数据功能点(DFP,系统的处理逻辑复杂度)
- TFP:EI(输入)、EO(输出)和EQ(查询)
- DFP:ILF(内部逻辑文件)、EIF(外部接口文件)

■ 估算步骤



■ 用途和用法

➤ 项目规划阶段估算功能点

- 输入：需求和历史数据
- 用途：项目规划

➤ 项目进行过程中估算功能点

- 输入：需求和设计文档
- 用途：重新评估需求和计划调整

➤ 项目交互阶段估算功能点：

- 用途：在交付文件中报告实际的功能点数、度量项目范围的变更（如需求变更率）、回顾项目实施情况、发布统计数据并作为组织的基线。

■ FP的基本概念

➤ **ILF: Internal Logical File**, 用户确认的, 在应用程序内部维护的、逻辑上相关的数据块或者控制块。ILF的主要目的是通过应用程序的一个或者多个基本处理来保存数据。

➤ 识别规则:

- The group of data or control information is logical and user identifiable.
- The group of data is maintained through an elementary process within the application boundary being counted.

■ FP的基本概念

➤ ILF的例子:

- 应用程序内部的业务数据。并不是数据库中一个表就是一个ILF，是否是一个ILF要根据定义和识别规则
- 应用程序维护的安全数据和口令数据
- 应用程序维护的参数数据
- 应用程序维护的出错误信息文件及其描述

➤ EIF: 另外一个系统的ILF,在本系统内被引用

■ FP基本概念(ILF/EIF的度量)

➤ **DET: Data Element Type**, 数据元素类型, ILF或EIF中的记录的字段, 或界面上的一个输入/输出字段

- DET也可以是控制信息, 如界面上的一个按钮, 一组Radio, 一个CheckBox

➤ **RET: Record Element Type**, 记录元素类型, 用户确认的ILF/EIF中的数据元素的分组(subgroup, 这些分组可以是可选的或强制的)。计算规则:

- 在ILF/EIF中每一组可选的或必须的数据元素, 记为一般认为, EI中一个TAB页面
- 如(FRAME、上一步...)是一个分组为一个RET

功能点FP

■ FP的基本概念

➤ **ILF/EIF的复杂度**。包含的字段越多，包含的记录类型越多，处理ILF/EIF的逻辑越复杂

数据文件 (EIF/ILF) 复杂度

RET \ DET	DET		
	1-19	20-50	>50
1	低	低	中
2-5	低	中	高
>5	中	高	高

数据功能点的计算方法

功能点类型 \ 复杂度	复杂度		
	低	中	高
EIF	5	7	10
ILF	7	10	15

功能点(FP)


■ Primavera中计算功能点的方法

功能点估算 - 数据和处理

元素	最低复杂度		平均复杂度		最高复杂度		总计
内部逻辑文件 (ILF)	<input type="text" value="1"/>	X 7	<input type="text" value="1"/>	X 10	<input type="text" value="1"/>	X 15 =	32
外部接口文件 (EIF)	<input type="text" value="0"/>	X 5	<input type="text" value="0"/>	X 7	<input type="text" value="0"/>	X 10 =	0
外部输入 (E)	<input type="text" value="0"/>	X 3	<input type="text" value="0"/>	X 4	<input type="text" value="0"/>	X 6 =	0
外部输出 (EO)	<input type="text" value="0"/>	X 4	<input type="text" value="0"/>	X 5	<input type="text" value="0"/>	X 7 =	0
外部查询 (EQ)	<input type="text" value="0"/>	X 3	<input type="text" value="0"/>	X 4	<input type="text" value="0"/>	X 6 =	0
未调整的功能点计数							32

 帮助

 清除 (L)

 关闭

ILF/EIF例子

招聘系统的职位维护功能

Action: _ 7=Prior 8=Following 9=Save

Job Data

Job number: RD15379305

Job name: May Issue - Print Covers

Pay grade: JRNY05A

Line No	Job Description
<u>01</u>	<u>Print Covers 4-Up, Lacquer Finish.</u>

F1=Help F7=Scroll up F8=Scroll down F12=Cancel

Enter: returns to calling screen.

F12: returns to calling screen.

F1: shows screen or field level help.

Action 7: shows prior job data, if present.

F7: scrolls up 10 job description lines.

Action 8: shows following job data, if present.

F8: scrolls down 10 job description lines.

ILF数量

DET和RET数量

■ 1个ILF/2个ILF?

➤ 职位表和职位描述需要一起维护，不可能单独维护一个，所以是一个ILF

■ DET

➤ 5个

■ RET

➤ 2个用户确认的分组，并在全部在一个EIF中，所以有两个RET

■ DFP计算

➤ **DFP数**：计算方法如上。如某系统有**3个ILF**，难度为低，有**2个EIF**，难度分别为中和高，则系统的**DFP**= $3*5 + 1*10 + 1*15 = 40$

■ EI:

➤ 输入包括业务数据，也包括按钮、**Checkbox**等控制数据。**EI**通常输入数据到**ILF**，一个**ILF**的维护通常包括增删改三个**EI**。

功能点FP

■ EO:

➤ 报表输出、画面初始化的输出。**EO**输出的数据或者来自**ILF**或**EIF**，或者组合**ILF/EIF**的字段而来的新数据，或者通过算法计算而来的数据。通常确认**Message**或错误提示不算作独立的**EO**，但通知**Mail**之类的算作独立的**EO**

■ EQ:

➤ 外部查询，从外部输入一些条件，系统返回一定的输出

■ FTR

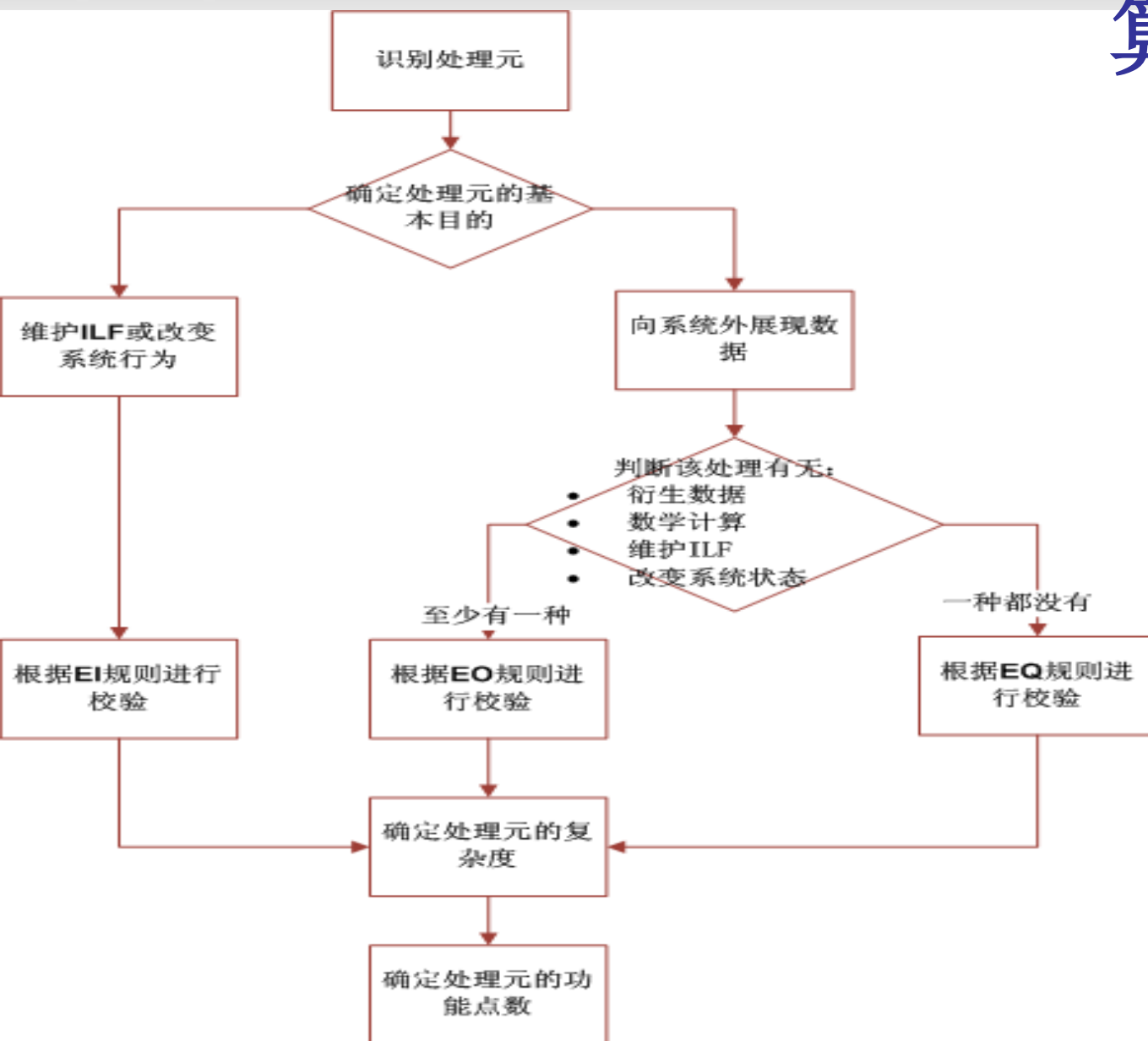
➤ 引用的文件数，**EI/EO/EQ**的处理过程中通常要用到**ILF/EIF**，**FTR**指事务处理用到的**ILF/EIF**数

■ EI/EO和EQ复杂度

➤ 用该事务的输入/输出数据中的**DET**（即数据字段）数和事务的**FTR**（引用的文件）数衡量

表三: Transaction File(EI)複雜度				表四: Transaction File(EO/EQ)複雜度			
DET \ FTR	1-19	20-50	>50	DET \ FTR	1-5	6-19	>19
0-1	低	低	中	0-1	低	低	中
2	低	中	高	2-3	低	中	高
>2	中	高	高	>3	中	高	高

EI/EQ/E0计算步骤



IPUG定义了13种处理逻辑，EI/EQ/EO必须满足以下规定：

处理逻辑	EI	EO	EQ
1) 需要执行的校验			
2) 需要执行了数学公式和运算		M*	N
3) 等值转换的算法，如货币转换，单位转换			
4) 筛选数据的条件			
5) 用于判断是否适用的条件，也就是入口条件。			
6) 一个或多个ILF的修改逻辑	M*	M*	N
7) 一个或多个被引用的ILF或EIF			M
8) 从系统内获取了数据或控制信息			M
9) 从已有数据衍生的额外数据		M*	N
10) 改变了系统的行为或状态	M*	M*	N
11) 向边界外准备或展现了信息		M	M
12) 从边界外接收数据或控制信息的能力	M		
13) 数据重排。注意：重排数据不能作为处理功能唯一性的识别依据			

M*:必须包含
N: 不能包含

常见的 EI/EQ/EO

从屏幕录入的数据	EI
批处理输入	EI
从磁带等设备输入	EI
扫描输入	EI
增，删，改等事务处理。	EI
导航菜单	EO?
登陆界面	EQ
从ILF读取列表的列表框或下拉框	EQ
筛选条件。	不是完整的处理元
快捷键，命令键。	不是处理元?
导出报表	EO
在线报表	EQ
详细报表中的统计字段	EQ?
由具体报表汇总得出的总结报表	EO
查看帮助	EQ
输出文件	EO

功能点FP

■ 招聘系统的职位维护功能

- 几个EI?
- 每个EI的复杂度是多少?

■ 更多信息参见以下文档

- [..\..\TK\(技术知识\)\CMM\功能点度量方法\IFPUG Counting Practices Manual 4.1.pdf](#)
- 功能点度量,清华大学出版社, 2003

■ 功能点计算方法

➤ $AFP = (0.65 + 0.01 \times \sum F_i) \times UFP$

➤ **UFP** : 未调整的功能点

➤ **AFP**: 调整后的功能点

➤ F_i : 14个因素的“复杂性调节值” ($i = 1..14$)

➤ 0.65, 0.01都是经验常数

功能点FP

■ Primavera中计算功能点

功能点估算值

未调整的功能点计数(UFP)

计算(O...)

37

总的影响度(TDI)

特征(H...)

42

UFP

TDI

=

最终 FP计数

$0 \times (0.65 + (0.01 \times 0)) =$

0

最终调整功能点计数

40

平均产量 (功能点 / 人工月)

0

估算数量

0.00h

?

帮助

关闭

功能点中的复杂度调整因子

F_i 的取值(0,1,2,3,4,5):0-没有影响, 1-偶有影响, 2-轻微影响, 3-平均影响, 4-较大影响, 5-严重影响

复杂度调整因子			
F1	可靠的备份和恢复	F2	数据通信
F3	分布式函数	F4	性能
F5	大量使用的配置	F6	联机数据输入
F7	操作简单性	F8	在线升级
F9	复杂界面	F10	复杂数据处理
F11	重复使用性	F12	安装简易性
F13	多重站点	F14	易于修改

功能点中的复杂度调整因子

2.6.4.3. 性能

性能指的是处理时间，吞吐量等指标对开发的影响。

用户所提出的关于处理时间，吞吐量的要求直接影响到系统的设计，实施，安装和支持。

影响程度	影响程度的描述
0	用户没有提出特定的性能要求
1	提出了性能和设计方面的要求，但不需要采取特定行动
2	响应时间和吞吐量在系统峰值时是关键，但是不需要采取相应的CPU使用方面的特殊设计。交易的完成期限是下一个营业日
3	响应时间和吞吐量在所有时间都是关键的，但是不需要采取相应的CPU使用方面的特殊设计。交易的完成期限比较严格
4	除了3中的要求之外，因为对需求的要求比较严格，在设计阶段需要进行性能分析
5	除了4中的要求之外，在设计和实施阶段需要使用性能分析工具来判断性能要求的完成情况

功能点中的复杂度调整因子

2.6.4.5. 交易速度

交易速度指的是业务交易处理速度的要求对应用开发的影响。

用户所提出的关于交易速度的要求直接影响到系统的设计，实施，安装和支持。

影响程度	影响程度的描述
0	没有预期的业务交易的高峰期
1	有预期的交易高峰期（例如，每月，每季度，每年）
2	每周都有预期的交易高峰期
3	每天都有预期的交易高峰期
4	用户在需求或者协议中提出了较高的交易速度要求，需要在设计阶段进行性能分析
5	用户在需求或者协议中提出了较高的交易速度要求，需要在设计阶段进行性能分析。此外，在设计，实施以及安装阶段还需要使用性能分析的工具

2.6.4.6. 在线数据输入

在线数据输入指的是数据通过交互的方式输入应用的程度。

应用提供在线数据输入和控制功能。

影响程度	影响程度的描述
0	所有的交易都是用批处理的方式完成
1	交互式的数据输入占总交易的 1%~7%
2	交互式的数据输入占总交易的 8%~15%
3	交互式的数据输入占总交易的 16%~23%
4	交互式的数据输入占总交易的 24%~30%
5	交互式的数据输入占总交易的 30%以上

功能点中的复杂度调整因子

2.6.4.7. 最终用户的效率

最终用户效率指的是对应用的人文因素以及使用简便性的考虑程度。

下列的功能设计是强调最终用户效率的：

- 导航帮助（例如功能键，跳转，动态菜单等）
- 菜单
- 在线帮助和文档
- 光标自动移动
- 滚屏
- 在线远程打印
- 预定义的功能键
- 在线方式提交的批处理任务
- 屏幕数据的光标选取
- 大量使用反白，高亮，颜色表示以及其他的标示方法
- 用户手册
- 鼠标界面
- 弹出窗口
- 尽可能少的屏幕
- 双语言支持（当 4 项）
- 多语言支持（当 6 项）

影响程度	影响程度的描述
0	以上一项都没有
1	1~3 项
2	4~5 项
3	6 项以上，但是没有用户对于效率的要求
4	6 项以上，用户对效率的要求足以使设计时必须考虑人文要素
5	6 项以上，用户对效率的要求使得开发人员必须使用特定的工具和流程以判定用户对效率的要求已经被达成

功能点

■ 优点

- 与程序设计语言无关, 在开发前就可以估算出软件项目的规模(事前)

■ 不足

- 没有直接涉及算法的复杂度, 不适合算法比较复杂的软件系统;
- 功能点计算和估算人员托经验有很大关系
- 数据不好采集

功能点FP和代码行方法的互换

代码行度量和功能点度量间的关系

语言	代码行/FP
Assembly	320
C/C++	150
COBOL	105
FORTRAN	105
PASCAL	91
ADA	71
PL/1	65
PROLOG/LISP	64
SMALLTALK	21
SPREADSHEET	6

工作量估算-UCP (Use case Point)

该方法涉及到4个要素

- Unadjusted Use Case Weight (UUCW) –未调整用例点数
- Unadjusted Actor Weight (UAW) – 未调整参与者数
- Technical Complexity Factor (TCF)—技术复杂度
- Environmental Complexity Factor (ECF)-环境复杂度
- Productivity Factor (PF).—生产率

该方法考虑了用例模型中以下贴这个特征

- The number of steps to complete the use case.
- The number and complexity of the actors.
- The technical requirements of the use case such as concurrency, security and performance.
- Various environmental factors such as the development teams' experience and knowledge.

工作量估算-UCP (Use case Point)

UCP计算公式

➤ $UCP = (UUCW + UAW) \times TCF \times ECF$

工作量

➤ 工作量 = $UCP \times \text{评分}$

工作量估算-UCP (Use case Point)

UUCW

表1 用例分类权重对应表

用例类别	说明	权重
简单（小）	基本流的步骤不超过3步，备选流或异常不超过3个。 比如简单用户界面或一般API	5
普通（中）	基本流的步骤有4~7步，备选流或异常不超过6个。 比如普通界面或复杂API	10
复杂（大）	基本流的步骤 8~ 12步，备选流或异常不超过9个。 比如复杂的用户界面或过程	15

- Ø 步骤——步骤定义为单个角色的原子操作。在一个步骤之内，只说明一个角色的连续动作，角色不发生转移；角色变换，是新的步骤。
- Ø 基本流——也有称为主成功场景，达成用例目标的事件流。
- Ø 备选流——也有称为失败场景，基本流之外的不能达到用例目标的事件流。
- Ø 异常——在基本流中直接说明的异常情况。

**UUCW = (Total No. of Simple Use Cases x 5) +
(Total No. Average Use Case x 10) +
(Total No. Complex Use Cases x 15)**

工作量估算-UCP (Use case Point)

UAW

Actor Classification	Type of Actor	Weight
Simple	External system that must interact with the system using a well-defined API	1
Average	External system that must interact with the system using standard communication protocols (e.g. TCP/IP, FTP, HTTP, database)	2
Complex	Human actor using a GUI application interface	3

$$UAW = (\text{Total No. of Simple actors} \times 1) + (\text{Total No. Average actors} \times 2) + (\text{Total No. Complex actors} \times 3)$$

序号	复杂度级别	复杂度标准	权值
1	simple	角色通过 API 与系统交互	1
2	average	角色通过协议与系统交互	2
3	complex	用户通过 GUI 与系统交互	3

工作量估算-UCP (Use case Point)

TCF

$TCF = 0.6 + (TF/100)$

Factor	Description	Weight
T1	Distributed system	2.0
T2	Response time/performance objectives	1.0
T3	End-user efficiency	1.0
T4	Internal processing complexity	1.0
T5	Code reusability	1.0
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portability to other platforms	2.0
T9	System maintenance	1.0
T10	Concurrent/parallel processing	1.0
T11	Security features	1.0
T12	Access for third parties	1.0
T13	End user training	1.0

序号	技术因子	说明
1	TCF1	分布式系统
2	TCF2	性能要求
3	TCF3	最终用户使用效率
4	TCF4	内部处理复杂度
5	TCF5	复用程度
6	TCF6	易于安装
7	TCF7	系统易于使用
8	TCF8	可移植性
9	TCF9	系统易于修改
10	TCF10	并发性
11	TCF11	安全功能特性
12	TCF12	为第三方系统提供直接系统访问
13	TCF13	特殊的用户培训设施

工作量估算-UCP (Use case Point)

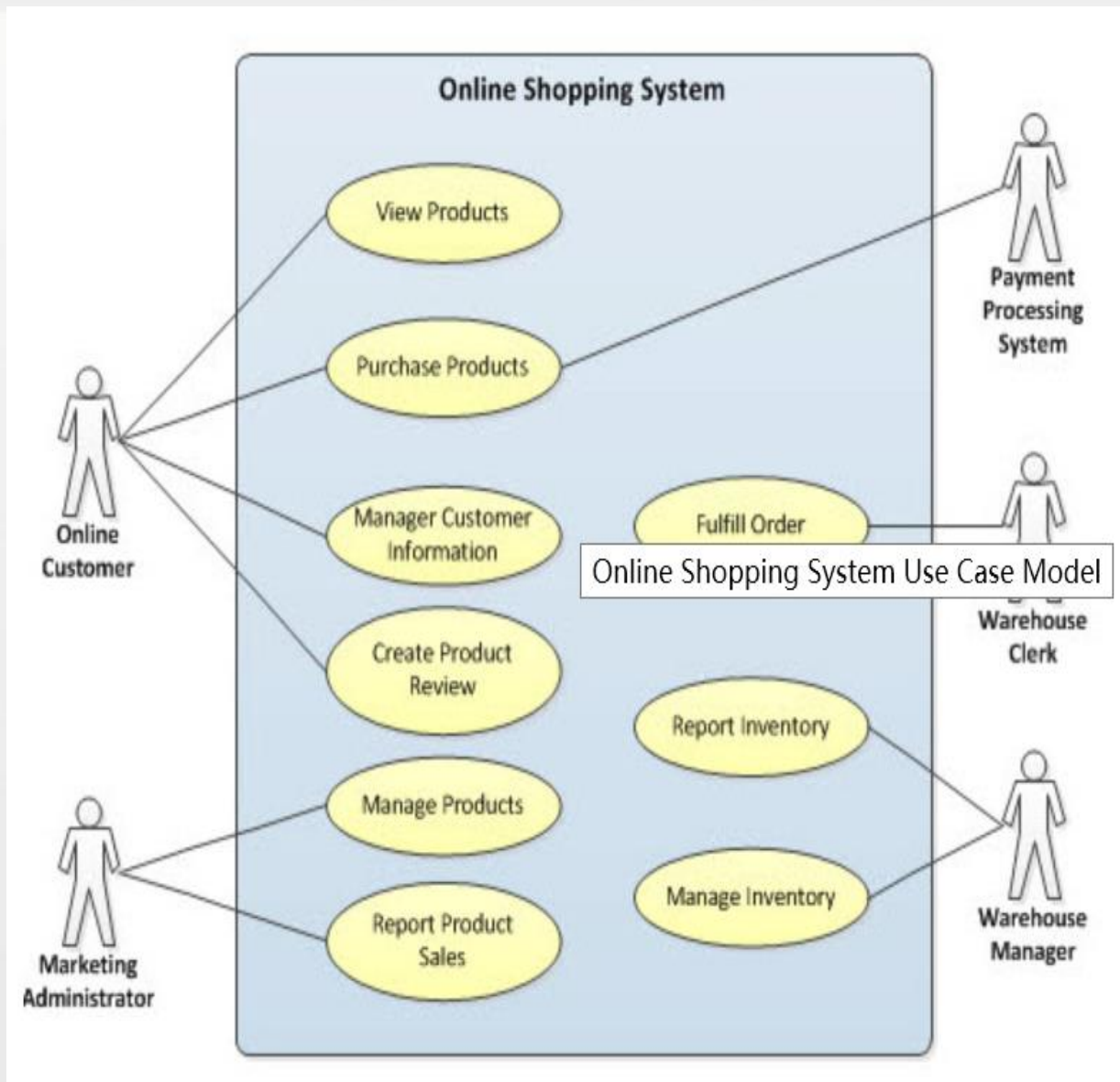
$$ECF = 1.4 + (-0.03 \times EF)$$

ECF

Factor	Description	Weight
E1	Familiarity with development process used	1.5
E2	Application experience	0.5
E3	Object-oriented experience of team	1.0
E4	Lead analyst capability	0.5
E5	Motivation of the team	1.0
E6	Stability of requirements	2.0
E7	Part-time staff	-1.0
E8	Difficult programming language	-1.0

序号	环境因子	说明
1	ECF1	UML 精通程度
2	ECF2	系统应用经验
3	ECF3	面向对象经验
4	ECF4	系统分析员能力
5	ECF5	团队士气
6	ECF6	需求稳定度
7	ECF7	兼职人员比例高低
8	ECF8	编程语言难易程度

工作量估算-UCP (Use case Point) —例子



UCP (Use case Point) — 例子

UUCW和UAW计算

$$\text{UUCW} = (\text{Total No. of Simple Use Cases} \times 5) + (\text{Total No. Average Use Cases} \times 10) + (\text{Total No. Complex Use Cases} \times 15)$$

For the Online Shopping System, the $\text{UUCW} = (2 \times 5) + (3 \times 10) + (4 \times 15) = 100$

$\text{UUCW} = 100$

T

$$\text{UAW} = (\text{Total No. of Simple Actors} \times 1) + (\text{Total No. Average Actors} \times 2) + (\text{Total No. Complex Actors} \times 3)$$

For the Online Shopping System, $\text{UAW} = (1 \times 1) + (0 \times 2) + (4 \times 3) = 13$

$\text{UAW} = 13$

UCP (Use case Point) — 例子

TCF计算

Factor	Description	Weight	Assigned Value	Weight x Assigned Value
T1	Distributed system	2.0	5	10
T2	Response time/performance objectives	1.0	5	5
T3	End-user efficiency	1.0	3	3
T4	Internal processing complexity	1.0	2	2
T5	Code reusability	1.0	3	3
T6	Easy to install	0.5	1	0.5
T7	Easy to use	0.5	5	2.5
T8	Portability to other platforms	2.0	2	4
T9	System maintenance	1.0	2	2
T10	Concurrent/parallel processing	1.0	3	3
T11	Security features	1.0	5	5
T12	Access for third parties	1.0	1	1
T13	End user training	1.0	1	1
Total (TF):				42

Next, the TCF is calculated:

$$TCF = 0.6 + (TF/100)$$

$$\text{For the Online Shopping System, } TCF = 0.6 + (42/100) = 1.02$$

$$TCF = 1.02$$

UCP (Use case Point) — 例子

ECF计算

Factor	Description	Weight	Assigned Value	Weight x Assigned Value
E1	Familiarity with development process used	1.5	3	4.5
E2	Application experience	0.5	3	1.5
E3	Object-oriented experience of team	1.0	2	2
E4	Lead analyst capability	0.5	5	2.5
E5	Motivation of the team	1.0	2	2
E6	Stability of requirements	2.0	1	2
E7	Part-time staff	-1.0	0	0
E8	Difficult programming language	-1.0	4	-4
Total (EF):				10.5

Next, the ECF is calculated:

$$ECF = 1.4 + (-0.03 \times EF)$$

$$\text{For the Online Shopping System, } ECF = 1.4 + (-0.03 \times 10.5) = 1.085$$

$$ECF = 1.085$$

UCP (Use case Point) — 例子

UCP计算和工作量计算

$$\text{UCP} = (\text{UUCW} + \text{UAW}) \times \text{TCF} \times \text{ECF}$$

For the Online Shopping System, $\text{UCP} = (100 + 13) \times 1.02 \times 1.085 = 125.06$

$$\text{UCP} = 125.06$$

$$\text{Estimated Effort} = \text{UCP} \times \text{Hours/UCP}$$

For the Online Shopping System, $\text{Estimated Effort} = 125.06 \times 28$

$$\text{Estimated Effort} = 3501 \text{ Hours}$$

PF 一般是采用历史数值, If no historical data has been collected, a figure between 15 and 30 is suggested by industry experts. A typical value is 20.

User Story Point (USP) 故事点

Story points 是描述用户故事 (user story)、特性 (feature)或者其他工作的总体规模的计量单位.是对于完成一个User Story所需要的工作量的一个相对估计。是敏捷开发时用来估算工作量的一种方法。

**故事点方法是一种粗略的估计。背后的理论依据是：
：对于复杂的未知事物，人们做出的相对估计，往往比“精确”估计更加合理。**

(User Story Point)

故事点：也是一种计量单位

你去拜访一个朋友的家。有人问“你的朋友家有多大啊？你往往很难说出具体的平米数。但是你很容易说出，这是一居室的房子，还是二居室的房子。这样大家就可以estimate一下房子的面积。

这比直接estimate房子的确切平米数字更加靠谱，不是么？

Story Point也是这样。Story Point=2, 意味着它比Story Point为1的User Story要大一些，可能大一倍左右

甲：你家离学院多远啊？地铁3号线5站地点

己：你家呢？我家地铁3号线15站地

这样我们就知道乙比较甲要远很多，但是具体远多少公里不知道也不需要准确知道

■ 做版本交付和项目迭代的计划

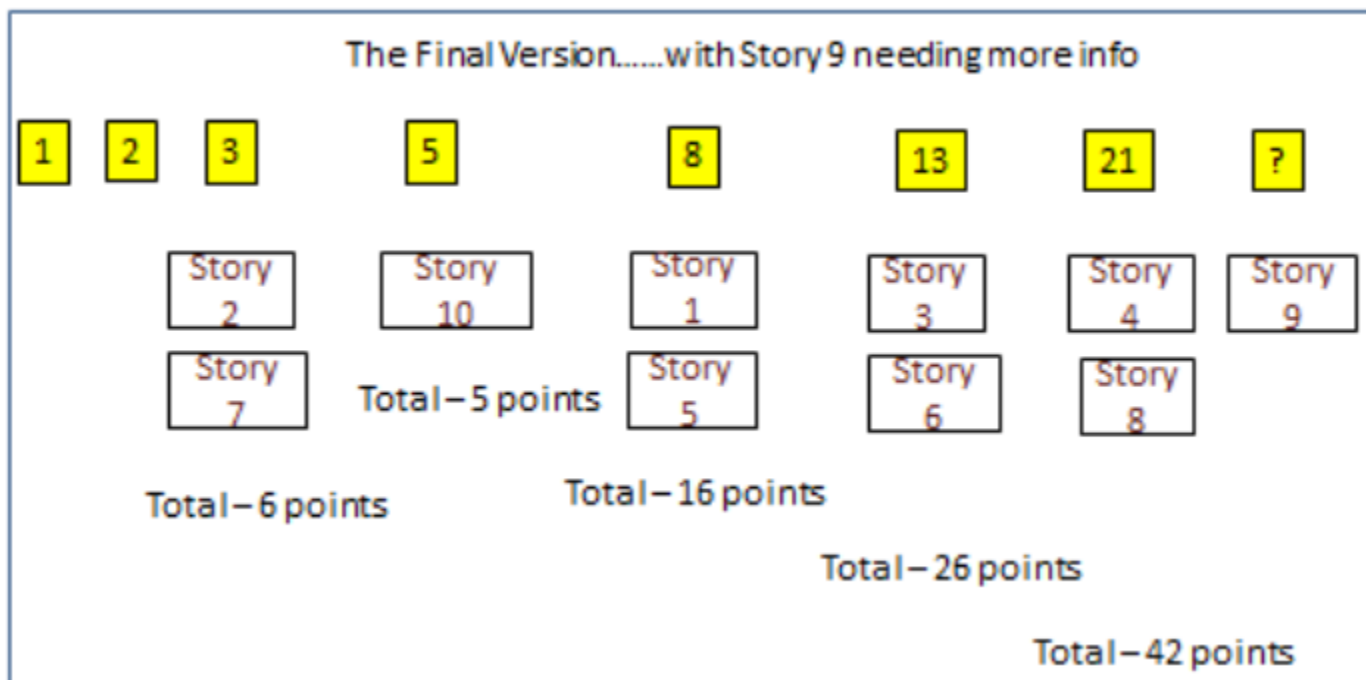
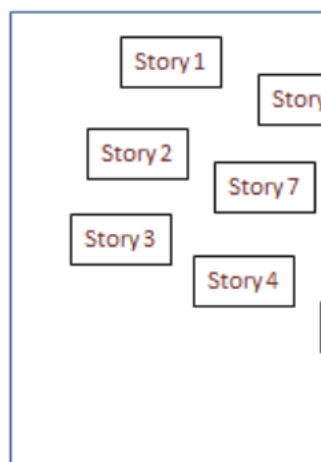
➤ 版本交付

- Team的Velocity是相对稳定的，有了所有User Story的Story Point总和，除我们的velocity,这样很容易就能算出需要几个迭代需要做完

➤ 迭代计划

- 根据历史数据，一个release的容量为150 Story Point，团队很有可能在下一个Release里面完成150左右的User story，这样优先级最高的总和为150 Story Point左右的那些User story是最有可能完成的。

故事点估算方法



估算过程

- 团队人员排队。从第一名成员开始，将用户故事放到他认为可以正确反应故事点值的那一列上
- 第一名成员做完后排团队成员的最后一个位置
- 下一个团队成员可以挪动已经摆好的用户故事，也可以选择另外的用户故事，把它挪到他认为可以正确反应故事点值的那一列
- 继续这个过程，直到所有用户故事都摆放完毕。

故事点估算方法

- **Fibonacci只是度量的一种，Scrum本身并没有规定用什么方式来度量，每个团队甚至可以创造自己的度量方式。**
 - 一个是用**T-shirt size**，分为**XS, S, M, L, XL, XXL**。每个**User Story**都可以用一种**size**来表示大小。每个**Sprint**的**velocity**可以是**2M+1XL**，或者是**1S+2L**之类的，长期来看也可以达到 **Team**自己熟悉的稳定值。
 - 另一个是用酒的容器，比如一小杯，一大杯，一瓶，一大瓶，一缸，等等，使用方法和**T-shirt size**类同
- **使用个性化度量的好处是Manager没有办法比较各团队之间的velocity，可以避免一些无意义的审核。当然也有不好的地方，Product Owner比较难做release plan，**

故事点用于做迭代计划

Story	Story Point
Story A	3
Story B	5
Story C	5
Story D	3
Story E	1
Story F	8
Story J	5
Story H	5
Story I	5
Story J	2

由于开发团队期望每个迭代完成规模为13的User Story，所以每轮迭代所计划的User Story的总规模不能超过13

迭代	User Story	规模
迭代一	A, B, C	13
迭代二	D, E, F	12
迭代三	G, H, J	12
迭代四	I	5

....

问题思考？

- 在软件开发的阶段，我们是否应该用不同的方法来进行总量的估算呢？

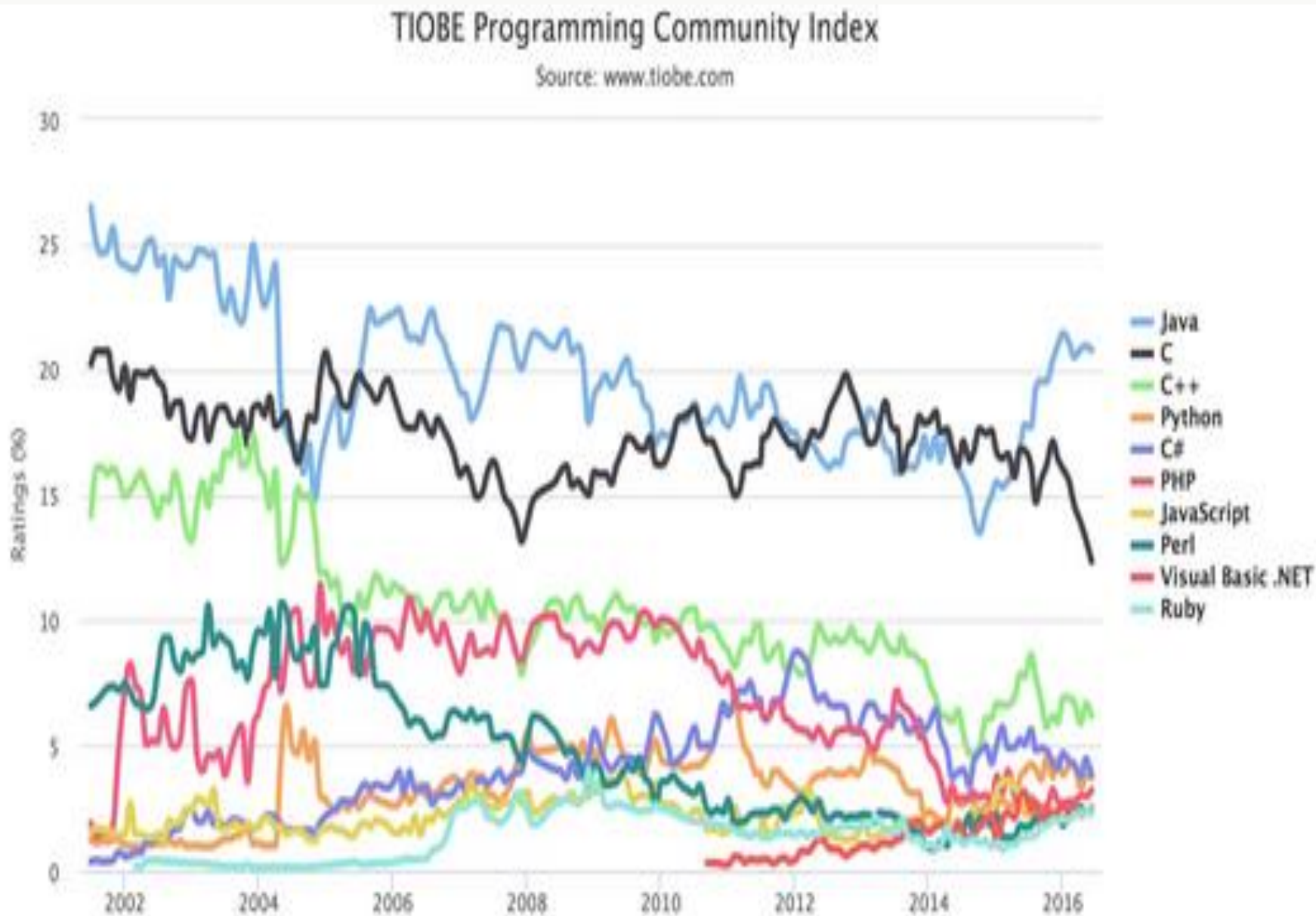
个体开发技术

- 程序设计语言排行榜
- 编程规范
- 代码简洁之道
- 程序效能分析
 - 代码自动检查
 - 方法调用次数分析
 - 方法运行时间分析

语言排行榜(TIOBE 2016.06)

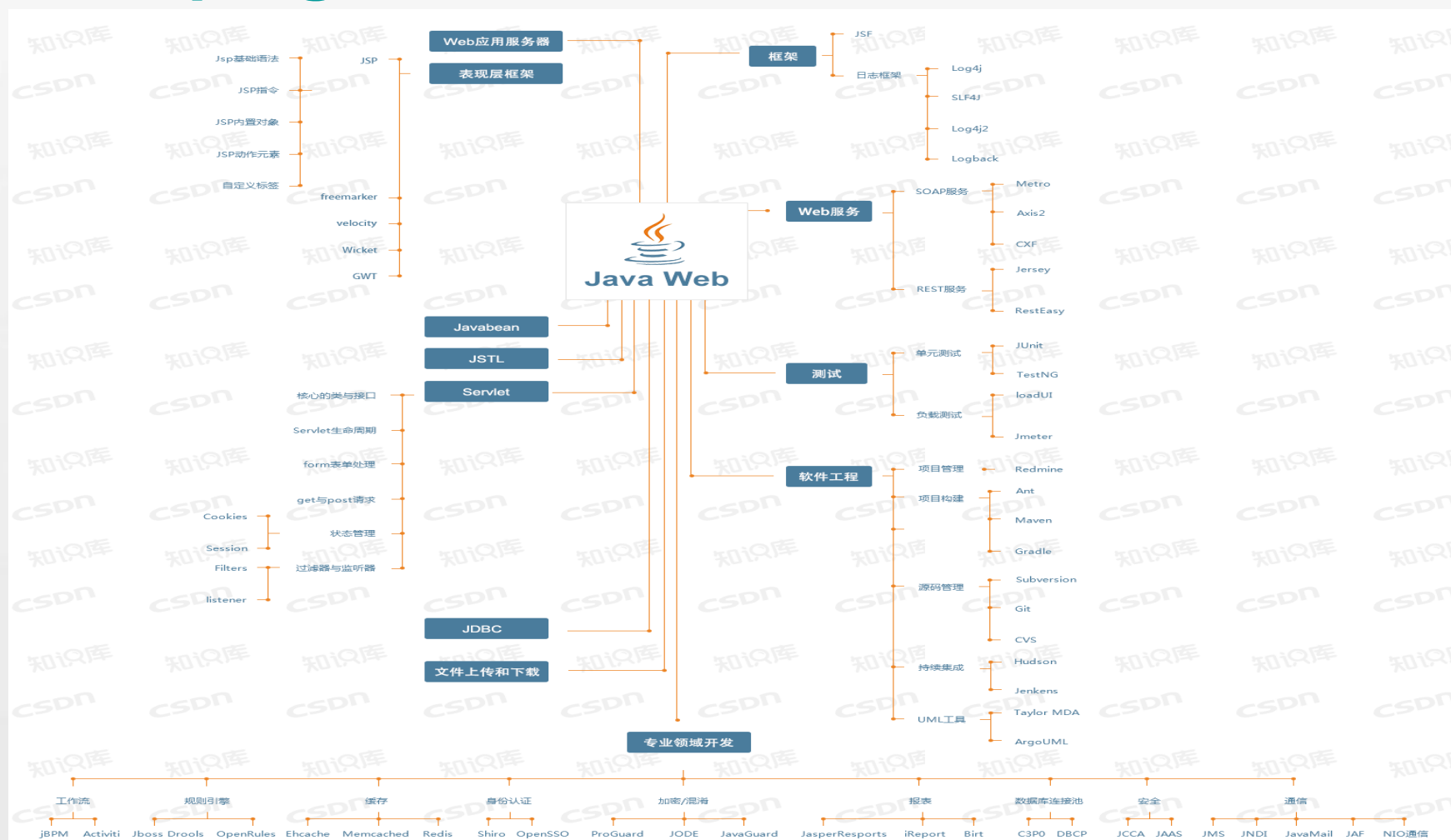
Jun 2016	Jun 2015	Change	Programming Language	Ratings	Change
1	1		Java	20.794%	+2.97%
2	2		C	12.376%	-4.41%
3	3		C++	6.199%	-1.56%
4	6	▲	Python	3.900%	-0.10%
5	4	▼	C#	3.786%	-1.27%
6	8	▲	PHP	3.227%	+0.36%
7	9	▲	JavaScript	2.583%	+0.29%
8	12	▲	Perl	2.395%	+0.64%
9	7	▼	Visual Basic .NET	2.353%	-0.82%
10	16	▲	Ruby	2.336%	+0.98%
11	11		Visual Basic	2.254%	+0.41%
12	23	▲	Assembly language	2.119%	+1.36%
13	10	▼	Delphi/Object Pascal	1.939%	+0.07%
14	14		Swift	1.831%	+0.39%
15	5	✖	Objective-C	1.704%	-2.64%
16	13	▼	R	1.540%	+0.02%
17	15	▼	MATLAB	1.447%	+0.01%
18	17	▼	PL/SQL	1.346%	+0.12%

各种语言发展的长期趋势



CSDN上各种语言的知识图谱

■ <http://geek.csdn.net/news/detail/96883>



编程规范

- [..\软件工程\(2014-2015 1\)\概要设计\设计规范\C++ 编程标准.doc](#)
- [..\软件工程\(2014-2015 1\)\概要设计\设计规范\Java编程标准.pdf](#)
- [..\软件工程\(2014-2015 1\)\概要设计\设计规范\华为软件编程规范.pdf](#)

编程规范

代码清单10-1 badly formatted code – big C

```
#include "stdafx.h"
#include "stdio.h"
void test();
int tmain
(int argc,
TCHAR* argv[])
{ test(); return
0; }

char C[25][40]; void d(int x,int y)
{C[x][y]=C[x][y+1]=32;} int f(int x){return (int)x*x*.08;}
void test(){int i,j; char s[5]="TEST";
for(i=0;i<25;i++)
for(j=0;j<40;j++)
C[i][j]=s[(i+j)%4];
for(i=1;i<=7;i++)
{d(18-i,12);
C[20-f(i)][i+19]=
C[20-f(i)][20-i]=32;
}d(10,13);d(9,13);
d(8,14);d(7,15);
d(6,16);d(5,18);d(5,20);
d(6,23);d(6,25);d(7,25);for(i=0;i<25;i++,printf("\n"))
for(j=0;j<40;j++,printf("%c",C[i][j++]))};
```

同学们纷纷发言，基本上有如下的反应：

- (1) Faint! !
- (2) 重写程序! !
- (3) 找到原作者，暴打一顿! ! !
- (4) 让此人从公司辞职! ! ! !

编程规范---C语言混乱代码大赛

```
#define _ -F<OO||—F—OO—;
int F=OO, OO=OO:main() {F_OO():printf(“%1.3f\n”, 4.*-F/OO/OO):}F_OO()
{
```

A diagram consisting of a large number of horizontal dashed lines. The lines are arranged in a triangular pattern, with the number of lines increasing from top to bottom. The top row has 1 line, and the bottom row has 15 lines. The lines are evenly spaced and extend across the width of the diagram.

编程规范——从某公司的代码审查标准来看

3.2 代码检查项目：

		检查项	分数	
				单项分
代码组织与风格	观感	1、 代码中是否进行了缩进 本项重点要求代码中进行缩进排版，保证代码的层次分布，发现有不合格的情况，每一处扣 1 分，扣完为止	5	10
		2、 函数之间是否有空行分割 要求函数之间有空行，空行不全的情况，只得 1 分	2	
		3、 函数内部是否分段组织 要求函数内部的代码按逻辑块，分段组织，各段之间空行；有分段组织但不全，可得 1 分	2	
		4、 函数过程是否太长，超过 100 行 在一个单元文件有一个函数不符合本要求，本项不得分	1	
	规范	1、 在同一源文件中，代码的 if/for/while 的缩进、组织风格是否一致 代码编写风格会因人而异，但是在同一个源文件中，要求编写人员如一保持同一风格；本项重点检查 if/for/while 的处理格式，发现有不一致的情况，每一处扣 1 分，扣完为止	5	10
		2、 代码续行符合规范的要求 代码续行指一行代码超长时，进行换行处理；重点检查 SQL 语句的组合、逻辑判断的情况；发现一处不符合要求，本项不得分	3	
		3、 代码空格使用符合规范的要求 代码空格重点检查运算符前后的空格处理，发现没有空格的运算符，就可扣除 1 分；大部分没有合格，本项不得分	2	
代码命名	观感	1、 是否有描述性的名称 代码编写中鼓励使用描述性的名称，一般变量的意义应该与变量注释相一致；发现有不一致的情况，每一处扣 1 分，扣完为止	5	10
		2、 有没有单字符变量名 发现一个单字符变量，扣除 1 分；同一源文件达到 3 个以上，本项不得分	2	
		3、 变量名是否采用大小写混合 大小写混合可以提高名称的可读性，发现单词组合的变量没有采用大小写混合，一处扣 1 分；大部分没有，本项不得分	2	
		4、 是否使用英文描述符 代码中大部分使用英文描述符，本项得 1 分	1	

编程规范——从某公司的代码审查标准来看

代码命名	观感	1、 是否有描述性的名称 代码编写中鼓励使用描述性的名称，一般变量的意义应该与变量注释相一致；发现有不一致的情况，每一处扣 1 分，扣完为止	5	10
		2、 有没有单字符变量名 发现一个单字符变量，扣除 1 分；同一源文件达到 3 个以上，本项不得分	2	
		3、 变量名是否采用大小写混合 大小写混合可以提高名称的可读性，发现单词组合的变量没有采用大小写混合，一处扣 1 分；大部分没有，本项不得分	2	
		4、 是否使用英文描述符 代码中大部分使用英文描述符，本项得 1 分	1	
	规范	1、 使用类型前缀 变量、属性应该按照规范加上类型前缀，有一处可得 1 分；大部分都有，可得 2 分	2	10
		2、 文件名、类名、方法名符合规范 本项主要强调各种名称的描述性，名称的意义应该与注释相一致，命名方法也要符合规范；其中， 文件名 1 分 类名 1 分 方法名 2 分 酌情给分	4	
		3、 变量名、常量名符合规范 变量名、常量名都有明确的规定，发现一处不合格扣除 1 分，扣完为止	4	
声明		1、 对代码的声明（类、方法、变量）是否进行了分类分组； 源代码中的声明可以按照类别给予区分，各个类别之间的应该用空行分割，重点是大批变量的声明；发现一处分组分类，可以给 3 分，全部有给 5 分	5	10

编程规范——从某公司的代码审查标准来看

		2、 声明的顺序是否符合规范 声明的顺序应该按照规范来安排，本顺序重点指类成员的声明顺序；完全符合规范得 5 分，部分不符合扣 2 分，完全错乱不得分	5	
注释	观感	1、 是否有文件头部的注释 发现源文件没有头注释，本项不得分；头部注释不全，得 5 分	10	30
		2、 是否有函数过程的注释 要求每个函数都应该有注释，重点在函数功能的说明；发现一处函数注释没有扣 3 分，扣完为止	10	
		3、 是否有变量注释 发现一处变量没有注释扣 2 分，扣完为止 全部变量没有注释，不得分	10	
	规范	1、 注释格式是否符合规范 按照规范，本项重点检查文件头和函数注释，其中 文件头： 2 分 函数： 3 分 其他： 1 分 酌情给分	6	20
		2、 作者、变更人员是否署名 作者署名加 3 分 发现有变更情况并有署名的加 3 分	6	
4、 注释量是否达到 10%（在任何 20 行连续代码中有一行注释） 发现一处扣除 2 分，扣完为止		8		
总分：				100

- 以上检查项，如果源代码非常的简单，或者该项不用于此源代码，可以认为合格（得分）。

代码简洁之道(1/5)

《代码整洁之道》知识体系剖解(1)

编写代码的难度，取决于周边代码的阅读难度。想要快速实现需求，想要快速完成任务，想要轻松的写代码，先让你书写的代码整洁易读。

保持整洁的习惯，发现脏代码就要及时纠正。花时间保持代码代码整洁，这不但有关效率，还有关项目的生存。

程序员遵从不了解混乱风险的产品经理(策划)的意愿，都是不专业的做法。

让代码比你来时更干净：如果每次签入时，代码都比签出时干净，那么代码就不会腐坏。

赶上期限的唯一方法，做得更快的唯一方法，就是始终尽可能保持代码的整洁。

<http://blog.csdn.net/>

代码简洁之道(2/5)

《代码整洁之道》知识体系剖解(2):命名

【要点一】要名副其实。一个好的变量、函数或类的名称应该已经答复了所有的大问题：可以大概告诉你这个名称所代表的内容，为什么会存在，做了什么事情，应该如何使用等。

【要点二】要避免误导。我们应该避免留下隐藏代码本意的错误线索，也应该避免使用与本意相悖的词。

【要点三】尽量做有意义的区分。尽量避免使用数字系列命名（a1、a2.....aN）和没有意义的区分。

【要点四】尽量使用读得出来的名称。如名称读不出来，讨论的时候会不方便且很尴尬。

【要点五】尽量使用可搜索的名称。名称长短应与其作用域大小相对应，若变量或常量可能在代码中多处使用，应赋予其以便于搜索的名称。

【要点六】取名不要绕弯子。取名要直白，要直截了当，明确就是王道。

【要点七】类名尽量用名词。类名尽量用名词或名词短语，最好不要是动词。

【要点八】方法名尽量用动词。方法名尽量用动词或动词短语。

【要点九】每个概念对应一词，并一以贯之。对于那些会用到你代码的程序员，一以贯之的命名法简直就是天降福音。

【要点十】通俗易懂是王道。应尽力写出易于理解的代码，想把代码写得让别人能一目了然，而不必让人去非常费力地去揣摩其含义。

【要点十一】尽情使用解决方案领域专业术语。尽管去用那些计算机科学领域的专业术语、算法名、模式名、数学术语。

【要点十二】要添加有意义的语境。需要用有良好命名的类，函数或名称空间来放置名称，给读者提供语境。若没能提供放置的地方，还可以给名称添加前缀。

<http://blog.csdn.net/>

代码简洁之道 (3/5)

《代码整洁之道》知识体系剖解(3):函数

第一原则：短小。最好将单个函数控制在10行以内。

第二原则：单一职责。函数应该只做一件事情。只做一件事，做好这件事。

第三原则：具描述性的名称。长而具有描述性的名称，比短而令人费解的名称好。当然，如果函数在已经足够说明问题，还是越短越好。

第四原则：参数尽可能少。最理想的函数参数形态是零参数，其次是单参数，再次是双参数，应尽量避免三参数及以上参数的函数，有足够的理由才能用三个以上参数。

第五原则：尽力避免重复。重复的代码会导致模块的臃肿，整个模块的可读性可能会因为重复的消除而得到提升。

<http://blog.csdn.net/>

代码简洁之道(4/5)

《代码整洁之道》知识体系剖解(4):格式

第一原则：像报纸一样一目了然

优秀的源文件也要像报纸文章一样，名称应当简单并且一目了然，名称本身应该足够告诉我们是否在正确的模块中。

源文件最顶部应该给出高层次概念和算法。细节应该往下渐次展开，直至找到源文件中最底层的函数和细节。

第二原则：恰如其分的注释

带有少量注释的整洁而有力的代码，比带有大量注释的零碎而复杂的代码更加优秀。

第三原则：合适的单文件行数

尽可能用几百行以内的单文件来构造出色的系统，因为短文件通常比长文件更易于理解。

第四原则：合理地利用空白行

在每个命名空间、类、函数之间，都需用空白行隔开。

第五原则：让紧密相关的代码相互靠近

靠近的代码行暗示着他们之间的紧密联系。所以，紧密相关的代码应该相互靠近。

第六原则：基于关联的代码分布

变量的声明应尽可能靠近其使用位置。

循环中的控制变量应该在循环语句中声明。

短函数中的本地变量应当在函数的顶部声明。

对于某些长函数，变量也可以在某代码块的顶部，或在循环之前声明。

实体变量应当在类的顶部声明。

若某个函数调用了另一个，就应该把它们放到一起，且调用者应该尽量放到被调用者之上。

概念相关的代码应该放到一起。相关性越强，则彼此之间的距离就越短。

第七原则：团队遵从同一套代码规范

一个好的团队应当约定与遵从一套代码规范，并且每个成员都应当采用此风格。

<http://blog.csdn.net/>

代码简洁之道 (5/5)

《代码整洁之道》知识体系剖解(5):类

原则一：合理地分布类中的代码。类中代码的分布顺序大致是：

1. 公有静态常量
2. 私有静态变量
3. 私有普通变量
4. 公共函数
5. 私有函数

原则二：尽可能地保持类的封装。尽可能使函数或变量保持私有，不对外暴露太多细节

原则三：类应该短小，尽量保持单一权责原则。类或模块应有且只有一条加以修改的理由。

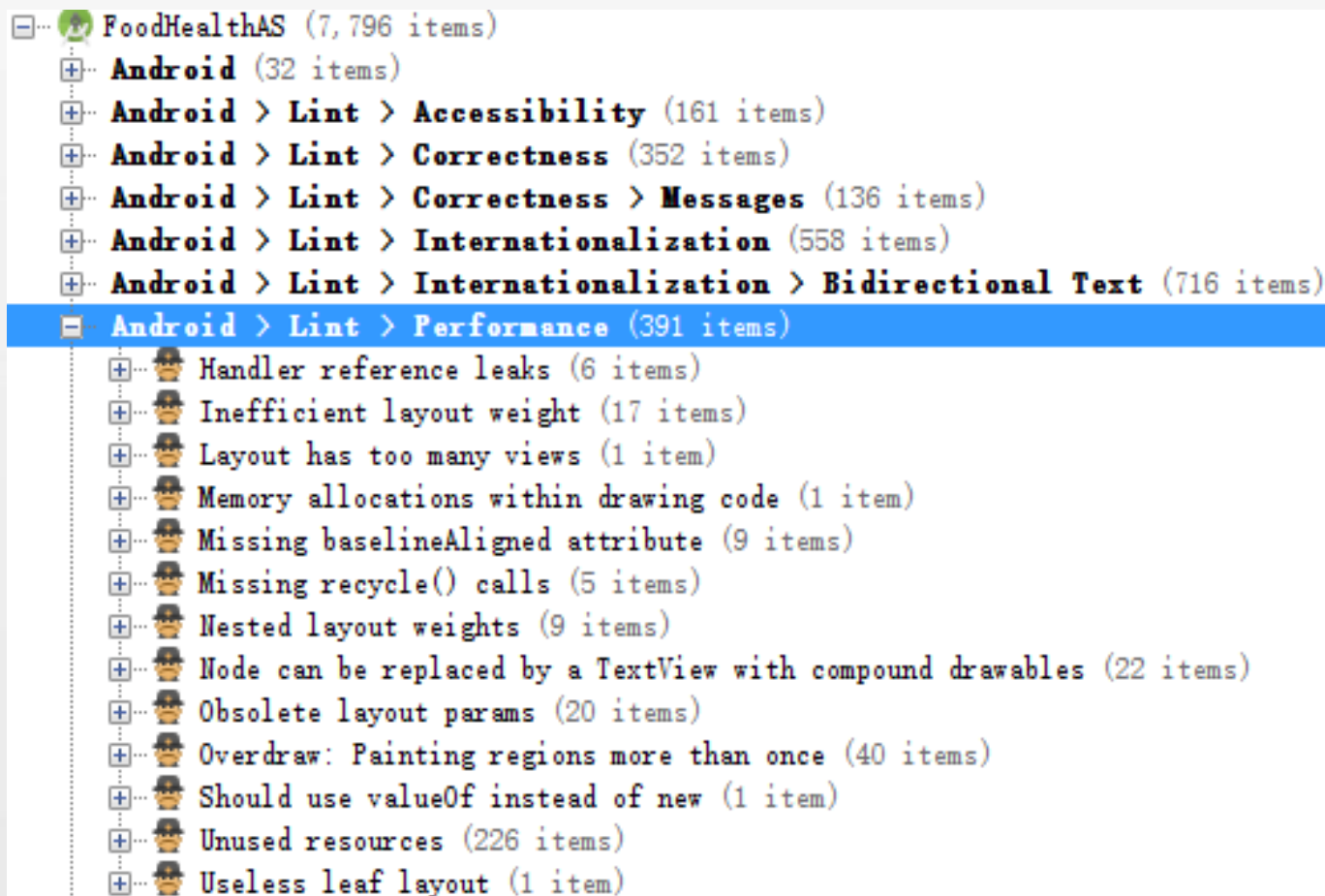
原则四：合理提高类的内聚性。我们希望类的内聚性保持在较高的水平。内聚性高，表示类中方法和变量相互依赖，相互结合成一个逻辑整体。

原则五：有效地隔离修改。类应该依赖于抽象，而不是依赖于具体细节。尽量对设计解耦，做好系统中的元素的相互隔离，做到更加灵活与可复用。

<http://blog.csdn.net/>

程序效能分析-以android开发为例

■ 代码检查(inspect code)---其中的性能问题



程序效能分析-代码检查-潜在的BUG

找到的潜在BUG类型

展开其中的某种类型
(可能会导致NULL值)

Probable bugs (359 items)

- Constant conditions & exceptions (84 items)
- Mismatched query and update of collection (1 item)
- Mismatched query and update of StringBuilder (1 item)
- Mismatched read and write of array (1 item)
- @NotNull/@Nullable problems (1 item)
- Result of method call ignored (18 items)
- Statement with empty body (90 items)
- String comparison using '=', instead of 'equals()' (17 items)
- Suspicious variable/parameter name combination (1 item)
- Unused assignment (145 items)

Constant conditions & exceptions (84 items)

- AboutActivity (1 item)
- AddMonitorDataActivity (1 item)
- Blood_pressureActivity (2 items)
- Blood_SugarActivity (2 items)
- Caffe_test (4 items)
- CameraActivity (3 items)
- Method invocation 'getExternalFilesDir(Environment.DIRECTORY_DCIM).getPath()' may produce 'java.lang.NullPointerException'
- Condition 'fileExists == false' is always 'true'
- Value 'fileExists' is always 'false'

Name

```
public method void onPictureTaken(byte[] data, Camera camera)
```

Location

```
class anonymous in void captrue() (com.shlr0.caffe_android_demo.CameraActivity)
```

Problem synopsis

```
Method invocation 'getExternalFilesDir(Environment.DIRECTORY_DCIM).getPath()' at line 610 may produce 'java.lang.NullPointerException'
```

Suppress

[Suppress for statement](#)

[Suppress for statement with comment](#)

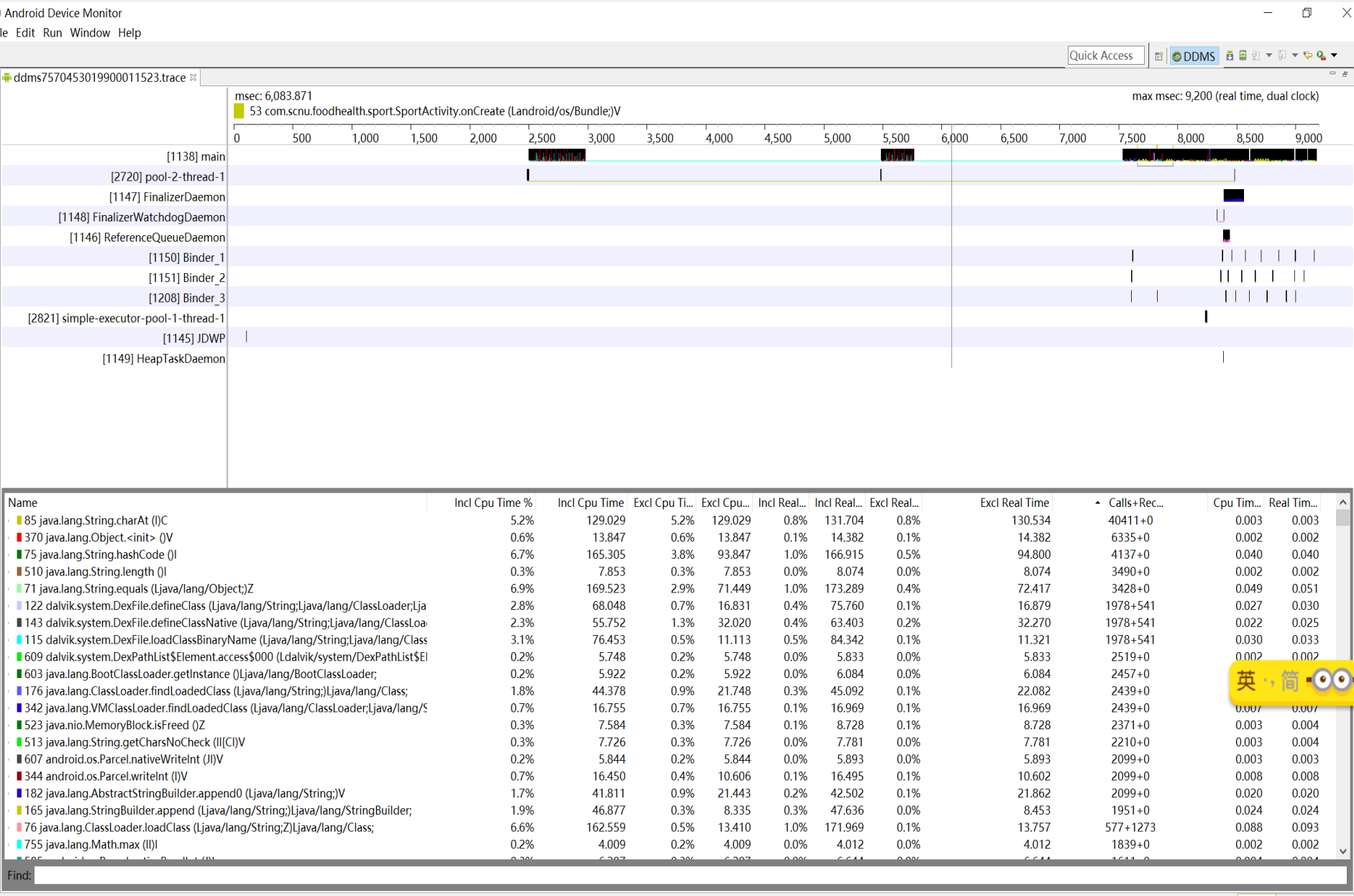
[Suppress for member](#)

[Suppress for class](#)

[Suppress all inspections for class](#)

定位到具体的代码并给出推荐的解决方法

程序性能分析—android studio中的TraceView



程序性能分析—android studio中的TraceView

■ 各个字段的含义

➤ **Incl Cpu Time** –方法总的执行时间

➤ **Calls + Recur Calls / Total**

- **Call** –方法的调用次数

- **Recur Calls**-方法的递归调用次数

➤ **Cpu Time / Call** –方法的**CPU**占用时间

- **CPU**占用时间/方法的调用次数，可得到方法每次调用小号的时间

程序性能分析—方法调用次数分析

Name	Incl Cp...	Incl Cp...	Excl ...	Excl Cpu...	Incl Real...	Incl Real...	Excl Real...	Excl Real...	Calls+Rec...
75 com.scnu.foodhealth.utils.MyThread.isFinishData ()Z	2.6%	160.300	2.6%	160.300	0.1%	160.426	0.1%	160.426	62240+0
84 java.lang.String.charAt (I)C	2.3%	140.449	2.3%	140.449	0.1%	152.360	0.1%	141.405	24287+0
339 java.lang.String.length ()I	0.3%	15.429	0.3%	15.429	0.0%	18.443	0.0%	15.891	4100+0
338 java.lang.Object.<init> ()V	0.3%	15.432	0.3%	15.432	0.0%	19.113	0.0%	16.056	3092+0
305 java.lang.String.getCharsNoCheck (II[CI)V	0.3%	16.867	0.3%	16.867	0.0%	17.176	0.0%	16.902	2758+0

调用次数分析：

1、方法isFinishData调用了62240次数，应该仔细分析该方法的调用方式，找到一下代码：

```
while (!mThread.isFinishData())
```

分析代码发现，，程序的设计目的的等待线程执行完成。大家看看应该如何修改；如果不修改的话，在修改isFinishData这个函数时，应注意什么问题。

2.标准库中的charAt()和String.length函数调用了24287和4100次，应检查是否在循环调用是否可以优化，如：

```
(for i=0;i<String.length();i++)
```


程序性能分析—方法调用时间分析

Name	Incl Cp...	Incl Cp...	Excl ...	Excl Cpu...	Incl Real...	Incl Real...	Excl Real...	Excl Real...	Calls+RecurCalls/Total	▲ Cpu Time/Call	Real Time/Call
2 android.os.AsyncTask\$SerialExecutor\$1.run ()V	33.6%	2037.768	0.0%	0.013	2.4%	3266.389	0.0%	0.015	1+0	2037.768	3266.389
3 java.util.concurrent.FutureTask.run ()V	33.6%	2037.741	0.0%	0.024	2.4%	3266.360	0.0%	0.022	1+0	2037.741	3266.360
4 android.os.AsyncTask\$2.call ()Ljava/lang/Object;	33.6%	2037.652	0.0%	0.075	2.4%	3266.272	0.0%	0.074	1+0	2037.652	3266.272
5 com.sh1r0.caffe_android_demo.CameraActivity\$CNNTask.doloInBackground ()I	33.6%	2037.297	0.0%	0.019	2.4%	3265.502	0.0%	0.017	1+0	2037.297	3265.502
6 com.sh1r0.caffe_android_demo.CameraActivity\$CNNTask.doloInBackground ()I	33.6%	2037.278	0.0%	0.052	2.4%	3265.485	0.0%	0.053	1+0	2037.278	3265.485
7 com.sh1r0.caffe_android_lib.CaffeMobile.predictImage (Ljava/lang/String;)I	33.6%	2037.190	0.0%	0.040	2.4%	3265.396	0.0%	0.042	1+0	2037.190	3265.396
8 com.sh1r0.caffe_android_lib.CaffeMobile.predictImage (Ljava/lang/String;)I	33.6%	2037.150	33.6%	2037.150	2.4%	3265.354	1.5%	2037.150	1+0	2037.150	3265.354

有多个方法的调用时间均为2000毫秒以上，展开方法onPictureTaken方法占用时间比较多，展开该方法，发现ProgressDialog.show方法占用了57.9%的时间，所以应优化或取消ProgressDialog。

Name	Incl Cp...	Incl Cp...	Excl ...
20 com.sh1r0.caffe_android_demo.CameraActivity\$6.onPictureTaken ((Landroid/hardware/Camera;)V	13.4%	809.607	0.2%
Parents			
19 android.hardware.Camera\$EventHandler.handleMessage (Landroid/os/Message;)V	100.0%	809.607	
Children			
self	1.2%	9.739	
26 android.app.ProgressDialog.show (Landroid/content/Context;Ljava/lang/CharSequence;Ljava/lang/CharSequence;Z)Landroid/app/ProgressDialog;	57.9%	469.001	
105 com.sh1r0.caffe_android_demo.CameraActivity.access\$1600 (Lcom/sh1r0/caffe_android_demo/CameraActivity;Landroid/hardware/Camera;Landroid/view/Si	13.4%	108.463	
106 com.sh1r0.caffe_android_demo.utils.BitmapUtils.saveJPG_After (Landroid/content/Context;Landroid/graphics/Bitmap;Ljava/lang/String;)V	13.3%	107.693	
131 com.sh1r0.caffe_android_demo.utils.CameraUtil.setTakePicktrueOrientation ((Landroid/graphics/Bitmap;)Landroid/graphics/Bitmap;	9.0%	72.778	
229 android.graphics.BitmapFactory.decodeByteArray (([B])Landroid/graphics/Bitmap;	3.0%	24.235	
368 java.lang.ClassLoader.loadClass (Ljava/lang/String;)Ljava/lang/Class;	1.1%	8.560	
546 android.content.ContextWrapper.getExternalFilesDir (Ljava/lang/String;)Ljava/io/File;	0.8%	6.794	
1464 android.os.AsyncTask.execute ((Ljava/lang/Object;)Landroid/os/AsyncTask;	0.1%	0.802	
1257 android.hardware.Camera.cancelAutoFocus ()V	0.0%	0.283	
2324 java.lang.String.valueOf ()Ljava/lang/String;	0.0%	0.178	
1563 android.util.Log.d (Ljava/lang/String;Ljava/lang/String;)I	0.0%	0.158	
111 java.lang.StringBuilder.append (Ljava/lang/String;)Ljava/lang/StringBuilder;	0.0%	0.150	
2449 com.sh1r0.caffe_android_demo.CameraActivity\$CNNTask.<init> (Lcom/sh1r0/caffe_android_demo/CameraActivity;Lcom/sh1r0/caffe_android_demo/CNNI	0.0%	0.141	
2479 android.graphics.Bitmap.recycle ()V	0.0%	0.134	

缺陷管理

- 软件测试中经常使用各种术语来描述软件出现的问题, 如下一些通用的术语:
 - ◆ 软件错误(Software Error)
 - ◆ 软件缺陷(Software Defect)
 - ◆ 软件故障(Software fault)
 - ◆ 软件失效(Software failure)

- 区分这些术语很重要, 它关系到测试工程师对软件失效现象与机理的深刻理解. 由于软件内部逻辑复杂, 运行环境动态变化, 且不同的软件差异可能很大, 因而软件失效的机理可能也有不同的表现形式, 但总的来说, 软件失效的机理可描述为:
 - 软件错误→软件缺陷→软件故障→软件失效

缺陷管理

软件错误:在可以预见的时期内, 软件将有人来开发. 在整个生存期的各个阶段, 都贯穿 着人的直接或间接的干预. 然而人难免犯错误, 这必然给软件留下不良的痕迹. 软件错误是指在软件生存期内的不希望或不可接受的人为错误, 其结果是导致软件缺陷的产生. 可见, 软件错误是一种人为过程, 相对于软件本身, 是一种外部行为.

软件缺陷:软件缺陷是存在于软件(文档, 数据, 程序)之中的那些不希望或不可接受的偏差. 其结果是软件运行于某一特定条件时出现软件故障, 这时称软件缺陷被激活.

软件故障:软件故障是指软件运行过程中出现的一种不希望或不可接受的内部状态. 比如: 软件处于执行一个多余循环过程时, 我们可以软件出现故障. 若此时没有适当的措施(容错)加以处理, 便产生软件失效. 软件故障是一种动态行为.

软件失效:软件失效是指软件运行时产生的一种不希望或不可接受的外部行为结果.

综上所述, 软件错误是一种人为错误. 一个软件错误必定产生一个或多个软件缺陷. 当一个软件缺陷被激活时, 便产生一个软件故障; 同一个软件缺陷在不同条件下被激活, 可能产生不同的软件故障. 软件故障如果没有及时容错措施加以处理, 便不可避免地导致软件失效.

什么是Bug?

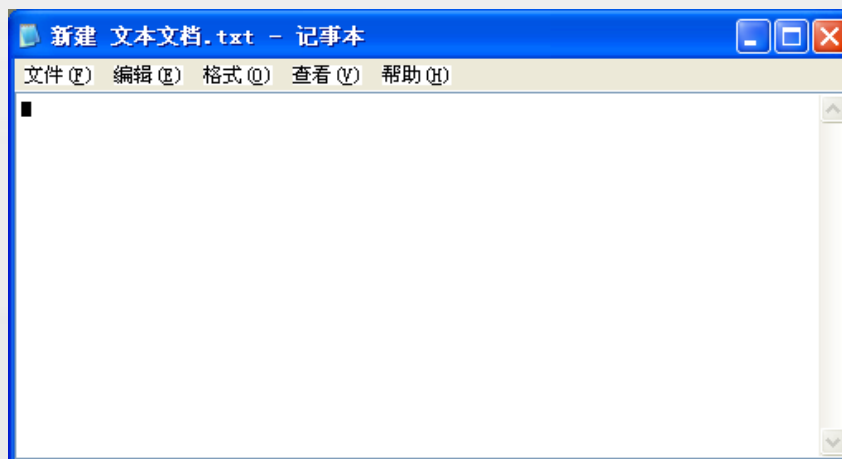
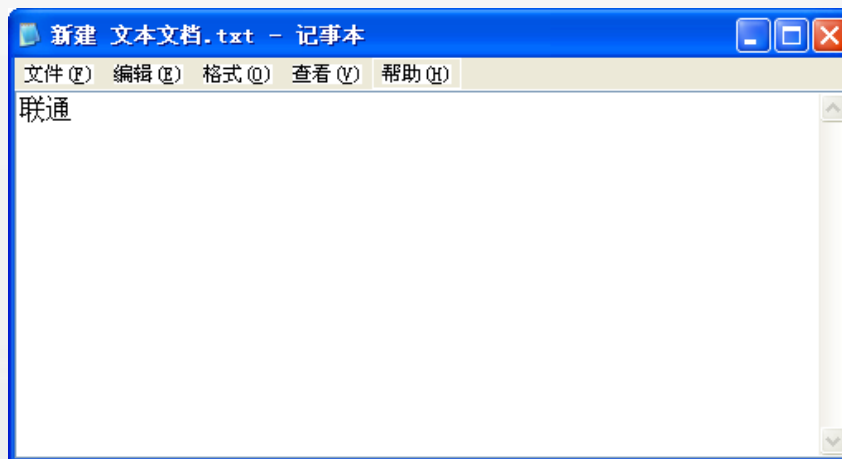
- 功能没有实现或与规格说明不一致的问题是bug;
- 不能工作(死机、没反应)的部分是bug;
- 不兼容的部分是bug;
- 边界条件未做处理是bug;
- 界面、消息、提示、帮助不够准确是bug;
- 屏幕显示、打印结果不正确也是bug;
- 有时把尚未完成的工作也作为一个bug。

Bug举例₁

文本文件保存错误:

在WindowsXP桌面上新建一个文本文档，输入“联通”两个字，并保存退出。

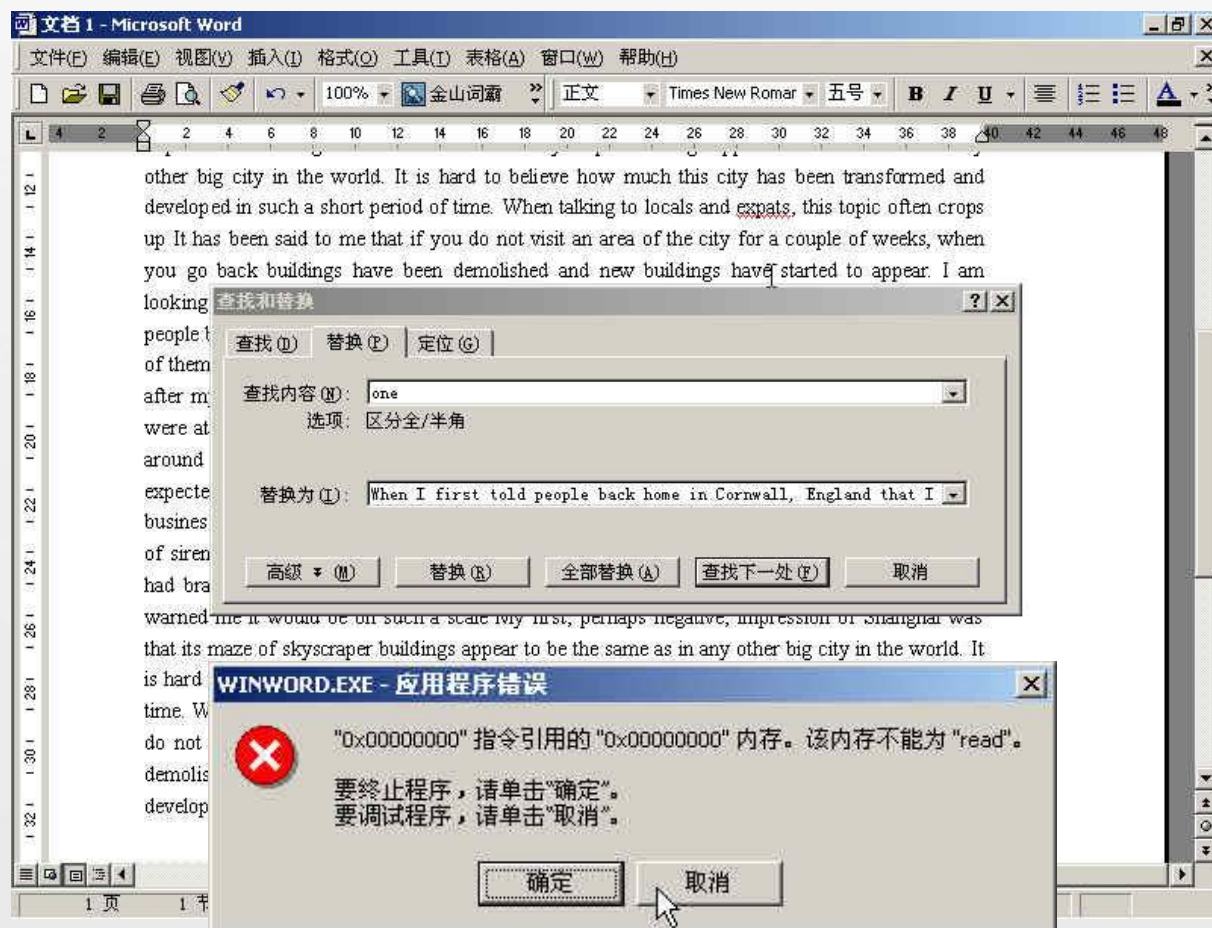
退出后再次打开这个文本文件时，刚才输入的内容变成了乱码。



Bug举例₃

替换字符串长度未作限定：

Word2000中，如果替换字符串长度过长，则会引起程序崩溃。

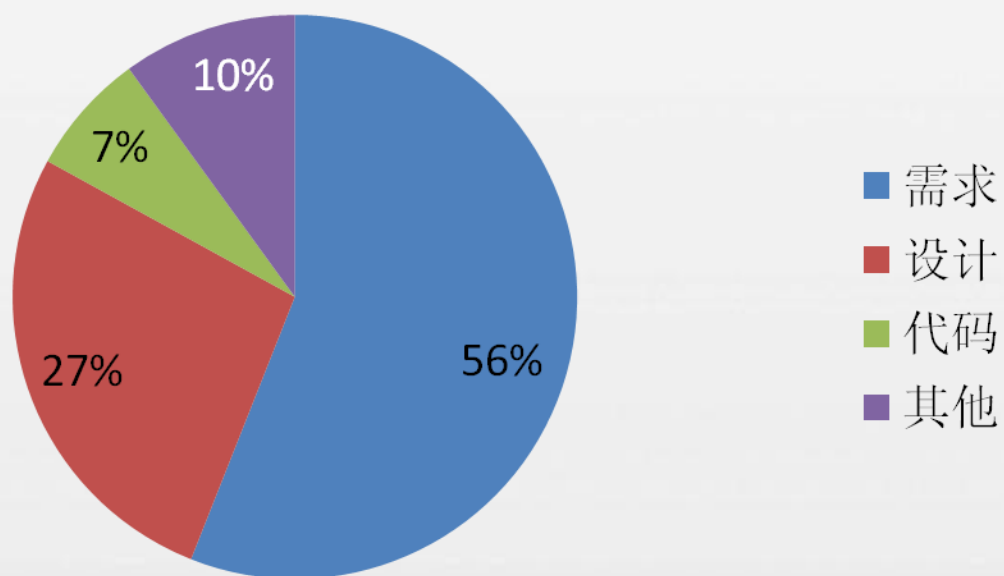


软件缺陷的判定

- (1) 软件未达到产品说明书中已经标明的功能；
- (2) 软件出现了产品说明书中指明不会出现的错误；
- (3) 软件未达到产品说明书中虽未指出但应当达到的目标；
- (4) 软件功能超出了产品说明书中指明的范围；
- (5) 软件测试人员认为软件难以理解、不易使用，或者最终用户认为该软件使用效果不良。

■ 总之，软件缺陷是软件开发过程中的“副产品”，会导致软件产品在某种程度上不能满足用户的需要，导致对软件产品预期属性的偏离。

缺陷原因分布图



缺陷管理-缺陷来源介绍

⑩ 缺陷来源	描述	缩写
⑩ Cause-Requirement	由于需求的问题引起的缺陷	C-R
⑩ Cause - Design	由于设计的问题引起的缺陷	C-D
⑩ Cause - Code	由于编码的问题引起的缺陷	C-C
⑩ Cause - Test	由于测试的问题引起的缺陷（测试用例设计问题等）	C-T
⑩ Cause - Integration & Other	由于集成或其它问题引起的缺陷	C-I&O

软件缺陷的范畴

■ 包括检测缺陷和残留缺陷

- 检测缺陷：软件在进入用户使用之前被检测出的缺陷
- 残留缺陷：软件发布后存在的缺陷，包括在用户安装前未被检测出的缺陷以及检测出但未被修复的缺陷。

造成残留缺陷的原因

- 软件错误/缺陷很难看到
- 软件错误/缺陷看到了但很难抓到
- 软件错误/缺陷抓到了但无法修改或很难修改

缺陷管理-目的

- 缺陷管理目的：
- 缺陷管理目的是对各阶段测试发现的缺陷进行跟踪管理，以保证各级缺陷的修复率达到标准。主要实现以下目标：
 - 及时了解并跟踪每个被发现的缺陷；
 - 确保每个被发现的缺陷都能被处理；
 - 收集缺陷数据并根据缺陷趋势曲线识别测试过程阶段；
 - 收集缺陷数据并在其上进行分析，作为组织过程的财富。

软件问题报告（Bug报告）

- 软件问题（Bug）报告是软件测试过程中最重要的文档。它记录了Bug发生的环境，如各种资源的配置情况，Bug的再现步骤以及Bug性质的说明。
- 更重要的是它还记录着Bug的处理过程和状态。Bug的处理进程从一定角度反映了测试的进程和被测软件的质量状况以及改善过程。

报告Bug的基本原则

- 尽快报告Bug;
- 有效描述Bug;
- 对Bug不做任何评价;
- 确保Bug可以重现。

有效描述Bug

- 短小：只解释事实和演示、描述Bug必需的细节；
- 单一：每一个报告中针对一个Bug；
- 步骤清晰：要清楚地描述出Bug的发生场景，包括前置条件和操作的详细步骤；
- 必要的时候可以添加注释（remarks）；
- 可以上载屏幕抓图和其他附件。

有效描述Bug

■ 以下是一个Bug描述的例子

1. 操作步骤:
2. 使用MappingBuilder对URL为“jdbc:mysql://10.0.0.12/test”的数据库进行映射, 虚拟数据库名称设置为“VMysql”。
3. 进入DataView主页面, 在DAS List中点击“VMysql”右侧的“高级查询”链接。
4. 在高级查询页面底端的输入框中, 输入SQL语句“select * from empinfo”, 点击查询按钮。
5. 在得到的查询结果页面中, 点击“下一页”链接。
6. 翻页到下一页后, 没有出现“保存当前页面的查询结果”链接, 无法保存当前页面结果。

缺陷管理-缺陷相关属性

缺陷属性	描述
缺陷描叙 (Summary)	简单描述缺陷，主要是什么缺陷
缺陷发现提交者 (Detected By)	描叙缺陷是由谁发现提出的。
缺陷发现时间 (Detected on Date)	描叙缺陷发现提出时间。
缺陷严重性 (Severity)	描述缺陷的严重性。
缺陷分给谁 (Assigned to)	指缺陷分派给谁。
缺陷在哪个版本发现 (Detected in Version)	描叙缺陷发现的版本
缺陷被修改的时间 (Modified)	描叙缺陷被修改的时间。
计划修复时间 (Plan fixed Data)	描叙缺陷计划完成修复的时间。
缺陷优先级 (priority)	描述缺陷的优先级。
缺陷所属项目 (Project)	描述缺陷所属的工程。
是否是重现缺陷 (Reproducible)	描述缺陷是否是重现缺陷。
缺陷的状态 (Status)	描述缺陷的状态
缺陷所属于的模块 (subject)	描述缺陷所属的模块。
缺陷详细描述 (Description)	缺陷详细描述，包括缺陷产生的步骤，缺陷的实际结果，缺陷的理想结果，建议等。
缺陷实际关闭的版本 (Closed in Version)	描述缺陷实际关闭的版本。
缺陷实际修复所花的时间 (Actual Fixed Time)	描述缺陷实际修复所花的时间
缺陷修复完成时间 (Closing Date)	描述缺陷实际关闭的时间。
注释 (Comments)	描叙对缺陷的注释。
附件 (Attachments)	添加缺陷附件。

缺陷管理-缺陷等级定义

等级	说明	现象描述（部分例子）	优先级
A类	致命错误	<ul style="list-style-type: none">✧ 由于程序所引起的死机, 非法退出;✧ 死循环;✧ 数据库发生死锁;✧ 因错误操作导致的程序中中断;✧ 与数据库连接错误;✧ 数据通讯错误;✧ 导致测试无法继续执行。✧ 可能影响其他模块功能。	立即处理或解决
B类	很严重的错误	<ul style="list-style-type: none">✧ 程序错误;✧ 程序接口错误;✧ 数据库的表、业务规则、缺省值未加完整性等约束条件;✧ 关键功能完全不能实现;✧ 程序运行不稳定, 如出现不可继续进行操作的错误;✧ 程序运行出现难以捕捉和不可再现的错误;✧ 响应其他业务流程的错误。	在发现的两天内完成。
C类	一般严重错误	<ul style="list-style-type: none">✧ 操作界面错误（包括数据窗口内列名定义、含义是否一致）✧ 打印内容、格式错误✧ 简单的输入限制未放在前台进行控制✧ 删除/退出操作未给出提示✧ 数据库表中有过多的空字段✧ 功能不完整, 如菜单、按钮不响应✧ 对错误没有处理信息	系统上线前必须修复完成
D类	一般性错误	<ul style="list-style-type: none">✧ 界面不规范;✧ 辅助说明描述不清楚;✧ 输入输出不规范;✧ 提示窗口文字未采用行业术语;✧ 可输入区域和只读区域没有明显的区分标志。	正常排队等待修复或方便时修复
E类	较小错误	<ul style="list-style-type: none">✧ Tab键跳转不正常;✧ 窗口控件的Z-Order不正确; ;✧ 窗口中的按钮或者控件缺少快捷字母, 或快捷字母冲突;✧ 文字表述中有错别字或歧义;✧ 测试人员所提出的建设性意见。	方便时再修复

缺陷管理-缺陷修复优先级

优先级	描述
紧急 (5-Urgent)	缺陷很紧急且很严重，得立即修复。
很高优先级 (4-very High)	例如，软件的主要功能错误或者造成软件崩溃，数据丢失的缺陷。
较高优先级 (3-High)	例如，影响软件功能和性能的一般缺陷。
一般优先级 (2-Medium)	例如，本地化软件的某些字符没有翻译或者翻译不准确的缺陷。
低优先级 (1-Low)	例如，对软件的质量影响非常轻微或出现几率很低的缺陷。

Bug的生命周期

Bug的生命周期就是指Bug从开始提出到最后完全解决，并通过复查的过程。在这个过程中Bug报告的状态不断发生着变化，记录着Bug的处理进程。

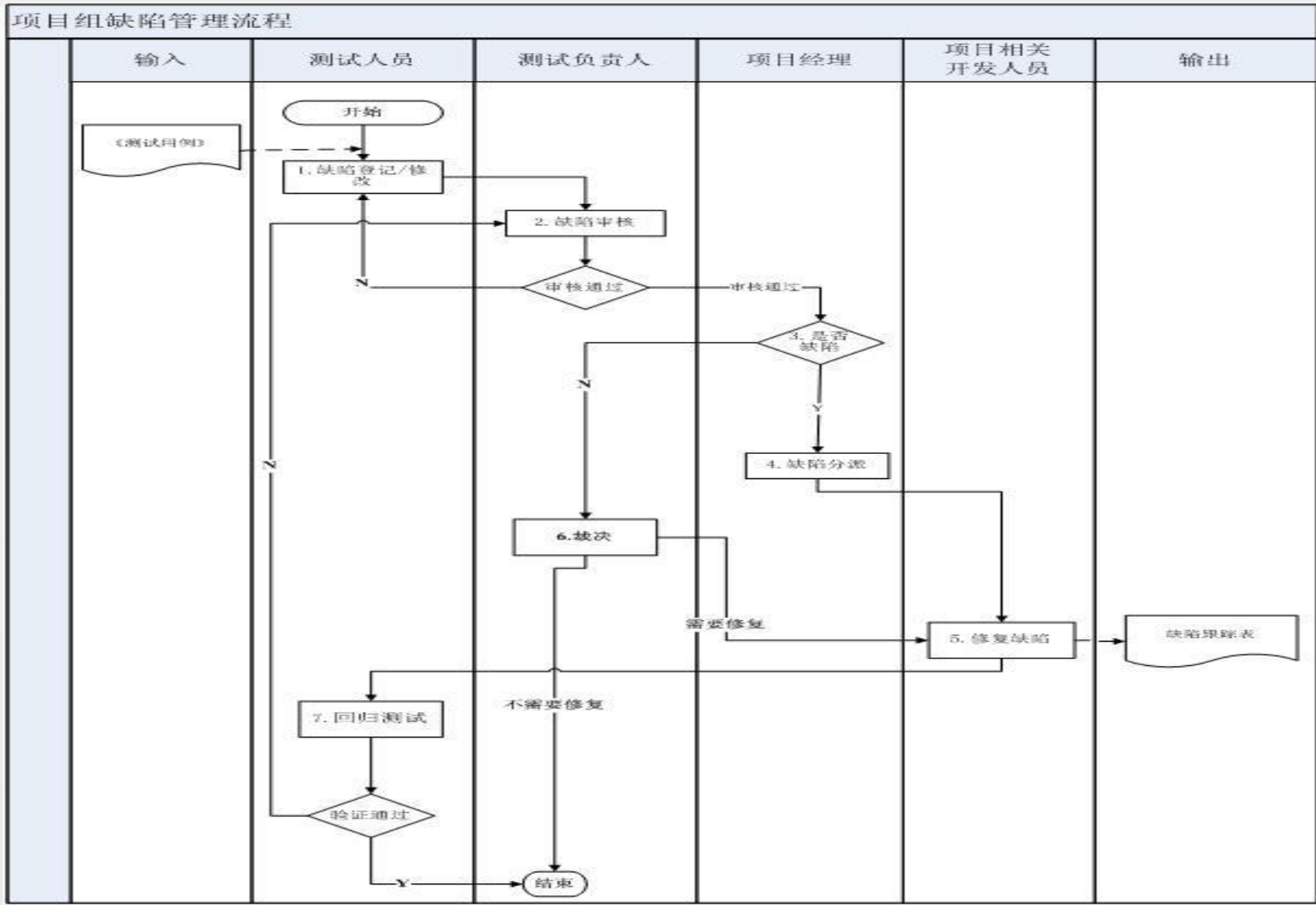
缺陷管理-缺陷状态

缺陷状态	描述
新提交 (New)	新提交的缺陷状态
激活 (Open)	缺陷已提交，正在处理
已拒绝 (Rejected)	拒绝“已提交的缺陷”，不需要修改或不是缺陷
已解决 (Fixed)	缺陷已修改
重激活 (Reopen)	缺陷修改未通过再测试， 或因其他原因造成缺陷再次打开
重复缺陷 (Duplicate)	缺陷重复出现，已经被提交过。
已关闭 (Closed)	确认缺陷已被修复，将其关闭

缺陷管理-人员职责

- 项目经理（PM）
 - 负责指派缺陷给相关责任人。
- 项目测试负责人（TM）
 - 决定缺陷管理方式和工具，拟定决策评审计划；
 - 管理所有缺陷关闭情况；
 - 审核测试人员提交的缺陷；
 - 对测试人员的工作质量进行跟踪与评价。
- 测试人员（TE）
 - 负责报告系统缺陷记录，且协助项目人员进行缺陷定位；
 - 负责验证缺陷修复情况，且填写缺陷记录中相应信息；
 - 负责执行系统回归测试；
 - 提交缺陷报告；
 - 负责被测软件进行质量数据和分析。
- 项目相关开发人员（DE）
 - 修改测试发现的缺陷，并提交成果物做再测试；
 - 负责接收各自的缺陷记录，并且修改；
 - 负责提供缺陷记录跟踪中其它相应信息。
- 质量保证人员（SQA）
 - 监控项目组缺陷管理规程执行情况。
- 监控项目组缺陷管理规程执行情况。

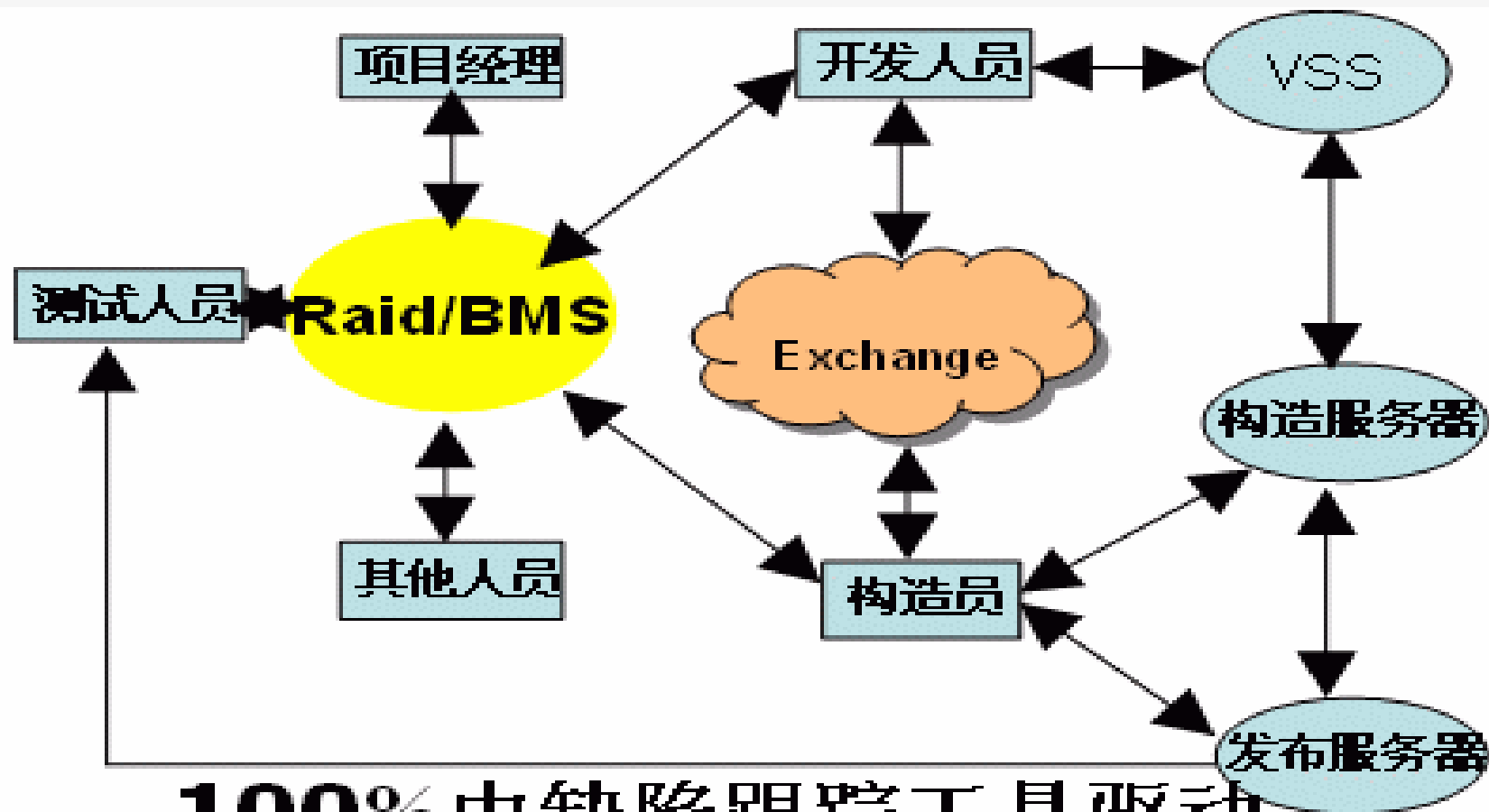
缺陷管理-流程图



缺陷管理-过程介绍

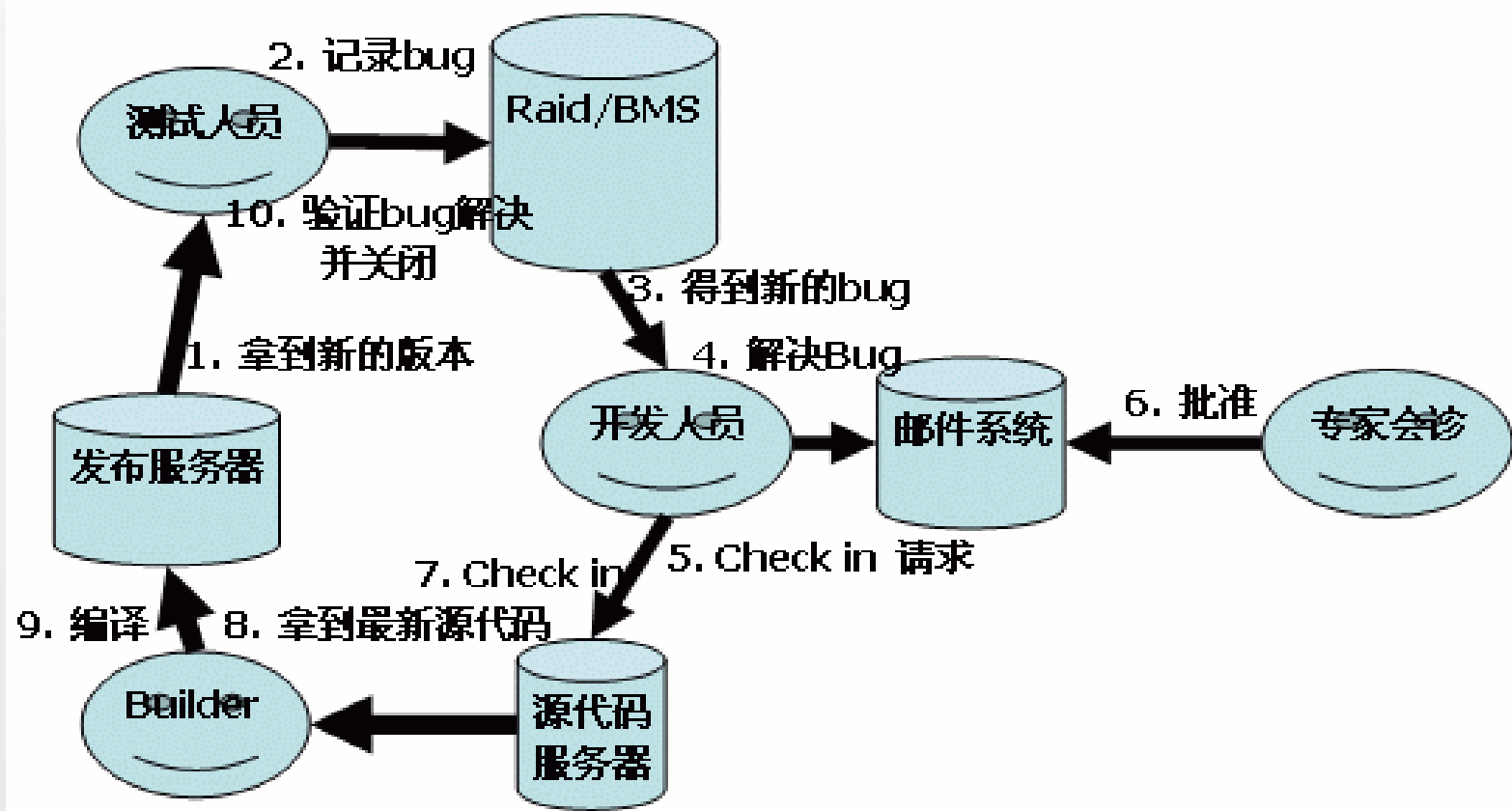
- 缺陷登记:
- 缺陷审批:
- 是否缺陷:
- 缺陷分派:
- 修复缺陷:
- 缺陷回归测试:

微软的BUG管理



100%由缺陷跟踪工具驱动

微软的一天

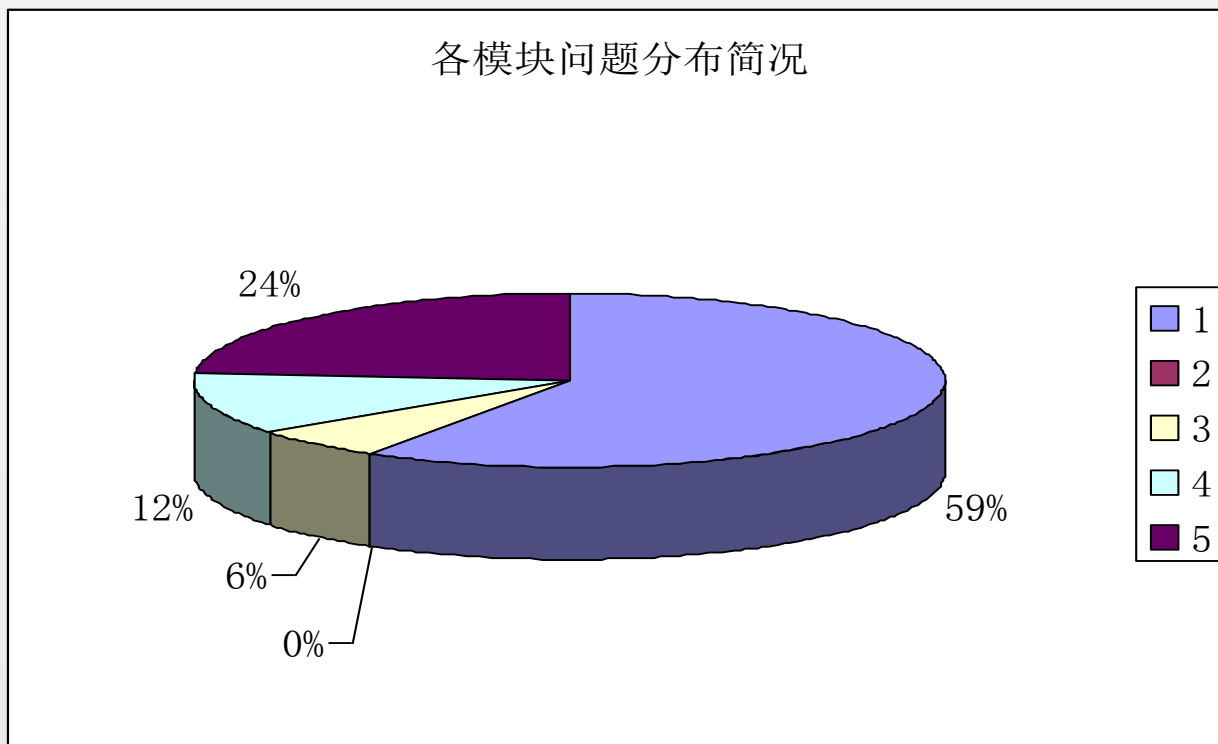


编写Bug摘要

- Bug的摘要是要用一句话的形式简明扼要地将Bug描述出来，要清晰指出Bug所在部位以及其错误类型，不能太笼统。
- 如“页面对非法输入有问题”可以修改为“流量信息查询页面对于非法输入没有进行校验”。

Bug统计₁

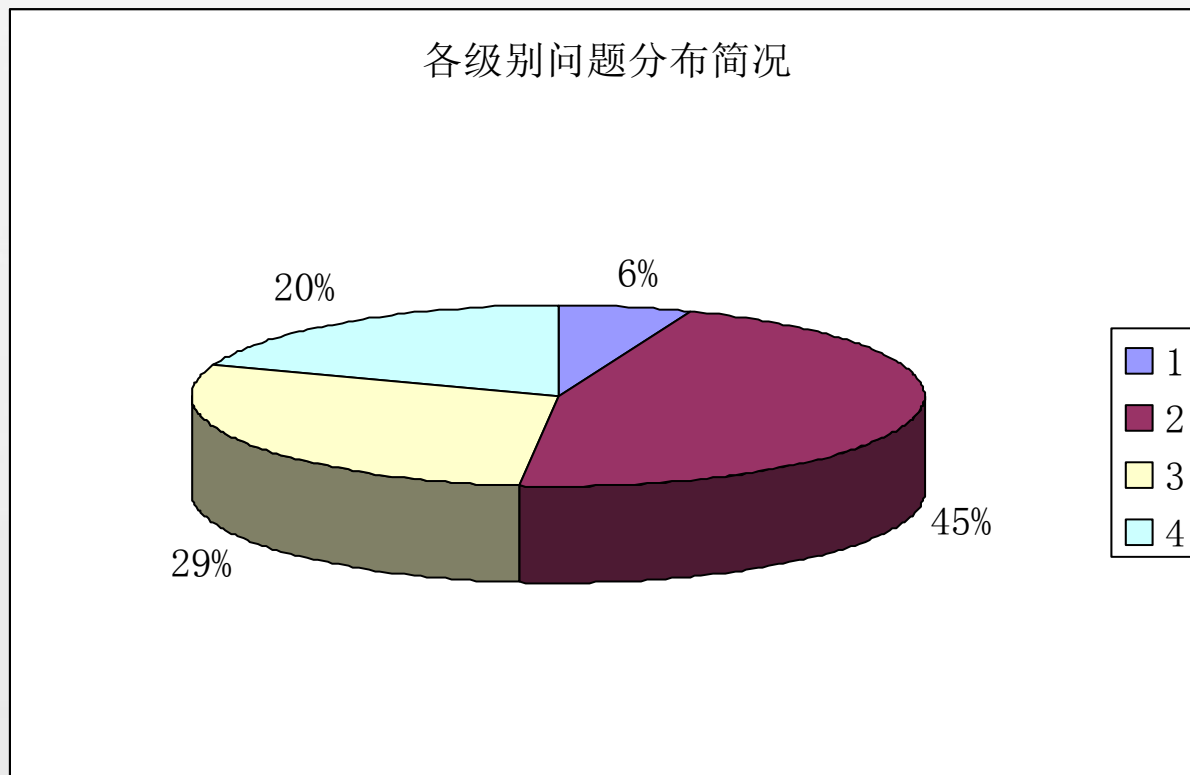
各模块问题分布简况



- 1. 流量信息统计查询模块 (10个, 59%)
- 2. 数据采集模块 (0个, 0%)
- 3. 数据整理模块 (1个, 6%)
- 4. 系统配置模块 (2个, 12%)
- 5. 邮件告警模块 (4个, 24%)

Bug统计₂

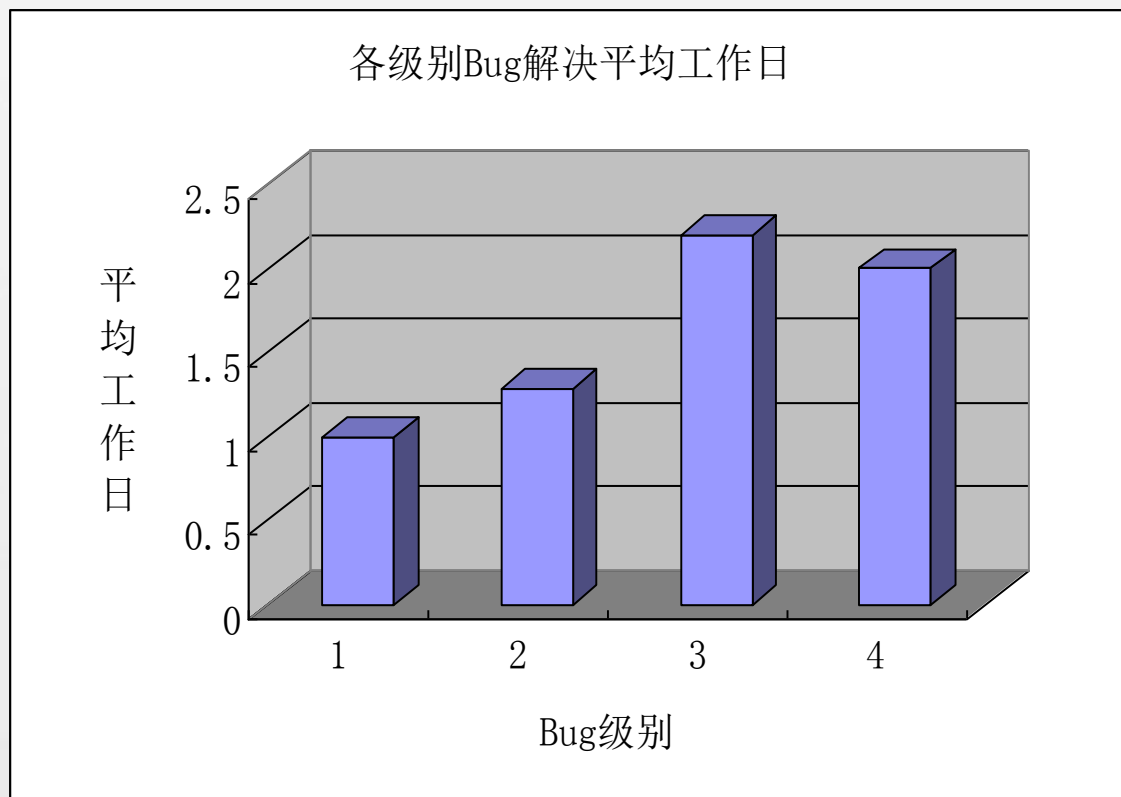
各级别问题分布简况



1. 一级bug(blocker, critical) (1个, 6%)
2. 二级bug (major, normal) (8个, 45%)
3. 三级bug(minor, trivial) (5个, 29%)
4. 四级bug(enhancement) (3个, 20%)

Bug统计₃

各级别Bug解决平均工作日



1. 一级bug (blocker, critical) (平均 1 天)
2. 二级bug (major, normal) (平均1.29天)
3. 三级bug (minor, trivial) (平均2.20天)
4. 四级bug (enhancement) (平均2天)

注：标记为LATE状态的问题不在统计之列