系统测试的执行

Execute of System Test

目录

CONTENTS

目录

CONTENTS

13.1 基本观点

系统测试的执行是软件开发的一个关键阶段：

1. 面临逼近软件交付日期的时间上的压力。

2. 有必要在交付产品前检测出绝大部分缺陷。

3. 关键是要验证缺陷在修复后能正常工作，并且不会导致新的缺陷。

13.1 基本观点

测试过程需要被监控，确定两种主要的**度量指标**：

1、系统测试执行状态；

2、缺陷状态（缺陷模式）。

在修复缺陷的过程中，软件开发团队对于**缺陷的分析**是一个至关重要的活动。

最后，还需要评估**系统测试有效性**的相关度量指标。

目录

CONTENTS

13.2 缺陷建模

为了能够跟踪和管理**缺陷**，并以此支持系统测试执行的**过程监控**，需要进行**缺陷建模**。

同时，缺陷的**利益相关人员**构成了一个跨功能组，包括**市场销售组**、**客户支持组**、**开发组**、**系统测试组**和**产品维护组**，要考虑缺陷对他们的不同**意义**。

13.2 缺陷建模-模型的状态

状态	语义	描述
N	**新建**(New)	发布一个带有**严重性**和**优先级**等级的问题报告
A	**已分配**(Assigned)	已经将问题分配给一个适当的人员
O	**打开**(Open)	被分配人员正在积极致力于解决该问题
R	**已解决**(Resolved)	被分配人员已经解决该问题并等待提交人员验证和关闭
C	**关闭**(Close)	提交人员验证了问题的解决方案
W	**等待**(Wait)	被分配人员正在等待提交人员提供更多的信息
F	**功能设计**(FAD)	报告的缺陷不是真正的缺陷，它是一个设计的功能
H	**阻塞**(Hold)	需要等其他问题解决后才能解决该问题
S	**被搁置**(Shelved)	明确决定该问题近期不会被解决（管理决定）
D	**重复**(Duplicate)	报告的问题与已报告的问题重复
I	**无法重现**(Irreproducible)	报告的问题不能由被分配人员重现
P	**延期**(Postponed)	这个问题会在下个版本中解决

13.2 缺陷建模

优先级(priority)：度量缺陷需要**多久才能修复**

1. **关键(critical)：**这种级别的缺陷必须**尽快**解决；直到**关键缺陷**被解决，系统才能继续测试。
2. **高(high)：**这种级别的缺陷必须**优先**解决；直到**高级别缺陷**被解决，系统的**部分**功能才能测试。
3. **中(medium)：**这种级别的缺陷不会对测试进程产生任何影响，随时可以解决该缺陷。
4. **低(low)：**如果时间允许，这种级别的缺陷将会解决。

13.2 缺陷建模

严重性(severity)等级：度量缺陷对于**影响产品的运行**的危害程度

1. **关键(critical)：**如果缺陷造成一个或多个**关键系统功能**被损害，且该缺陷没有变通方案可绕过进行下一步测试。
2. **高(high)：**如果**基本系统功能**被损害，但有变通方案可绕过缺陷进行下一步测试。
3. **中(medium)：**没有基本系统功能被损害，且有变通方案可绕过缺陷进行下一步测试。
4. **低(low)：**问题只是和系统的表面特性相关。

13.2 缺陷建模

优先级和**严重性等级**的区别

缺陷描述：“当系统时间被设置为12月32日时，将会造成系统崩溃”

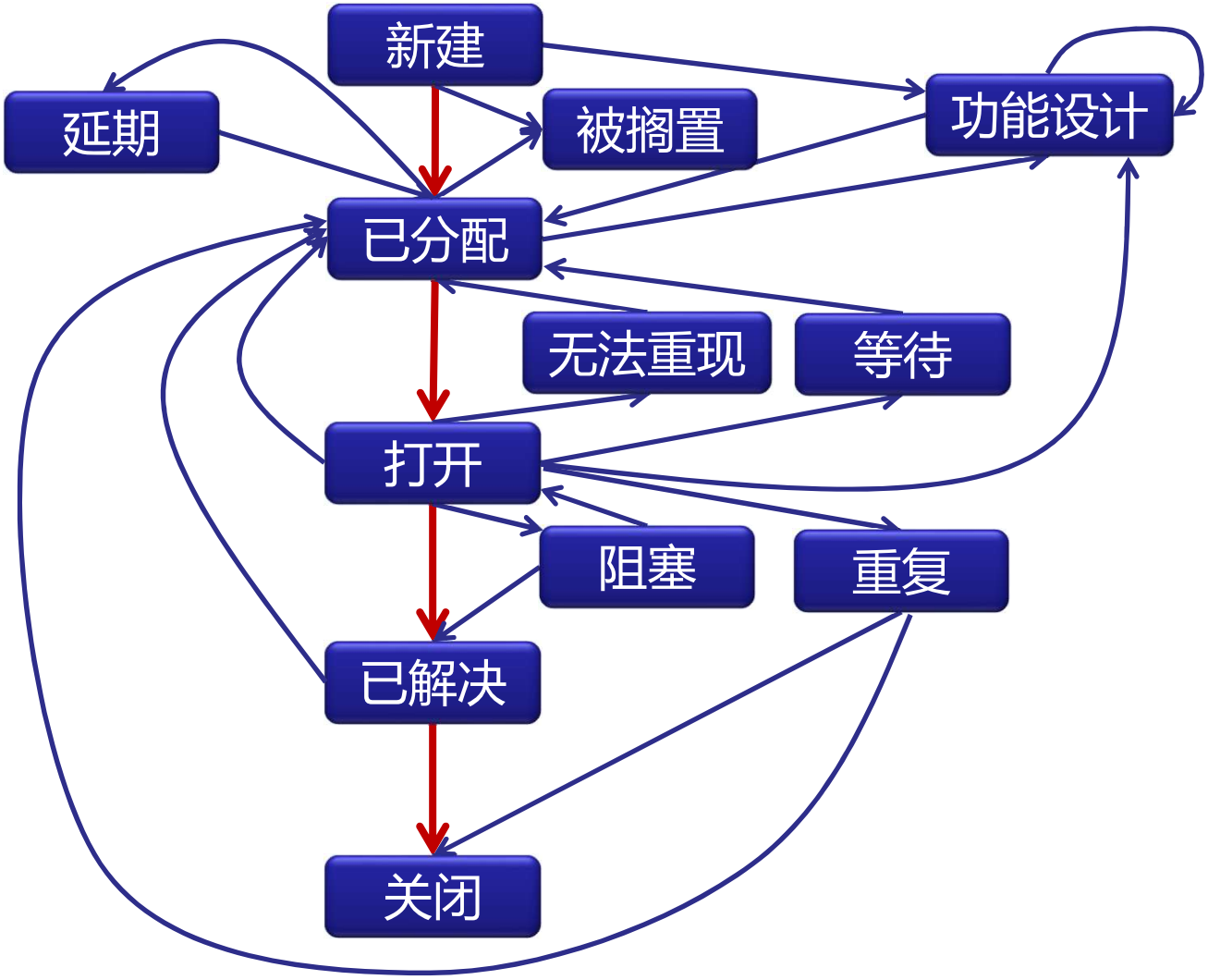
该缺陷**严重性等级**为“**关键**”，而**优先级**等级为“**低**”。

13.2 缺陷建模

缺陷的**严重性等级**的在它的整个生命周期中都是不变的，而**优先级**等级会发生变化。

例：**图形用户界面**的一个拼写错误（必须修复但不着急）在系统测试早期是**低**优先级缺陷，但这个缺陷的优先级会随着系统接近发布日期而变得越来越**高**。

13.2 缺陷建模-状态转移图



过程参与者
缺陷提交人
开发经理
开发人员
评审小组
验证人员

缺陷生命周期的状态转移图

13.2 缺陷建模-模型的状态及所有者

状态	语义	状态描述及所有者
N	**新建**	发布一个带有**严重性**和**优先级**等级的问题报告（**软件开发经理**）
A	**已分配**	已经将问题分配给一个适当的人员（**软件开发经理**或**开发人员**）
O	**打开**	被分配人员正在积极致力于解决该问题（**软件开发人员**）
R	**已解决**	被分配人员已经解决该问题并等待提交人员验证和关闭（**提交人员**）
C	**关闭**	提交人员验证了问题的解决方案
W	**等待**	被分配人员正在等待提交人员提供更多的信息 （**提交人员**）
F	**功能设计**	报告的缺陷不是真正的缺陷，它是一个设计的功能（**开发组、客户支持组、产品管理组、项目经理**）
H	**阻塞**	需要等其他问题解决后才能解决该问题（**软件开发经理**）
S	**被搁置**	明确决定该问题近期不会被解决（管理决定）
D	**重复**	报告的问题与已报告的问题重复（**提交人员**）
I	**无法重现**	报告的问题不能由被分配人员重现（**提交人员**）
P	**延期**	这个问题会在下个版本中解决（**软件开发经理**）

13.2 缺陷建模-状态转移图

用**状态转移图**描述了一个缺陷模型的各种状态(见图13.1)，并且阐述了缺陷如何从一个状态转移到另一个状态。当发现一个**新的缺陷**时，将其**记录**到一个缺陷跟踪系统中，并且该缺陷被赋予初始状态”**新建**”。这个状态的所有者是**软件开发经理**。

13.2 缺陷建模-状态转移图

软件开发经理是**新建状态**的所有者，通过把缺陷所有权变成软件开发人员或者他自己，他把这个缺陷的状态**由新建改为已分配**。软件开发经理可以**拒绝**这个缺陷，只要解释清楚它不是一个真正的缺陷，并且把这个缺陷的状态改为**FAD**，而所有权移交给**缺陷的提交人员**。

13.2 缺陷建模-状态转移图

开发经理可以把**缺陷状态**改为**被搁置状态**，也就是这确实是一个真正的缺陷，并且明确决定它在近期内不会修复。当缺陷改为被搁置状态时，需要得到**所有相关组**，尤其是组织的客户支持组的**同意**。

把缺陷状态改为被搁置状态有几点原因：

例如，在客户的运营环境下，一个缺陷对系统运行影响不大并且修复它会消耗很多的资源。

实际上，很少有缺陷会标记为被搁置状态。

13.2 缺陷建模-状态转移图

已分配状态表示已经指定某人修复这个缺陷，但是需要**修复**这个缺陷的实际工作**尚未开始**。软件开发经理或者某个开发人员，是在已分配状态下的缺陷所有者。所有权会转移到修复这个缺陷的开发人员。

一旦**修复缺陷的工作开始**，那么软件开发人员就把已分配状态改为**打开状态**。注意，只有开发经理可以把缺陷**从已分配状态改成**如下状态：**被搁置**，**延期**，**功能设计(FAD)**。当一个缺陷改为FAD状态时，缺陷的所有权改为**提交人员**。

13.2 缺陷建模-状态转移图

在**打开状态**时，软件开发人员作为缺陷的所有者正在积极进行**缺陷修复**工作。开发人员初始化缺陷模式的forecast_fix_version 和 forecast_build_number

这两项(在表13.2中给出)，让**提交人员**可以计划并安排重新测试这个补丁。

缺陷处于打开状态可能会**持续数个星期**。软件开发人员可以把缺陷从这个状态改为以下5种状态之一: **无法重现**，**已解决**，**等待**，**阻塞**，**重复**。以上5种状态连同改变这5种状态所需的行动都在表13.3 中给出。

13.2 缺陷建模-打开状态之后的5种状态

表 13.3　自打开状态之后的 5 种可能状态的状态转移

下一个可能状态	行　动
无法重现	如果所有者，即某个开发人员不能重现一个缺陷，这个缺陷就改为无法重现状态，并且所有权改回给提交人员
等待	如果提交人员提供的信息量对于理解缺陷或者重现缺陷不够充分，那么所有者会向提交人员要求有关缺陷的更多信息。本质上，开发人员会在 notes 字段中要求更多的关于缺陷的信息。所有权改回给提交人员

13.2 缺陷建模-打开状态之后的5种状态

（续表）

下一个可能状态	行　动
已解决	1. 所有权改回给提交人员 2. 以下字段需要填写： ● actual_fix_version：修复缺陷的补丁是可用的软件版本号 ● actual_build_number：修复缺陷的补丁是可用的构件版本号 ● fixed_description：对于该补丁的简要描述。开发人员必须列举所有已修改的文件，以及为了修复问题，文件中已经修改的部分 ● fixed_date：缺陷被修复并且改为已解决状态的日期
阻塞	在 notes 字段中已经解释了为什么该缺陷会改为阻塞状态。阻塞状态的意思是，由于问题依赖于其他问题，因而没有做出决策，例如一个在第三方产品中的软件缺陷。除非其他的问题得以解决，否则这个缺陷描述的问题是无法解决的
重复	如果这个缺陷和另一个缺陷相同，那么原缺陷的标识符会记录到 duplicate_defect_id 字段中

13.2 缺陷建模-状态转移图

如果处理缺陷的软件开发人员认为该缺陷已经修复了，那么他会把这个缺陷的状态改为**已解决状态**，并且把缺陷的所有权移交给提交人员。不过，一个缺陷处于已解决状态时，仅仅代表缺陷已被修复并使开发人员满意，因此需要在更大的背景下进行相关的测试来进一步验证这个声明。

13.2 缺陷建模-状态转移图

提交人员在验证并确认这个缺陷修复之后，可以把缺陷状态从**已解决**改为关闭状态，并且将以下的一些缺陷字段进行初始化：verifier,closed_in_version,closed_in_build,closed_date,verification description. 本质上，verification_description 描述了验证过程的细节。

关闭状态预示着已经解决该缺陷并且验证完毕。

13.2 缺陷建模-状态转移图

另一方面，如果**验证人员**确信这个缺陷没有真正或者完全解决，那么可以驳回这个缺陷的补丁，而缺陷的状态将改回**已分配状态**，缺陷的所有权也改为负责相关缺陷修复的**软件开发人员**。

验证人员在**驳回一个修复补丁**时需要提供如下的信息:

1. 解释驳回该补丁的**原因**
2. 验证时发现的**新情况**或者遇到的**新问题**。

13.2 缺陷建模-状态转移图

如果开发组认为一个缺陷**不是真正的缺陷**而是系统本身功能所期望的，那么这个缺陷的状态就会改为**FAD状态**。这个缺陷在FAD状态下的所有者可能包括开发组、客户支持组、产品管理组和项目经理，从而一起评审这个状态的缺陷。

13.2 缺陷建模-状态转移图

在缺陷评审工作会议上有以下选择结果:

1.报告这个缺陷是因为提交人员方面的**程序误差**。对于这种结果，该缺陷保持FAD状态，缺陷保持FAD状态而不改为关闭状态，表明提交人员已经指定它为一个缺陷。

2.报告的缺陷是一个**真正的缺陷**，当软件开发经理在缺陷注释上表明这个缺陷得到所有部门的同意时，该缺陷将改为**已分配状态**。

13.2 缺陷建模-状态转移图

在缺陷评审工作会议上有以下择结果:

3. 做出建议，提出**加上一个新的特性的请求**，以便接受此时当做一个缺陷的系统行为。提交人员把这个缺陷改为已分配状态，并加上开发经理的注释，注释提到这个缺陷将通过创建一个新的需求来解决，并且在缺陷模式中提交需求的标识符requirement_id。

13.2 缺陷建模-状态转移图

如果一个缺陷无法在特定的发布中修复，则将其改为**延期状态**，**开发经理**是缺陷的**所有者**。由于在某个版本中没有时间来修复缺陷，缺陷的修复工作被延期。

在缺陷修复补丁的发行版本已知之后，将缺陷改为**已分配状态**,并且以下缺陷字段需要填写:(i)forecast_build_number，可以在测试中使用的构件,(ii)forecast_fix_version，可以在测试中使用的版本号。

13.2 缺陷建模-状态转移图

在产品发布给客户后，**所有仍处于打开状态并逾期未修复的缺陷将改为延期状态**，从而使这些缺陷可以安排在未来的软件发布中修复。在最终决定这些缺陷将在哪个版本中修复后，他们的状态将改为**已分配状态**。延期状态对于跟踪系统中逾期未修复的缺陷并把它们安排在未来将被修复的版本中很有帮助。

13.2 缺陷建模-状态转移图

如果**开发人员**发现提交人员所提供的信息并不足以修复缺陷，那么他会向提交人员要求更多的信息，并把缺陷的状态由**打开状态**改为**等待状态**。

在等待状态中，提交人员是缺陷的所有者。提交人员通过初始化缺陷模式中的notes字段来提供开发人员所需要的信息。然后把缺陷状态改为**已分配状态**，并且把所有权修改回软件开发人员。

13.2 缺陷建模-状态转移图

开发人员会尝试重现缺陷所产生的后果以便理解缺陷，如果开发人员**无法重现报告的缺陷**那么缺陷的状态将改为**无法重现状态**，并且把所有权改为提交人员。

提交人员努力重现这个缺陷，收集所有与缺陷报告相关的信息，如果提交人员重现这个缺陷，那么这个缺陷就改为**已分配状态**。如果提交人员无法重现这个缺陷，那么这个缺陷**保持无法重现状态**。

13.2 缺陷建模-状态转移图

有时会发现有些缺陷是由外部因素引起的，例如正在使用的第三方产品的一个缺陷。报告的缺陷是无法修复的，除非解决外部组件的缺陷。在这种情况下，将缺陷挂起并移至**阻塞状态**，此处软件开发经理是缺陷的所有者。

开发经理可以把缺陷的状态改为**已解决状态**或**打开状态**，这分别取决于外部组件的主要缺陷是否**解决**或**正在修复**。

13.2 缺陷建模-状态转移图

当一个缺陷处于**重复状态**时，**提交人员**作为它的所有者。这意味着与该问题重复的缺陷之前已经报告过。最初那个缺陷的标识符应该填入对应的字段 duplicate_defect_id 中。一旦重复的缺陷处于已解决状态，**提交人员**应该**验证**该缺陷。如果验证成功，提交人员把缺陷改为**关闭状态**。

13.2 缺陷建模-状态转移图

另一方面，如果**验证失败**，那么提交人员驳回这个缺陷的重复性声明，并把缺陷状态改为**已分配状态**，此时软件开发人员是缺陷的所有者。**验证人员必须给出驳回这个缺陷作为重复缺陷的原因**。该信息包含验证缺陷时所观察到的任何**情况**或遇到的**问题**。

目录

CONTENTS

13.3 系统测试开始前的准备工作

确认《测试执行工作文档》就绪并完整（*首次系统测试周期的进入标准*）。

由**测试组负责人**进行创建、控制并跟踪。

13.3 系统测试开始前的准备工作

系统测试执行工作文档大纲

1、测试工程师
2、测试用例分配
3、测试台分配
4、自动化进程
5、预计测试的执行速率
6、执行失败的测试用例
7、开发新的测试用例
8、系统镜像的实验
9、缺陷评审会议的时间表

13.3 系统测试开始前的准备工作

系统测试执行工作文档大纲

1、测试工程师

2、测试用例分配

3、测试台分配

4、自动化进程

5、预计测试的执行速率

6、执行失败的测试用例

7、开发新的测试用例

8、系统镜像的实验

9、缺陷评审会议的时间表

记录**测试工程师**的信息，主要包括工程师在各专属领域的工作可参与性和经验，以及工程师的培训需求。

这部分主要记录人力资源可用性的相关信息。

13.3 系统测试开始前的准备工作

系统测试执行工作文档大纲

1、测试工程师

2、测试用例分配

3、测试台分配

4、自动化进程

5、预计测试的执行速率

6、执行失败的测试用例

7、开发新的测试用例

8、系统镜像的实验

9、缺陷评审会议的时间表

召开团队会议，理解测试用例，确定测试用例的优先级，根据测试工程师的经验和兴趣**分配测试用例**，确认测试目标的相应开发人员。记录并跟踪上述信息。

13.3 系统测试开始前的准备工作

系统测试执行工作文档大纲

1、测试工程师
2、测试用例分配
3、测试台分配
4、自动化进程
5、预计测试的执行速率
6、执行失败的测试用例
7、开发新的测试用例
8、系统镜像的实验
9、缺陷评审会议的时间表

阐述可用测试台的数量和种类，以及测试台的分布方式；在测试台和测试用例子集建立映射（基于执行成组测试用例的配置需求）；**分配测试台**给相应负责测试工程师。记录并跟踪相应信息。

13.3 系统测试开始前的准备工作

系统测试执行工作文档大纲

1、测试工程师
2、测试用例分配
3、测试台分配
4、自动化进程
5、预计测试的执行速率
6、执行失败的测试用例
7、开发新的测试用例
8、系统镜像的实验
9、缺陷评审会议的时间表

采用表格跟踪记录测试用例的**自动执行进程**和那些用于执行的自动化测试用例的可用性。目的是减少手动执行的测试用例以提升测试效率。

13.3 系统测试开始前的准备工作

系统测试执行工作文档大纲

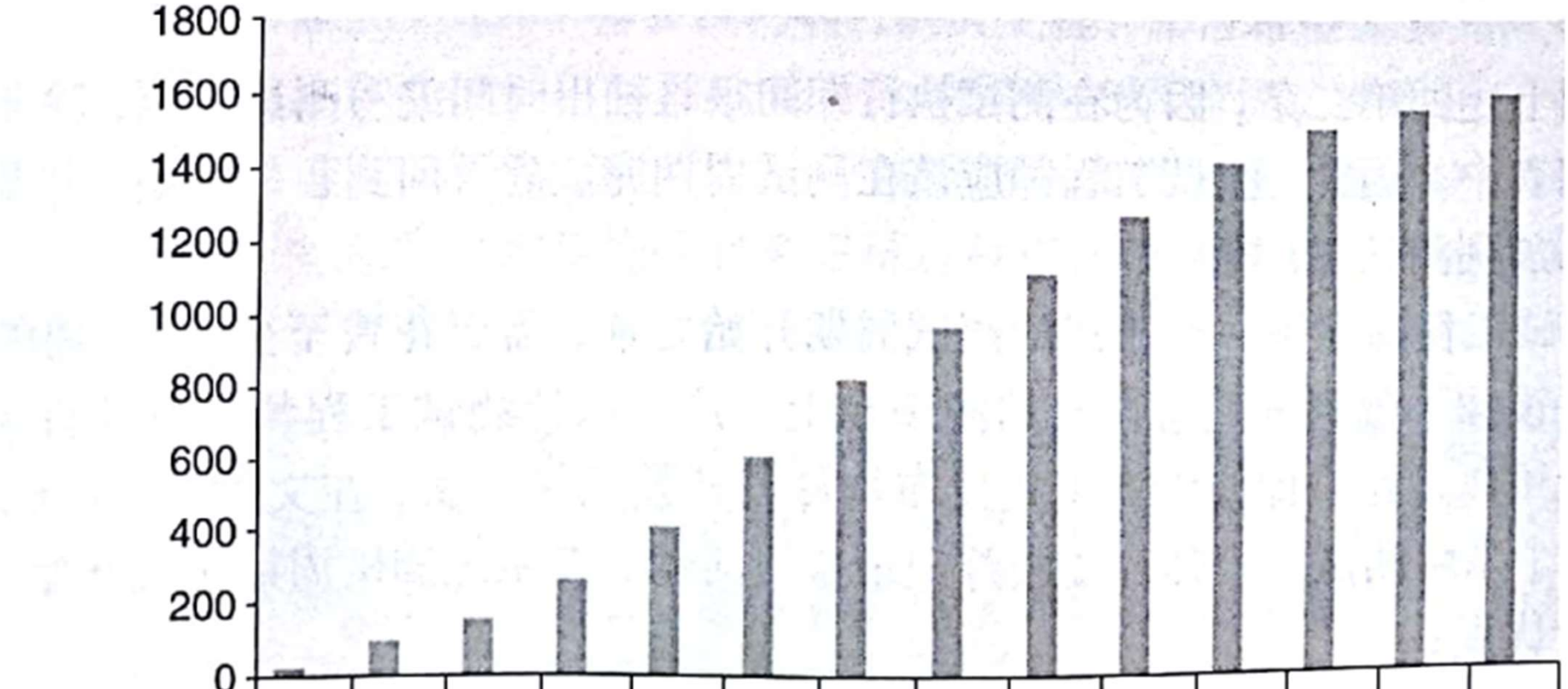
1、测试工程师
2、测试用例分配
3、测试台分配
4、自动化进程
5、预计测试的执行速率
6、执行失败的测试用例
7、开发新的测试用例
8、系统镜像的实验
9、缺陷评审会议的时间表

描述在测试周期中预计**每周将会执行的测试用例数目**。之后将会与实际执行的测试用例数目进行跟踪比较，目的是使得测试组可以掌握测试进度。

13.3 系统测试开始前的准备工作-例

例如，设有一个 Bazooka 测试项目，该项目一共选择了1592个测试用例。在14周测试周期中以周为单位预计执行的测试用例数如图13.2所示，该表以柱状图的形式给出。执行1592个测试用例预计需要14周;第一周将会执行25个测试用例,第二周将执行75个，这样在第二周结束时一共会执行100个测试用例。

13.3 系统测试开始前的准备工作-例



	第1周	第2周	第3周	第4周	第5周	第6周	第7周	第8周	第9周	第10周	第11周	第12周	第13周	第14周
■ 计划执行量	25	100	150	250	400	600	800	950	1100	1250	1400	1500	1550	1592
□ 实际执行量														

图 13.2　以周为单位的测试用例计划执行量

13.3 系统测试开始前的准备工作-例

当执行推进时，图13.2的第二行也就是实际执行的测试用例数将会更新。这样使得测试组可以掌握测试进度。如果连续几周实际执行的测试用例远落后于预计的测试用例，那么测试经理将会知道这个项目有可能被延期。

13.3 系统测试开始前的准备工作

系统测试执行工作文档大纲

1、测试工程师
2、测试用例分配
3、测试台分配
4、自动化进程
5、预计测试的执行速率
6、执行失败的测试用例
7、开发新的测试用例
8、系统镜像的实验
9、缺陷评审会议的时间表

描述**失败的测试用例**执行及缺陷修复的补丁验证策略。策略如下：

1、在下一周期执行失败的测试用例；
2、一旦修复补丁可用，就立即在当前测试周期执行失败的测试用例；
3、进行回归测试。

13.3 系统测试开始前的准备工作

系统测试执行工作文档大纲

1、测试工程师
2、测试用例分配
3、测试台分配
4、自动化进程
5、预计测试的执行速率
6、执行失败的测试用例
7、开发新的测试用例
8、系统镜像的实验
9、缺陷评审会议的时间表

描述在执行测试用例期间**编写新测试用例**的策略：
1、在测试周期中开发新测试用例；
2、在缺陷报告中添加记录确认针对该缺陷需要新增测试用例，按照时间表继续执行本测试周期的测试用例。

13.3 系统测试开始前的准备工作

系统测试执行工作文档大纲

1、测试工程师
2、测试用例分配
3、测试台分配
4、自动化进程
5、预计测试的执行速率
6、执行失败的测试用例
7、开发新的测试用例
8、系统镜像的实验
9、缺陷评审会议的时间表

跟踪记录活动的信息：在测试周期开始前，测试工程师应将**软件镜像提前下载**到分配给自己的测试台；其目的是让工程师熟悉软件镜像，并确保测试台是可用的。

13.3 系统测试开始前的准备工作

系统测试执行工作文档大纲

1、测试工程师
2、测试用例分配
3、测试台分配
4、自动化进程
5、预计测试的执行速率
6、执行失败的测试用例
7、开发新的测试用例
8、系统镜像的实验
9、缺陷评审会议的时间表

记录**缺陷评审会议的时间表**，目的是提前计划缺陷评审工作（因为参与人员会来自多个不同的跨功能组）。

目录

CONTENTS

13.4 跟踪系统测试的度量指标

在**系统测试**的执行过程中，需要监督真实的、反映**系统测试进程**的**度量**，并揭示系统的**质量水平**。

并以此帮助管理团队制定有效的过程管理策略。

13.4 跟踪系统测试的度量指标

执行度量指标分为两类：

监督测试执行的指标： 关注执行测试用例的过程。

监督缺陷的指标： 关注作为测试执行结果所发现的缺陷以确定产品的质量等级。

这些**度量指标**需要进行**周期性**的跟踪和分析

13.4.1监测测试用例执行的度量指标

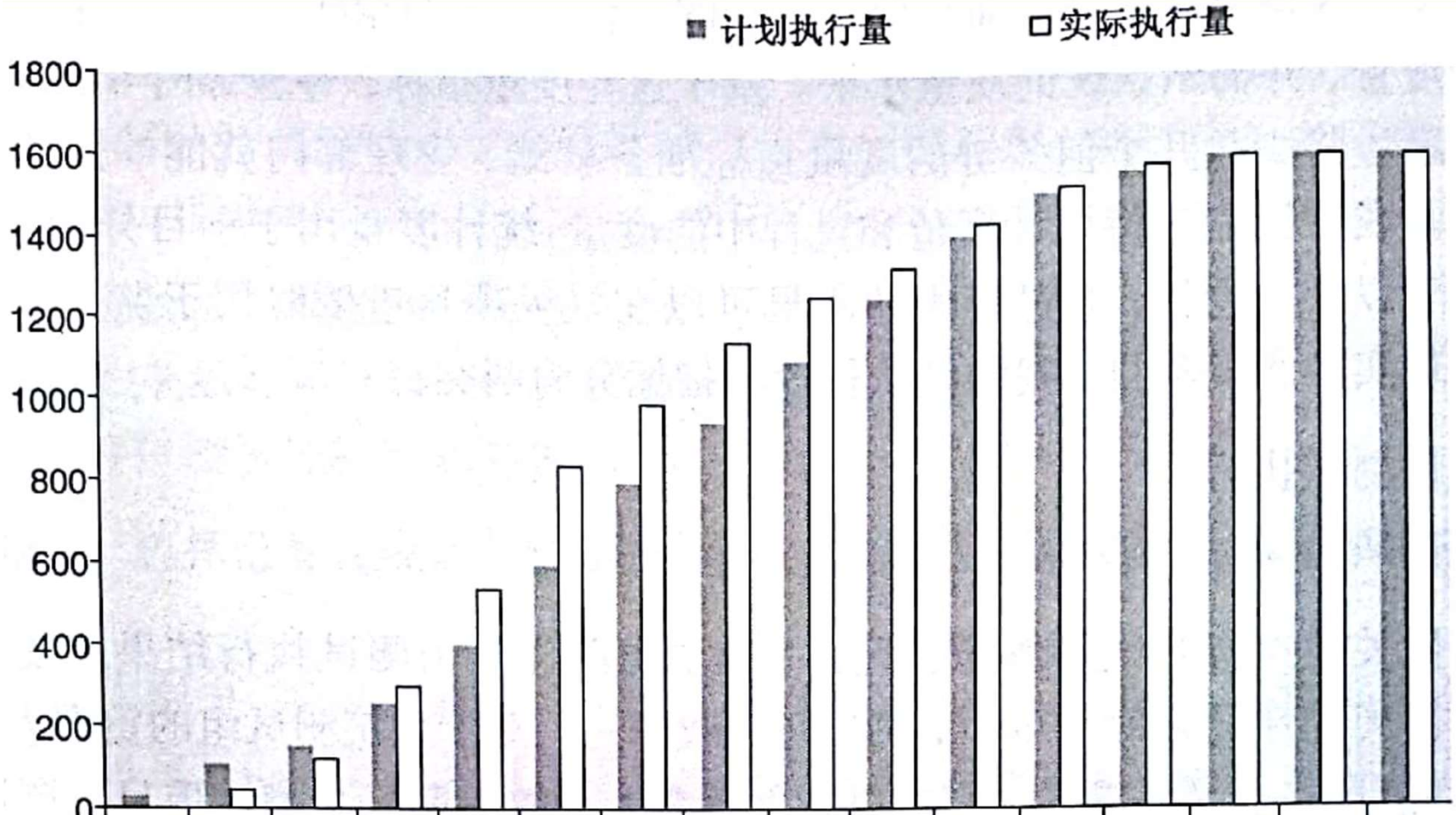
测试用例逃逸(TCE,Test Case Escape)：在测试过程中需要设计新测试用例即称为**测试用例逃逸**。**TCE**快速增长说明在测试设计阶段准备不充分且会影响项目进度。

13.4.1监测测试用例执行的度量指标

计划执行率对实际执行率(PAE,Program Execution and **A**ctual **E**xecution)**:** 每周实际执行的测试用例数和计划执行的测试用例数进行比较，体现测试用例执行的生产率。

我们给出一个测试用例执行度量指标的示例，它来自一个叫做Bazooka的真实测试项目。该项目总共设计了1592个测试用例。在14周的测试周期中，1592个测试用例每周计划执行的数量如图13.3所示，以PAE度量指标的柱状图形式表示。

13.4.1监测测试用例执行的度量指标



	第1周	第2周	第3周	第4周	第5周	第6周	第7周	第8周	第9周	第10周	第11周	第12周	第13周	第14周	第15周	第16周
PE	25	100	150	250	400	600	800	950	1100	1250	1400	1500	1550	1592	1592	1592
AE	5	45	121	297	542	841	991	1148	1254	1324	1426	1515	1566	1581	1581	1591

图 13.3　Bazooka 项目中的 PAE 度量（PE：计划执行量；AE：实际执行量）

13.4.1监测测试用例执行的度量指标

第一周计划执行25个测试用例，第二周执行75个。按计划在前两周将会有100个测试用例执行完。图13.3底部的第2行是实际执行的测试用例的个数。在测试执行期间，应该以周为单位来监测测试用例的计划执行量和实际执行量。如果二者之间有很大的差距，应该立即采取行动来找出其问题来源，并且尽早提高测试执行率。如表中前三周所示，测试的执行率低于计划执速率，测试工程师理解并达到相应的速率花费了大约三周时间。

13.4.1监测测试用例执行的度量指标

为了在一个执行周期完成所有的测试用例，其中包括重新执行失败的测试用例，这个测试项目**实际执行了16周而非计划的14周**。最初，测试工程师没有运行压力、负载和稳定性这类测试用例，因为一些有关内存泄漏的补丁只是在第13周末才加入产品代码。因此，**测试周期延长了两周多**，从而确保系统能够在一个相当长的时间里持续运行而没有内存泄漏。

13.4.1监测测试用例执行的度量指标

测试用例的执行状态(EST, Execution Status of Test cases)：监测不同执行状态（失败，通过，阻塞，无效，未测试）的测试用例。

让我们考虑先前的例子，即Bazooka测试项目，该项目选择了1592个测试用例。每周对每个测试组检测的**EST度量**关注以下状态中测试用例的数量：通过，失败，无效，阻塞，未测试。举例来说，表13.5展示了第四周的执行状态。

13.4.1监测测试用例执行的度量指标

表 13.5　Bazooka 项目中第四周的 EST 度量

测试组	测试用例总数	通过	失败	阻塞	无效	已执行	未测试	通过的/执行的(%)
基础	156	64	9	0	0	73	83	87.67
功能性	480	56	7	0	0	63	417	88.89
健壮性	230	22	1	0	0	23	207	95.65
互操作性	149	15	2	0	0	17	132	88.24
负载和稳定性	43	1	0	0	0	1	42	100.00

(续表)

测试组	测试用例总数	通过	失败	阻塞	无效	已执行	未测试	通过的/执行的(%)
性能	54	0	0	0	0	0	54	0.00
压力	36	0	0	0	0	0	36	0.00
可收缩性	54	5	0	0	0	5	49	100.00
回归	356	93	13	0	0	106	250	87.74
文档	34	8	1	0	0	9	25	88.89
总计	1592	264	33	0	0	297	1295	88.89

13.4.1监测测试用例执行的度量指标

我们解释了Bazooka团队是如何控制系统测试的进度。测试周期的恢复准则是有关一个产品的质量水平的谓词，是根据执行通过的测试用例的百分比。

如果该准则满足则测试阶段停止。给出一个重做准则的例子：**在测试周期的任何时刻，如果累积失败率达到20%，在报告的缺陷已经解决之后，该周期应该全部重新执行一遍**。

在表13.6的最右侧一列给出执行的测试用例的通过率。在Bazooka的第2周周末通过率是71.11%，也就是说失败率是28.89%。考虑之前给出的例子中的重做准则，这个测试阶段应该放弃。

13.4.1监测测试用例执行的度量指标

然而，作为一个特例，Bazooka项目管理团队**做出了一个不放弃这个测试阶段的决定**，原因如下:

1. 1592个测试用例只有45个被执行。

2. 一些测试用例失败的原因，是测试工程师方面的**操作错误**造成的。因此最终认为这些测试用例是通过的。

Bazooka团队做了一致决定，再等待一周并且每天监测测试的结果。如表13.6所示，在第3周周末，通过率达到89.26%。由于失败率远低于达到重做准则中的阈值20%，所以该测试阶段继续执行直至结束。

13.4.1监测测试用例执行的度量指标

表 13.6　Bazooka 项目中以周为单位跟踪的 EST 度量

周	执行	通过	失败	执行的总数(%)	通过的总数(%)	通过的/执行的(%)
1	5	5	0	0.31	0.31	100.00
2	45	32	13	2.83	2.01	71.11
3	121	108	13	7.60	6.78	89.26
4	297	264	33	18.66	16.58	88.89
5	542	451	91	34.05	28.33	83.21
6	841	677	164	52.83	42.53	80.50
7	991	835	156	62.25	52.45	84.26
8	1148	1009	139	72.11	63.38	87.89
9	1254	1096	158	78.77	68.84	87.40
10	1324	1214	110	83.17	76.26	91.69
11	1426	1342	84	89.57	84.30	94.11
12	1515	1450	65	95.16	91.08	95.71
13	1566	1519	47	98.37	95.41	97.00
14	1581	1567	14	99.31	98.43	99.11
15	1581	1570	11	99.31	98.62	99.30
16	1591	1580	11	99.94	99.25	99.31

注释：有效测试用例总数为 1592。

13.4.3监测缺陷报告的度量指标

处于FAD状态的缺陷的数量：FAD数量越低，说明测试工程师对系统给的理解程度越高。

无法重现的缺陷数量：是系统**不可靠性**的一个度量。

缺陷到达率DAR(Defects **A**rrival **R**ate**)的计数：**来自不同功能组（客户支持组、市场营销组和文档组）的缺陷的百分比。

13.4.3监测缺陷报告的度量指标

对于上述大部分度量指标，使用第二个测试项目的数据给出一些例子，该项目名为Stinger，在表13.7中展示出，该组织的各组在16周的第二个测试周期获得系统级测试的**DAR**。表的第1行作为第0周，给出了第一个测试周期**仍处于打开状态的缺陷总和**。每周由ST、SW、SIT和其他组填写的缺陷的平均值分别是57、15、10和18。此处，其他组包括客户支持组、市场营销组和文档组。在前6周中，缺陷的到达率远高于该测试周期的剩余部分。这归结于一个事实，也就是在测试周期的初期，测试用例按照优先级在系统更容易出错的部分找出缺陷。

表 13.7　Stinger 项目中的 DAR 度量指标

周	ST		SW		SIT		其他组		缺陷总数
	缺陷数量	百分比	缺陷数量	百分比	缺陷数量	百分比	缺陷数量	百分比	
0	663	69.21	172	17.95	88	9.19	35	3.65	958
1	120	79.47	28	18.54	2	1.32	1	0.66	151
2	99	55.00	44	24.44	31	17.22	6	3.33	180
3	108	53.73	54	26.87	31	15.42	8	3.98	201
4	101	65.16	16	10.32	22	14.19	16	10.32	155
5	107	57.22	21	11.23	26	13.90	33	17.65	187
6	108	56.25	16	8.33	35	18.23	33	17.19	192
7	59	57.84	0	0.00	1	0.98	42	41.18	102
8	15	31.91	9	19.15	3	6.38	20	42.55	47
9	28	59.57	0	0.00	1	2.13	18	38.30	47
10	14	50.00	4	14.29	0	0.00	10	35.71	28
11	12	48.00	1	4.00	0	0.00	12	48.00	25
12	59	53.64	5	4.55	9	8.18	37	33.64	110
13	28	54.90	5	9.80	0	0.00	18	35.29	51
14	23	51.11	0	0.00	0	0.00	22	48.89	45
15	12	31.58	21	55.26	3	7.89	2	5.26	38
16	24	38.71	23	37.10	0	0.00	15	24.19	62
总和	917		247		164		293		1,621
平均	57	56.57	15	15.24	10	10.12	18	18.08	

13.4.3监测缺陷报告的度量指标

缺陷拒绝率DRR(Defects Rejection Rate)的计数：对原本要修复但并没有被真正修复的缺陷的计数。其代表软件开发组修复缺陷的生产率和成功修复缺陷的程度。

在Stinger项目中，**DRR的数目**如表13.8所示，平均缺陷拒绝率是5.24%。在前8周中拒绝率几乎是6%，而在之后的4周中大约在5%左右。在该测试周期余下的时间中，拒绝率下降到接近1.5%。

表 13.8　Stinger 测试项目中每周的 DRR 状态

周	验证的缺陷数量	关闭的缺陷数量	拒绝的缺陷数量	拒绝的缺陷(%)
1	283	269	14	4.95
2	231	216	15	6.49
3	266	251	15	5.64
4	304	284	20	6.58

周	验证的缺陷数量	关闭的缺陷数量	被拒绝的缺陷数量	被拒绝的缺陷(%)
5	194	183	11	5.67
6	165	156	9	5.45
7	145	136	9	6.21
8	131	122	9	6.87
9	276	262	14	5.07
10	160	152	8	5.00
11	52	49	3	5.77
12	80	74	6	7.50
13	71	71	0	0.00
14	60	58	2	3.33
15	121	119	2	1.65
16	97	96	1	1.03
总和	2636	2498	138	
平均	165	156	9	5.24

13.4.3监测缺陷报告的度量指标

缺陷关闭率DCR(Defects Closure Rate)计数：表示验证缺陷修复声明的有效性。

显著缺陷OD(Obvious Defects)的数量：如果一个缺陷持续存在，则称该缺陷为**OD**。这个指标体现了系统的质量水平（减少代表质量提高）。

显著缺陷(**OD**)的数目如表13.9所示。**关键、高、中和低优先级的缺陷分别表示为P1、P2、P3和P4**。优先级等级是对缺陷多久能够修复的一个度量。显著缺陷的数量从第1个测试周期转入第二个测试周期，该数量在表的第1行作为第0周的统计数据。可以看出，在16周中，显著缺陷的总数逐渐从958降到81。

表 13.9　Stinger 测试项目中基于优先级的每周的显著缺陷

周	P1		P2		P3		P4		缺陷总数
	缺陷数量	百分比	缺陷数量	百分比	缺陷数量	百分比	缺陷数量	百分比	
0	14	1.46	215	22.44	399	41.65	330	34.45	958
1	19	2.26	164	19.52	280	33.33	377	44.88	840
2	17	2.11	138	17.16	236	29.35	413	51.37	804
3	11	1.46	108	14.32	181	24.01	454	60.21	754
4	7	1.12	44	7.04	123	19.68	451	72.16	625
5	5	0.79	28	4.45	123	19.55	473	75.20	629
6	10	1.50	33	4.96	123	18.50	499	75.04	665
7	11	1.74	83	13.15	52	8.24	485	76.86	631
8	5	0.90	34	6.12	32	5.76	485	87.23	556
9	8	2.35	21	6.16	39	11.44	273	80.06	341
10	4	1.84	21	9.68	36	16.59	156	71.89	217
11	4	2.07	26	13.47	38	19.69	125	64.77	193
12	3	1.31	28	12.23	84	36.68	114	49.78	229
13	3	1.44	40	19.14	84	40.19	82	39.23	209
14	10	5.10	42	21.43	68	34.69	76	38.78	196
15	2	1.74	34	29.57	38	33.04	41	35.65	115
16	3	3.70	25	30.86	24	29.63	29	35.80	81

13.4.3监测缺陷报告的度量指标

造成崩溃的缺陷CD(Crash Defects)的数量：系统**稳定性度量**指标。

在测试的最后8周，由不同组观察到的**崩溃的数量**如表13.10所示。从表中可以明显地观察到，崩溃的总数从每周的20个逐渐降到每周1个。应该仔细研究这些崩溃，从而确定导致每个崩溃产生的原因。开发团队必须单独调查这些崩溃。

13.4.3监测缺陷报告的度量指标

表 13.10 Stinger 测试项目中每周由不同组观察到的崩溃缺陷数量

周	SW		ST		SIT		其他组		缺陷总数
	缺陷数量	百分比	缺陷数量	百分比	缺陷数量	百分比	缺陷数量	百分比	
9	6	30.00	8	40.00	6	30.00	0	0.00	20
10	5	31.25	6	37.50	3	18.75	2	12.50	16
11	4	25.00	9	56.25	0	0.00	3	18.75	16
12	3	33.33	4	44.44	0	0.00	2	22.22	9
13	2	22.22	6	66.67	0	0.00	1	11.11	9
14	1	14.29	2	28.57	1	14.29	3	42.86	7
15	0	0.00	2	66.67	1	33.33	0	0.00	3
16	0	0.00	1	100.00	0	0.00	0	0.00	1
总和	21		38		11		11		81
平均	3	25.93	5	46.91	1	13.58	1	13.58	10

13.4.3监测缺陷报告的度量指标

缺陷的到达和消除ARD(Arrival and **R**emove of **D**efects**)计数：**对缺陷消除的成本进行度量，以估算修复所有缺陷所用的时间。

13.4.3监测缺陷报告的度量指标

最后，通过考虑第三个测试项目Bayonet，我们给出**ARD**度量指标的一个例子。在表13.11的上半部分，给出了在一个测试周期的前4周预计的缺陷数量，包括提交的、已解决的和仍然处于打开状态的缺陷。这些**预计的数量来自于一个早期的测试项目**，它和Bayonet有很多**相似**特性。在这个测试周期的开始阶段，新打开的缺陷总数是184个。其中50个缺陷的优先级是P1或者P2，63个缺陷的优先级是P3，71个缺陷的优先级是P4。提交并解决的缺陷实际数量在同一个表的下半部分。在第一周结束后，新打开的缺陷的实际数量略低于预计的数量。

13.4.3监测缺陷报告的度量指标

然而在第二周解决的缺陷的实际数量远低于预计的数量。这表明软件开发人员在解决缺陷时比预计的要慢。在第二周结束时，打开的缺陷的总数要高于估计的数量。在这个阶段，软件开发经理必须采取措施来提高缺陷消除率，可以通过增加工作时间或者把其他项目的软件开发人员调到Bayonet项目中。在这种情况下，经理让有经验的软件开发人员加入到该项目中来加快显著性缺陷的解决过程。建议在测试周期开始之前，先实现表格的上半部分。表格的剩下部分随着测试周期的进展而逐步展开，以便开发团队中的每个人都能注意到缺陷消除的进度。

13.4.3监测缺陷报告的度量指标

表 13.11　Bayonet 项目的 ARD 度量指标

周	构件	提交的				已解决的				打开的			
		总数	P1 + P2	P3	P4	总数	P1 + P2	P3	P4	总数	P1 + P2	P3	P4
						计划的							
										184	50	63	71
1	build10	75	24	36	15	118	38	60	20	141	36	39	66
2	build11	75	24	36	15	118	38	60	20	98	22	15	61
3	build12	75	24	36	15	95	38	37	20	78	8	14	56
4	build13	14	5	7	2	20	10	5	5	72	3	16	53
						实际的							
1	build10	67	23	27	17	119	37	49	33	132	36	41	55
2	build11	77	26	36	15	89	37	37	15	120	25	40	55
3	build12	62	18	32	12	100	36	52	12	82	7	20	55
4	build13	27	11	12	4	33	15	8	10	76	3	24	49

目录

CONTENTS

13.6 缺陷原因分析

缺陷原因分析DCA(Defect Causes Analysis)，用来有效地在低成本下提高产品的质量。

13.6 缺陷原因分析

缺陷原因分析DCA可以做到：

- 决定/发现错误发生的原因；
- 采取措施阻止类似错误在将来发生；
- 排除可能存在于系统中的类似缺陷，或者于软件开发阶段在最早可能的时间点进行甄别。

13.6 缺陷原因分析

驱动**缺陷原因分析**的三个关键原理：

- **减少缺陷数量以提高质量：**缺陷多不代表软件质量低；通过关注缺陷预防和早期缺陷检测可以提高软件质量。
- **当缺陷发生时请教本地专家：****DCA**应在发生源执行（缺陷出现在哪/由谁产生，就在哪/由谁进行**DCA**）
- **关注系统性错误：**类似或同样的错误在很多场合下经常发生，则认为该错误为**系统性错误**；识别和预防系统性错误能有效提高产品质量。

13.6 缺陷原因分析

识别系统性错误的五个步骤（问题）

1

何时检测到失败，相关失败是何时注入的？确定开发阶段引入的相关缺陷。辨识相关的问题域。

13.6 缺陷原因分析

识别系统性错误的五个步骤（问题）

2

用什么方式对错误进行分类?

通过回答这个问题识别出重要错误的类别。错误的分类基于执行的工作性质、产生错误的概率和项目的当前进展。

13.6 缺陷原因分析

识别系统性错误的五个步骤（问题）

3

什么是公共的问题（系统性的错误）？第2步识别出的一个错误分类给出了一个系统性错误的提示。

13.6 缺陷原因分析

识别系统性错误的五个步骤（问题） 4

什么是错误的主要原因？这个问题的回答需要对产品的深入认识和充足的经验指导。

一般原因：

工具：复杂、不可靠或工具存在缺陷

输入：不完整、模棱两可或存在错误

方法：不完整、模棱两可、有错误或不可靠

人员：缺乏专业的培训或者不能理解项目

13.6 缺陷原因分析

识别系统性错误的五个步骤（问题）

5

将来如何防止错误的发生?

三种行动：

预防：早期监测问题并将其修复

纠正：修复问题

减轻：试图缓解问题的不利后果

例：培训、改进沟通方式、使用工具、过程改进。

目录

CONTENTS

13.7 beta测试

beta测试，是由潜在的买家在正式版本发布前进行的。

目的：不是为了发现缺陷，而是为了从现场获得有关产品的易用性的反馈。

13.7 beta测试

beta测试的三种类型：

- **市场beta**：在潜在买家中建立对产品的早期意识和兴趣。
- **技术beta**：为了在真实的环境下，使用不同的配置，从**少数友好用户**获取有关产品易用性的反馈。
- **验收beta**：确保该产品是满足验收标准的，即产品履行了买卖双方的合同。

13.7 beta测试

beta测试的**发布时机**和**发布标准（表13.13）**

1.所有测试用例的 98%已经通过

2.系统在最后一周的所有测试中没有宕机

3.所有已解决的缺陷都必须处于关闭状态

4.仍处于打开状态且具有变通措施的所有缺陷的版本注释必须是可用的。如果不存在变通措施，那么有关对于客户影响的说明必须放入版本注释中

5.没有严重性为“关键”的缺陷处于打开状态

6.没有严重性为“高”的缺陷有高于0.1的概率在客户现场发生

13.7 beta测试

beta测试的**发布时机**和**发布标准（表13.13）**

7.产品没有多于一定数量的严重性为“中”的缺陷

8.性能测试的所有测试用例必须都通过，而且结果必须对于beta 客户是可用的

9.源于所有的潜在beta客户的beta测试计划必须是可用的

10.用户指南草案必须是可用的

11.有关系统的培训资料对于所有现场工程师来说是可用的

12.必须有beta支持团队与每个beta客户的周例会

13.所有识别的阻塞缺陷都处于关闭状态

目录

CONTENTS

13.9 系统测试报告

在所有测试周期结束后，创建最终的**系统测试报告**。

13.9 系统测试报告

最终系统测试报告的结构

1、测试项目导言
2、测试结果综述
3、性能特征
4、扩展限制
5、稳定性观察
6、系统的互操作性
7、硬件/软件兼容性矩阵
8、遵守需求状态

13.9 系统测试报告

最终系统测试报告的结构

1、测试项目导言

2、测试结果综述

3、性能特征

4、扩展限制

5、稳定性观察

6、系统的互操作性

7、硬件/软件兼容性矩阵

8、遵守需求状态

概述报告的目的，包括：

- 测试项目名称
- 软件镜像名称
- 文档的修正历史
- 术语和定义
- 测试人员
- 文件的范围
- 参考文献

13.9 系统测试报告

最终系统测试报告的结构

1、测试项目导言

2、测试结果综述

3、性能特征

4、扩展限制

5、稳定性观察

6、系统的互操作性

7、硬件/软件兼容性矩阵

8、遵守需求状态

- 开始和结束日期；
- 归档的缺陷数量；
- 在不同状态的缺陷数量；
- 系统中仍然存在的问题，包括任何不足之处或可能发生的失败，根据它们的严重性等级和对客户的潜在影响进行总结。

13.9 系统测试报告

最终系统测试报告的结构

1、测试项目导言
2、测试结果综述
3、性能特征
4、扩展限制
5、稳定性观察
6、系统的互操作性
7、硬件/软件兼容性矩阵
8、遵守需求状态

描述系统的响应时间、吞吐量、资源利用率和延迟测量结果，以及在测试环境和测量过程中使用的工具。

13.9 系统测试报告

最终系统测试报告的结构

1、测试项目导言
2、测试结果综述
3、性能特征
4、扩展限制
5、稳定性观察
6、系统的互操作性
7、硬件/软件兼容性矩阵
8、遵守需求状态

限制部分的声明。包括设备和工具。

13.9 系统测试报告

最终系统测试报告的结构

1、测试项目导言
2、测试结果综述
3、性能特征
4、扩展限制
5、稳定性观察
6、系统的互操作性
7、硬件/软件兼容性矩阵
8、遵守需求状态

- 系统正常工作的小时数；
- 不同组观察到的崩溃次数；
- 缺陷造成的系统崩溃（处于无法重现状态）的描述；

13.9 系统测试报告

最终系统测试报告的结构

1、测试项目导言
2、测试结果综述
3、性能特征
4、扩展限制
5、稳定性观察
6、系统的互操作性
7、硬件/软件兼容性矩阵
8、遵守需求状态

针对第三方软件和硬件系统的互操作性测试。

13.9 系统测试报告

最终系统测试报告的结构

1、测试项目导言
2、测试结果综述
3、性能特征
4、扩展限制
5、稳定性观察
6、系统的互操作性
7、硬件/软件兼容性矩阵
8、遵守需求状态

软硬件兼容情况描述

13.9 系统测试报告

最终系统测试报告的结构

1、测试项目导言
2、测试结果综述
3、性能特征
4、扩展限制
5、稳定性观察
6、系统的互操作性
7、硬件/软件兼容性矩阵
8、遵守需求状态

- 遵守、部分遵守或不遵守软件镜像的需求；
- 通过测试、分析、展示和审查验证的需求；
- 测试套件中每个测试用例连同测试用例的状态所包含的需求；

目录

CONTENTS

系统测试的执行

End