单元测试

Unit Test

目录

CONTENTS

目录

CONTENTS

目录

CONTENTS

3.1 单元测试的概念

问题：

1. 什么是单元测试?
2. 为什么要独立进行单元测试?
3. 如何测试?
4. 测试效果如何?
5. 由谁来测试?
6. 单元测试的阶段?

3.1 单元测试的概念

单元测试是指**独立（孤立）**地测试**程序单元**（一个**程序单元**可以看成是实现“低”层次函数的一段代码）

3.1 单元测试的概念

为什么要**独立（孤立）**测试?

1、测试中发现的错误一般仅限于一个特定的单元，错误更易修复。单元测试移除了对于其他单元的依赖性

2、在单元测试中，需要验证程序单元每次不同的执行是否得到了相应的期望结果（通过直接访问单元的输入向量，能更容易进行多路径测试）

3、多单元同时测试容易出现下述**问题**：单元间关系使得被测单元中不同路径的执行变得比较困难

3.1 单元测试的概念

如何测试?

执行每一行代码（充分认识程序）

执行单元中每个谓词，判断是否为true或false

观察单元执行了它应有的功能，并确保没有已知错误发生

3.1 单元测试的概念

测试效果?

不能保证一个满意的被测单元在系统视角上功能正确

(这意味着某些错误只能在集成测试和系统测试该单元与其他单元集成时才能发现)

3.1 单元测试的概念

由谁测试?

由负责构建程序单元的程序员来执行

(程序员熟悉单元细节且需要对自己工作的质量负责，也只能由其负责)

3.1 单元测试的概念

单元测试的阶段	**静态单元测试**（基于非执行的）： 代码评审（主要方式）
	动态单元测试（基于执行的）： 通过实际运行代码来测试代码

静态单元测试与动态单元测试需要都被执行，静态单元测试必须在最终的动态单元测试之前执行

目录

CONTENTS

- 单元测试的概念
- **静态单元测试**
- 动态单元测试
- 调试
- 单元测试工具

3.2 静态单元测试

为什么要执行静态单元测试?

基于这样一种想法，软件产品在其生命周期的每个里程碑处都需要经过一个**检查和修复**阶段。检查的思想是尽可能在缺陷出现点附近发现这些错误，从而可以花费更少的精力来移除这些缺陷。静态单元测试（代码审查）的最大作用是因为其社会化属性。

3.2 静态单元测试

静态单元测试的方法：

代码评审

审查（inspection）：一步步的组内结对来检查一个产品，每一步都按照预先确定好的标准进行检查

走查（walkthrough）：作者带领团队使用预先定义好的场景来手动或模拟执行产品

3.2 静态单元测试

代码评审特点	检查源代码细节
	参与者是对软件开发项目拥有不同兴趣程度的设计和开发人员
	代码评审非常耗时
	一个设计良好的代码评审过程能够找到基于执行的测试可能无法找到的错误
	没有完美的检查过程（人的因素）

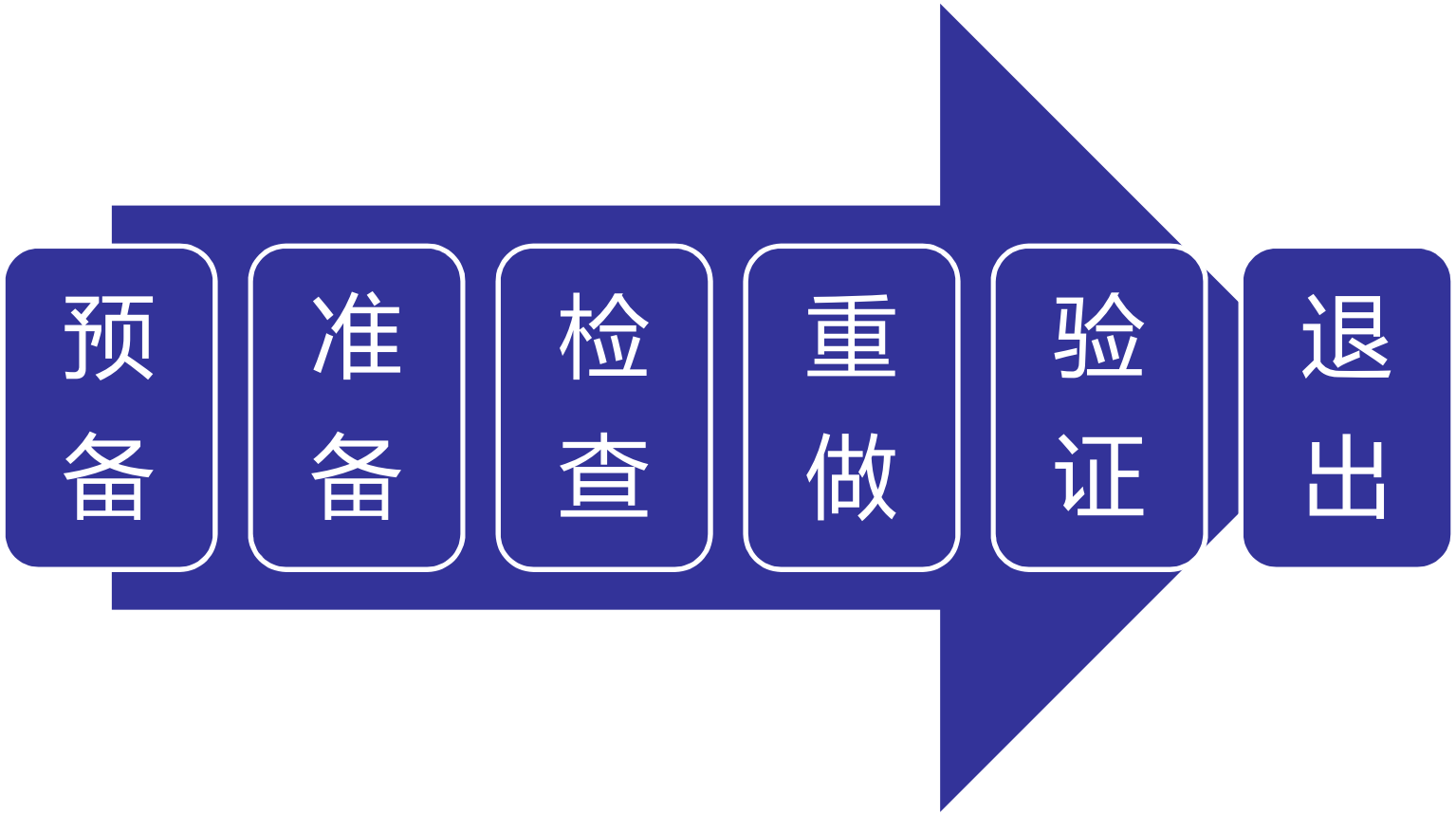
3.2 静态单元测试

人的因素

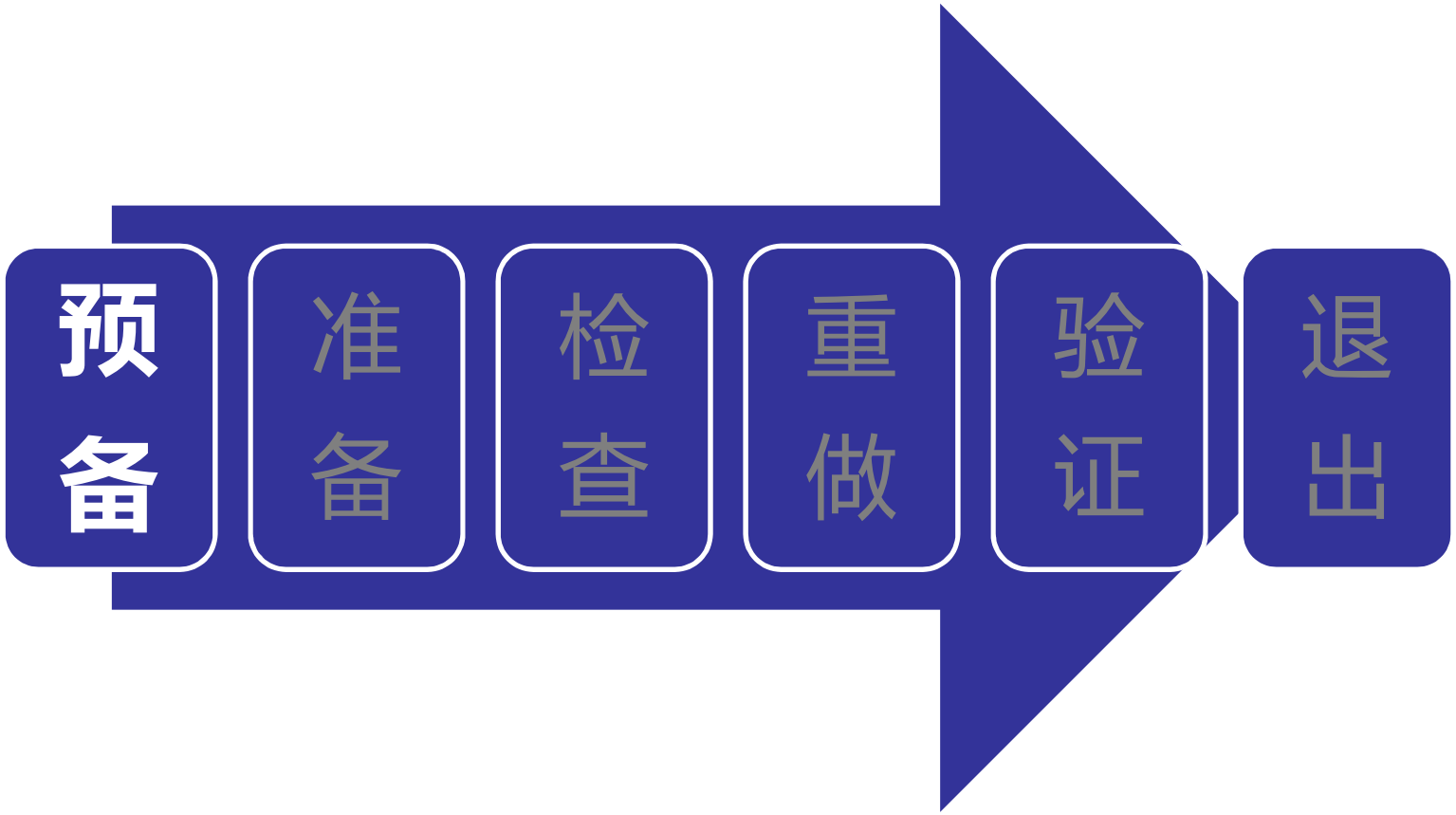
代码作者和检查人员之间可能会发生冲突，这样会使得讨论会议效率低下

需要专业的计划和管理，组内需要相互的尊重、公开、信任和专业技术的共享

3.2 静态单元测试-执行代码评审的步骤



3.2 静态单元测试-执行代码评审的步骤



执行代码评审的步骤1：预备

1、单元的作者确保被测单元已经可以进行检查

完整性：所有与单元检查有关的代码均是可用/完成的

最小功能性：代码需要能够通过编译和链接，必须已经经过了一定程度的测试以保证它能完成基本的功能

可读性：代码规范

复杂性：指代码中条件语句的数量，单元中输入数据元素的数量，单元生成的输出数据元素的数量，代码的实时过程，代码与其他单元交互的数量。**（过于简单则无需代码评审）**

需求文档和设计文档：低层次的设计规范（单元的详细设计）或其他对于程序需求的合适描述

执行代码评审的步骤1：预备

2、构造检查小组（角色分工）

主持人：负责检查会议的主持工作（项目无关人员，保持客观性）

作者：待检查的代码的编写人员

展示人员：预先阅读并理解代码的非作者人员（备份和鼓励角色）

记录人员：记录问题和建议采取的行动（非作者或主持人）

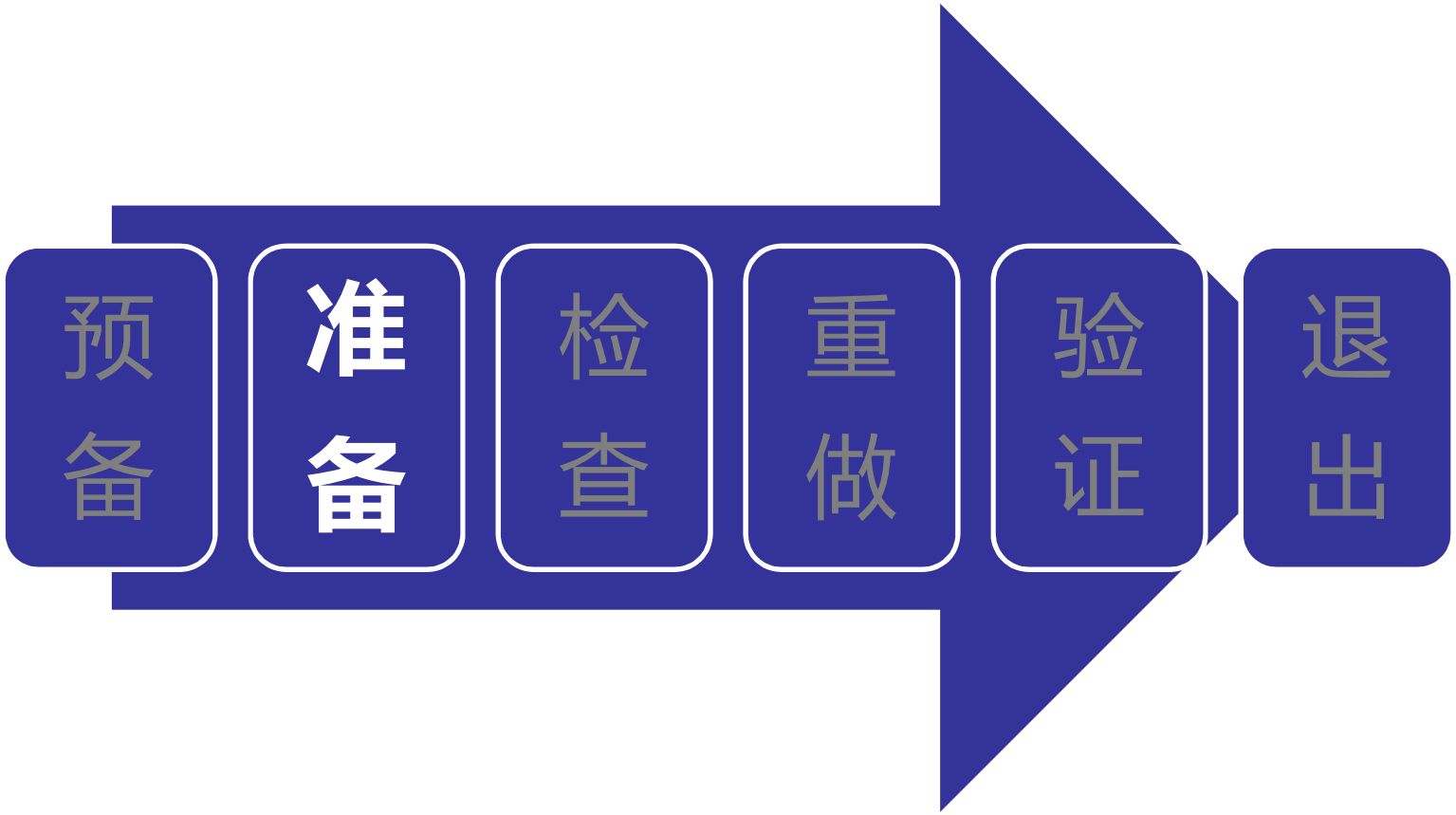
检查人员：代码评审领域的专家小组，3-7人构成

观察人员：想要学习待检查的代码，不参与检查过程

执行代码评审的步骤1：预备

3、提前两三天通知小组成员，提供待检查工作包副本以便小组提前阅读

3.2 静态单元测试-执行代码评审的步骤



执行代码评审的步骤2：准备

会议开始之前，检查人员应做到：

列出问题：想要向作者或其他组员提出的问题(表3.2)

表 3.2　代码评审的检查列表(checklist)

1. 代码确实实现了设计规范中指定的功能吗?
2. 模块中使用的过程是否正确地解决了问题?
3. 是否存在某个软件模块与另一个已经存在的、可以重用的模块重复?
4. 如果使用了库模块，是否使用了正确的库模块? 是否使用了库模块的正确版本?
5. 每个模块是否都只有一个进入点和结束点? 有多个进入点和结束点的程序更难测试。
6. 模块的环形复杂度是否大于 10? 如果是，那么充分测试该模块是异常困难的。
7. 每个原子函数可以在 10 ~ 15 分钟内检查和理解吗? 如果不是，则认为其过于复杂了。
8. 对于所有的标识符，比如指针、索引、变量、数组和常量，是否都遵循了命名规范? 实施编码标准对于系统的开发团队更易于引入一位新人(程序员)是非常重要的。
9. 代码是否有足够的注释?
10. 所有的变量和常量是否正确初始化了? 是否检查了正确的类型和范围?
11. 是否小心控制了全局和共享变量?
12. 数值是否硬编码在程序中? 应该将其声明为变量。
13. 指针是否正确使用?
14. 动态分配的内存块在使用后是否释放了?
15. 模块是否正常结束? 模块最终会结束吗?
16. 是否存在无限循环、永远不会执行的循环或异常退出的循环?
17. 在结束时，所有打开使用的文件是否都关闭了?
18. 是否有计算使用了数据类型不一致的变量? 是否有溢出(overflow)和下溢(underflow)的可能?
19. 当访问消息的一个公共表时，是否会产生错误代码和条件消息? 每个错误代码应该有一个含义，并且所有含义应该存储在表的一个位置以供查询，而不是分散在程序代码的各处。
20. 代码可移植吗? 源代码在其生命周期中，可能会在多种处理器结构和不同的操作系统上执行。它不能预先排除多种执行环境的可能实现。
21. 代码效率高吗? 一般来说，为了效率目标，清晰、易读或者正确性不能都满足。代码评审的目的在于发现不利于系统性能实现的选择。

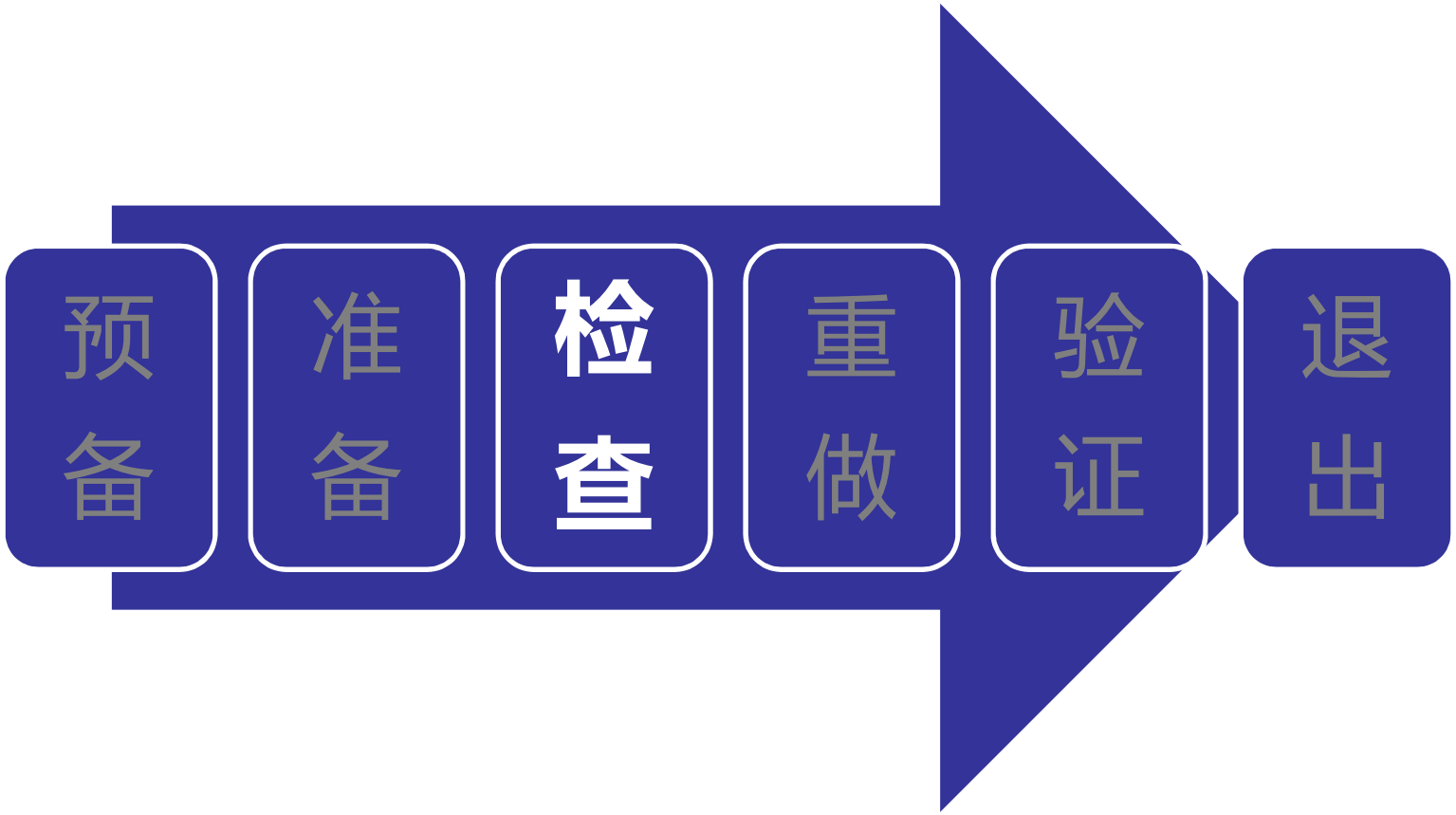
执行代码评审的步骤2：准备

会议开始之前，检查人员应做到：

提出潜在变更请求（CR）：不是错误报告。不会被包含在与产品相关的错误统计中

建议改进的机会：建议如何修复被检查代码中的问题

3.2 静态单元测试-执行代码评审的步骤



执行代码评审的步骤3：检查

检查

（会议）

1、作者讲解代码使用的逻辑，主要计算的路径，以及被检查单元与其他单元的依赖关系

2、展示人员逐行阅读代码。此时检查人员可以提出问题及修复错误的大概建议，由作者在会后决定采取何种修正措施

执行代码评审的步骤3：检查

检查

（会议）

3、记录人员记录变更请求和修复问题的建议。其中变更请求需包含如下细节：

a、简要描述问题和动作项

b、为该变更请求赋予一个优先级（主要或次要）

c、指定一个人员（可为提出该变更请求的检查人员）跟进这个问题

d、设定一个解决该变更请求的最后期限

执行代码评审的步骤3：检查

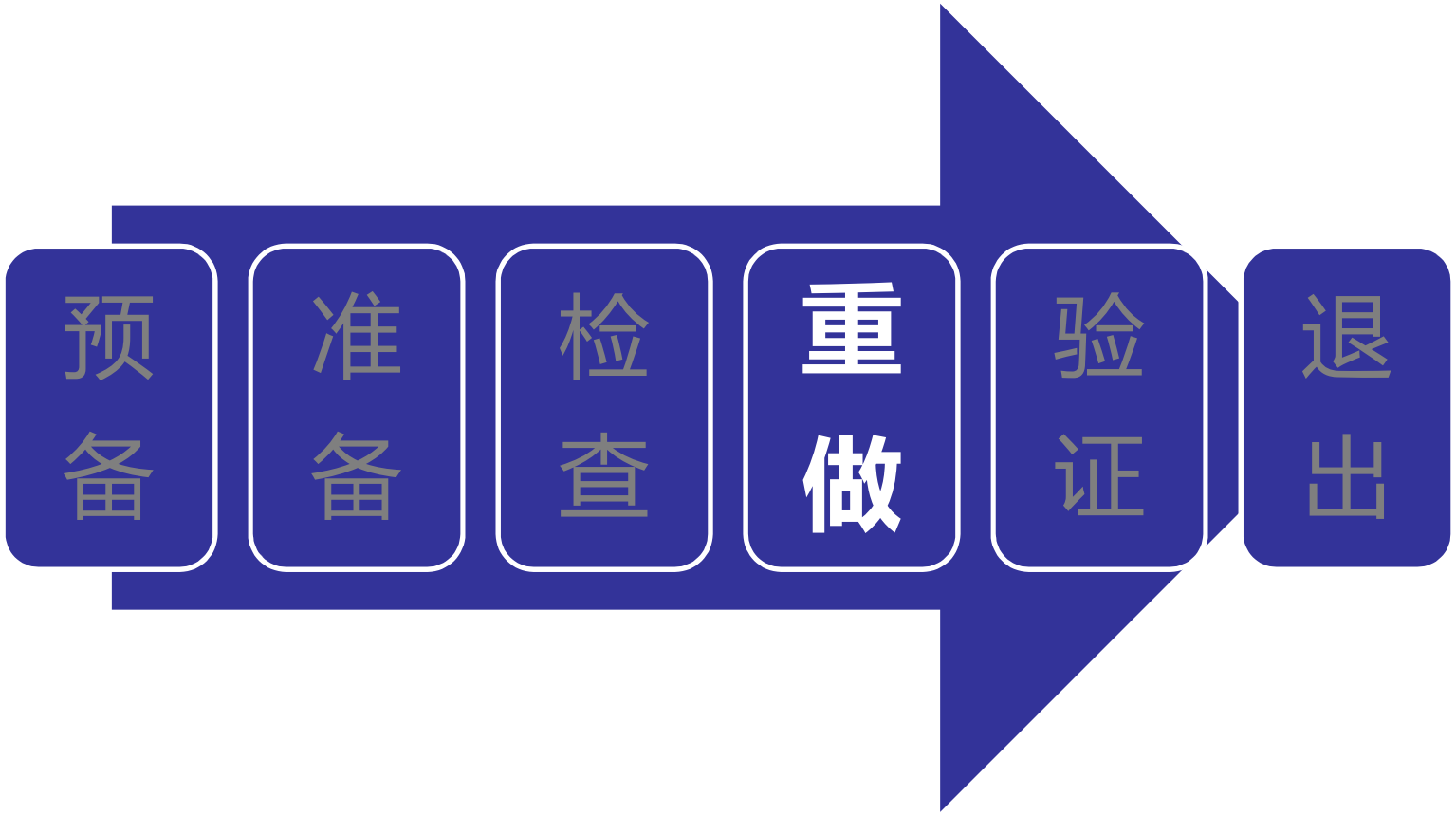
检查

（会议）

4、主持人确保会议始终围绕检查过程以实现会议的目标

5、会议末尾，决定是否需要进行另一次会议继续检查代码。如无新会议，则进入步骤4：重做

3.2 静态单元测试-执行代码评审的步骤



执行代码评审的步骤4：重做

重做

1、记录人员生成会议总结，包含如下信息：

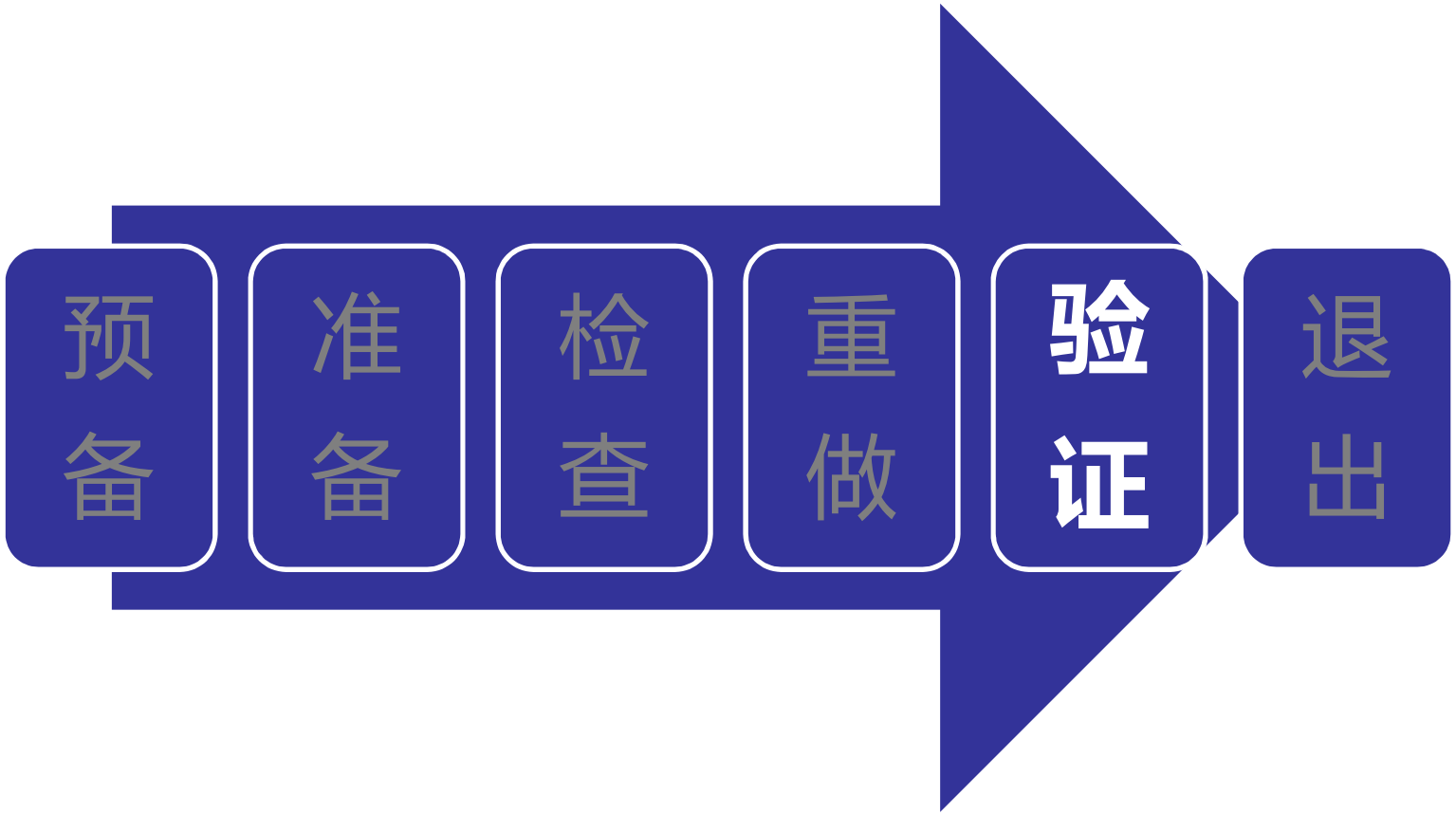
一个所有变更请求的列表，包括将被修复的日期，以及负责验证这些变更请求的人员名单；

一个改进建议列表；会议记录(可选)

2、分发总结副本给所有小组成员

3、会议结束后，作者争取在指定时间内解决变更请求，记录对代码的改进

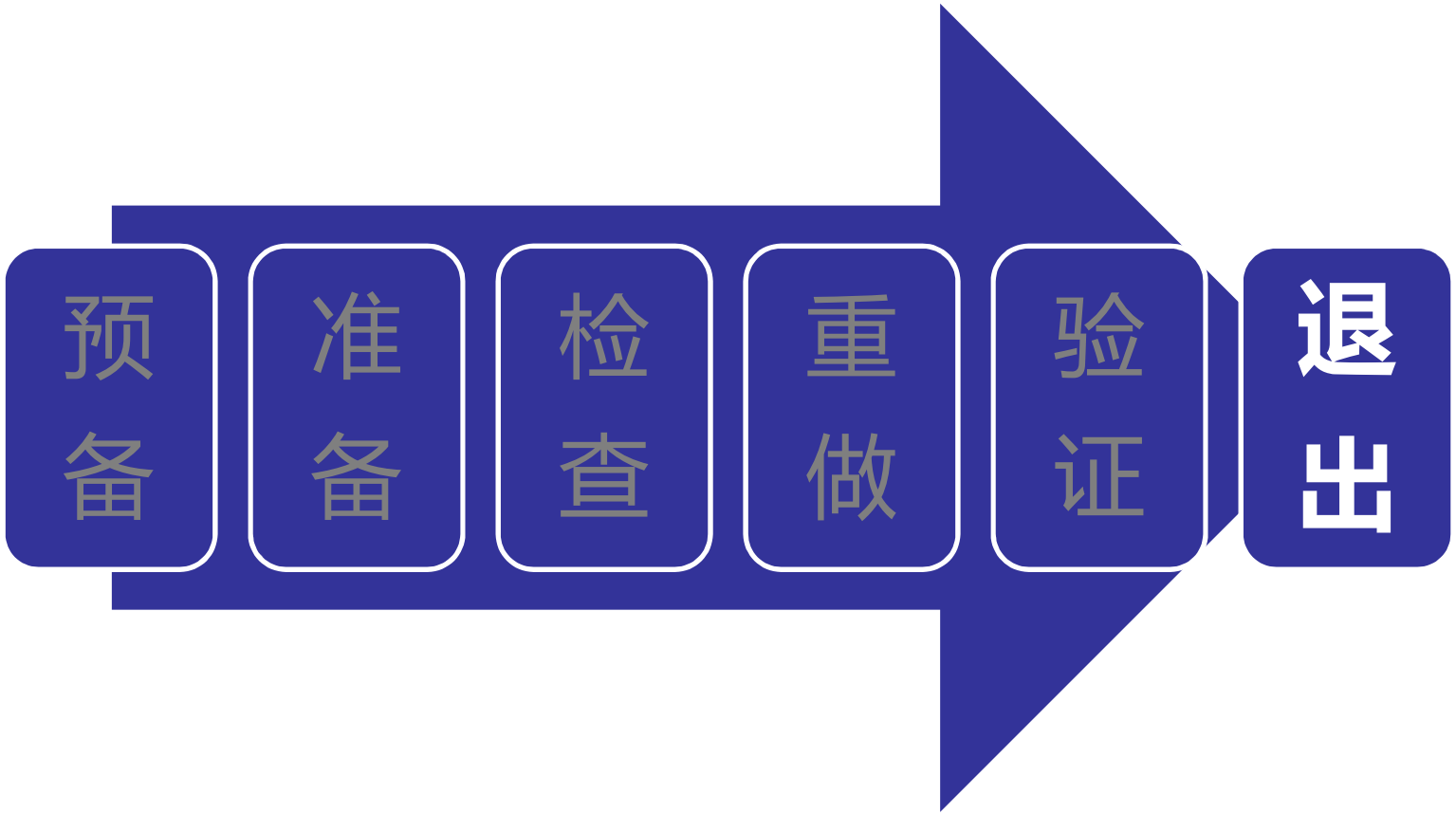
3.2 静态单元测试-执行代码评审的步骤



执行代码评审的步骤5：验证

由主持人或者之前指定人员独立进行验证。过程包括检查变更请求中记录的修改后的代码，确保建议的改进已被正确实现

3.2 静态单元测试-执行代码评审的步骤



执行代码评审的步骤6：退出

总结检查过程，并执行下述步骤

1、确保单元中的每行代码都被检查了

2、如果错误过多（超过5%的代码行数），则在作者修正错误之后需要二次检查

3、作者和检查人员达成一致

4、所有的变更请求都已存档，并由主持人或其他人员验证

5、分发会议总结给所有检查小组人员

3.2 静态单元测试-代码评审指标

代码评审指标： 用于监测和度量代码评审的有效性	每小时检查的代码行数（LOC）
	每千行代码（KLOC）产生的变更请求数
	每小时产生的变更请求数
	每个项目产生的变更请求总数
	每个项目在代码评审上花费的时间

3.3 缺陷预防

为了减少在代码评审阶段可能出现的变更请求而制定的一系列编码指导

3.3 缺陷预防

1、将内部诊断工具，称为插桩代码（instrumentation code），写入单元中（实现内置跟踪和追踪机制的被动代码）

3.3 缺陷预防

2、使用标准控制来发现错误条件可能出现的情况（比如下标越界检查：assert(index>=0||index<=n)）

3.3 缺陷预防

3、确保对于所有返回值（有效或无效）均有代码处理

3.3 缺陷预防

4、确保计数的数值与缓冲区上溢和下溢的问题得到很好的处理

3.3 缺陷预防

5、由统一来源（知识背景）提供错误消息和帮助文本，以便文本的变化不会引发不一致性

3.3 缺陷预防

6、验证输入数据（比如传递给函数的参数）

3.3 缺陷预防

7、使用**断言**来发现不可能的条件、未定义的数据使用和异常的程序行为。

断言是对一个应该为真的条件的检查。例如：

在执行单元前，确保前置条件（执行前对单元状态的期待状态）得到满足。

当从单元退出时，确保预期的后置条件（执行后对单元状态的期待状态）为真。

确保不变量保持一定状态。

3.3 缺陷预防

8、在代码中放置**断言**(Debug和Release)

3.3 缺陷预防

9、对于意义不明的**断言**，应用充分的文档说明。

3.3 缺陷预防

10、在每次主要的计算后，反过来在代码中由结果计算输入并与实际输入进行比较验证（可能要容许误差）

3.3 缺陷预防

11、在涉及消息传递的系统中，对缓冲区进行管理是一项非常重要的内部活动

3.3 缺陷预防

12、开发计时器例程，以保证程序能结束

3.3 缺陷预防

13、在每个循环中包含一个循环计数器，用于判定循环次数是否小于最小可能执行次数和大于最大可能次数

3.3 缺陷预防

14、定义一个指示哪一个判定逻辑分支将会执行的变量，用于发现判定条件可能出现的问题

目录

CONTENTS

- 单元测试的概念
- 静态单元测试
- **动态单元测试**
- 调试
- 单元测试工具

3.4 动态单元测试

基于执行的单元测试称为动态单元测试，其特点为：

1、被测单元从其实际执行环境中**分离**出来

2、实际的执行环境通过编写更多的代码来**模拟**，被测单元和模拟环境编译到一起

3、上述经过编译的集合体用选择的输入来执行，执行的结果采用多种方式收集（直接显示在屏幕，记录在文件，插桩代码查看运行时状态等），将结果与期望结果比较以判定是否失败（代码中存在错误）

3.4 动态单元测试

如何构建单元测试环境？（模拟被测单元的上下文）

1、单元测试的**上下文**包括：a、单元的调用者；b、该单元调用的所有单元

2、模拟环境必须简单，以便运行该单元发现的任何错误都仅仅是被测单元**自身**的原因

3、设计**测试驱动(test driver)**来作为调用者单元，设计**桩(stub)**来作为被测单元调用的单元；称测试驱动和桩为**脚手架(scaffolding)**

3.4 动态单元测试

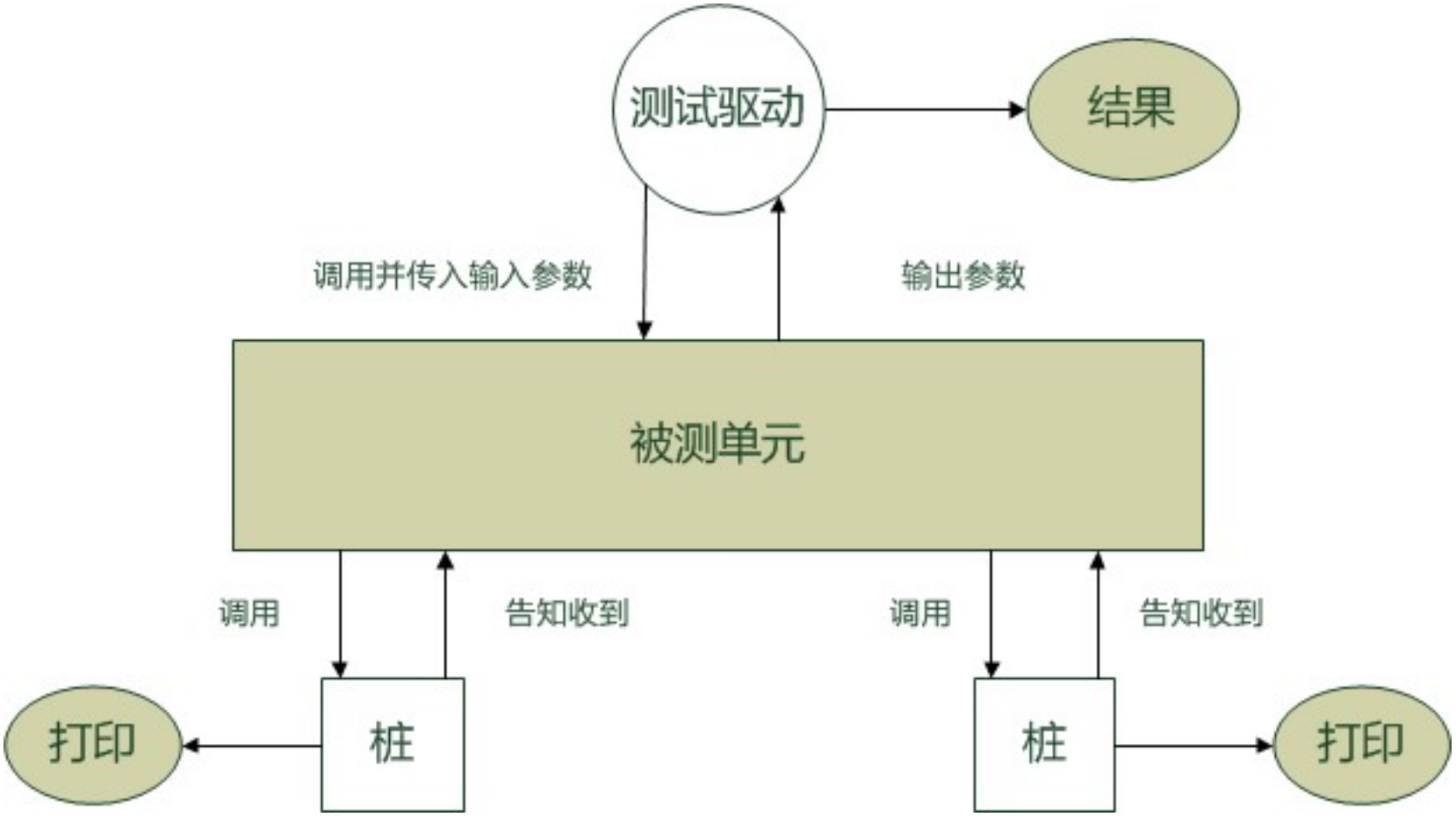


图3.2 动态单元测试环境

3.4 动态单元测试

测试驱动

1、测试驱动是调用被测单元的程序（主单元）

2、为被测单元提供预期格式的输入数据

3、被测单元根据从驱动得到的输入值来执行，在结束前将返回值传递给驱动

4、测试驱动比较实际输出与期望输出，报告测试结果

5、帮助被测单元的编译

3.4 动态单元测试

桩

1、是替代被测单元调用的单元的“哑子程序”(dummy subprogram)

2、通过打印一条消息表明桩确实被调用了

3、桩向被测单元返回一个事先计算好的值，以便被测单元可以继续其执行过程

3.4 动态单元测试

一般情况下**被测单元、测试驱动和桩**为**1:1:n**的关系，它们一起构成一个**单元测试套件**，有时输入数据独立为单独文件

三者**紧密耦合**

1、每次修改单元时，均需确认是否要对测试驱动和桩进行对应修改

2、每当发现了单元中的一个新错误，测试驱动应该更新其输入数据集以发现该错误和与其相似的错误

3、如果单元运行在不同平台上，也需要开发测试驱动和桩以便在新平台上测试该单元

4、确认测试驱动、桩和被测单元间的交叉引用，然后一起作为一个产品放入版本控制系统

3.4 动态单元测试-具体方法

控制流测试(Ch4)	a、画出程序单元**控制流图**
	b、选择若干**控制流测试标准**
	c、在控制流图中确定满足选择标准的**路径**
	d、从选择的路径生成**路径谓词表达式**
	e、通过**计算**路径谓词表达式，为程序单元生成输入值，即执行对应路径的测试用例

3.4 动态单元测试-具体方法

数据流测试(Ch5)	a、画出程序单元**数据流图**
	b、选择若干**数据流测试标准**
	c、在数据流图中确定满足选择标准的**路径**
	d、从选择的路径生成**路径谓词表达式**
	e、通过**计算**路径谓词表达式，为程序单元生成输入值，即执行对应路径的测试用例

3.4 动态单元测试-具体方法

域 测 试 (Ch6)	定义**域错误**概念，专门选择测试用例数据来寻找这类错误（主动发现错误）

3.4 动态单元测试-具体方法

功能程序测试(Ch9)	a、确定程序的输入域和输出域
	b、对于一个给定的输入域，选择一些特殊的值，并计算预期结果
	c、对于一个给定的输出域，选择一些特殊的值，计算将使单元产生这些输出值的输入值
	d、考虑以上选择的输入值的各种组合

3.5 变异测试

变异测试是一种考察测试数据（测试用例）**完备性**的技术，是**衡量测试用例质量**的一个手段（附带实现更多的单元测试），通常用于传统单元测试技术的补充

3.5 变异测试

概念

程序的一次**变异**：对程序源代码**一次单一、微小、符合语法**的更改。修改后的程序称为一个**变异体**

当测试用例执行引起变异体失败时，则称为变异**被杀死(killed)**，该变异体称为“**死亡了**”

如果一个变异体总是产生和原始程序相同的输出，则称它与原始程序**等价**，反之则为**不等价**

如果现有测试用例没有杀死一个变异体，则称这个变异体为“**可杀死的**”或“**顽强的**”

变异分数(mutation score)：不等价的变异体被一组测试用例杀死的百分比。如果一个测试套件的变异分数是100%，则称为**变异充分**

3.5 变异测试

变异测试/分析的一般步骤

1、测试套件是否能区分原始程序和变异体决定了它的充分程度。一个测试套件可能无法杀死所有的非等价变异体

2、向测试套件中添加新的测试用例，从而杀死这些顽强变异体。直到达到期望的变异分数为止。**（迭代增强测试套件的测试能力）**

3.5 变异测试-例子

程序P

查找数组中最大值**第一次**出现的位置

```
1. main(argc,argv)
2. int argc, r, i;
3. char *argv[];
4. { r=1;
5.   for i=2 to 3 do
6.       if (atoi(argv[i])>atoi(argv[r])) r=i;
7.   printf( “Value of the rank is %d \n” , r);
8.   exit(0);}
```

3.5 变异测试-例子

程序P

设计测试套件

测试用例1：

输入：1 2 3；输出：Value of the rank is 3

测试用例2：

输入：1 2 1；输出：Value of the rank is 2

测试用例3：

输入：3 1 2；输出：Value of the rank is 1

3.5 变异测试-例子

程序P

生成变异体

变异体1：把第5行改为：**for** i=**1** to 3 do

变异体2：把第6行改为：**if** (**i**>atoi(argv[r])) r=i;

变异体3：把第6行改为：

if (atoi(argv[i])**>=**atoi(argv[r])) r=i;

变异体4：把第6行改为：

if (atoi(argv[**r**]>atoi(argv[r])) r=i;

3.5 变异测试-例子

程序P	变异体1和变异体3没有被杀死
对变异体执行测试套件	变异体2：测试用例2失败 **用例2**-输入：1 2 1；输出：Value of the rank is 3
	变异体4：测试用例1和2失败 **用例1**-输入：1 2 3；输出：Value of the rank is 1 **用例2**-输入：1 2 1；输出：Value of the rank is 1

3.5 变异测试-例子

程序P　变异体1实际上是**等价变异体**

分析　**变异体1**：把第5行改为：**for** i=**1** to 3 do

添加新测试用例：

输入：2 2 1

变异体3被杀死

变异体3：**if** (atoi(argv[i])**>=**atoi(argv[r])) r=i;

变异分数达到100%

3.5 变异测试

使用变异测试获取健壮测试套件的步骤：

步骤1：为程序P建议一组正确的测试用例集合T

步骤2：在程序P上执行T中的每个测试用例，如有失败，则修改程序，再测试。如果没有测试失败，则进入步骤3

步骤3：生成一组变异体$\{P_i\}$

3.5 变异测试

使用变异测试获取健壮测试套件的步骤：

步骤4：在每个P_i上执行T中的测试用例。则会出现如下情况：

a、P_i被杀死（现有测试用例是足够的，但不确定）

b、P_i没有被杀死，则要么P_i与P是等价的，这时不需做任何处理；要么P_i是顽强的，这时可能需要添加新的测试用例

3.5 变异测试

使用变异测试获取健壮测试套件的步骤：

步骤5：计算T的变异分数，**=100*D/(N-E)**，其中D是被杀死的变异体，N是变异体的总数，E是等价的变异体数目

步骤6：如果T的变异分数不够高，则需要设计新的测试用例添加到T中，回到步骤2，重复直到变异分数超过某个适当的阈值

3.5 变异测试

变异测试的两条假设：

1、开发人员能力假设，即认为开发人员的错误均为小错误

2、耦合效应（已经过理论证明）：复杂的错误都是和简单的错误相耦合，如果一个测试套件能够发现程序中所有的简单错误，它也将发现绝大多数复杂错误

3.5 变异测试

变异测试的基本前提

如果软件中包含错误，那么通常将有一组变异体，它们只能被某个测试用例杀死，并且该测试用例也能发现这个错误

3.5 变异测试

变异测试的成本耗费较高，一般需要自动化测试工具支撑

目录

CONTENTS

3.6 调试

调试是检验故障原因的**过程**，**目标**是分离错误并精确地识别其失败原因

3.6 调试

- 最好由编写代码的程序员**执行**
- 非常消耗**时间**和容易**出错**
- 通常有外界的**压力**
- 调试包括**系统性的评估**或者**直觉**
- 需要**运气**

3.6 调试

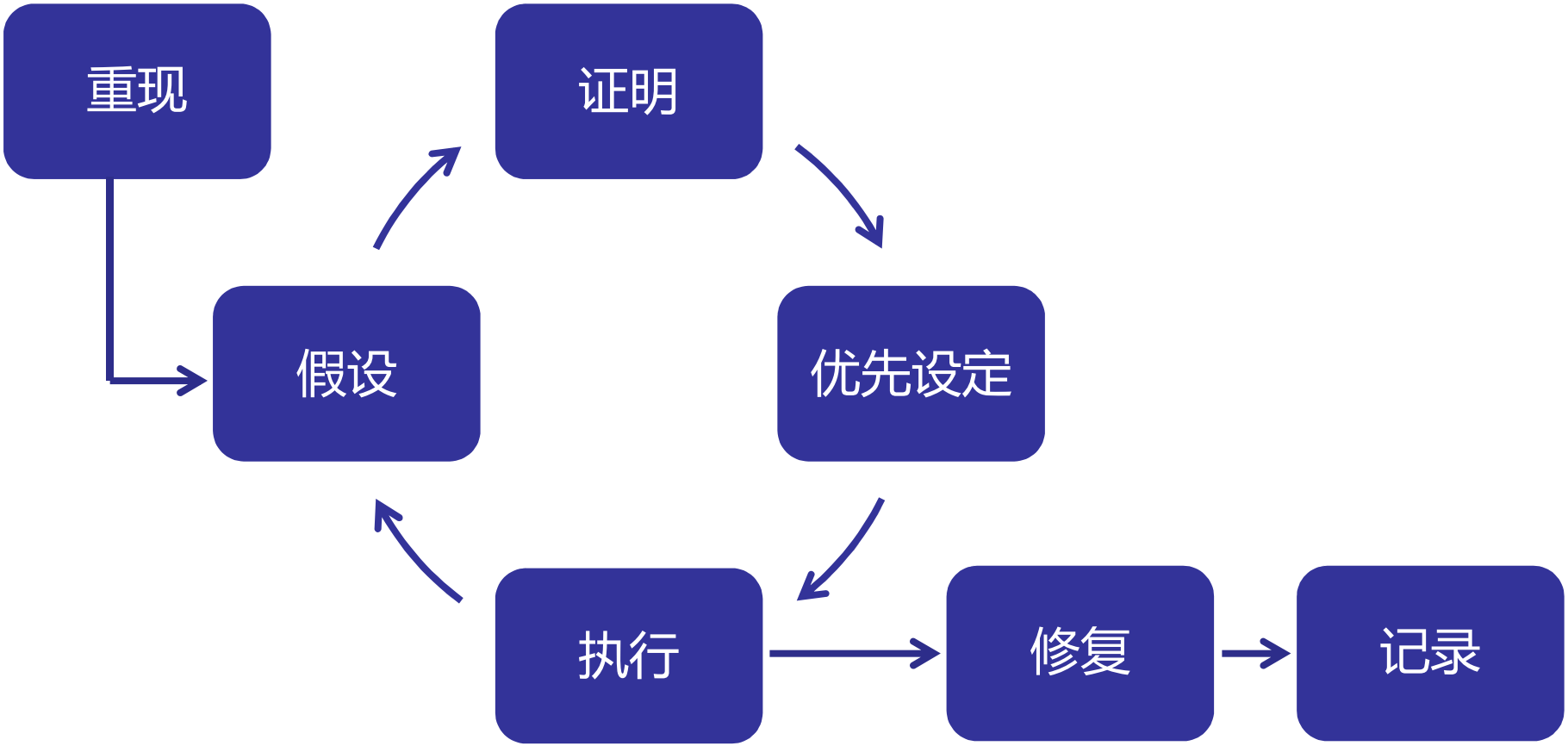
一般调试方法：

蛮力调试法（让电脑发现错误）：打印语句（跟踪），逐步执行代码（变量赋值注入），变量监测，桩代码，内存转储等

原因排除法（思维模式）：归纳和推理；利用测试来消除或证实依据可能性降序排列的每一个原因

回溯调试法（人力）：从失败的代码作为起始点，追溯执行的代码到达发生问题的点

3.6 调试-启发式方法



3.6 调试

调试启发式方法：

步骤1：**重现**问题现象（有时很难甚至不可能）

阅读产品的故障排除指南

利用诊断代码重现问题现象

收集所有信息，执行因果分析（目标是发现导致问题的根源）

3.6 调试

调试启发式方法：

步骤2：在因果分析的基础上制定一些**假设**的问题原因

步骤3：建立一个测试场景**证明**或**反驳**每个假设（新测试用例：无破坏性，低成本）

步骤4：制定测试用例**优先级**（可能性及成本）

步骤5：**执行**测试用例，以便找出现象的原因（执行用例->检查结果->寻找证据，分离缺陷或消除假设或回到步骤2）

3.6 调试

调试启发式方法：

步骤6：修复问题

1、简单修复或复杂修复（有严格步骤）

2、如果接受修改的单元已经完成代码审查，则需重做代码审查

3、代码审查后，应用修复部分

4、重新测试该单元以确保失败的实际原因已经发现。如果测试表明所观察到的失败不再发生，说明该单元已被正确调试和修复了

5、如果没有动态单元测试用例来揭示该问题，那么添加一个新测试用例，以发现可能的重复发生或其他类似的问题

6、回归测试

3.6 调试

调试启发式方法：

步骤7：**记录**变更

记录源代码本身的改变，以反映变化

更新整个系统的文件

动态单元测试用例的变更

如果代码已经提交至版本控制系统后发现问题，则在缺陷跟踪库记录故障

目录

CONTENTS

3.9 单元测试工具

编程工具
(代码编辑器、编译器、操作系统、调试器)

提高单元测试效率的工具

3.9 单元测试工具

1、代码审查器

用于检查软件质量以保证其符合最低编码标准（编码规范）；辨别不能移植的代码；对代码的结构和风格的改进提供建议；计算单位时间生产的LOC，计算错误密度（每KLOC的错误数量）

3.9 单元测试工具

2、边界条件检测器

检查意外写入内存指令区或数据存储区之外的数据。例如数组越界

3.9 单元测试工具

3、文档生成器

读取源代码，自动产生描述及源代码调用/被调用树状图或数据模型。例如类图，以及JavaDoc

3.9 单元测试工具

4、交互调试器

单步执行工具，断点设置工具（基于推理逻辑），万能调试器（ODB，非推理模式，简单追踪值至其出处）等

3.9 单元测试工具

5、内电路仿真器

ICE，用于嵌入式系统测试的模拟过程

3.9 单元测试工具

6、内存泄漏检测器

不合法的内存读取；读取还未初始化的内容；动态覆盖还没有分配给应用程序的内存

描述内存使用情况（源码级，即描述某行代码访问了哪个内存地址）

3.9 单元测试工具

7、静态代码（路径）分析器

基于代码结构识别需要测试的程序路径

(基于诸如McCabe环路复杂度测量标准等)

3.9 单元测试工具

8、软件审查支持工具

辅助制定小组的审查时间表；提供评审项目状态和接下来要开展的工作，以及分发问题解决的报告

3.9 单元测试工具

9、测试覆盖率分析器

测量内部测试覆盖率，报告覆盖率标准。跟踪和报告动态单元测试期间执行了那些路径，识别出动态单元测试中从未执行的源代码部分

3.9 单元测试工具

10、测试数据生成器

辅助程序员选择引起程序产生期望行为的测试数据

a、基于输入工具的特征描述，生成大量符合期望的数据集的变种；b、可以从源代码生成测试输入数据；c、可以生成与边界值等价的类和数值；d、可以计算边界值测试的期望范围；e、可以估计能够揭示错误的测试数据的可能性；f、可以生成数据，辅助变异分析

3.9 单元测试工具

11、测试装置

支持动态单元测试的执行，并使其可以

1.将测试单元安装的测试环境中

2.通过期望格式的输入数据来驱动测试单元

3.生成桩来模拟子模块行为

4.获取被测单元生成的输出

高级工具可以将实际结果和期望结果的比较

3.9 单元测试工具

12、性能监视器

监测和评估单元的时间特性，如延迟和吞吐量等

3.9 单元测试工具

13、网络分析器

测试网络操作系统，具有分析流量和识别问题区域的能力。提高网络安全监控能力

3.9 单元测试工具

14、模拟与仿真器

用于替代当前不能获得的真实软硬件（基于训练、安全和经济的考虑）

3.9 单元测试工具

15、流量生成器

生成大量数据，对接口和集成系统进行压力测试

3.9 单元测试工具

16、版本控制

提供存储开发中一系列软件修改和相关信息文件（源代码，编译代码，文档，环境信息等）的功能。目标是保证系统的和可追溯的软件开发过程。一些产品：SVN,GIT,VSS,TFS

进阶产品是配置管理系统(CMS)，控制全系统生命周期的变更控制

3.9 单元测试工具

16、版本控制和配置管理工具的特性

访问控制：监测和控制对组件的访问（读和写）

交叉引用：维护相关组件之间的联系，包括问题报告、组件、修改和文档

跟踪修改：维护组件所有的修改记录

发行版本生成：自动创建新的系统发行版本，以区分产品开发、测试和发行版本

系统版本管理：允许在不同系统版本间共享通用组件和系统版本的受控使用。支持并行开发和项目协作

压缩文档：支持已退役组件和系统版本的自动压缩存档

3.10 本章小结

本章主要是对单元测试的概念做了介绍。单元测试通过隔离分析的和执行的程序单元来识别错误；两种主要的，互补的测试类型：**静态单元测试**（代码审查）和**动态单元测试**（以受控的方式执行单元）

3.10 本章小结

代码评审（静态单元测试）过程分为六步：预备、准备、检查、重做、验证和退出

3.10 本章小结

阐述了一些在代码开发期间的预防性措施（**缺陷预防**），以减少程序中的错误

3.10 本章小结

在动态单元测试中，设计**测试用例**，设计**测试驱动**和**桩**辅助测试过程，通过实际运行程序来发现错误

3.10 本章小结

变异分析方法提高了测试套件的测试能力，可以作为单元测试的补充

3.10 本章小结

调试是检验故障原因的过程，**目标**是分离错误并精确地识别其失败原因，最好由编写代码的程序员进行，通常是一个非常消耗时间和容易出错的过程，而且有外界的压力。调试包括系统性的评估或者直觉，有时候还需要运气

3.10 本章小结

最后，本章介绍了一些可以改进单元测试效率的**工具**

单元测试

End