系统集成测试

System Integration Test

目录

CONTENTS

目录

CONTENTS

- 集成测试的概念
- **接口的不同类型和接口错误**
- 系统集成测试的粒度
- **系统集成技术**
- 系统集成的测试计划

目录

CONTENTS

7.1 集成测试的概念

软件**模块**或**组件**是软件系统中自我包含的元素

模块间有定义明确的**接口**

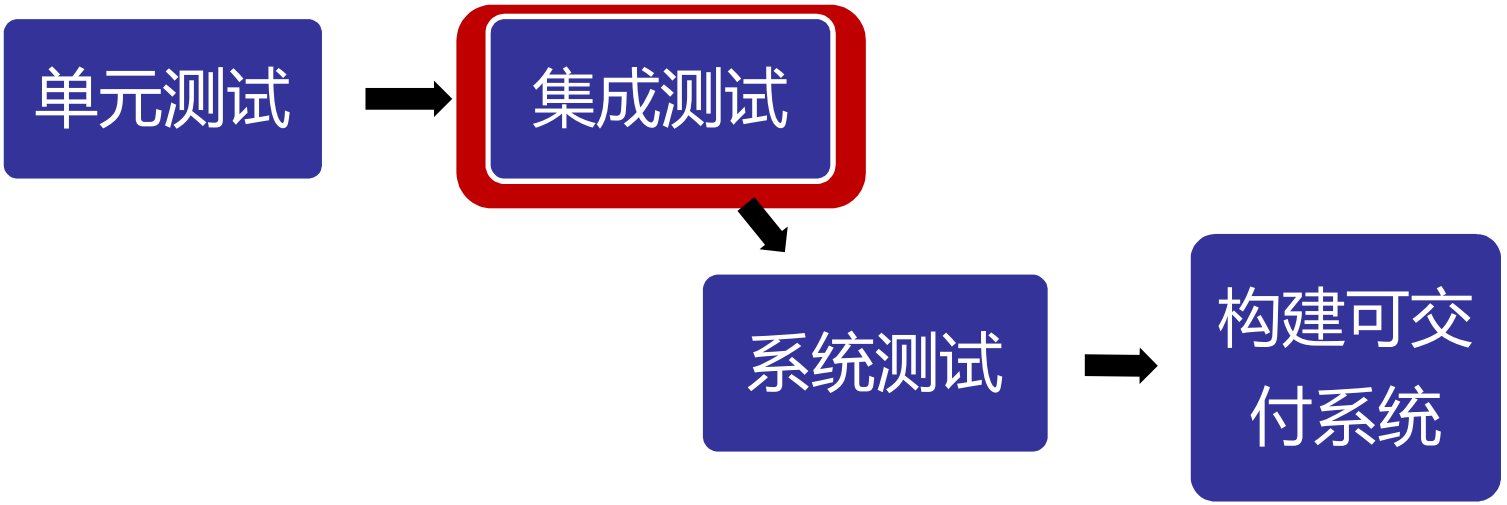
一个**模块**可以是一个子程序、函数、过程、类，或者由这些基本元素构成的组合，用以提供一个更高层次的服务

7.1 集成测试的概念

子系统是一个没有充分集成所有模块的过渡性系统，也称为**子组装体(subassembly)**

一个**系统**由以某种方式相互联系的模块组成，用以完成一个具体目标

7.1 集成测试的概念



7.1 集成测试的概念

集成测试是一项系统性的技术，用以在组装一个软件系统的同时，进行与**接口**相关的错误的测试。确保进行了单元测试的模块组合到一起能够按照设计的要求正常工作

7.1 集成测试的概念

集成测试通常由包含少量模块的小型子组装体扩大到包含越来越多模块的大型组装体

7.1 集成测试的概念

当系统得到了完全的整合，所有测试用例都已执行，发现并修复了所有严重的错误，以及重新测试了系统之后，就完成了**集成测试**

7.1 集成测试的概念

集成测试的目标：

在实验室环境中组装出一个相对稳定的系统，使其能够承受现实环境中严格、全方位的系统测试

7.1 集成测试的概念

集成测试的重要性

由不同开发人员，甚至是同一开发人员创建的模块间的接口错误是无法控制的

单元测试的有效性受开发人员有效测试所有路径的能力所制约。单元测试完成后也很难预测一个模块在真实环境中的行为

由于内部复杂性，一些模块比其它模块更容易出错。识别那些引起最大失败的模块是至关重要的

目录

CONTENTS

7.2 接口的不同类型和接口错误

系统的功能是通过模块间的**接口**交互来完成，**接口**实现了模块间的控制和数据的传递机制

7.2 接口的不同类型和接口错误

三种常用模块间接口

过程调用接口：模块中的一个过程调用其他模块中的一个过程。控制由调用方转到被调用方，同时传递数据。结束时，控制由被调用方转到调用方，同时回传数据

共享内存接口：两个模块共享一块内存。一个模块负责分配和写内存，另一个模块负责读

消息传递接口：一个模块在将数据结构的值域初始化之后，准备好一条消息并将其发送到另一个模块（B/S和C/S系统）

7.2 接口的不同类型和接口错误

接口错误是一些和结构相关的错误，这些结构位于模块的局部环境之外但为该模块所用

7.2 接口的不同类型和接口错误

接口错误的分类：

构造： 接口说明和实现分离（C或C++语言的头文件模式）

7.2 接口的不同类型和接口错误

接口错误的分类：

功能缺乏： 模块提供的功能少于其他模块所期望

7.2 接口的不同类型和接口错误

接口错误的分类：

功能位置：对功能位置的误解

7.2 接口的不同类型和接口错误

接口错误的分类：

功能改变：修改了一个模块，但相关模块没有同步改变

7.2 接口的不同类型和接口错误

接口错误的分类：

功能增加： 增加了新的功能模块但未向版本控制系统提交相关信息

7.2 接口的不同类型和接口错误

接口错误的分类：

接口误用：接口参数的个数、类型和顺序错误

7.2 接口的不同类型和接口错误

接口错误的分类：

接口误解：被调用模块可能假设了某些传入参数要满足一些特定条件，但是调用方并不能保证这些要求

7.2 接口的不同类型和接口错误

接口错误的分类：

数据结构变更：当一个数据结构的容量不够或者没有包含足够数量的信息域时就会发生这个问题

7.2 接口的不同类型和接口错误

接口错误的分类：

不充分的错误处理：调用模块可能没有合理地处理被调模块返回的错误码

7.2 接口的不同类型和接口错误

接口错误的分类：

错误处理的附加项：某个模块对于其他模块的错误处理在当其他模块发生变化时没有做相应变更

7.2 接口的不同类型和接口错误

接口错误的分类：

不充分的后处理： 未能释放某些不再使用的资源

7.2 接口的不同类型和接口错误

接口错误的分类：

不充分的接口支持：所提供的实际功能不足以支持接口规定的功能（例：将摄氏温度值传递给只能接收华氏温度的模块）

7.2 接口的不同类型和接口错误

接口错误的分类：

初始化/值错误： 未能对变量的数据结构进行初始化及赋值

7.2 接口的不同类型和接口错误

接口错误的分类：

违反数据约束： 程序的实现不支持数据项之间的某些关系

7.2 接口的不同类型和接口错误

接口错误的分类：

时效性/性能问题： 通信进程的不适当同步

7.2 接口的不同类型和接口错误

接口错误的分类：

变更协同：没有将一个模块的变更同步到和其相关的模块中

7.2 接口的不同类型和接口错误

接口错误的分类：

硬件/软件接口： 软件不能很好地处理硬件设备

7.2 接口的不同类型和接口错误

进行集成测试的优点

- 尽早发现错误
- 越早发现，越容易处理
- 对单个模块和整个系统的性能和可接受度有一个尽早的反馈
- 灵活地安排错误处理，并且能和开发同步

7.2 接口的不同类型和接口错误

集成测试的团队组成

创建模块的工程师：了解模块细节

系统集成测试人员：熟悉接口机制

系统架构师：对系统有比较宏观的把握

目录

CONTENTS

7.3 系统集成测试的粒度

1、系统内测试

这是**低层次**的基础测试，目标是将模块组合成为一个整体系统。测试用例来源于低层次的设计文档，其中详细描述了系统架构中这些模块的规范

7.3 系统集成测试的粒度

2、系统间测试

这是**较高层次**的测试阶段，需要被测系统间独立地进行交互。所有的系统都连接在一起，进行端到端测试。

端到端测试的目的是保证系统间的交互可以工作，而不是进行一个全面的测试，一次只在一定范围内测试一个功能。然后，在**系统测试**阶段根据需求进行全面测试。

测试用例来源于高层次的设计文档，这些文档详细描述了整个系统架构

7.3 系统集成测试的粒度

3、成对测试

在**系统内测试**和**系统间测试**之间的层次

一次只测试整个系统中相互连接的两个系统

其目的是保证待测的两个系统能够一起工作（此时假定系统中其它系统都能正常工作）

目录

CONTENTS

7.4 系统集成技术

集成测试从什么时候开始?

一旦相关模块可用（通过单元测试），就可以开始集成测试

7.4 系统集成技术

常用的集成测试方法

- **增量法（基本方法）**
- **自顶向下**
- **自底向上**
- **三明治方式**
- **大爆炸方式**

7.4 系统集成技术-增量法

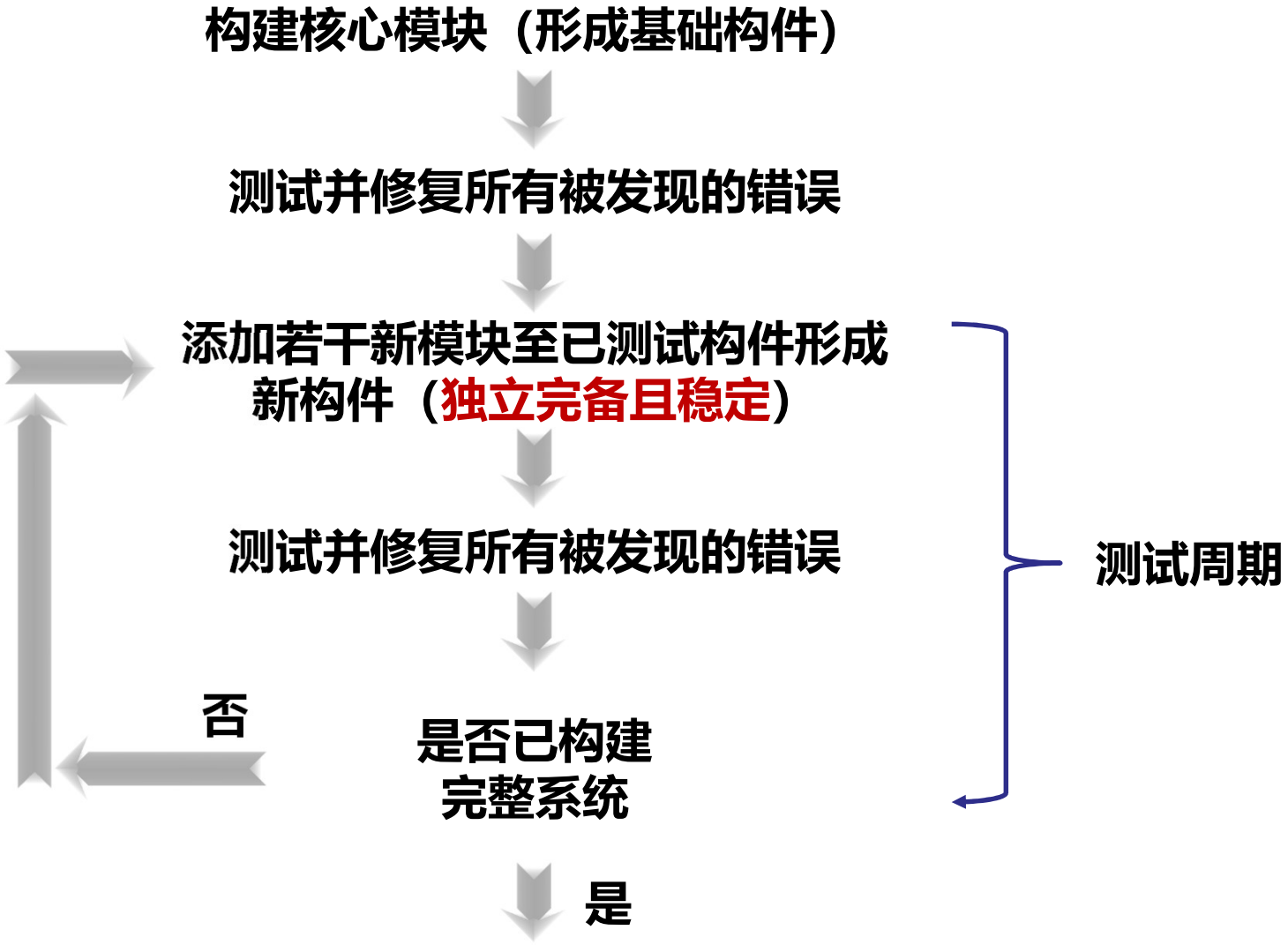
增量法： 是一个**循环迭代过程**

系统开始在核心模块开始进行构建，然后进入循环过程。在每一个测试循环中，更多的模块和已有的**被测构件**进行集成，从而创造一个更大的**构件**（**独立完备且稳定的**）

每完成一个测试**周期**，让开发人员修复所发现的所有错误，并进行下一个测试**周期**

整个系统就这样**周期地**进行增量构建，直到整个系统成为可操作的并能进行系统级的测试为止

7.4 系统集成技术-增量法



7.4 系统集成技术-增量法

集成测试的**周期数**和**整个集成时间**由以下几个参数决定

- 系统的模块数
- 模块的相对复杂程度（环形复杂度）
- 模块间接口的相对复杂程度
- 在每个测试周期中需要加入的模块数
- 待加入的模块是不是已得到充分的测试
- 每个测试-调试-修复周期的运行时间

7.4 系统集成技术-增量法

采用增量法进行集成测试的最终结果是一个过渡的**软件镜像。**其将会成为系统测试的对象

7.4 系统集成技术-增量法

构建**软件镜像**的步骤（管理过程）

1、收集最新的已测试单元及已认证的模块版本

2、编译这些模块的源代码

3、向代码库中提交这些编译后的代码

4、将这些已编译的模块链接到子系统中

5、验证子系统工作正常

6、**进行版本管理**

7.4 系统集成技术-增量法

创建**“日构件”**：强调小增量测试，稳定增加的测试用例数目，以及构件到构件的**回归测试**

7.4 系统集成技术-增量法

检入请求表（检入请求机制，表7.1）：审批通过后，模块方可进入集成环节

1、明确需要更新的所有文件，并且这些信息要为其他团队成员所知

2、在集成之前，新的代码必须经过检查

3、新的代码必须经过单元测试

4、明确检入的范围

7.4 系统集成技术-增量法

表 7.1　检入请求表

作者	申请这次检入的人员姓名
日期	年、月、日
检入请求日期	年、月、日
类型(标识所有适用的)	新功能:(是、否)
	改进:(是、否)
	缺陷:(是、否);如果是:缺陷数
	有主要缺陷吗?(是、否)
	有一般缺陷吗?(是、否)
检入的简短描述	用一个小段落描述待检入的功能、改进和缺陷修复
检入的文件数	给出待检入的文件数,如果可能包括文件名
代码评审人员姓名	提供代码评审人员的姓名
是否有命令行接口变更	(是、否);如果是,它们是:
	是否被记录?(是、否)
	是否被评审?(是、否、挂起)
这次检入是否包含全局头文件的变更?	(是、否);如果是,包含头文件名
这次检入是否包含输出日志的变更?	(是、否);如果是,它们被记录了吗?(是、否)
单元测试描述	所进行的单元测试的描述
备注	其他备注和问题

7.4 系统集成技术-增量法

构件对应的发布记录包含

- 从上一个构件以来，什么改变了?
- 哪些显著缺陷得到了修复?
- 这个构件中的显著缺陷有哪些?
- 加入了哪些新的模块和功能?
- 改进、修改或者删除了哪些现有的模块或功能?
- 有没有一些地方发生了不为人知的变更?

7.4 系统集成技术-自顶向下

自顶向下的步骤：

步骤1:令IM表示那些已经完成集成的模块集合和所需的桩。初始情况下，IM包含顶层模块和所有其下层模块对应的桩。假设顶层模块已经通过了进入条件。

步骤2:在IM集合中选择一个桩成员M ‘。令M是和桩M’ 对应的实际模块。我们将桩M ‘替换成M，并在CM中包含M的下层模块对应的所有的桩，这样可以从IM 得到一个新的集合CM。CM是四个集合的组合:{M}，CMs，Cmi，CMr。其中CMs是桩的集合，CMi是和M有着直接接口的模块集合，CMr是 CM 中余下的块。

7.4 系统集成技术-自顶向下

步骤3:现在，测试CM的组合行为。测试CM意味着对系统的顶层模块进行输入。需要注意的是，虽然集成团队能访问系统的顶层模块，但任何类型的测试都无法进行。这是很明显的事实，因为CM不代表整个系统。在这一步中，集成团队将测试由CM中实际模块所实现的系统功能的一个子集。

7.4 系统集成技术-自顶向下

集成团队进行两种测试:

1.运行测试用例以发现M和CMi 成员之间任何的接口缺陷。

2.进行回归测试以保证在M模块出现的情况下，CMi集合和CMr集合的模块集成符合要求。在前面的迭代中，CMi和CMr中模块间的接口已经过了测试和缺陷修复。但是，这个测试是基于M'(M的桩)而不是M本身。此时集成系统中M的出现能让我们对CMi和CMr并集中的模块间进行接口测试，这是因为系统现在可能支持M的更多的功能。

上述两类测试一直进行，直到集成团队确认没有已知的接口错误。一旦发现一个接口错误在进行下一阶段之前必须要修改这些错误。

步骤4: 如果CMs为空，则停止；否则，令IM=CM，跳至步骤2。

7.4 系统集成技术-自顶向下

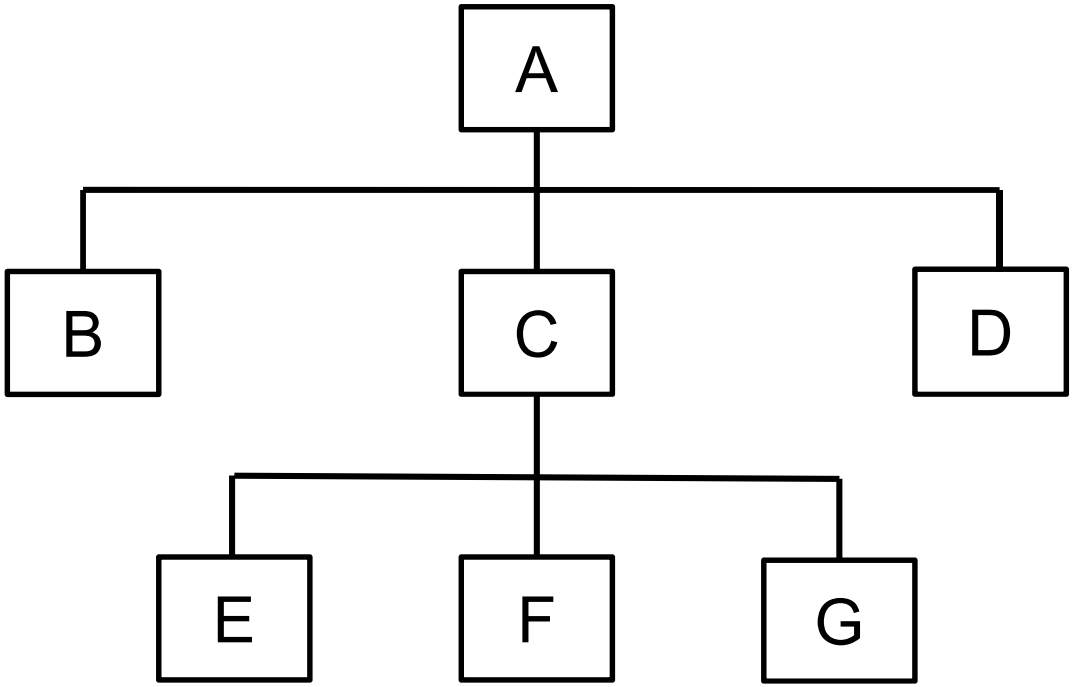


图7.1 一个有着3层、7个模块的模块层次

7.4 系统集成技术-自顶向下

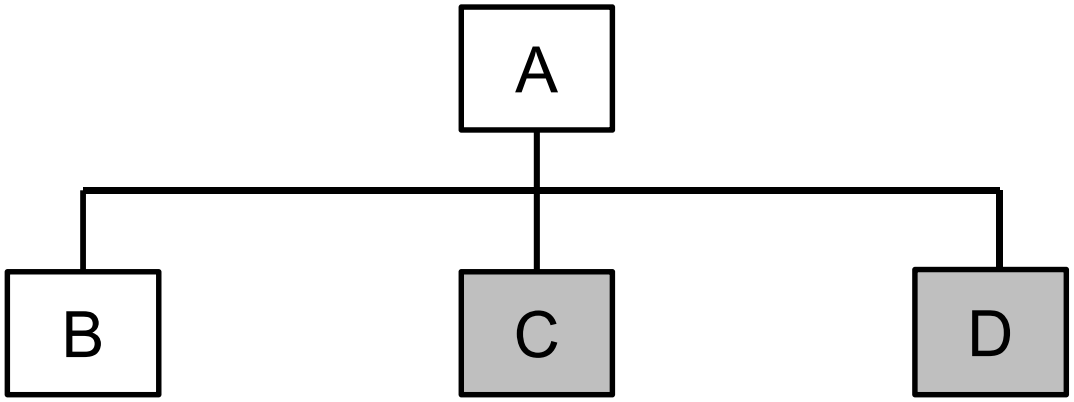


图7.2 模块A和B自顶向下集成

7.4 系统集成技术-自顶向下

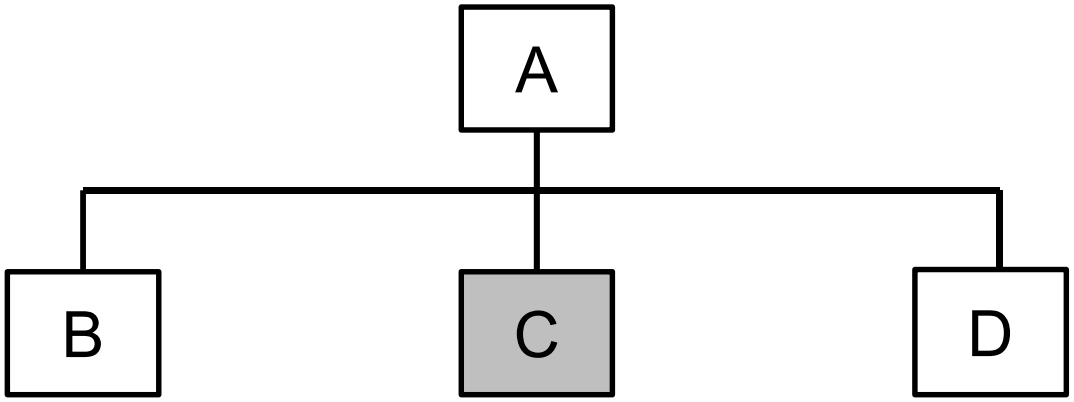


图7.3 模块A、B和D自顶向下集成

7.4 系统集成技术-自顶向下

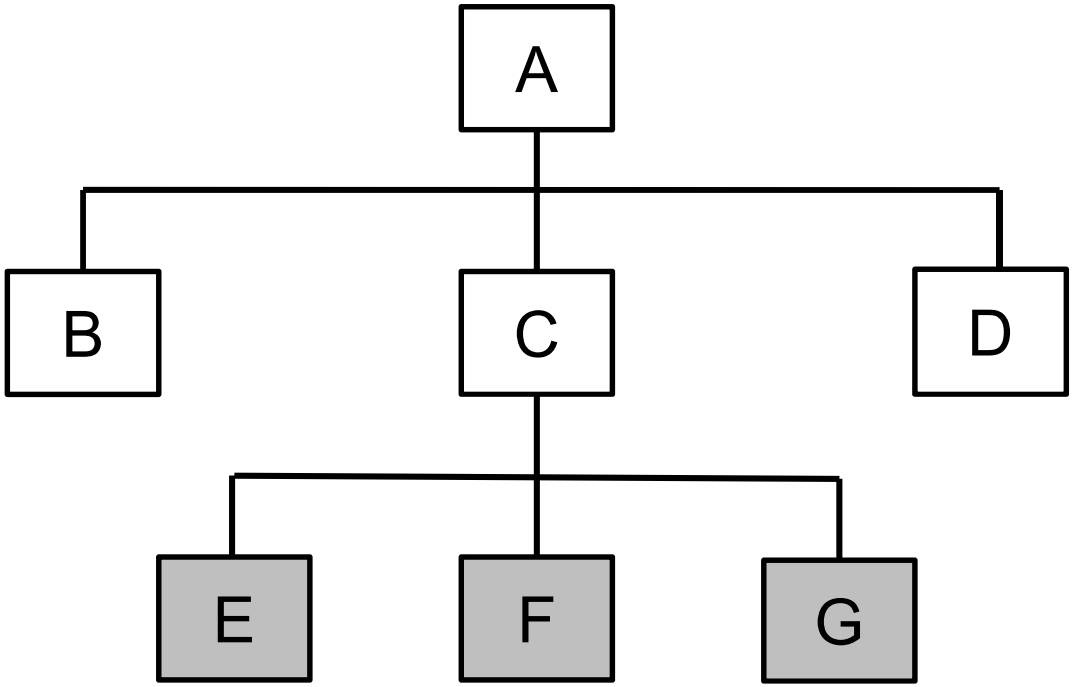


图7.4 模块A、B、D和C自顶向下集成

7.4 系统集成技术-自顶向下

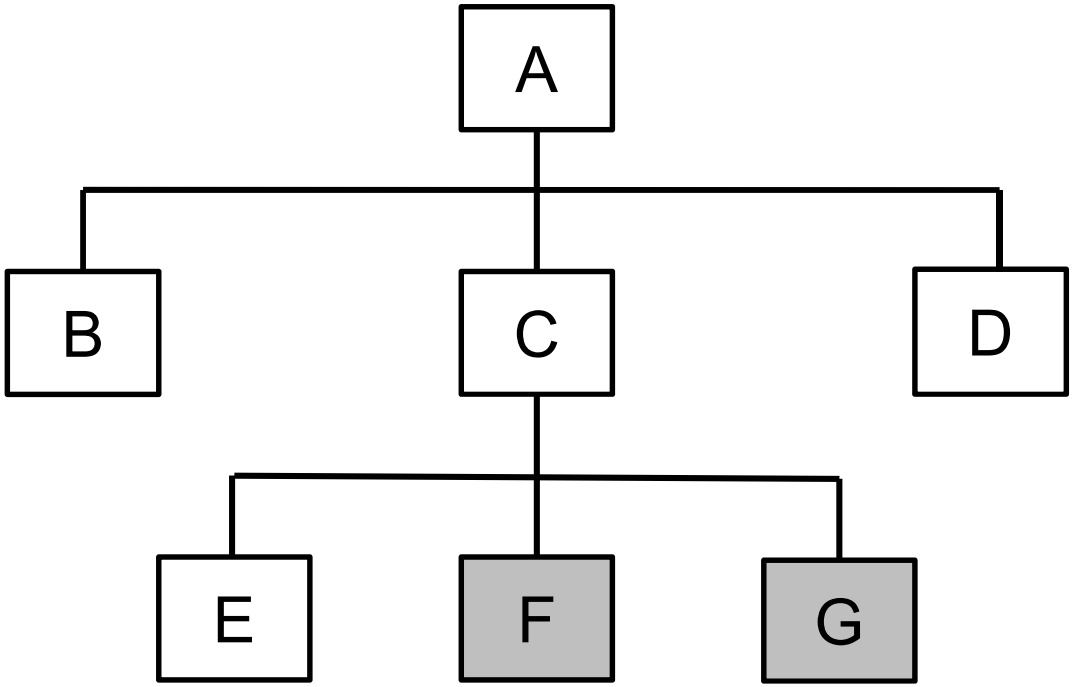


图7.5 模块A、B、C、D和E自顶向下集成

7.4 系统集成技术-自顶向下

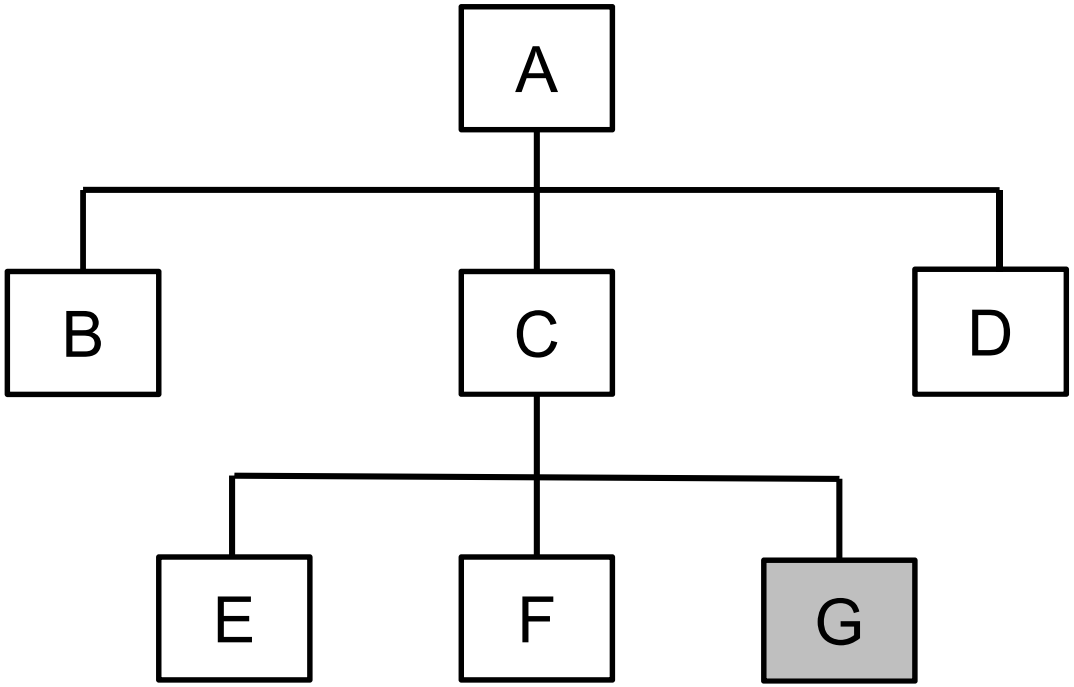


图7.6 模块A、B、C、D、E和F自顶向下集成

7.4 系统集成技术-自顶向下

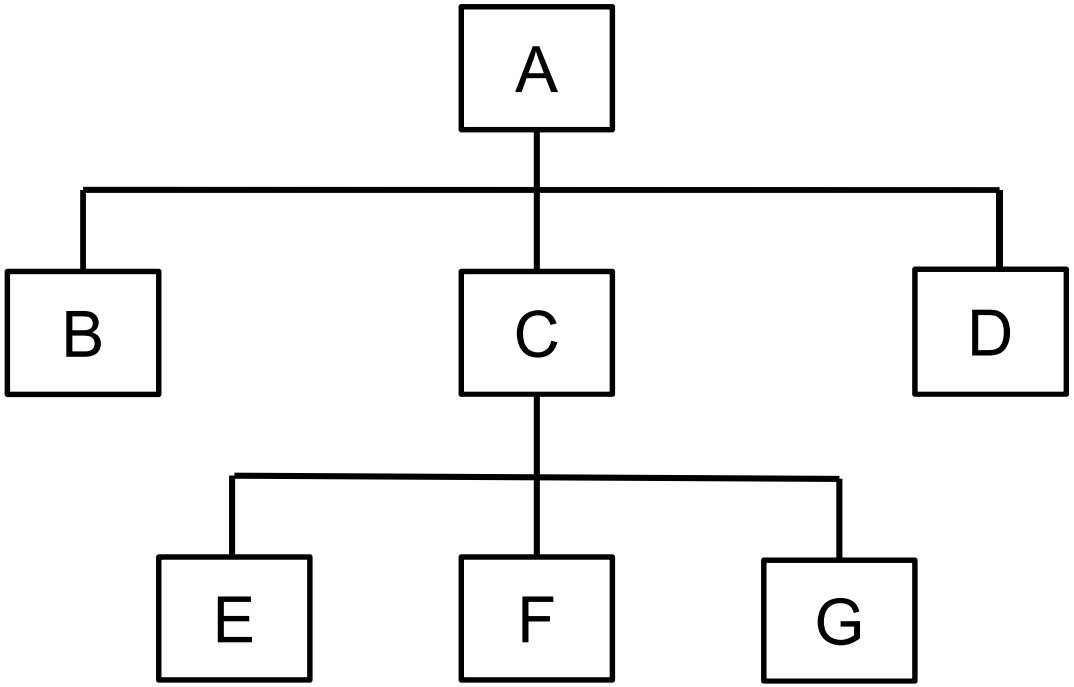


图7.7 模块A、B、C、D、E、F和G自顶向下集成

7.4 系统集成技术-自顶向下

优点

随着集成过程的进行，系统集成测试工程师持续观察到系统级的功能

由于自顶向下集成的渐进性质，接口错误的隔离变得容易

用于测试模块M集成的用例可以在集成其他模块后的回归测试中重用

由于测试输入是应用于顶层模块中，因此那些测试用例是与系统功能相关的

7.4 系统集成技术-自顶向下

局限性

在一定的模块集合的集成完毕前，可能无法观察到一些有意义的系统功能，这是由于一些底层模块的缺失和桩的存在

当桩的层次和顶层模块相差很远时，测试用例的选择和桩的设计会变得越来越困难

7.4 系统集成技术-自底向上

自底向上集成：系统集成开始于对系统最底层模块的整合

需要创建一个测试驱动器，一旦确认所需的一组低层模块集成是符合要求的，就用真实的模块来代替这个驱动器

然后用另一个测试驱动器来集成已整合模块和其他更多的模块。过程持续到所有模块都被集成为止

7.4 系统集成技术-自底向上

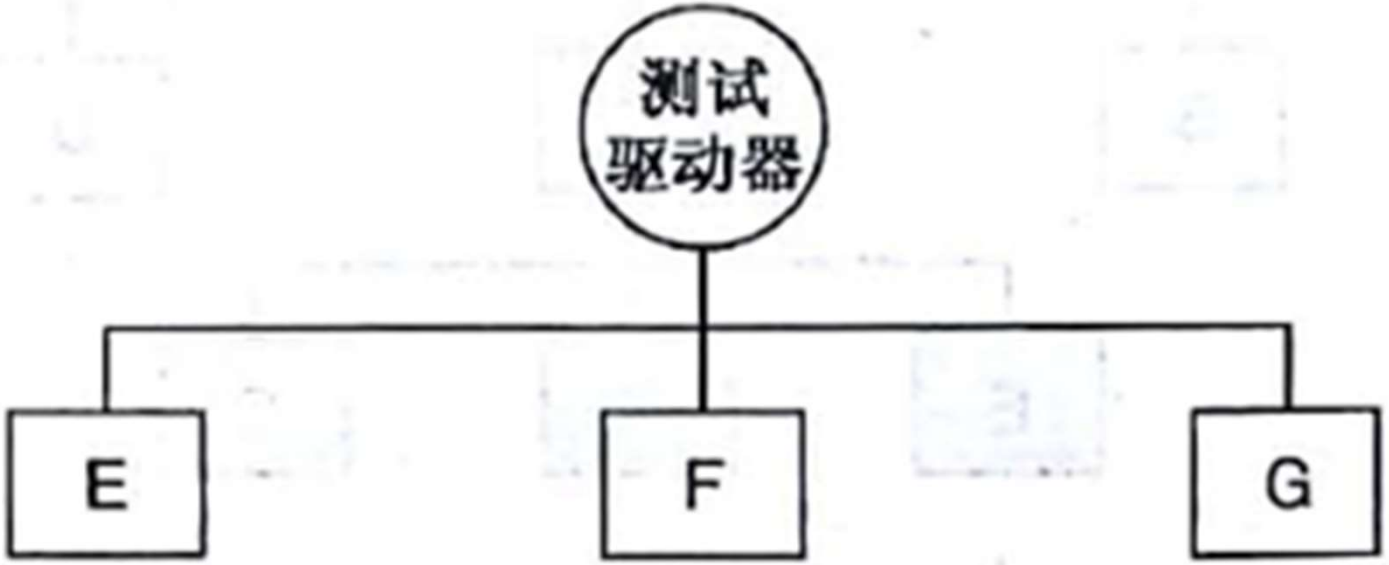


图 7.8　模块 E、F、G 自底向上的集成

7.4 系统集成技术-自底向上

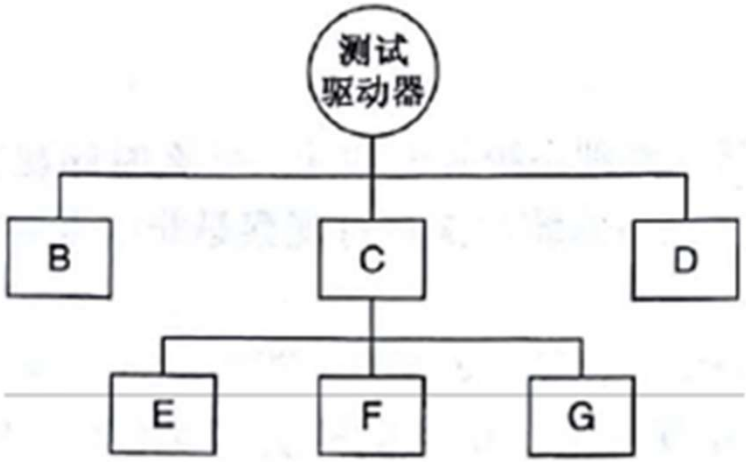


图 7.9　模块 B、C、D 与 E、F、G 自底向上的集成

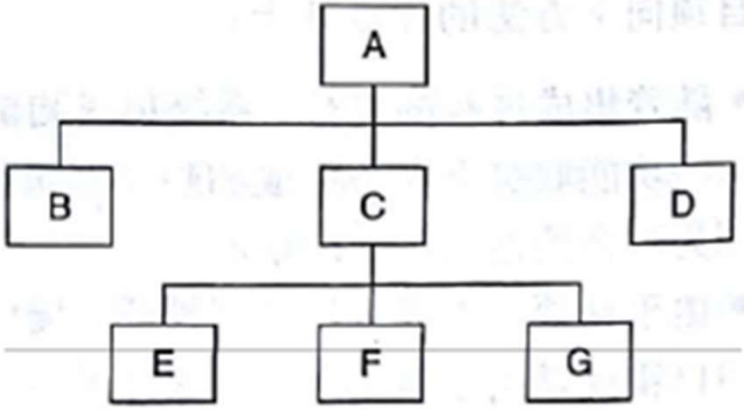


图 7.10　使用所有其他模块自底向上的集成

7.4 系统集成技术-自底向上

优点

低层模块的先测试可以使得高层模块的集成更有效

7.4 系统集成技术-自底向上

缺点

测试工程师从部分集成的系统中无法观察到系统级的功能

直到集成顶层模块，才会发现系统设计中的主要缺陷

7.4 系统集成技术-比较

自顶向下和自底向上的比较

主要设计决策的确认：自顶向下先发现其错误，而自底向上后发现

系统级的功能观察：自顶向下先观察到，而自底向上后观察到

设计测试用例的难度：自顶向下比较难，而自底向上相对容易

测试用例的可复用性：自顶向下策略可复用为新集成模块设计的测试用例，而自底向上不能复用

7.4 系统集成技术

三明治方式： 采用自顶向下和自底向上混合模式

7.4 系统集成技术

大爆炸方式：直接用所有已测试的模块构建整个系统，然后测试（大型系统不宜使用）

7.4 系统集成技术

自顶向下方式和**大爆炸**方式（应用难度和成本高）最可靠

自底向上效率最差

三明治方式拥有中等可靠性

目录

CONTENTS

7.6 集成测试的计划

表7.3 SIT计划框架

1.测试范围
2.集成等级结构
- a.集成测试阶段
- b.每个阶段要集成的模块或子系统
- c.每个阶段中的构件流程和进度表
- d.每个阶段中要建立的环境和需要的资源

3.每个集成测试阶段的标准
- a.进入标准
- b.退出标准
- c.使用的集成技术
- c.设置测试配置

4.每个集成测试阶段的测试规范
- a.测试用例ID编号
- b.输入数据
- c.初始条件
- d.期望结果
- e.测试流程
- 如何执行该测试?
- 如何得到并解释测试结果?

5.每个集成测试阶段的真实测试结果
6.参考文献
7.附录

7.6 集成测试的计划

表7.4 系统集成的进入标准的框架

软件功能和设计规范必须是书面的，并经过评审和批准
代码必须经过评审和审批
每一个模块的单元测试计划必须是书面的，通过评审，并能执行
通过了所有的单元测试
完整的检入请求表必须填写完整，提交并通过
硬件设计规范是书面的，并通过批准
硬件设计验证测试是书面的，通过评审，并能执行
通过所有的设计验证测试
硬件/软件集成测试计划是书面的，通过评审，并能执行
通过所有的硬件/软件集成测试

7.6 集成测试的计划

表7.5 系统集成的退出标准的框架

所有代码已经完成，并被冻结，不会有更多的模块参与集成
通过所有的系统集成测试
没有未解决的重大缺陷
所有在SIT阶段发现的中度缺陷都已修复并通过了重新测试
未解决的轻度缺陷不超过25个
在系统集成测试环境中，系统正常运行两个星期没有任何异常情况
系统集成测试结果记录在案

目录

CONTENTS

- 集成测试的概念
- **接口的不同类型和接口错误**
- 系统集成测试的粒度
- **系统集成技术**
- 系统集成的测试计划

系统集成测试

End