第22章 软件测试策略

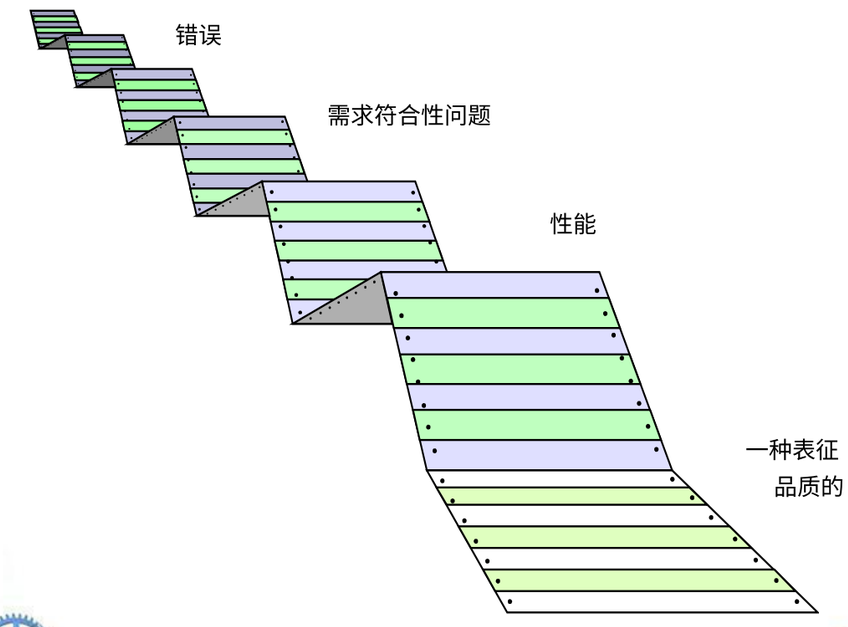
•软件测试

•测试是通过运行程序来

专门发现错误的过程，旨在

交付最终用户前完成。

•测试结果展示



•战略方法

•进行有效测试需执行严谨的技术

评审。通过该流程可在早期消除大量

测试开始。

•测试从组件层面启动，逐步"向外"推进至

整个计算机系统的集成。

•不同的软件工程方法需采用相应测试技术

•测试时机亦影响技术选择

•测试由软件开发人员执行（针对

大型项目）独立测试小组。

•测试与调试是不同活动，但调试必须

纳入任何测试策略中。

•验证与确认

•验证指确保软件正确实现特定功能的任务集合

软件正确实现了特定功能

•确认则指另一组确保

所构建的软件需能追溯到

客户需求。Boehm [Boe81]对此有如下

表述：

•验证："我们是否正确地构建了产品？"

•确认："我们是否构建了正确的产品？"

•谁来测试软件?

开发人员

理解系统

但会"温和"测试

且以"交付"为导向

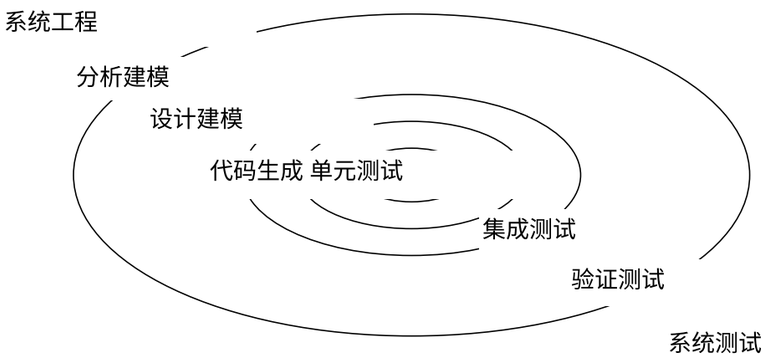
独立测试员

必须学习系统知识

但会试图破坏系统

且以质量为导向

•测试策略



•测试策略

•我们首先进行"小规模测试"

再逐步转向"大规模测试"

•针对传统软件

•模块(组件)是我们的初始重点

•随后进行模块集成

•面向对象软件

•"微观测试"的关注点从

独立模块（传统视角）转向包含

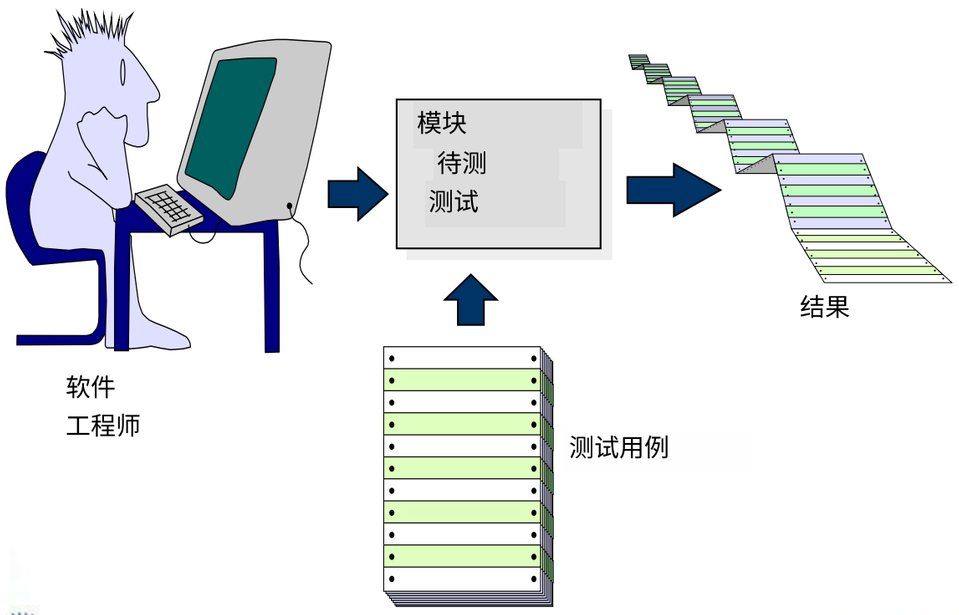
属性与操作的OO类，并涉及

通信与协作

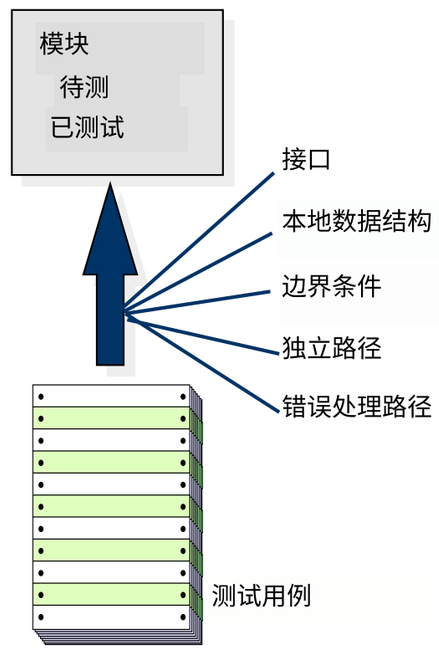
战略问题

- 在测试开始前，以可量化的方式明确产品需求
- 明确表述测试目标
- 理解软件用户并建立用户画像
  为每个用户类别
- 制定强调"快速循环测试"的测试计划
- 构建具备自检能力的"健壮"软件
- 在测试前采用高效技术评审作为过滤手段
- 通过技术评审评估测试策略及
  测试用例本身。
- 制定测试的持续改进方法
  流程。

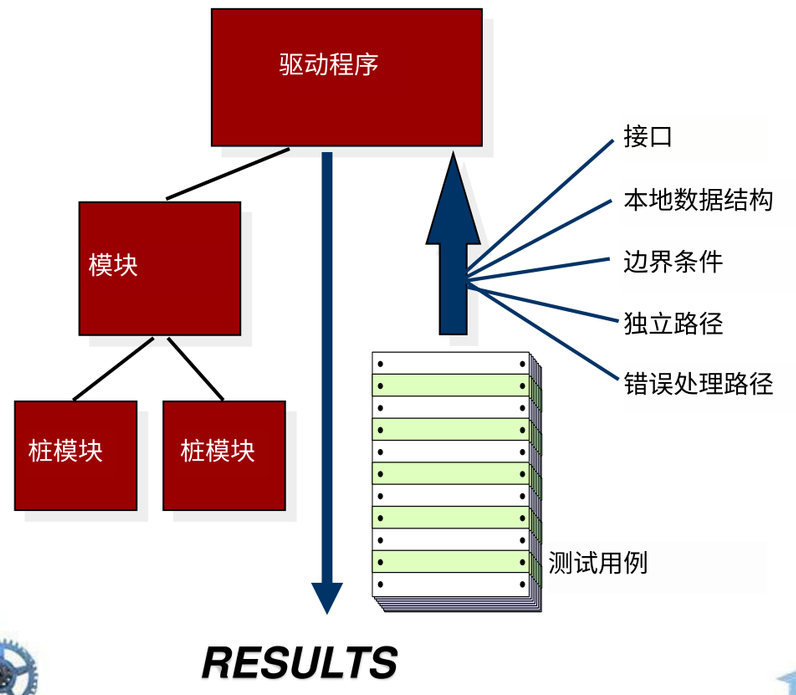
单元测试



单元测试

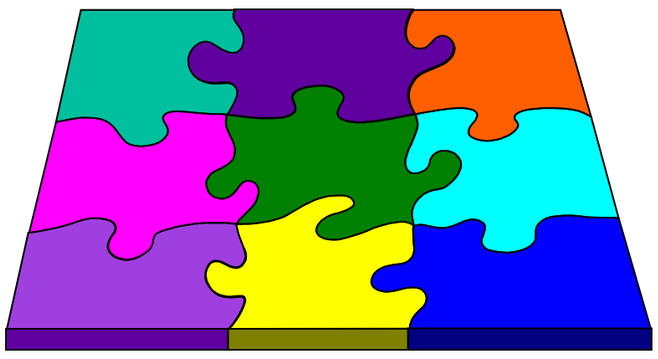


单元测试环境

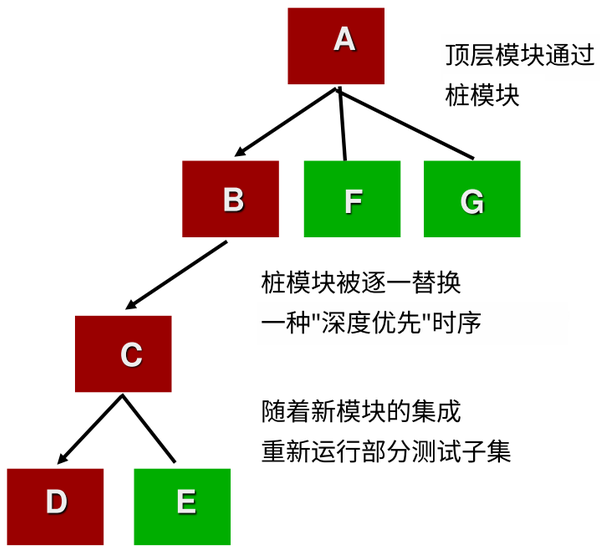


集成测试策略

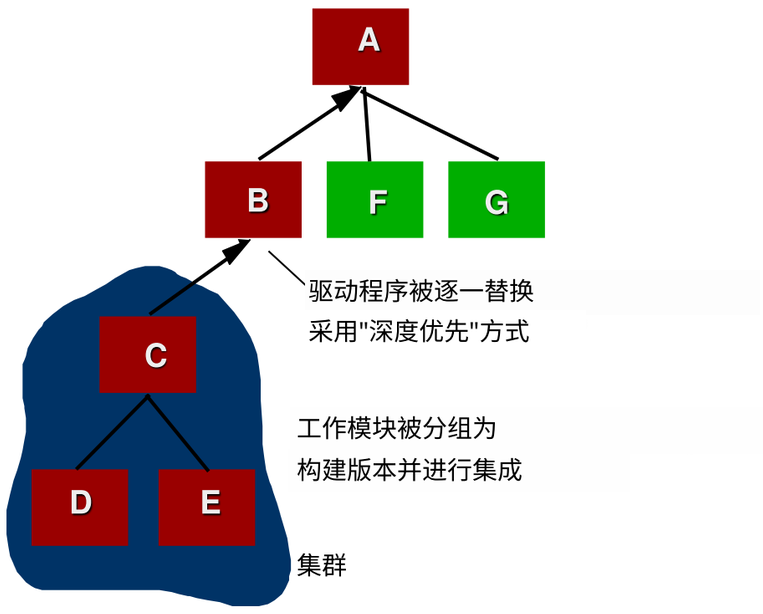
- 选项：
  - "大爆炸"式方法
  - 增量构建策略



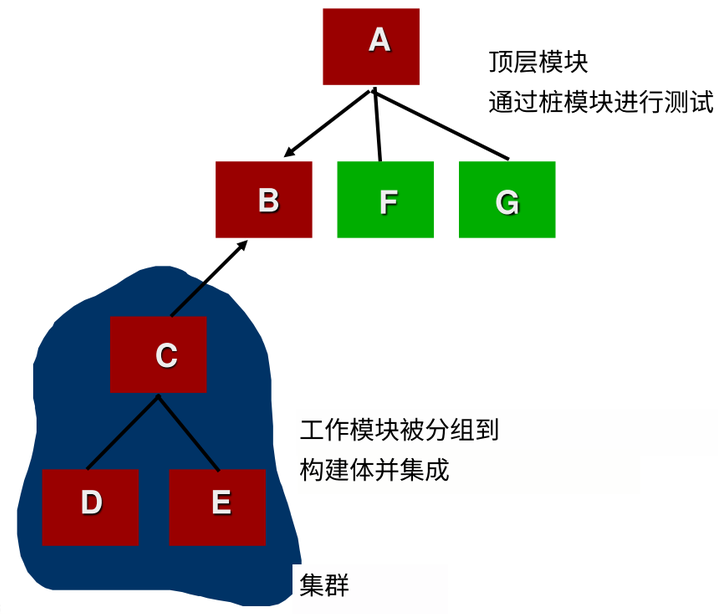
自顶向下集成



自底向上集成



三明治测试法



•回归测试

•回归测试是重新执行部分已完成的测试用例，以确保变更未引发非预期的副作用

•每当软件被修正时，其配置（程序、文档或支持数据）的某些方面都会发生变更

•回归测试有助于确保变更（因测试或其他原因）不会引入非预期行为或额外错误

•回归测试可通过手动重新执行部分测试用例，或使用自动化捕获/回放工具来完成

•冒烟测试

•为产品软件创建"每日构建"的通用方法——冒烟测试步骤：

•已转换为代码的软件组件被集成到"构建"中

•构建包含实现一个或多个产品功能所需的所有数据文件、库、可复用模块和工程组件

•设计一系列测试来暴露可能导致构建无法正常运行的错误——旨在发现最可能延误软件项目进度的"致命性"错误——将构建与其他构建集成，并对整个产品（当前版本）进行每日冒烟测试

•集成方法可采用自顶向下或自底向上

•通用测试标准

•接口完整性 - 每当模块或集群被添加到软件时，都会对内部及外部模块接口进行测试

•功能有效性 - 用于发现软件中功能缺陷的测试

•信息内容——检测本地或全局数据结构中的错误

•性能 - 验证是否测试了规定的性能界限

•面向对象测试

•首先评估分析与设计模型的正确性和一致性- 测试策略发生变化- 由于封装性，"单元"概念被拓宽- 集成测试聚焦于类及其在"线程"或使用场景中的执行- 验证环节采用传统黑盒测试方法- 测试用例设计既沿用常规方法，又涵盖特殊功能特性

•拓展"测试"的视野

•可以论证，面向对象分析与设计模型的评审尤为重要，因为相同的语义结构（如类、属性、操作、消息）会贯穿分析、设计和编码阶段。在分析阶段发现类属性定义问题，能够避免该问题若遗留至设计或编码阶段（甚至下一轮分析迭代）可能引发的连锁反应。

•CRC模型测试

•1. 重新审视CRC模型与对象关系模型
•2. 检查每张CRC索引卡的描述，确认被委托的职责是否属于协作者的定义范畴
•3. 逆向验证连接关系，确保每个被请求服务的协作者都接收到合理来源的请求
•4. 利用步骤3中检查的反向连接，判断是否需要其他类，或职责是否在各类间得到合理分组。

•5. 判断被频繁请求的职责是否可合并为单一职责。

•6. 步骤1至5需迭代应用于每个类，并贯穿分析模型的每次演进。

•面向对象测试策略

•类测试等同于单元测试
  •测试类内部的操作
  •检验该类的状态行为
•集成应用了三种不同策略
  •基于线程的测试——整合响应单一输入或事件所需的类集合
  •基于用例的测试——整合响应单一用例所需的类集合
  •集群测试——整合展示单一协作所需的类集合

•Web应用测试-I

•审查WebApp的内容模型以发现错误。

•审核界面模型以确保其能容纳所有用例。

•检查WebApp的设计模型以发现导航错误。

•测试用户界面以发现呈现和/或导航机制中的错误。

•对每个功能组件进行单元测试。

•Web应用测试 - 第二部分

•测试整个架构的导航功能
•Web应用在多种不同环境配置中部署并测试各配置的兼容性

•执行安全测试以尝试利用
Web应用或其环境中的漏洞
•执行性能测试
•该Web应用通过受控且受监控的环境进行测试
终端用户群体。他们与系统的交互结果会从内容与导航错误、可用性问题、兼容性问题，以及Web应用可靠性与性能等方面进行评估。

•移动应用测试

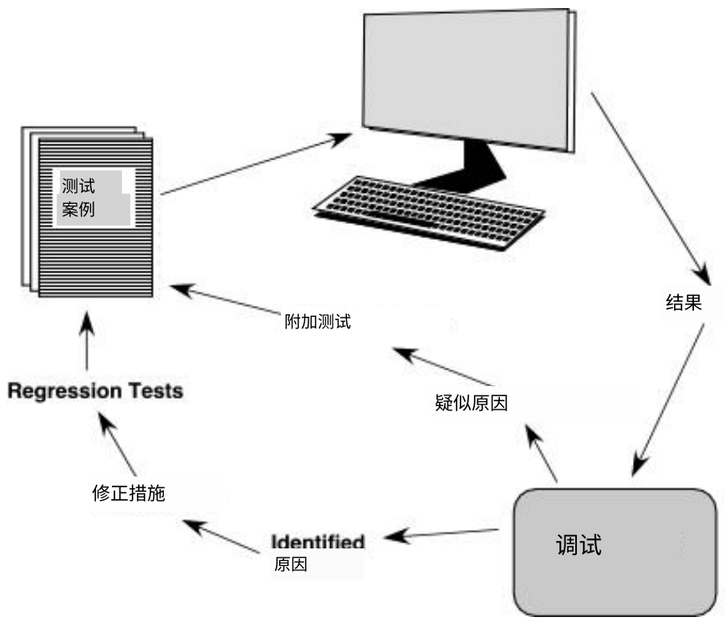
•用户体验测试 - 确保应用满足利益相关方需求
可用性与无障碍性预期
•设备兼容性测试 - 在多台设备上进行测试
•性能测试 - 验证非功能性需求
•连接测试 - 检验应用程序的可靠连接能力
•安全测试 - 确保应用满足利益相关方的安全预期

•真实环境测试 - 在用户设备及实际使用环境中测试应用-认证测试 - 验证应用是否符合分发标准

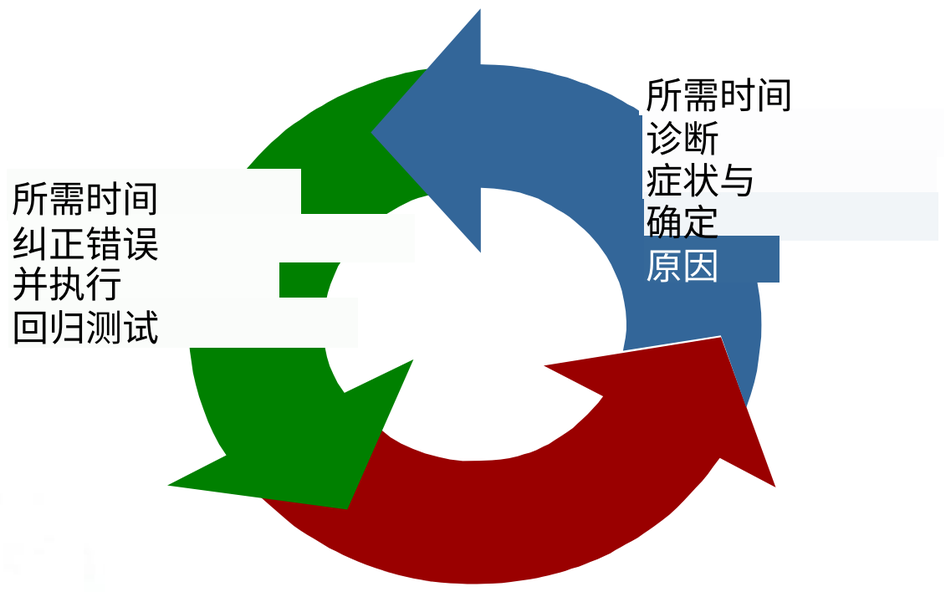
•高阶测试
•验证测试
•聚焦软件需求
•系统测试
•侧重系统集成
•Alpha/Beta测试
•关注用户实际使用
•恢复性测试
•通过多种方式迫使软件失效，并验证其能否正确恢复
performed
•安全性测试
•验证系统内置的保护机制能否有效防御非法渗透
• *Stress testing*
•以异常资源需求量的方式执行系统，频率或音量
• 性能测试
•在集成系统环境中测试软件的运行时性能

•调试：诊断性过程

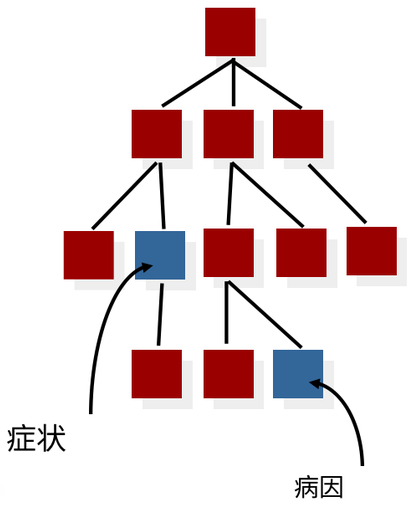
•调试过程



•调试工作

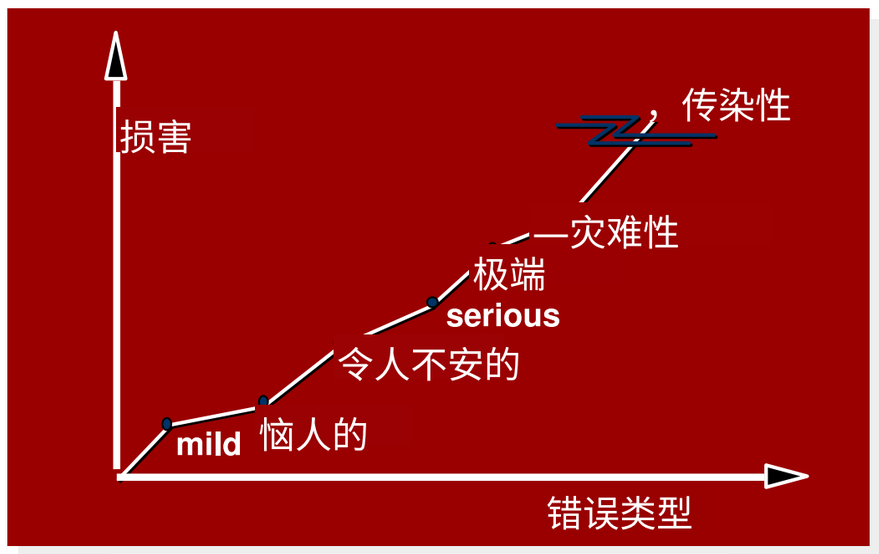


•症状与原因



- ☐ 症状与病因可能存在地理分隔
- ☐ 当满足条件时症状可能消失
另一问题已修复
- ☐ 原因可能源于
非错误组合
- ☐ 原因可能源于系统
或编译器错误
- ☐ 原因可能在于
假设所有人
认为
- ❑ 症状可能间歇性出现

缺陷后果



缺陷分类：功能相关缺陷、
系统相关缺陷、数据缺陷、编码缺陷、
设计缺陷、文档缺陷、标准
违规等

•调试技术

- 暴力测试
- 回溯
- 归纳
- 推理

•**错误修正**

•该bug的成因是否在程序其他部分复现

？多数情况下，程序缺陷源于
可能被复现的错误逻辑模式
别处。

•我即将引入的修复可能会带来什么"下一个错误"

？在实施修正前，应评估源代码（或更理想的是设计）以分析其
耦合性
逻辑与数据结构

•我们本可以采取哪些措施来预防这个缺陷

的发生？这个问题是建立统计软件质量保障体系的第一步。若您能改进流程与产品，该缺陷不仅会从当前程序中移除，更可能从未来所有

程序中彻底消除

。

•最终建议

•三思而后行——修改前请深思

•善用工具获取深层洞察

•若陷入僵局，及时寻求协助——修复缺陷后，务必通过回归测试排查潜在副作用