



SOFTWARE TESTING

苏临之

sulinzhi029@nwu.edu.cn



JUnit

- JUnit-4安装和使用
- 注解
- 断言
- 异常抛出



Typical E-Business Structure

- Web Browser
- User Interface Layer
- Business Logic Layer
- Data Access Layer



Debugging

❖ 调试指的是定位错误和修改错误的技术。某种程度上属于灰盒测试。

❖ Basic Steps

- 第一步：定位错误，确定程序中错误的准确性质和位置，占调试的绝大部分工作量（大概95%）。
- 第二步：修改错误，有了明确的位置和预期判断就可以将错误改正。



Five Debugging Techniques

- Brutal Force Debugging (暴力调试法)
- Inclusive Debugging (归纳调试法)
- Deductive Debugging (演绎调试法)
- Back-Tracking Debugging (回溯调试法)
- Test Debugging (测试调试法)



Five Debugging Techniques

- Brutal Force Debugging (暴力调试法)
- Inclusive Debugging (归纳调试法)
- Deductive Debugging (演绎调试法)
- Back-Tracking Debugging (回溯调试法)
- Test Debugging (测试调试法)



Brutal Force Debugging

- ❖ 暴力调试法指的逐条列举的方法来一点点定位错误位置的方法。
- ❖ 暴力调试法特点
 - 使用比较普遍，不需要过多思考
 - 效率低下，成功率低
- ❖ 何时适宜使用？
 - 其他方法都失败
 - 作为思考过程的补充，但不能替代思考过程



Three Types


❖ 暴力调试法的三种类型

- 利用内存信息输出来调试
- 根据一般的“在程序中插入打印语句”建议来调试
- 使用自动化的调试工具进行调试



Output of the Internal Storage

- ❖ 使用内存信息输出调试能很模糊地了解软件遇到的少量错误，是最缺乏效率的暴力调试法
 - 难以在内存区域与源程序中的变量之间建立对应关系
 - 内存信息输出会产生非常庞大的数据，大多是无关的
 - 内存信息输出产生的是程序的静态快照，而发现错误还需要研究程序的动态状态
 - 通过分析输出的内存信息来发现问题的方法并不太多



Insert "printf" in the Program

- ❖ 可以显示程序的动态状态，让检查的信息可以相对容易地与源程序联系起来。
- ❖ 存在的缺点
 - 主要是碰运气，而不是鼓励我们去思考程序中的问题
 - 所产生的需要分析的数据量非常庞大
 - 要求我们修改源代码，这些修改可能会掩盖症状或引入新的错误
 - 可能对小型程序有效，但如果应用到大型程序，成本就相当高，对某些类型的程序，甚至无法使用这种方法



Automatic Tools

- ❖ 使用编程语言的调试功能，或使用特殊的交互式调试工具来分析程序的动态状态
 - 使用的语言功能：产生可打印的语句执行轨迹的机制，子程序调用和/或对特定变量的修改
 - 调试工具的设置断点功能：程序执行到某条特定语句或改动了某个特定变量的值时暂停执行，然后程序员就可以检查程序的当前状态
- ❖ 存在的缺点
 - 仍然是碰运气为主
 - 有时会生成数量过于庞大的无关数据

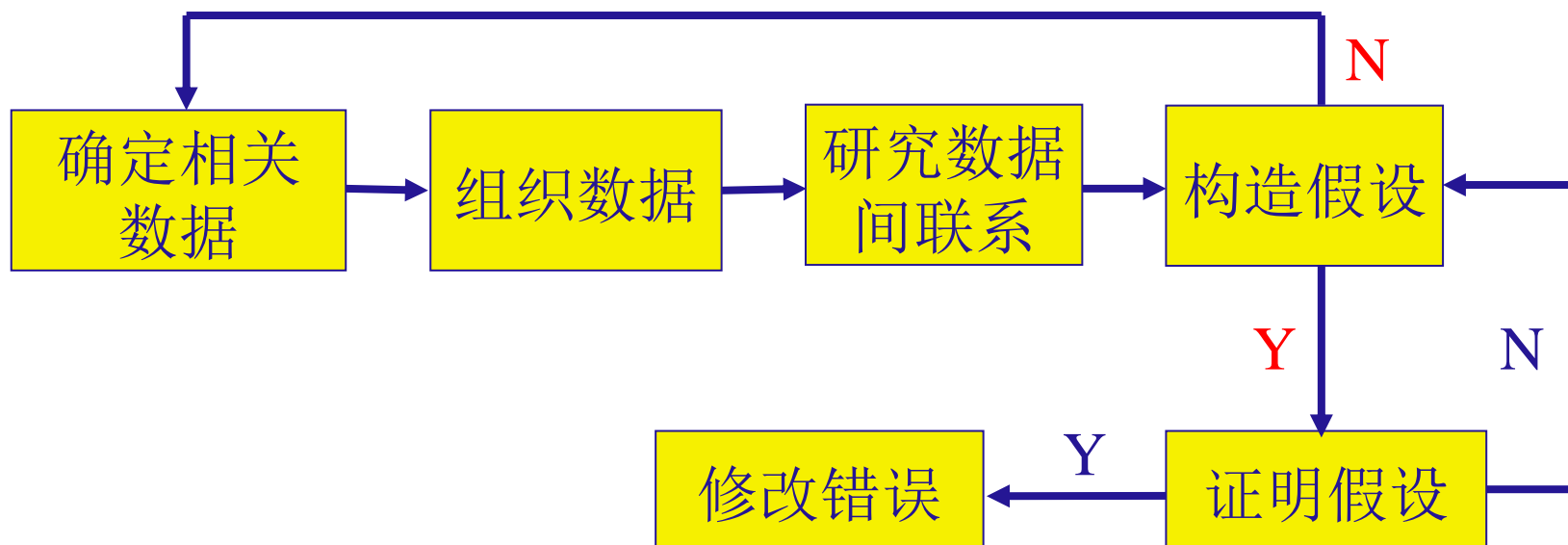


Five Debugging Techniques

- Brutal Force Debugging (暴力调试法)
- Inclusive Debugging (归纳调试法)
- Deductive Debugging (演绎调试法)
- Back-Tracking Debugging (回溯调试法)
- Test Debugging (测试调试法)

Inclusive Debugging

- 归纳调试即以错误的症状为线索，寻找线索之间的联系。





Procedure of Inclusive Debugging

1. 确定相关数据：列举出所有知道的程序执行的正确和不正确之处，这些不正确之处即是症状。
2. 组织数据：组织相关数据，以便观察线索间的模式，找到矛盾、事件。用What、Where、When、How对症状进行描述。
3. 作出假设研究线索之间的联系，利用线索结构里可能的模式作出一个或多个关于错误原因的假设，选择最有可能的假设。
4. 证明假设将假设与其最初的线索或数据相比较，证明假设的合理性，确定这些假设完全可以解释这些线索的存在。

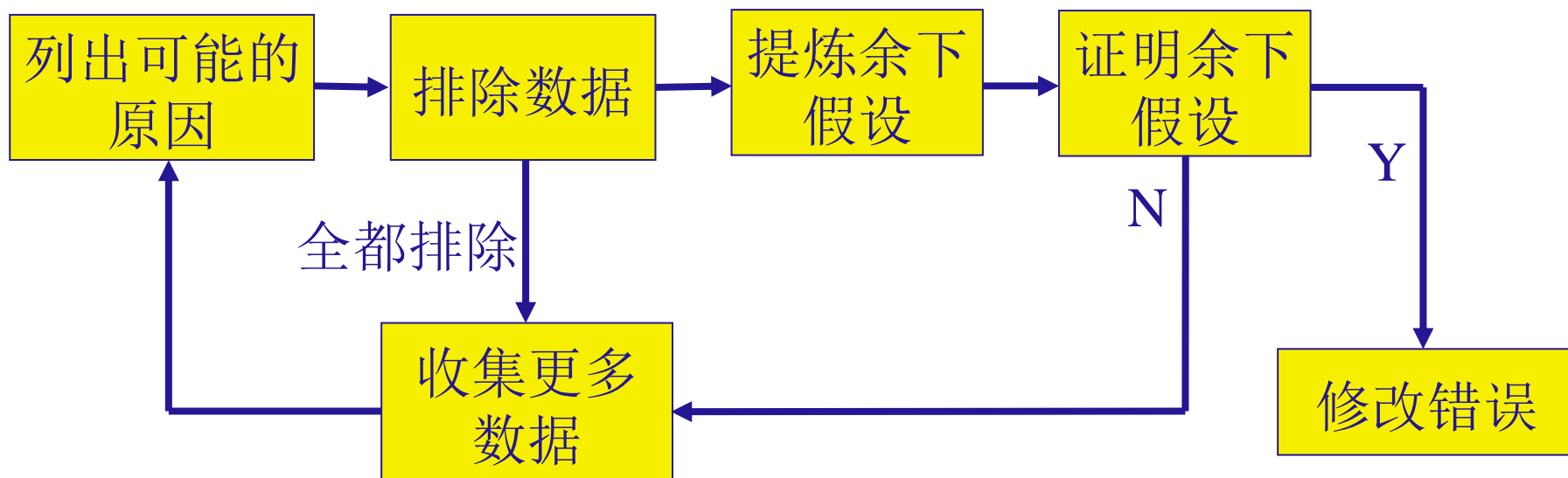


Five Debugging Techniques

- Brutal Force Debugging (暴力调试法)
- Inclusive Debugging (归纳调试法)
- Deductive Debugging (演绎调试法)
- Back-Tracking Debugging (回溯调试法)
- Test Debugging (测试调试法)

Deductive Debugging

- 演绎调试即从普遍的理论和前提出发，使用排除和精炼的过程，达到一个结论。





Procedure of Deductive Debugging

1. 列出所有可能的原因或假设，即建立一份所有想像得到的错误原因的清单。
2. 利用数据排除可能的原因，要求详细检查所有数据，尤其是寻找存在矛盾的地方，然后尽量排除所有可能的原因，仅留一条。
3. 如果所有的原因都被排除，再增加额外的测试用例，得到更多的数据来设计新的推测。
4. 提炼剩下的假设，即用现有的线索来提炼这个推测，以具体到能够指出错误，最后证明剩下的假设。



Five Debugging Techniques

- Brutal Force Debugging (暴力调试法)
- Inclusive Debugging (归纳调试法)
- Deductive Debugging (演绎调试法)
- Back-Tracking Debugging (回溯调试法)
- Test Debugging (测试调试法)



Back-Tracking Debugging

- ❖ 回溯法就是沿着程序的逻辑结构回溯错误的结果，直到找到程序逻辑出错的位置为止。
- ❖ 从程序产生不正确结果的地方开始，从该处观察到的结果推断出程序变量应该是什么值，并从这个位置开始逆向执行程序，重复使用“如果程序在此处的状态是这样的，那么程序在上面位置的状态就必然是那样的”过程，就能很快定位出错误。
- ❖ 回溯法的适用于小型程序



Five Debugging Techniques

- Brutal Force Debugging (暴力调试法)
- Inclusive Debugging (归纳调试法)
- Deductive Debugging (演绎调试法)
- Back-Tracking Debugging (回溯调试法)
- Test Debugging (测试调试法)



Test Debugging

- ❖ 使用测试用例进行调试，当发现了某个被怀疑的错误的症状之后，我们需要编写与原先有所变化的测试用例，尽量确定错误的位置
- ❖ 两种类型的测试用例
 - 供测试的测试用例：目的是暴露出以前尚未发现的错误，每个测试用例尽量涵盖较多的条件。
 - 供调试的测试用例：目的是提供有用的信息，供定位某个被怀疑的错误使用，每个测试用例仅需要覆盖一个或几个条件。
- ❖ 测试调试法不是独立方法，需结合先前所述诸方法。



Principle: To Locate the Bugs

❖ 定位错误的原则

■ 动脑筋

- 对错误症状的有关信息动脑筋进行分析

■ 如果遇到了僵局，就留到稍后解决

- 发挥潜意识的作用

■ 如果遇到了困境，就把问题描述给其他人听

- 描述过程可能会帮助我们发现新的东西

■ 仅将暴力调试作为辅助手段

- 无计划、盲目、成功机会小，而且会将新错误引入程序



Technique: To Fix the Bugs

- ❖ 存在一个缺陷的地方，很有可能还存在其他缺陷
- ❖ 应纠正错误本身，而不仅是其症状
- ❖ 正确纠正错误的可能性并非100%
 - 修改错误的代码本身也可能是错误的，需要更严格测试
- ❖ 正确修改错误的可能性随着程序规模的增加而降低
- ❖ 应意识到改正错误会引入新错误的可能性
 - 不仅对原先的错误情景进行测试，还应该执行回归测试，以判定是否引入了新错误
- ❖ 修改错误的过程有些情况下也是临时回到设计阶段的过程
- ❖ 应修改源代码，而不是目标代码
 - 有可能是在使用试验法调试
 - 目标代码与源代码不同步，重新编译可能错误再次出现



Difficulties Encountered

❖ 调试的障碍

- 个人自尊心：调试说明程序员会在设计或编码时犯错误
- 热情耗尽：调试耗费脑力，承受较大压力
- 可能会迷失方向：错误实际上可能出现在程序的任何部位
- 必须自力更生：关于调试过程的研究、资料和正式的指南都比较少
-



THANK YOU!