

软件静态测试

知识回顾

- 缺陷预防——通过错误阻断或错误源的消除防止缺陷的引入
- 缺陷减少——通过故障检测和排除减少缺陷
- 缺陷遏制——即进行失效预防和遏制

软件测试课程安排

- 软件测试(上): 静态测试
- 软件测试(下): 动态测试

目标

- 了解软件测试的基本概念及分类
- 掌握人工审查的过程
- 了解文档审查、代码审查的内容
- 掌握静态结构分析的主要内容

内容

- 软件测试概述
- (人工)审查概述
- 文档审查
- 代码审查
- 静态分析
- 代码审查课堂实践

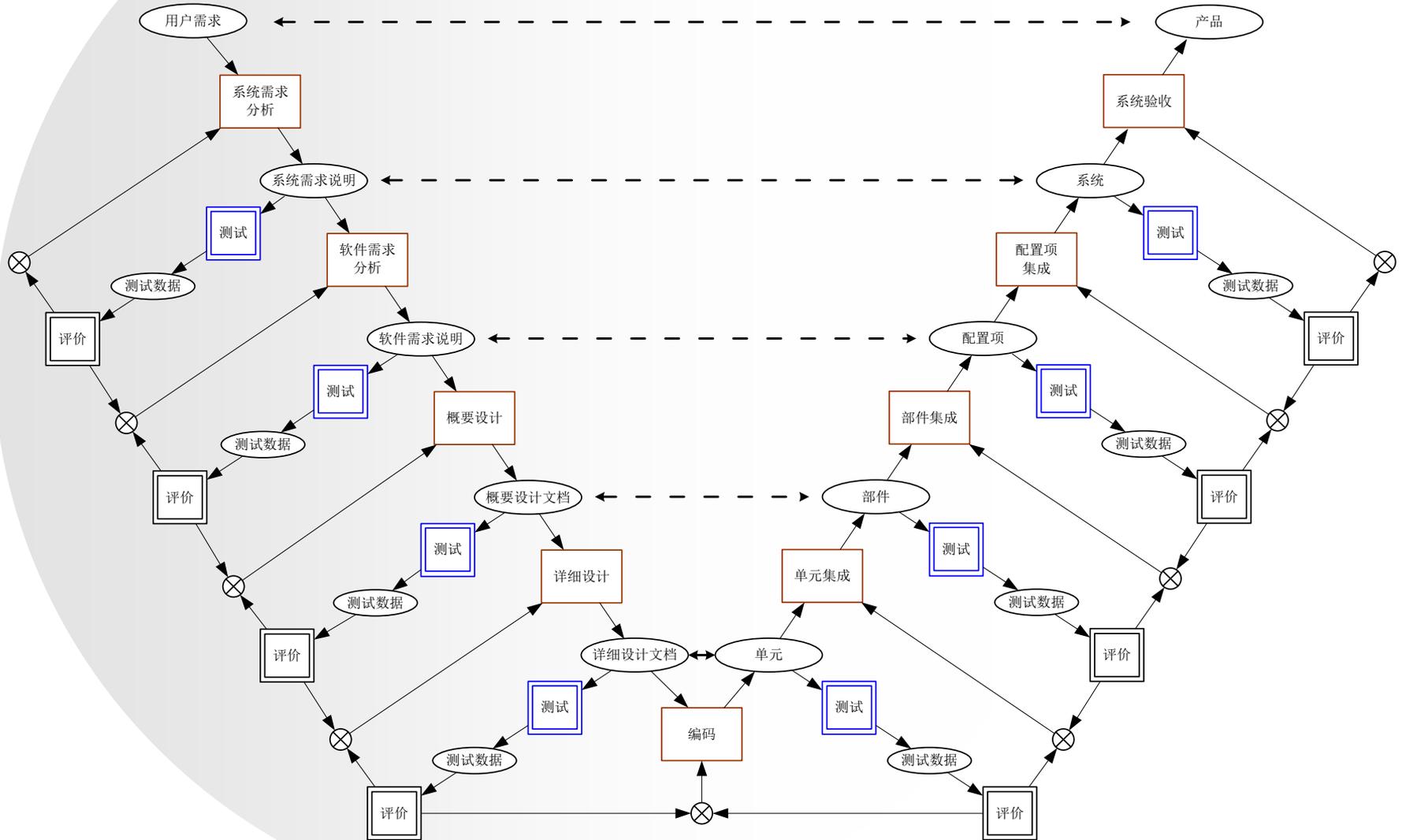
1 软件测试概述

软件测试的意义

- 近三十年的实践表明，软件工程的确是摆脱软件危机，确保软件质量的有效途径。IBM公司在总结其成功开发航天飞机飞行软件的经验时说明，其主要经验有两条，一是认真实施软件工程，二是特别加强测试。
- 在国内测试也带来巨大收益：
 - 某型号飞机倾斜角大于10度撞山时不能自动拉起。
 - 某型号飞机敌我斜距小于50米时，座舱多功能显示器激烈闪屏。
 -

软件测试的作用：发现错误

软件开发与测试的关系(2/2)



软件测试技术的分类

根据是否需要 内部信息

- 白盒测试
- 黑盒测试

根据是否运行 软件

- 静态测试
- 动态测试

根据开发的 阶段

- 需求/设计评审
- 单元测试
- 部件测试
- 配置项测试
- 系统测试

根据软件质量 特性

- 功能性测试
- 效率测试
- 可靠性测试
- 易用性测试
- 可维护性测试
- 可移植性测试

常见软件测试

	开发阶段	质量特性	动态/静态	白盒/黑盒
<u>文档审查</u>	<u>各阶段均可</u>	<u>均可涉及</u>	<u>静态</u>	<u>白盒</u>
单元测试	编码之后	均可涉及	动态/静态	白盒/黑盒
静态分析	编码之后	均可涉及	静态	白盒
<u>代码审查</u>	<u>编码之后</u>	<u>均可涉及</u>	<u>静态</u>	<u>白盒</u>
系统测试	配置项集成之后	均可涉及	动态	白盒/黑盒

2 审查概述

审查的基本概念

- 审查(Inspection)是同行评审(Peer Review)的一种技术。
- 同行评审(Peer Review)是一种通过作者的同行来确认缺陷的方法，除包括审查外，还包括走读(Walkthrough)和技术评审(Technical Review)。

审查的过程(1/3)

- 简单地说，审查的过程包括两部分：会前事先阅读材料、会中集中讨论。更详细地说，审查的过程包括审查计划、启动会议、审查准备、会议审查、修改缺陷、回归审查。

审查的过程(2/3)

- 审查计划

由主持人制定审查的计划。

- 启动会议

产品制作者介绍产品的背景情况；这里的产品可以是软件需求规格说明书、软件设计文档、软件代码等。

- 审查准备

每个审查人员要预先对被审查的产品进行个人的检查，标识出可能的缺陷。审查人员也可组成小组进行交流、讨论。

审查的过程(3/3)

● 会议审查

以会议的形式组织审查，在会议上产品制作者逐一介绍产品内容。在介绍过程中，审查小组的其他成员进行提问，判断是否存在错误。

● 修改缺陷

产品制作者根据会议审查确定的错误及修改计划对产品进行修改。

● 回归审查

对产品的修改部分进行重新审查，以确保对错误的修改是正确的，并且没有因为修改而引入其他的错误。

无论是文档审查，还是代码审查，都遵循以上过程。

审查工作的一个重要特点

- 审查过程对照检查表进行，检查表是过去经验的总结，是审查活动的核心技术。

人工审查的类别

- 文档审查

- 对软件文档的一种测试方法。通过对软件过程中文档的完整性、准确性、一致性等多方面进行检查，发现其中的各类软件文档问题。

- 代码审查

- 主要检查代码和设计的一致性，代码对标准的遵循、可读性，代码的逻辑表达的正确性，代码结构的合理性等方面。

3 文档审查

文档审查的内容(1/3)

- 文档的完整性审查：

验证所提交的软件文档是否完整，以及文档标识和签署是否完整性。

- 审查文档种类是否符合依据文件要求的种类。依据文件可以是软件开发计划或软件质量保证计划。若无规定，可参考国军标确定审查文档的种类。附件1是某单位依据国军标，结合自身情况，给出的不同级别软件需要提供的文档名称。
- 封面要包括文档名称、版本、密级、编号、编写人、单位、编写时间。
- 会签要完整，即包括校对、标准化、审核、审批等签署。

文档审查的内容(2/3)

● 文档的一致性审查

- 审查文档内容和术语的含义前后是否一致，有没有自相矛盾的地方
- 检查文档与程序的一致性
- 检查书面文档与联机帮助文档的一致性

文档审查的内容(3/3)

- 文档的准确性审查
 - 审查文档内容是否正确和准确
 - 是否有错别字
 - 是否有二义性的定义、术语或内容

文档审查的检查表

- 检查表是文档审查中一个非常重要的环节，在正式的文档审查之前就应该根据文档类型和审查的经验，将一次审查的主要检查点罗列在检查表中，以此作为文档审查的依据。P107-108分别给出了软件文档共性问题检查表和软件配置管理计划问题检查表。

需求避错准则可用于需求文档审查

● 需求避错准则

- 应采用统一的需求建模方法
- 要列出输入在范围之外的处理流程
- 对可能的异常情况，考虑相应的保护措施
- 定义软件功能所涉及的各种数据
- 须全面分析软件有哪些非功能需求
- 定量要求应合理、可达且可验证
- 对于需要处理避免敏感数据的软件，必须提出保密性要求
- ...

设计避错准则可用于设计文档审查

● 设计避错准则

- 模块化设计：高内聚、低耦合
- 模块单入口和单出口设计
- 模块功能应该可以预测
- 模块规模应该适中
- 深度、宽度、扇出和扇入都应适当
- 降低模块接口的复杂程度
- 限制程序圈复杂度
- 在数据模块接口数据时，先假定其为错误数据，并建立检测判据检测它
- 须要进行异常情况设计
- 确定实现软件容错的范围及容错的方式
- 安全关键信息不会因一位获两位差错而引起系统故障

4代码审查

代码审查

- 主要检查代码和设计的一致性，代码对标准的遵循、可读性，代码的逻辑表达的正确性，代码结构的合理性等方面。

代码审查的检查表内容(1/7)

● 数据声明错误

- 是否所有变量都已声明?
- 默认的属性是否被正确理解?
- 数组和字符串的初始化是否正确?
- 变量是否赋予了正确的长度、类型和存储类?
- 初始化是否与存储类相一致?
- 是否有相似的变量名?

代码审查的检查表内容(2/7)

● 比较错误

- 是否存在不同类型变量间的比较?
- 是否存在混合模式的比较运算?
- 比较运算符是否正确?
- 布尔表达式是否正确?
- 比较运算是否与布尔表达式相混合?
- 是否存在二进制小数的比较?
- 操作符的优先顺序是否被正确理解?
- 编译器对布尔表达式的计算方式是否被正确理解?

代码审查的检查表内容(3/7)

● 控制流错误

- 是否超出了多条分支路径?
- 是否每个循环都终止了?
- 是否每个程序都终止了?
- 是否存在由于入口条件不满足而跳过循环体?
- 可能的循环越界是否正确?
- 是否存在“仅差一个”的迭代错误?
- DO/END语句是否匹配?
- 是否存在不能穷尽的判断?
- 输出信息是否有文字或语法错误?

代码审查的检查表内容(4/7)

● 输入/输出错误

- 文件属性是否正确?
- OPEN语句是否正确?
- I/O语句是否符合格式规范?
- 缓冲大小与记录大小是否匹配?
- 文件在使用前是否打开?
- 文件在使用后是否关闭?
- 文件结束条件是否被正确处理?
- 是否处理了I/O错误?

代码审查的检查表内容(5/7)

● 接口错误

- 形参的数量是否等于实参的数量?
- 形参的属性与实参的属性相匹配?
- 形参的量纲是否与形参的量纲相匹配?
- 传递给被调用模块的实参个数是否等于其形参个数?

代码审查的检查表内容(6/7)

- 传递给被调用模块的实参量纲是否与其形参量纲相匹配?
- 调用内部函数的实参的数量、属性、顺序是否正确?
- 是否引用了与当前入口点无关的形参?
- 是否改变了某个原本仅为输入值的形参?
- 全局变量的定义在模块间是否一致?
- 常数是否以实参形式传递过?

代码审查的检查表内容(7/7)

● 其他情况

- 在交叉引用列表中是否存在未引用过的变量?
- 属性列表是否与预期结果的相一致?
- 是否存在警告或提示信息?
- 是否对输入的合法性进行了检查?
- 是否遗漏了某个功能?

编码避错准则可用于代码审查

讲义P84-96:

- if、for、do、while、case、switch、default等语句自占一行，且if、for、do、while等语句的执行语句部分无论多少都要加括号{ }
- 程序内部必须有正确的文档
- 数据说明应便于查阅易于理解
- 语句应该尽量简单清晰
- 头文件结构编码避错准则
- 顺序结构...
- 内存...
- IF语句...
- SWITCH语句...
- 循环语句...
- 函数...

5 静态分析

静态分析包括的内容(1/4)

- 静态分析(Static Analysis)一般包括：
 - 1.1 静态结构分析
 - 1.2 质量度量
 - 1.3 代码规则检查

静态分析包括的内容(2/4)

● 静态 结构 分析

- 包括 控制流分析、数据流分析、接口分析、表达式分析 等。
- 一般以 图形 的方式表示程序的内部结构。例如，控制流图显示一段程序的逻辑结构，它由许多节点组成，一个 节点 代表一条语句或数条语句，连接结点的叫边，边 表示节点间的 控制流向。

静态分析包括的内容(3/4)

● 质量度量

- 一般基于软件的质量模型进行，通常是通过对代码的圈复杂度、扇入/扇出、注释度量等进行统计，或者通过Halstead科学度量法等对程序的质量给出评价。

静态分析包括的内容(4/4)

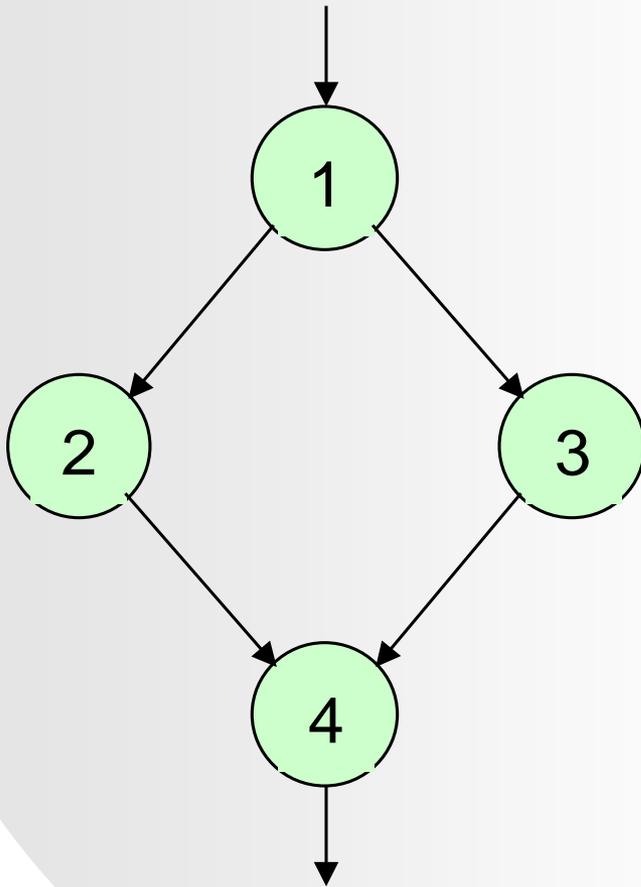
● 代码规则检查

- 可以发现违背程序编写标准的问题，程序中不安全、不明确和模糊的部分，找出程序中不可移植部分、违背程序编程风格的问题，包括变量检查、命名和类型审查、程序逻辑审查、程序语法检查和程序结构检查等内容。

1.1.1 静态结构分析-控制流分析

- 控制流图是对程序控制结构的一种有向图表示，用来表示程序的逻辑路径，从流程图转化而来，只保留了流程图中节点和判断，忽略了程序过程块。

控制流图示例



if then else

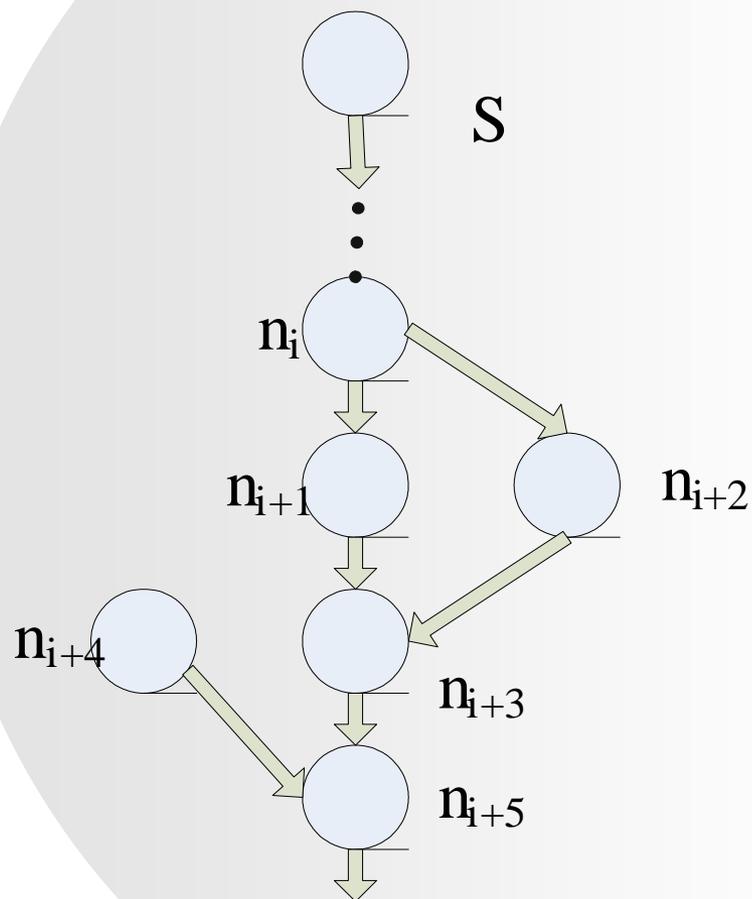
```
if (state_1.eof==0 &&
    status.eof==1)
{
    status.eof=0;
    k=1;
}
else
{
    status.eof=1;
    k=0;
}
a=4;
b=5;
```

控制流分析的内容

- 转向并不存在的语句标号。
- 没有使用的语句标号。
- 没有使用的子程序定义。
- 调用并不存在的子程序。
- 从程序入口进入后无法达到的语句。
- 死循环语句。
- 结构化编程验证。
- 不可达代码检查。

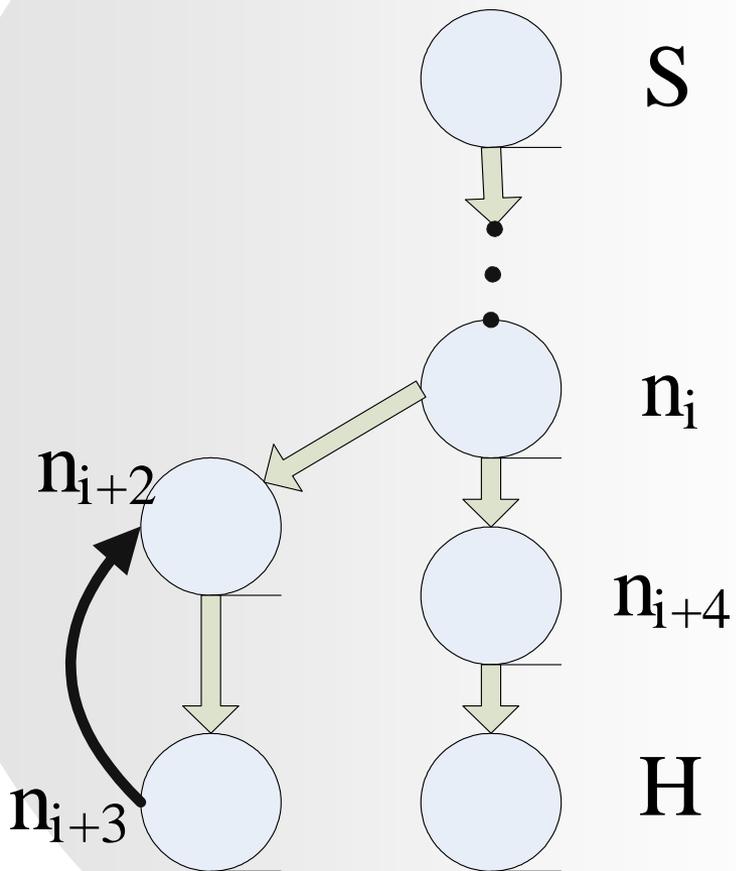
供静态分析的代码，不要求已经通过编译。

不可达代码检查



节点 n_i+4 属于不可达节点。这可能是一个未被调用的函数。

死循环代码检查

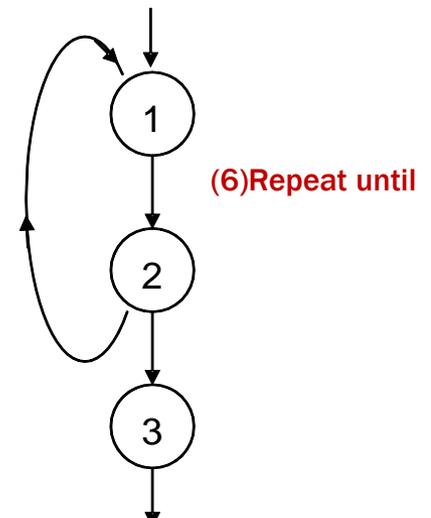
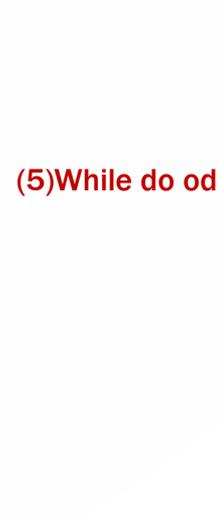
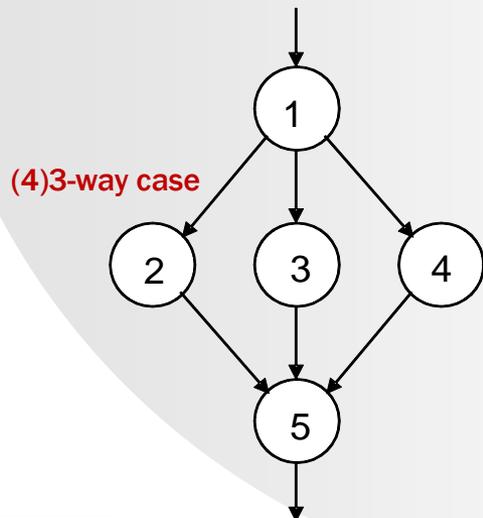
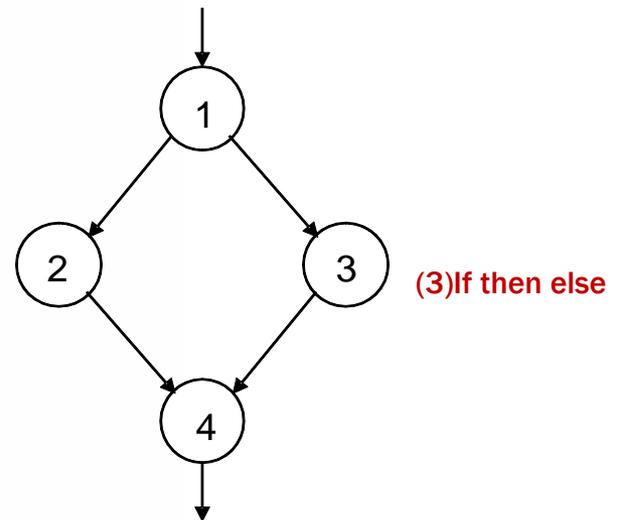
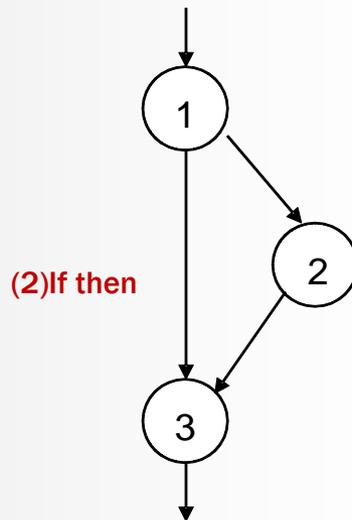
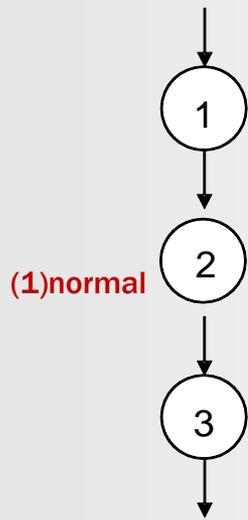


n_i+2 、 n_i+3 之间的执行就是死循环语句。

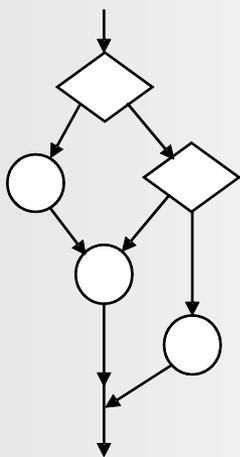
结构化编程验证

- 验证的思想：把典型的结构列出来，看看程序的编程是不是符合典型的结构。如果是，就是结构化编程。
- 通过对程序结构化编程验证可有效地排除一些程序错误，提高软件可靠性及可维护性。下图分别画出了典型的结构化编程结构和典型的非结构化编程结构。

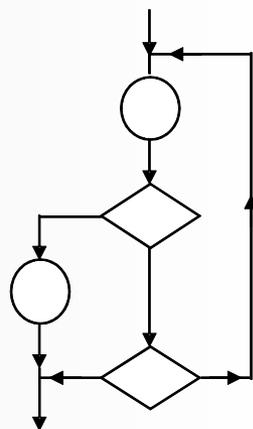
典型的结构化编程结构



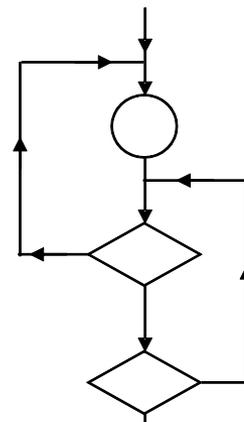
典型的非结构化编程结构



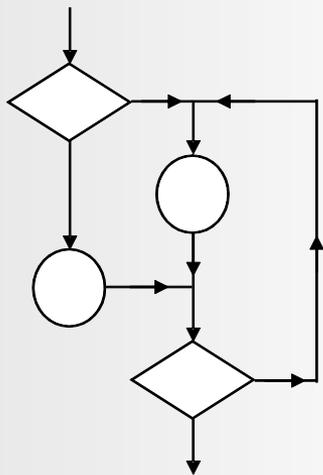
1) Abnormal selection



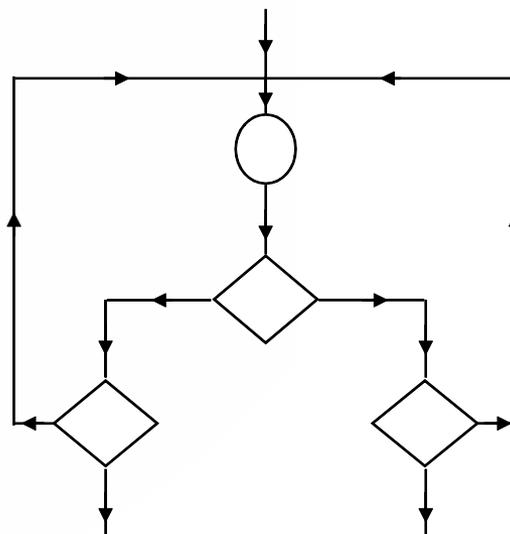
2) Loop with multiple exit points



3) Overlapping loops



4) Loop with multiple entry points

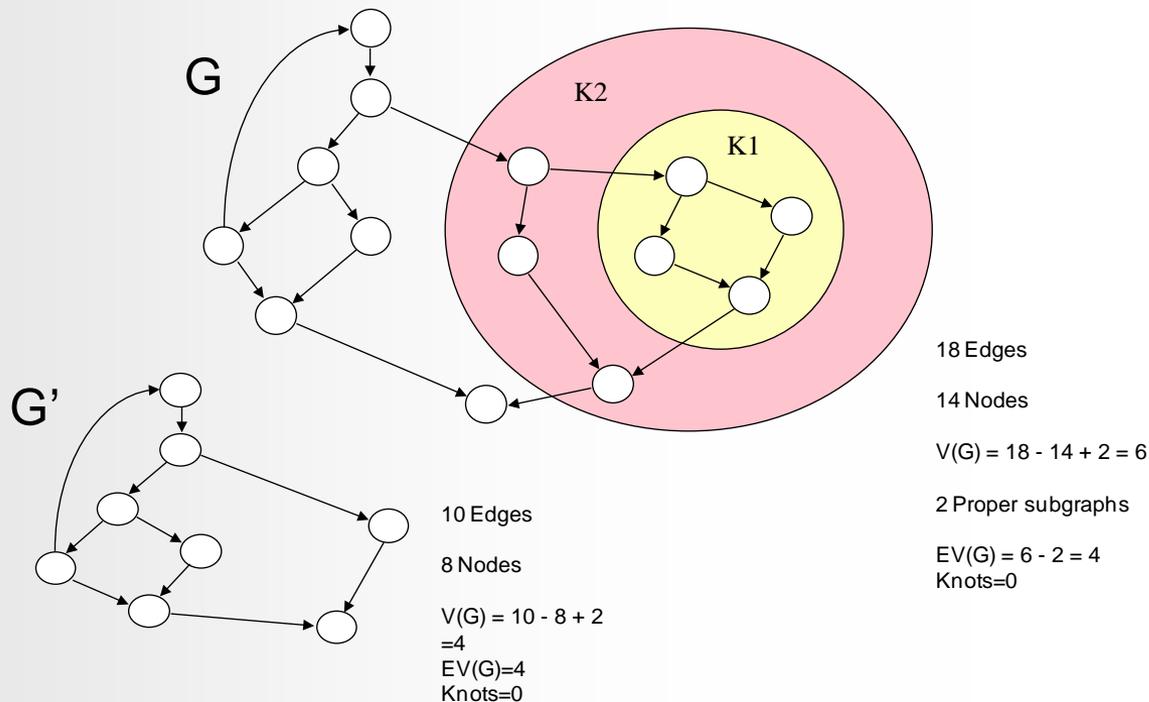


5) Parallel loops

结构化编程验证

- 如果一个程序是完全结构化的，则其控制流图可简化为最后一个节点。通过结构化流程简化后，程序的圈复杂度叫做基本圈复杂度(Essential Cyclomatic Complexity)。
- 完全结构化的程序，其基本圈复杂度为1。

结构化编程验证



- K1圈是一个典型的“if then else”结构化编程结构。因此可以简化成一个节点。
- 简化之后，K2圈中的结构也属于典型的结构化编程结构，因此也可以简化成一个节点。
- 经过简化后，控制流G编程控制流图G'。G'中没有典型的结构化编程结构了，因此能再简化。
- G'的圈复杂度是4，基本圈复杂度也是4。
- 简化前G的圈复杂度是6，经过简化后变为4，这也就是G的基本圈复杂度。

1.1.2 静态结构分析-数据流分析

- 通过检查变量的定义(Definition)和引用(Reference)关系来发现程序中的错误。
- 如果程序中某一语句的执行能改变程序变量X的值，则称X是被该语句定义的。如赋值运算 $x = y + 10$ 。
- 如果某一语句的执行引用了内存中变量X的值，则说该语句引用变量X。如 $s = 2 * (t + 46)$ 、 $if (x > 100)$ 、array[i]。
- 常见的数据流异常有：未定义就引用(UR)、定义后未引用(DU)、定义后再定义(DD)。

数据流分析

```
1 void proc ()
2 {
3   int x,y,z,t;
4   x = 1;
5   x = 3;
6   if (y > 0)
7     x = 2;
8   /* end if */
9   z = x + 1;
10 }
```

- y在第6行进行了引用，但是之前未被定义，属于UR问题。
- x在第4行与第5行均进行了定义，属于DD问题。
- z在第9行进行了定义，但是未被引用，因此属于DU问题。

1.1.3 静态结构分析-接口分析

程序的接口分析涉及子程序以及函数之间的接口一致性：

- 包括检查形参与实参类型、个数、维数、顺序的一致性
- 当子程序之间的数据或控制传递使用公共变量块或全局变量时，也应检查它们的一致性。

1.1.4 静态结构分析-表达式分析

表达式分析主要用于发现以下几种错误:

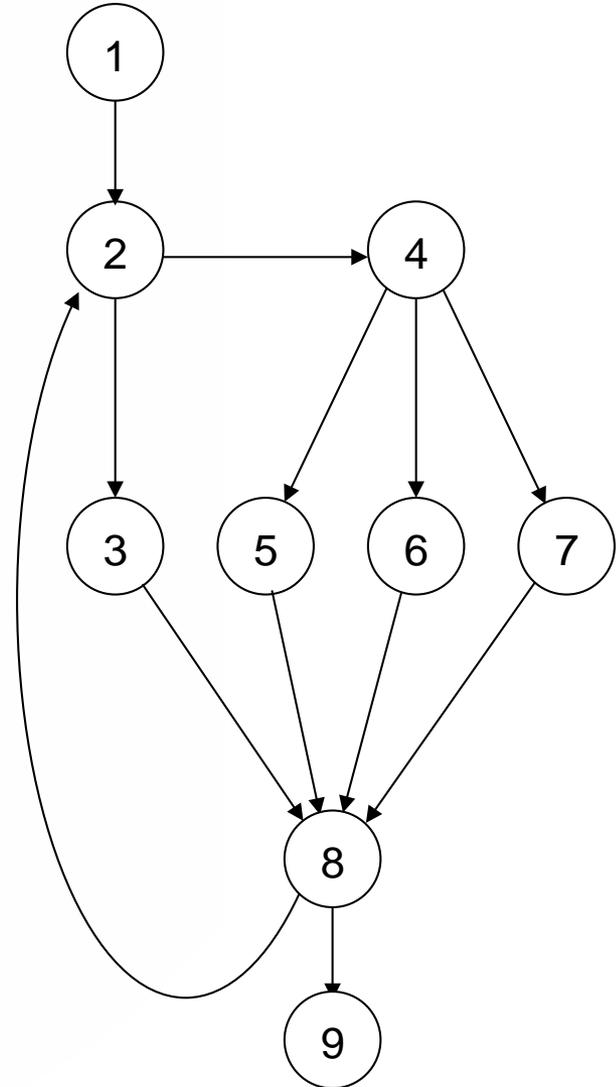
- 程序中括号的使用不正确
- 数组引用错误
- 作为除数的变量可能为零
- 作为开平方的变量可能为负
- 作为正切值的变量可能为 $\pi/2$
- 浮点数变量比较时产生的错误

1.2 静态分析-代码的质量度量

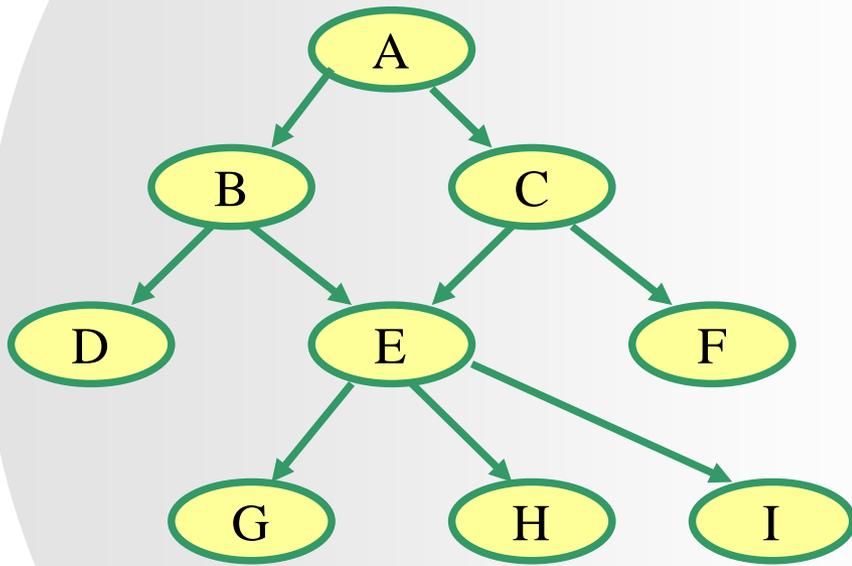
- 常见的质量度量有：
 - 1.2.1 圈复杂度
 - 1.2.2 扇入/扇出
 - 1.2.3 注释行

1.2.1 圈复杂度

- 从有向图中计算McCabe圈复杂度一般有三种方法：
1) $V(G)$ = 线性独立路径数；
2) $V(G)$ = Edge数 - 节点数 + 2；
3) $V(G)$ = 区域数。
圈复杂度越大，程序越复杂，可靠性越差。
- 在图中，有12条边，9个节点，因此 $V(G) = 12 - 9 + 2 = 5$ 。



1.2.2 扇入/扇出



- 扇入/扇出主要用于分析函数/过程间的调用层次。
 - **扇入数**指某一过程/函数被多少过程/函数调用。
 - **扇出数**指某一过程/函数调用多少过程/函数。
- 在图中，E函数扇入数为2，扇出数为3。

扇入/扇出分析

- 在扇入/扇出分析中，主要参考以下几点：
 - 扇出与程序结构宽度有关：
 - 扇出过大会增加程序结构宽度。
 - 扇出过小会增加程序结构深度。
 - 扇出过大增加过程或函数复杂性(需控制的模块过多)。
 - 扇出数最好控制在3-4，最高不超过6-7。
 - 扇入越大表明(该)模块通用性好，但过大程序聚合性差。
 - 底层过程或函数扇出数尽量小，扇入数尽量大。
 - 上层过程或函数扇出数尽量大，扇入数尽量小。

1.2.3 注释行

- 程序中注释行对提高代码维护性非常有帮助，同时也是度量程序代码可读性与可维护性的一个重要属性。根据注释在程序中出现的位置分为：
 - 头注释，是指过程或子程序开始前所有注释。
 - 执行行注释，是指出现在程序中可执行代码间注释。
 - 声明注释，是指出现在程序中变量或函数声明区域的注释。

6代码审查实践

课堂实践要求

- 对给定的一段代码进行代码审查
- 审查依据、参考：
 - 个人编程经验
 - 代码审查表(讲义P109~112)
 - 3.4节“软件编码阶段的避错编码准则”(讲义P84-96)。
- 个人审查 10分钟
- 小组讨论 10分钟
- 会议审查 15分钟
- 总结

要点

- 根据是否需要内部信息，软件测试可分为白盒测试和黑盒测试。根据是否允许软件，可分为动态测试和静态测试。根据开发阶段不同，可分为需求/设计评审、单元测试、部件测试、配置项测试、系统测试。根据质量特性不同，可分为功能测试、效率测试、可靠性测试、易用性测试、可维护性测试、可移植性测试。
- 简单地说，审查的过程包括两部分：会前事先阅读材料、会中集中讨论。更详细地说，审查的过程包括计划、启动会议、审查准备、会议审查、回归。
- 审查过程对照检查表进行，检查表是过去经验的总结，是审查活动的核心技术。
- 文档审查主要对文档的完整性、准确性、一致性等方面进行审查。代码审查主要检查代码与设计或与需求是否一致。

要点

- 静态结构分析一般包括控制流分析、数据流分析、接口分析、表达式分析。六种典型的结构化编程结构是：“normal”、“if then”、“if then else”、“3-way case”、“while do od”、“repeat until”。完全结构化的程序，可以简化成圈复杂度为1的控制流图，即基本圈复杂度为1。数据流分析常发现的问题有未定义引用、定义未引用以及定义后再定义。
- 常见的代码质量度量有圈复杂度、扇入/扇出、注释行。圈复杂度的计算方法有：1) $V(G)=$ 线性独立路径数；2) $V(G)=$ Edge数-节点数+2；3) $V(G)=$ 区域数。扇入数指某一过程/函数被多少过程/函数调用。扇出数指某一过程/函数调用多少过程/函数。