SOFTWARE TESTING

Software Testing

Complexity Analysis

School of Computer Science & Engineering
Sun Yat-sen University

Instructor: Guoyang Cai
email: isscgy@mail.sysu.edu.cn

Approaches & Technologies

中山大學
SUN YAT-SEN UNIVERSITY

OUTLINE

- 3.1 软件静态测试概述
- 3.2 软件代码检查
- 3.3 软件复杂性分析
  - 软件复杂性概述
  - 软件的结构复杂性
  - 软件复杂性控制
  - 软件复杂性度量
  - 面向对象软件的复杂性度量
- 3.4 软件质量度量
- 3.5 软件静态分析工具

软件复杂性概述

- 软件复杂性反映为分析、设计、实现、测试、维护和修改软件的困难程度或复杂程度。
- 软件复杂性是软件危机产生的最直接原因。
  - 研究表明，作为软件显著特点的软件复杂性是导致软件错误的主要原因，软件可靠性问题的本质就是软件复杂性问题。
  - 软件复杂性已经远远超出人们对复杂性的控制能力；软件的可维护性等质量特性也与之有极大关系。
    - 软件复杂性越高，软件隐含错误的概率越大，软件的可靠性和可维护性越差。
    - 当软件复杂性超过一定限度时，软件缺陷急剧上升，甚至引发软件开发项目的失败。
    - 软件的维护开销除与维护人员的素质等因素相关外，维护工作量是软件复杂性的一个指数函数。

软件复杂性概述

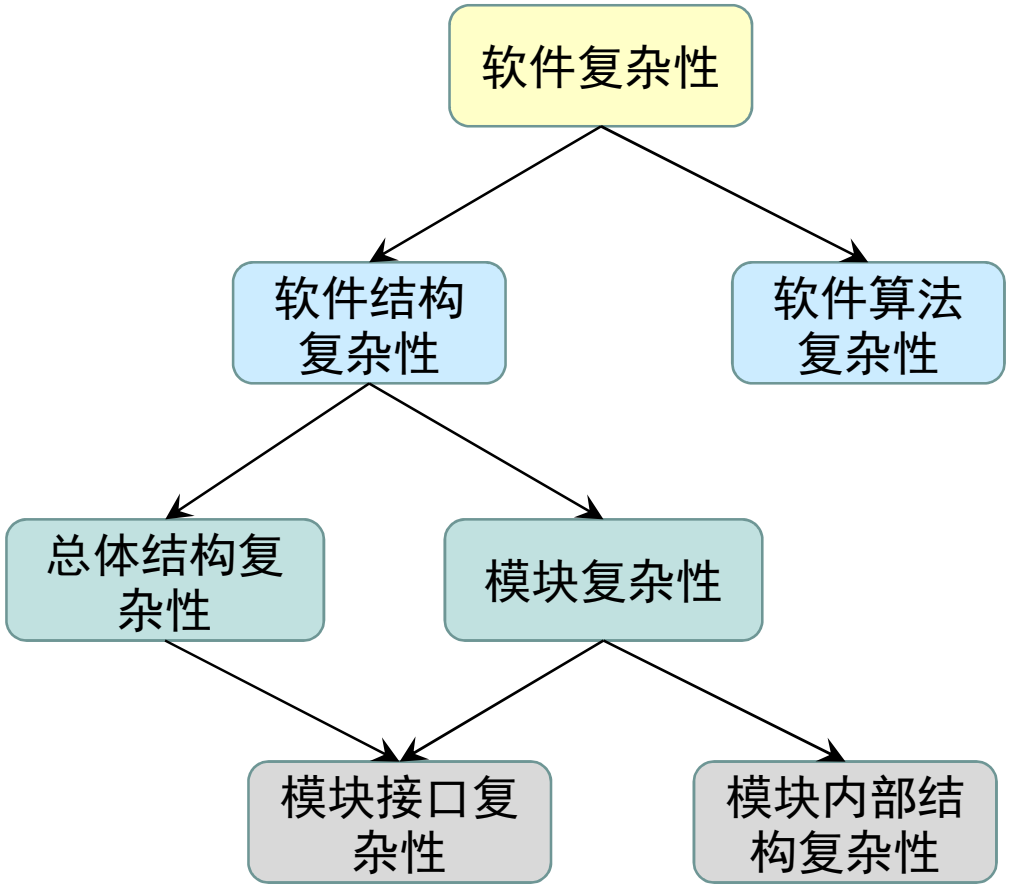
- 对软件复杂性进行分析、度量和控制，是软件可靠性工程亟待解决的重要问题，其目的是：
  - 降低由软件设计方法和技巧使用不当而带来的复杂性，更好地对软件开发过程进行控制；
  - 降低由复杂性引发软件错误的可能性，提高软件的可靠性和可维护性；
  - 最终确保软件产品的质量。
- 软件复杂性直接关系到对软件开发费用、开发周期和软件内部隐藏错误的评估，同时是软件可理解性的另一种度量。

软件复杂性概述

- 软件复杂性产生的主要原因：
  - 软件应用需求 (功能与效能) 的复杂性；
  - 软件应用框架、结构及模型的复杂性；
  - 软件开发环境 (包括编程、调试、测试、应用仿真等) 及应用环境的复杂性；
  - 软件开发过程和开发模型的复杂性；
  - 软件项目涉及的人的智力劳动管理的复杂性；
  - 软件设计与验证的复杂性 (尤其对于嵌入式应用软件)。
- 软件复杂性的主要体现
  - 软件复杂性最终体现在软件结构复杂性和算法复杂性等方面，目前主要根据软件结构复杂性来实现软件复杂性度量。
    - 软件结构复杂性包括总体结构复杂性和模块结构复杂性；模块结构复杂性又包括模块内部结构复杂性和模块接口复杂性。

软件复杂性概述

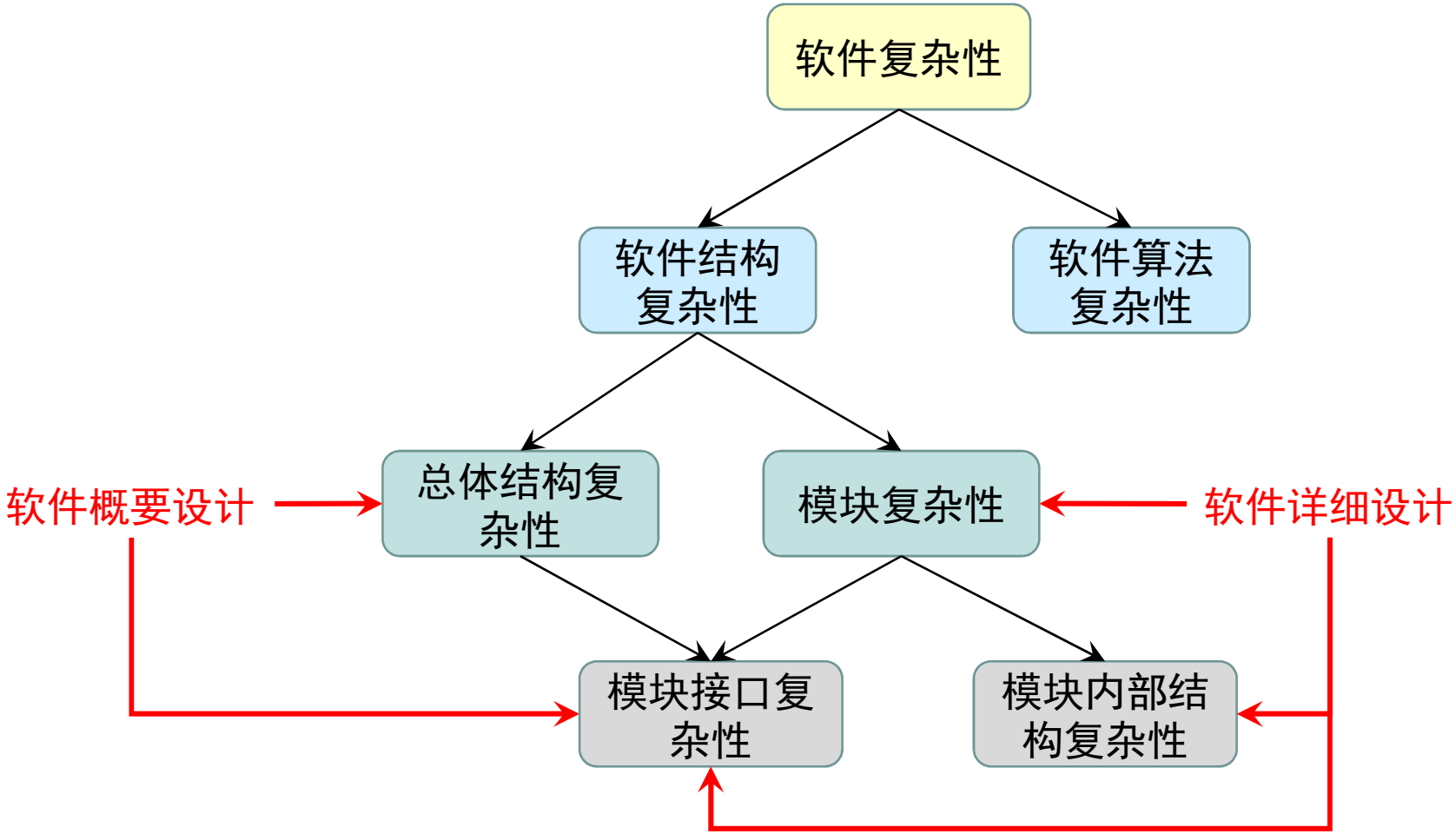
软件复杂性构成



- 软件复杂性概述
  - 软件设计过程是软件结构复杂性形成的根源。
    - 软件概要设计
      - 需要对软件总体结构复杂性进行有效控制，使之保持在一个合理的范围内；
    - 软件详细设计
      - 需要对模块内部结构复杂性、模块接口复杂性进行有效控制，使之保持在一个合理的范围内。

软件复杂性概述

软件复杂性构成



软件的结构复杂性

- 软件结构复杂性包含总体结构复杂性 (也称总体模块结构复杂性) 和模块复杂性两个方面。

(1) 模块复杂性

- 模块复杂性包含了模块内部结构复杂性和模块接口复杂性两部分内容。模块复杂性度量主要用来对模块中的数据流和控制流结构 (或模块信息流结构) 和模块之间互连的复杂程度等进行度量和评价。

(A) 模块内部结构复杂性度量

- 模块内部结构复杂性度量是软件结构复杂性度量的基础。
- 模块内部结构复杂性的度量方法主要有 *Halstead* 度量和 *McCabe* 度量等方法。

软件的结构复杂性

(1) 模块复杂性 (续)

(B) 模块接口复杂性度量

(*a*) 以模块调用关系图中所包含的有向路径数目来度量。

- 将一个模块对应一个结点，结点之间的连接关系就是模块之间的调用关系；所有调用关系的总和构成模块调用关系图。
- 模块接口复杂性定义为模块调用关系图中以起始模块为起点的图的所有有向路径的数目加1。

(*b*) 以模块或信息的扇入/扇出数量来度量。

- 模块的扇入数等于直接调用该模块的上级模块的个数，表达该模块的复用程度。
- 模块的扇出数等于该模块直接调用的下级模块的个数，表达该模块的复杂程度。

- 模块接口复杂性度量的另一方面是试图反映包括所有模块接口关系在内的整个软件的结构复杂性。

软件的结构复杂性

(2) 总体结构复杂性

- 总体结构复杂性也称总体模块结构复杂性。总体结构复杂性与模块复杂性之间往往相互矛盾。
  - 单个模块的划分越小，模块功能越简单，虽然可以降低模块内部结构复杂性，但模块之间的联系就越多，接口就越复杂，由此可能导致总体结构复杂性增加。
  - 反之，增加模块内部结构复杂性有可能降低总体结构复杂性。
- 总体结构复杂性度量
  - 在软件设计、尤其是软件总体设计中，通过对总体结构复杂性的度量来综合反映软件的功能要求及软件中的模块划分情况等，力求在模块复杂性与总体结构复杂性之间取得平衡。

- 软件的结构复杂性
  - 软件结构复杂性度量
    - 软件结构复杂性度量的研究需要寻找一个合适的复杂性指标，并使之最小化，从而达到改善软件结构复杂性的目的。
      - 软件结构复杂性是模块复杂性、总体模块结构复杂性，以及软件的重要程度、调用频率等的综合。
      - 目前尚没有一种权威的软件结构复杂性度量方法。

软件复杂性控制

- 软件复杂性控制是一个系统工程。
  - 软件复杂性控制是在对软件的各种复杂性度量的基础上，综合考虑软件开发成本、可靠性要求等一系列因素之后，对软件复杂性的一种平衡。
- 软件复杂性控制反映的是软件设计人员的综合能力。
  - 软件复杂性控制要求在软件设计中，通过对所有影响软件复杂性、进而影响软件可靠性的因素进行控制，将它们限制在最小范围内，以期改善软件可靠性。
  - 软件复杂性控制需要的不仅是技术、方法和工具，它也是一种艺术，反映了软件设计人员的综合能力。

- 软件复杂性控制
  - 模块内部结构复杂性控制
    - 模块化是结构化程序设计的基础，模块复杂性控制是软件复杂性控制的基础，而其中模块内部结构复杂性控制尤为重要。
    - 模块隔离可以有效防止错误蔓延，从而降低软件复杂性。
      - 单个模块进行单独的编制、调试、测试和维护。
    - 模块内部结构复杂性控制主要包括模块的划分策略和语言控制结构的合理使用。
      - 模块的划分由其功能和性能决定，需要避免模块化程度不够或过分模块化。
      - 模块划分带来的结构复杂性和模块之间接口的复杂性是相互矛盾的。适当地确定模块的大小和接口，使之保持适度的复杂性，是模块内部结构复杂性控制的基本原则。
      - 在三种典型的语言控制结构中，循环结构的复杂性最高。
      - 谨慎使用递归结构。

- 软件复杂性控制
  - 总体结构复杂性控制
    - 提高模块独立性
      - 独立的模块容易测试和维护。
      - 模块之间的耦合性描述了模块之间的相对独立性，是影响软件复杂性的一个重要因素。
      - 设计软件时，遵循尽量使用数据耦合、少用控制耦合、限制公共耦合范围、完全不用内容耦合这4个原则，就可能在很大程度上减少模块之间的耦合性，降低模块的复杂性。
      - 在软件设计中，尽量使所设计的模块具有高内聚性，并能识别出低内聚性。
        - 内聚是指模块功能的相对强度，用于衡量一个模块内部各个元素彼此结合的紧密程度，是信息隐蔽和局部化的自然延伸。

- 软件复杂性控制
  - 总体结构复杂性控制 (续)
    - 提高模块独立性
      - 数据耦合：模块之间通过参数传递数据，将某些模块的输出数据作为另一些模块的输入数据。数据耦合是耦合程度最低的耦合形式。
      - 控制耦合：一个模块通过接口向另一个模块传递一个控制信号，接受信号的模块根据信号值而进行适当的动作。
      - 公共耦合：两个或两个以上的模块共同引用一个全局数据项。
      - 内容耦合：一个模块直接修改或操作另一个模块的数据，或一个模块不通过正常入口转入另一个模块。内容耦合是程度最高的耦合。

- 软件复杂性控制
  - 总体结构复杂性控制 (续)
    - 保持适当的扇入/扇出
      - 扇出量是影响模块宽度的主要因素。扇出量越大，模块就越复杂。
        - 适当增加中间层次的控制模块，以降低模块的扇出量。
        - 扇出量太小时，可把下级模块进一步分解成若干子功能模块或合并到它的上级模块中去。

- 软件复杂性控制
  - 总体结构复杂性控制 (续)
    - 简化软件接口
      - 接口复杂性是错误产生的重要根源。
        - 在设计软件接口时，应尽量使软件接口传递的信息简单，并与模块的功能一致。
      - 通常设计要求是单入口、单出口模块。
        - 可以有效地防止模块之间的内容耦合；
        - 便于降低接口的复杂性和冗余度；
        - 有利于改善一致性。
      - 设计软件接口应尽量避免模块之间的病态连接，谨慎使用转移进入或引用一个模块的内部结构。

- 软件复杂性控制
  - 软件复杂性控制的基本出发点
    - 形成软件高复杂性的重要原因：
      - 控制结构和数据结构复杂；
      - 转向语句使用不当；
      - 非局部变量较多；
      - 模块及过程之间联系密切；
      - 嵌套深度大；
      - 按地址调用参数比按值调用参数的复杂性高；
      - 循环结构比选择结构和顺序结构的复杂性高；
      - 模块宽度是形成软件复杂性的主要原因。
    - 上述内容构成了软件复杂性度量的基本度量准则集，是软件复杂性控制的基本出发点。软件复杂性度量力图对这些准则进行定义和定量描述，找出影响和制约软件复杂性的所有关系。

软件复杂性度量

概述

- 软件复杂性度量的结果是软件复杂度，是对软件复杂性的定量描述，为软件复杂性的定量分析和控制提供依据，是软件复杂性分析和控制研究的基础。
- 软件复杂性度量的根本目的
  - 通过控制软件复杂性来改善和提高软件的可靠性。
- 软件复杂性度量的方法和标准主要分为两大类：
  - 面向过程的软件复杂性度量 (研究最为活跃)
    - 典型方法：Line Count 语句行度量；基于 FPA (function points analysis 功能点分析) 的度量；*Halstead* 软件科学度量法和 *McCabe* 结构复杂性度量。
  - 面向对象的软件复杂性度量
    - 典型方法：C&K，MOOD 等。

软件复杂性分析

软件复杂性度量

- 软件复杂性度量元的主要分类
  - 规模
    - 通常由指令总数目或源程序代码行数表示。
    - 例如：Line Count 复杂度。
  - 难度
    - 通常由程序中出现的操作数的数目所决定的量表示。
    - 例如： *Halstead* 复杂度。
  - 结构
    - 通常由与程序结构有关的度量表示。
    - 例如： *McCabe* 复杂度。
  - 智能度
    - 算法的难易程度。

- 软件复杂性度量
  - 规模度量元 (Line Count 复杂度)
    - 规模度量元统计程序的源代码行数，是以程序规模为基准度量程序复杂性的最简单的方法。
      - 程序复杂性随着程序规模的增加不均衡地增长。
      - 采用分治策略可以有效控制程序规模。
    - 基本思想：统计一个程序模块的源代码行数目，并以源代码行数做为程序复杂性的度量值。
      - 代码出错率是每100行源代码中可能存在的错误数目。一般代码出错率的估算范围是从 0.04%-7% 之间。
      - 代码的出错率与源程序行数之间不存在简单的线性关系：随着源程序代码量的增加，代码出错率将非线性增长。
      - 代码行度量法是一个简单估计的方法，实践中很少单独用于复杂度估计。

软件复杂性度量

- 难度度量元 (*Halstead* 复杂度)
  - *Halstead* 复杂度 (*Maurice H. Halstead*, 1977) 是软件科学提出的第一个计算机软件的分析“定律”，用以确定计算机软件开发中的一些定量规律。
    - *Halstead* 复杂度采用一组基本的度量值，这些度量值通常在程序源代码产生之后直接得到，或者在设计完成之后进行估算。
  - *Halstead* 复杂度根据程序源代码中语句行的操作符和操作数的数量计算程序复杂性。
    - 程序源代码中操作符和操作数的量越大，程序难度就越大。
    - 操作符统计范围通常包括语言保留字、函数调用、运算符，也可以包括有关的分隔符等。
    - 操作数统计范围可以是常量和变量等标识符。

软件复杂性度量

难度度量元 (*Halstead* 复杂度) (续)

Halstead 复杂度度量

- 设 n_1 表示程序中不同的操作符个数，n_2 表示程序中不同的操作数个数，N_1 表示程序中出现的操作符总数，N_2 表示程序中出现的操作数总数。
- *Halstead* 程序词汇表长度 Program vocabulary: $n = n_1 + n_2$.
- *Halstead* 程序长度或简单长度 Program length: $N = N_1 + N_2$.
  - 注意到 N 定义为 *Halstead* 长度，并非源代码行数。
- 以 $N^{\wedge}$ 表示程序的预测长度 Calculated program length:

$$N^{\wedge} = n_1 \log_2 n_1 + n_2 \log_2 n_2.$$

- *Halstead* 的重要结论之一是：程序的实际长度 N 与预测长度 $N^{\wedge}$ 非常接近，这表明即使程序还未编写完也能预先估算出程序的实际长度 N。

软件复杂性度量

难度度量元 (*Halstead* 复杂度) (续)

Halstead 的其它计算公式

- 程序体积或容量 Volume: $V = N\log_2(n)$，表明了程序在词汇上的复杂性。
- 程序级别 Level: $L\hat{} = (2/n_1) \times (n_2/N_2)$，表明了一个程序的最紧凑形式的程序量与实际程序量之比，反映了程序的效率。
- 程序难度 Difficulty: $D = 1/L\hat{}$，表明了实现算法的困难程度。
- 编程工作量 Effort: $E = V \times D = V/L\hat{}$.
- 语言级别: $L' = L\hat{} \times L\hat{} \times V$.
- 编程时间 (hours): $T\hat{} = E/(S \times f)$，这里 $S = 60 \times 60$, $f = 18$.
- 平均语句大小: N/语句数。
- 程序中的错误数预测值: $B = V/3000 = N\log_2(n)/3000$.

软件复杂性度量

难度度量元 (*Halstead* 复杂度) (续)

E.g. Day of Week with *Zeller*'s Congruence.

```
if (m < 3) {
    m += 12;
    y -= 1;
}
int k = y % 100;
int j = y / 100;
int dayOfWeek = ((d + (((m + 1) * 26)/10) + k + (k/4) +
                (j/4)) + (5 * j)) % 7;
```

- It's an implementation of *Zeller*'s congruence for the *Gregorian* calendar, which determines the day of the week of a given date. The inputs *d*, *m*, *y* are day, month, and year, respectively.

软件复杂性度量

难度度量元 (*Halstead* 复杂度) (续)

- E.g. Day of Week with *Zeller*'s Congruence.

Operator	Number of Occurrences	Operand	Number of Occurrences	Operand	Number of Occurrences
If	1	*m*	3	100	2
<	1	*y*	3	26	1
+=	1	*k*	3	10	1
-=	1	*j*	3	4	2
=	3	*dayOfWeek*	1	5	1
%	2	*d*	1	7	1
/	4	3	1		
+	6	12	1		
*	2	1	1		
$n_1 = 9$	$N_1 = 21$			$n_2 = 15$	$N_2 = 25$

- 软件复杂性度量
  - 难度度量元 (*Halstead* 复杂度) (续)
    - E.g. Day of Week with *Zeller*'s Congruence.
      - *Halstead* Metrics
        - $n_1 = 9$, $n_2 = 15$, $N_1 = 21$, $N_2 = 25$.
        - Program vocabulary: $n = n_1 + n_2 = 24$
        - Program length: $N = N_1 + N_2 = 46$
        - Program volume: $V = N\log_2(n) = 210.68$
        - Program level: $L^{\wedge} = (2/n_1) \times (n_2/N_2) = 0.1333$
        - Program difficulty: $D = 1/L^{\wedge} = 7.500$

- 软件复杂性度量
  - 难度度量元 (*Halstead* 复杂度) (续)
    - *Halstead* 方法的优点
      - 不需要对程序进行深层次的分析，就能够对错误率和维护工作量进行估计；
      - 有利于项目规划，衡量所有程序的复杂度；
      - 计算方法简单；
      - 与所用的高级程序设计语言类型无关。
    - *Halstead* 方法的缺点
      - 仅仅考虑程序数据量和程序体积，没有考虑程序控制流的情况；
      - 不能从根本上反映程序复杂性。

- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度)
    - *McCabe* 复杂度 (*Thomas J. McCabe, Sr.*, 1976) 方法对程序流程图 (Program Flow Chart) 进行静态分析，将其转化为程序控制流图 (Control Flow Graph, CFG 是一个有向图)，然后基于图论的方法对 CFG 进行严格的结构分析，是对程序拓扑结构复杂性的度量。
      - 程序控制流图的一个重要性质是它的可规约性 (reducibility)。如果程序中不存在从循环外跳到循环内的转移语句，那么这个程序对应的控制流图称为是可规约的 (reducible)，反之这个控制流图就是不可规约的 (irreducible)。因此，模块符合结构化设计准则是其控制流图可规约的基础。

- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 复杂度包括：
      - 环路复杂度 (Cyclomatic Complexity)、基本复杂度 (Essential Complexity)、模块设计复杂度、设计复杂度、集成复杂度、行数、规范化复杂度、全局数据复杂度、局部数据复杂度、病态数据复杂度。
    - *McCabe* 环路复杂度
      - 一个程序模块的环路复杂度用来衡量模块中判定结构的复杂程度，数量上可以表现为程序控制流图 (将在第4章白盒测试详细讨论) 中从开始点到终结点的独立路径条数，相当于合理预防错误所需测试的最少路径条数。
      - 程序的可能错误和环路复杂度有着密切的关系，高环路复杂度说明程序代码可能质量低而且难以测试和维护。

软件复杂性度量

结构度量元 (*McCabe* 复杂度) (续)

McCabe 环路复杂度 (续)

- 单入单出程序控制流图 G 的 *McCabe* 环路复杂度定义为：

$$V(G) = m - n + 2p$$

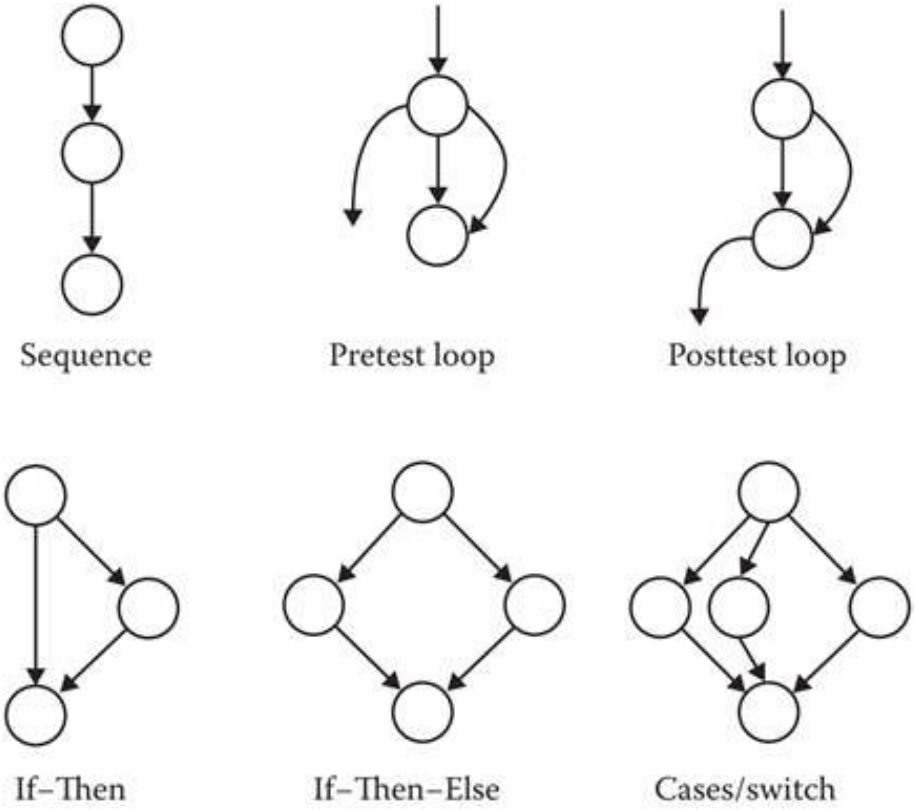
  - m 是 G 的边数目
  - n 是 G 的顶点数目
  - p 是 G 的连通分支数

- 简单程序控制流图是连通图，$p = 1$，此时：

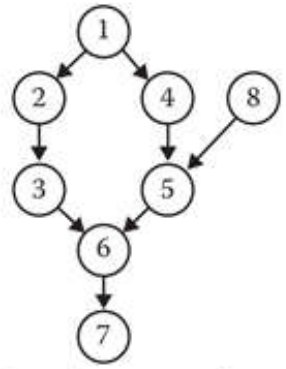
$$V(G) = m - n + 2$$

- G 是平面连通图时，由欧拉公式，$V(G) = R$。其中 R 是平面被控制流图划分成的区域数目 (包括外部面)。
- 对于简单的单入单出结构化模块，$V(G)$ 值等于程序控制流图中的单条件判断节点的个数 +1。多条件判断条件可以先转化为单条件复合结构再应用本结论。

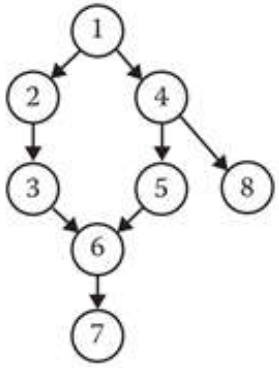
- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 环路复杂度 (续)
      - 结构化模块



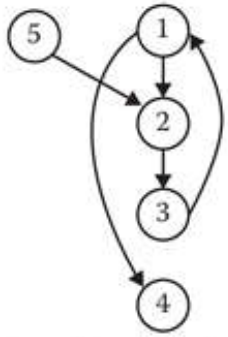
- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 环路复杂度 (续)
      - 非结构化模块



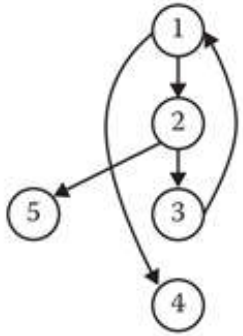
Branching into a decision



Branching out of a decision

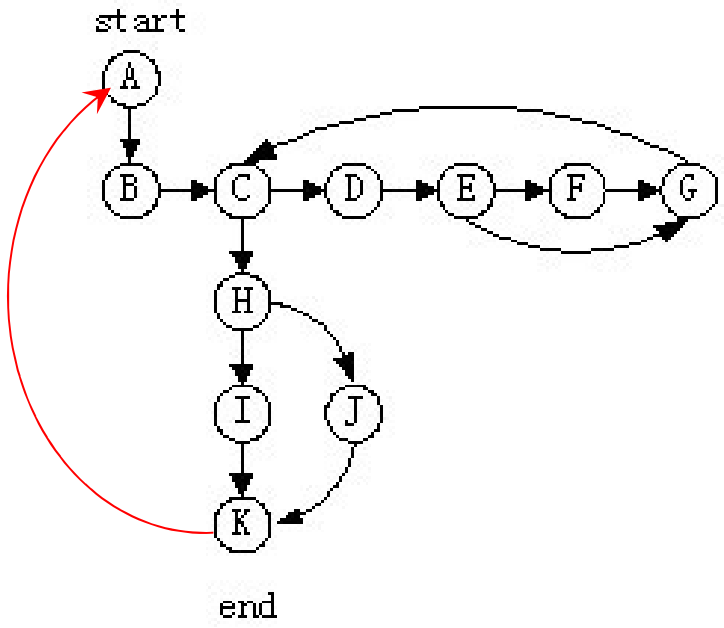


Branching into a loop



Branching out of a loop

- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 环路复杂度 (续)
      - 另外的讨论
        - 在单入单出的程序控制流图 *G* 中增加从出口指向入口的辅助边，得到一个强连通图 *G'*。



- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 环路复杂度 (续)
      - 另外的讨论
        - G' 的顶点数 $n' = n$，边数 $m' = m + 1$，连通分支数 $p' = 1$。强连通图的环路复杂度定义为图的秩数 (或圈数、第一贝蒂数)：

$$\begin{aligned} V(G') &= Rank(G') \\ &= m' - n' + p' \\ &= m' - n' + 1 \\ &= m - n + 2 \end{aligned}$$

        - G' 的秩数 $Rank(G')$ 是 G' 的线性无关独立回路数目。
        - G' 的环路复杂度和 G 的 *McCabe* 环路复杂度定义是一致的。

软件复杂性度量

结构度量元 (*McCabe* 复杂度) (续)

McCabe 环路复杂度 (续)

- 例：计算右下图的程序控制流图的 *McCabe* 环路复杂度。
- 解1：图中 $m = 13$，$n = 11$，故

  $V(G) = m - n + 2$

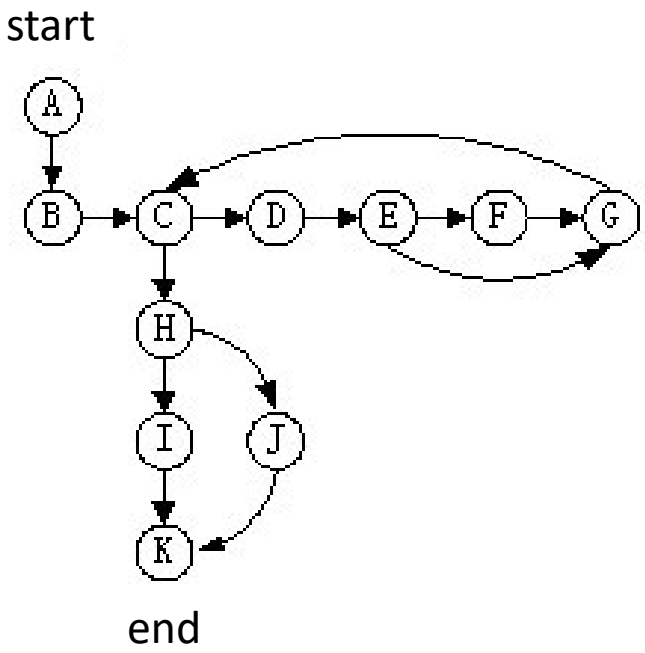
  $= 13 - 11 + 2$

  $= 4$
- 解2：图是平面的且有4个面，故

  $V(G) = 4$
- 解3：图中有3个单条件判定节点 C, H, E，故

  $V(G) = 4$

(单条件判定问题在第4章讨论)

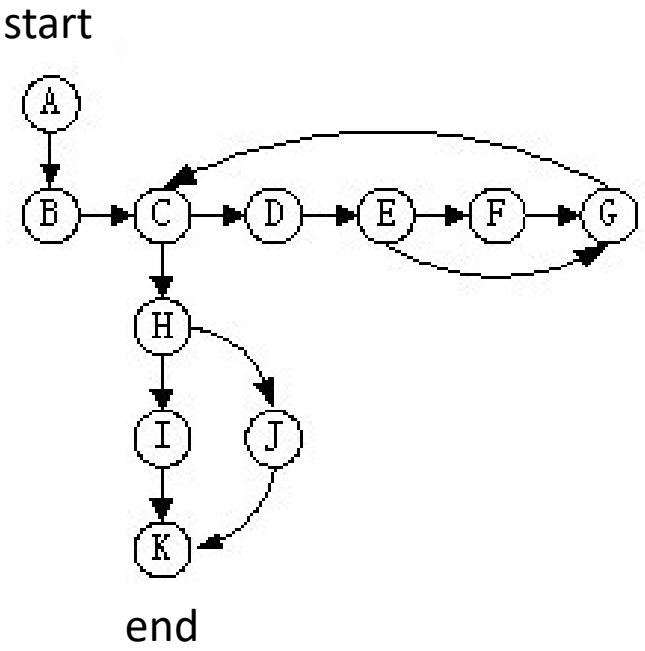


软件复杂性度量

结构度量元 (*McCabe* 复杂度) (续)

McCabe 环路复杂度 (续)

- 简单的单入单出模块的 *McCabe* 环路复杂度等于程序控制流图的最大独立路径数目，它指出为防止出错所需要的最少测试次数，典型应用于白盒测试的基本路径测试方法。
- 例：右图的4条基本路径：
  - A-B-C-H-I-K
  - A-B-C-H-J-K
  - A-B-C-D-E-F-G-C-H-I-K
  - A-B-C-D-E-G-C-H-I-K



- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 环路复杂度 (续)
      - *McCabe* 环路复杂度实质上是对程序控制流复杂性的度量，它并不考虑数据流，因而其科学性和严密性具有一定的局限。
      - 环路复杂性度量原理的应用
        - 适用于源代码分析
        - 避免引起可靠性问题的过度复杂化
        - 使用度量值驱动测试过程

- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 基本复杂度
      - 计算方法
        - 将程序控制流图中的结构化部分简化成一个点，基本复杂度定义为简化后的控制流图的环路复杂度。
      - 基本复杂度用来衡量程序的非结构化程度
        - 非结构成分降低了程序的质量，增加了代码的维护难度，使程序难以理解。
        - 基本复杂度高意味着非结构化程度高，难以模块化和维护，消除一个错误可能容易导致其他的错误。
      - 优点
        - 衡量非结构化程度；反映代码质量；预测代码维护量，辅助模块划分；与所用的高级程序设计语言类型无关。

- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 基本复杂度 (续)
      - 应用
        - 当基本复杂度为1时，这个模块是充分结构化的。
        - 当基本复杂度大于1而小于环路复杂度时，这个模块是部分结构化的。
        - 当基本复杂度等于环路复杂度时，这个模块是完全非结构化的。

- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 模块设计复杂度
      - 计算方法
        - 从模块控制流图中移去那些不包含调用子模块的判定和循环结构后得出的环路复杂度就是模块设计复杂度，它应该不大于环路复杂度，通常远小于环路复杂度。
      - 模块设计复杂度用于衡量模块和其他模块的调用关系
        - 软件模块设计复杂度高意味着模块耦合度高，这将导致模块难以隔离、维护和复用。
      - 优点
        - 衡量模块对其下层模块的支配作用；
        - 衡量一个模块到其子模块进行集成测试的最小数量；
        - 是设计复杂度 (S0) 和集成复杂度 (S1) 计算的基础。

- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 设计复杂度 S0
      - 计算方法
        - S0 是程序中所有模块设计复杂度之和。
      - 程序设计复杂度以数量来衡量组成程序的模块之间的相互作用关系
        - 它提供了系统级模块设计复杂度的概况。
        - 高 S0 的系统意味着系统各部分之间有复杂的相互关系。
      - 优点
        - 可应用于完整的软件，也可应用于任何子系统；
        - 揭示了程序中模块调用的复杂程度，指出一个模块的整体复杂度反映了该模块和其内部模块的控制关系；
        - 有助于集成复杂度 S1 的计算。

- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 集成复杂度 S1
      - 计算方法
        - S1 的计算公式：S1 = S0 - *N* + 1，*N* 是程序中模块的数目。
      - 程序的集成复杂度给出对程序进行完全测试所需要的集成测试的数目，即程序的独立线性子树的数目。
      - 优点
        - 有助于集成测试的实施；
        - 对集成测试工作进行量化且反映了系统设计复杂度；
        - 有助于从整体上隔离系统复杂度。
    - *McCabe* 的其它度量元 (略)

- 软件复杂性度量
  - 结构度量元 (*McCabe* 复杂度) (续)
    - *McCabe* 复杂性度量在软件工程中的应用
      - 作为测试的辅助工具
        - *McCabe* 复杂性度量的结果等于通过一个程序的独立路径数，因而需要设计同样多的测试案例以覆盖所有路径。
      - 作为程序设计和管理指南
      - 作为网络复杂性度量的一种方法

- 软件复杂性度量
  - 其它复杂度度量元
    - 函数参数个数、路径数、层次数、直接调用个数、RETURN 语句个数、调用者的个数、GOTO 语句个数、词汇频度、局部变量个数、注释率、函数中的可执行语句数、宏定义个数、扇入/扇出数等。

- 面向对象软件的复杂性度量
  - 传统的软件度量方法无法适应 OO 技术中采用的数据抽象、封装、继承、多态性、信息隐藏、重用等机制。
    - 例如，继承提供的重用中属于对象自身的代码较少，如果用源代码行 (LOC) 来度量整个对象，结果不能令人满意。
    - 把类看作模块是不恰当的。
      - 功能性的模块之间的耦合关系表现在接口上，主要是参数传递、对全程变量的访问以及模块间的调用关系。
      - 对象间的耦合关系主要表现为通过继承、通过消息传递和接收、通过对抽象数据类型的引用带来的耦合。
  - OO 度量的主要目的
    - 更好地理解产品的质量；
    - 提高评价过程的效率；
    - 改进项目完成工作的质量。

- 面向对象软件的复杂性度量
  - 面向对象度量的特性
    - 局域性是指信息被集中在一个程序内的方式。
      - 传统方法是数据与过程分离、功能分解和功能信息局域化
        - 典型的实现形式为过程模块，工作时由数据驱动。
        - 度量放在功能内部结构和复杂性上 (如模块规模、聚合度、环路复杂性等) 或放在该功能与其他功能 (模块) 的耦合方式上。
      - OO 方法的局域性是基于对象的。
        - 类是 OO 系统的基本单元：应把类 (对象) 作为一个完整实体来量度 。
        - 操作 (功能) 和类之间的关联是一对一的：在考虑类合作中的量度时，必须能适应一对多和多对一的关联。

- 面向对象软件的复杂性度量
  - 面向对象度量的特性 (续)
    - 封装性是指一个项集合的包装。
      - 传统方法：记录和数组中只有数据而没有过程，属于低层次的封装；过程、函数、子例程和段则只有过程而没有数据，属于中层次的封装。
        - 度量的重点分别在代码行的数据和环路的复杂性。
      - OO 方法：OO 系统封装拥有类的职责 (操作)，包括类的属性、操作和特定的类属性值定义的类 (对象) 的状态。
        - 度量对象不是单一的模块，而是包含数据 (属性) 和过程 (操作) 的包。

- 面向对象软件的复杂性度量
  - 面向对象度量的特性 (续)
    - 信息隐藏是指隐藏 (删除) 了程序构件操作的细节，只将访问该构件所必要的信息提供给访问该构件的其他构件。
      - OO 方法和传统方法基本一致。
      - OO 系统应支持信息隐藏，除提供隐藏等级说明的量度外，还应提供 OO 设计质量指标。
    - 继承性是指一个对象的属性和操作能够传递给其他对象的机制
      - 继承性的发生贯穿于一个类的所有层次；一般来说，传统软件不支持这种特性。
      - 对 OO 系统来说，继承性是一个关键性的特性。对OO系统进行继承性度量非常重要：如子的数量 (类的直接实例数量)、父的数量 (直接上一代数量) 以及类的嵌套层次 (在一个继承层次中，类的深度)。

- 面向对象软件的复杂性度量
  - 面向对象度量的特性 (续)
    - 抽象方法使设计者只关心一个程序构件的数据和过程，而不考虑底层的细节。
      - 抽象是一种相对概念，在 OO 和传统开发方法中都被采用。
        - 当处于抽象的较高层次时，可忽略更多的细节，只提供一个关于概念或项的一般看法。
        - 当处于抽象的较低层次时，可以引入更多的细节，即提供一个关于概念或项的更详细的看法。
      - 在 OO 中，类就是一个抽象。它可以从许多不同的细节层次和用许多不同的方式 (如作为一个操作的列表、一个状态的序列、一组合作) 来观察。
      - OO 量度可用一个类度量的项来表示抽象。
        - 如每个应用类的实例化的数量、每个应用类被参数化的数量，以及类被参数化与未被参数化的比例等。

- 面向对象软件的复杂性度量
  - 面向类的度量
    - CK 度量
      - 计算每个类的加权方法数 WMC、继承树的深度 DIT、孩子的个数NOC、对象类之间的耦合 CBO、类的响应 RFC、方法中的聚合缺乏 LCOM。
    - LK 度量
      - 规模、继承、内部 (特性) 和外部 (特性)
        - 基于规模的度量，主要集中在单一类的属性和操作的数量，以及作为整个 OO 系统的平均值；
        - 基于继承的度量，关注的是贯穿于类层次的操作被重用的方式；
        - 类的内部特性的度量是考察聚合和代码问题；
        - 外部特性的度量则是检查耦合和重用问题。

- 面向对象软件的复杂性度量
  - 面向类的度量 (续)
    - MOOD 度量
      - MOOD 方法从封装性、继承性、耦合性和多态性四方面提出了系统级度量指标。
    - CK、LK 和 MOOD 这三种基于类的度量方法中的度量元都反映出面向对象技术的特点，但度量的侧重点有所不同。
      - CK 度量组中，系统的多态性通过 RFC 和 WMC 间接度量，CBO 和 LCOM 评测封装性，系统的继承性用 DIT 和 NOC 评测，但它没有多态性度量指标。
      - LK 度量组中，CS 反映系统的封装性，NOO 是基于继承的度量着重于在整个类层次中操作被重用的方式，NOA 考察内聚和面向源代码的问题，SI 检查耦合和重用。
      - MOOD 度量方法则从封装性、继承性、耦合性、多态性四个方面给出六个度量指标，作用于类的属性和方法。

- 面向对象软件的复杂性度量
  - 面向操作的度量
    - 类是 OO 系统设计中的基础框架或模板设施，而 OO 系统真正活动的实体是类的对象。
      - 类对象具有状态和行为两大特性，分别用数据和操作表示，因此必须对类操作进行复杂性度量。
    - 类操作复杂性度量的三种度量元
      - 平均操作规模 OSavg：在 OO 软件中由操作所传送的消息数量提供了一个对操作规模度量可选的方法。操作规模增大，表示操作所传送的消息数量增加，数量越大，说明越复杂。
      - 操作复杂性 OC：一个操作的复杂性可以用传统软件所使用的任何复杂性量度进行计算。
      - 每个操作参数的平均数 NPavg：操作参数的数量越大，对象间的合作就越复杂。

- 面向对象软件的复杂性度量
  - 面向系统的度量
    - 封装性
      - 方法 (操作与服务) 中的聚合缺乏性度量 LCOM
        - LCOM 的值越高，表示更多的状态必须进行测试，才能保证该方法不产生副作用。
      - 公共与私有的百分比 PAP (Percent Public and Protected)
        - 公共属性从其他类继承，所以这些类是可见的。私有属性是专属的，为一特定子类所拥有。
        - PAP 说明类的公共属性的百分比，PAP 的值高则可能增加类间的副作用。
      - 数据成员的公共访问 PAD (Public Access to Data Member)
        - PAD 说明可访问其他类属性的类的数量，即封装的破坏程度。PAD 值高可能导致类间的副作用，测试的设计必须保证每一种这样的副作用能够被发现。

- 面向对象软件的复杂性度量
  - 面向系统的度量 (续)
    - 继承性
      - 根类的数量 NOR
        - NOR 描述在设计模型中，描述性质各不相同的类层次数量的计算方法。
        - 对每一个根类及其子类的层次必须开发相应的测试序列。随着 NOR 的增大，测试工作量也相应增加。
      - 扇入 FIN
        - 面向 OO 系统时，扇入是一种多重继承的指标。
        - FIN 大于1，说明一个类不只是从属一个根类，而是继承更多的根类的属性和操作。
      - 孩子的数量 NOC 和继承树的深度 DIT
      - 父类的方法 (操作、服务) 发生变化将导致需要对每个子类进行重新测试

Lecture 14. Complexity Analysis

End of Lecture