中山大學
SUN YAT-SEN UNIVERSITY

Chapter 3
Software Static Testing

Software Testing: approaches & Technologies

School of Data & Computer Science, Sun Yat-sen University

Outline

- 3.1 软件静态测试概述
- 3.2 阶段评审
- 3.3 软件的代码检查
  - 代码检查的基本方法
  - 软件编程规范检查
  - MISRA C 编程规范
  - 代码的自动分析和结构分析
  - 代码安全性检查
- 3.4 软件复杂性分析
- 3.5 软件质量控制
- 3.6 软件静态分析工具

3.3 软件的代码检查

- **软件代码检查概述**
  - 软件代码检查以小组为单位阅读代码，是一系列规程和错误检查技术的集合。
    - ✧ 代码检查通常将注意力集中在发现错误、而不是纠正错误上。
    - ✧ 代码检查研究分析代码而不是实际执行代码，一般采用静态白盒测试的方法。
      - ◌ 如桌面检查、代码走查和技术评审
    - ✧ 代码检查是众多软件测试方法中发现软件缺陷最有效的方法之一。
  - 软件代码检查的时机
    - ✧ 软件代码全部或部分完成后，应立即进行逐行代码检查以期尽早发现缺陷，使程序更具可测试性。
      - ◌ 大部分的程序错误可通过代码检查发现。
      - ◌ 80%的程序性错误是由20%的代码错误引起的。

3.3 软件的代码检查

- **软件代码检查概述**
  - 软件代码检查的组织约束
    - ✧ 评审人员具备程序开发语言专业知识
    - ✧ 具备用作参照物的程序基线和有关标准
  - 软件代码检查输出的信息
    - ✧ 度量标准、易产生错误的代码、代码规则的执行、流图和调用图的分析
      - ◌ 代码复杂性度量：McCabe、Halstead、嵌套级别 (最大/平均) 等
      - ◌ 规格度量：行数、语句数、注释数、声明数等

3.3 软件的代码检查

- **软件代码检查概述**
  - 软件代码检查内容
    1. 完整性检查
       - 代码是否完全实现了设计文档中提出的功能需求。
       - 代码中是否存在没有定义或没有引用到的变量、常数或数据类型。
    2. 一致性检查
       - 代码的逻辑是否符合设计文档。
       - 代码中使用的格式、符号、结构等风格是否保持一致。
    3. 正确性检查
       - 代码是否符合制定的标准。
       - 是否所有的变量都被正确定义和使用。
       - 是否所有的注释都是准确的。

3.3 软件的代码检查

- **软件代码检查概述**
  - 软件代码检查内容 (续)

4. 可修改性检查
   - 代码涉及到的常量是否易于修改。
   - 是否使用配置、定义为类常量、使用专门的常量类等。

5. 可预测性检查
   - 代码是否具有定义良好的语法和语义。
   - 代码是否可能无意陷入死循环。
   - 代码是否避免了无穷递归。

6. 健壮性检查
   - 代码是否采取措施避免运行时错误，如空指针异常等。

3.3 软件的代码检查

- **软件代码检查概述**
  - 软件代码检查内容 (续)

    7. 可理解性检查
       - 注释是否足够清晰的描述每个子程序，对于没用的代码注释是否删除。
       - 是否使用不明确或不必要的复杂代码，它们是否被清楚的注释。
       - 是否使用了一些统一的格式化技巧用来增强代码的清晰度，诸如缩进、空白等。
       - 是否在定义命名规则时采用了便于记忆，反映类型等方法
       - 循环嵌套是否太长太深。

    8. 可验证性检查
       - 代码中的实现技术是否便于测试。

3.3 软件的代码检查

- **软件代码检查概述**
  - 软件代码检查内容 (续)

9. 结构性检查
   - 程序的每个功能是否都作为一个可辩识的代码块存在。
   - 循环是否只有一个入口。
     - 例如，使用 goto 语句跳转到循环体内的一个标号，将使循环体在逻辑上增加一个入口

10. 可追溯性检查
   - 代码是否对每个程序进行了唯一标识。
   - 是否有一个交叉引用的框架可以用来在代码和开发文档之间相互对应。
   - 代码是否包括一个修订历史记录。
   - 记录中对代码的修改和原因是否都有记录，是否所有的安全功能都有标识。

3.3 软件的代码检查

- **软件代码检查概述**
  - 软件代码检查内容 (续)

    11.代码标准符合性检查

    - 标准是预先建立的必须遵守的规则：
      - 必须做什么
      - 不能做什么
    - 规范是建议的最佳做法。规范可以适当的放宽。
    - 有些代码虽然可以正常运行，但代码的编写不符合某种标准或规范。
    - 违反标准符合性的代码将严重影响软件的可靠性、可读性、可维护性和可移植性。

3.3 软件的代码检查

- **软件代码检查方法**
  - 软件代码检查方法
    - ✧ 代码检查方法主要有桌面检查、代码走查、代码评审和技术审查等类型
      - ◌ 桌面检查 (Desk Checking) 由程序员自己检查自己所编写的程序。
      - ◌ 代码走查和代码评审 (Walkthroughs & Reviews) 由若干个开发人员和测试人员组成一个小组，举行会议进行集体讨论，找出错误或遗漏，但不解决问题。
      - ◌ 技术审查 (Inspections) 是最正式的检查类型。
    - ✧ 可选择的方法是：采用代码评审作为核心代码的评审方式，采用走查和同级桌查作为一般代码的评审方式，条件成熟时举行技术审查会议。

3.3 软件的代码检查

- **软件代码检查方法**
  - 桌面检查 Desk Checking
    - 桌查通常指程序员自己检查自己所编写的程序。
      - 桌查是程序员根据相关的文档对源程序代码进行分析、检验，以期发现程序中错误的过程。
        - 用作自我检查以期发现一些较明显的错误和漏洞
    - 桌面检查的主要内容：
      - 代码和设计是否一致
      - 代码是否遵循标准、是否可读
      - 代码逻辑表达是否正确
      - 代码结构是否合理
      - 程序编写是否符合编写标准和规范
      - 程序中是否有不安全、不明确和模糊的部分
      - 程序编写风格是否符合要求

3.3 软件的代码检查

- **软件代码检查方法**
  - 桌面检查 (续)
    - ✧ 桌面检查的工作细分
      - ◌ 检查变量的交叉引用表：是否有未说明的变量和违反了类型规定的变量
      - ◌ 检查标号的交叉引用表：验证所有标号是否正确
      - ◌ 检查子程序、宏、函数：验证每次调用与所调用位置是否正确，调用的子程序、宏、函数是否存在，参数是否一致
      - ◌ 检查全部等价变量的类型的一致性
      - ◌ 确认常量的取值和数制、数据类型
      - ◌ 选择、激活路径：在设计控制流图中选择某条路径，在实际的程序中检查能否激活这条路径

3.3 软件的代码检查

- **软件代码检查方法**
  - 桌面检查 (续)
    - 桌面检查的缺陷
      - 效率不高
      - 自己很难发现一些习惯性错误
      - 程序员没有发现自己的程序错误的心理欲望
      - 难以 (无法) 纠正对程序功能的理解错误

3.3 软件的代码检查

- **软件代码检查方法**
  - 代码走查 Walkthrough
    - ✧ 代码走查是一类非正式的软件代码检查方法。
    - ✧ 在代码走查中，编写代码的程序员要向5人小组或者由其他程序员和测试人员组成的小组做正式陈述。
      - ◌ 在这个小组中至少有一位资深程序员
      - ◌ 测试人员应是具有经验或精通程序设计的程序设计人员
      - ◌ 通常需要一位没有介入到这个项目中的新人
        - 新人不会被已有的设计约束，容易发现问题。

3.3 软件的代码检查

- **软件代码检查方法**
  - 代码走查 (续)
    - ✧ 代码走查的流程：
      - ◌ 审查者阅读材料；
      - ◌ 会议期间由陈述者 (编写代码的程序员) 逐行或逐段通读代码，解释代码的工作原理；
      - ◌ 审查者提题；
      - ◌ 审查小组做出审查结果的书面报告和错误报告，交给程序开发人员。

3.3 软件的代码检查

- **软件代码检查方法**
  - 代码走查 (续)
    - ✧ 代码走查的17项检查内容：

- 文档和源程序代码
- 功能
- 界面
- 流程
- 提示信息
- 函数
- 数据类型与变量
- 条件判断
- 循环
- 输入输出
- 注释
- 程序 (模块)
- 数据库
- 表达式分析
- 接口分析
- 函数调用关系图
- 模块控制流图

3.3 软件的代码检查

- **软件代码检查方法**
  - 代码评审 Code Review
    - ✧ 代码评审要点
      - ◌ 代码和设计的一致性
      - ◌ 代码执行标准的情况
      - ◌ 代码的逻辑表达正确性
      - ◌ 代码结构的合理性
      - ◌ 代码的可读性等
    - ✧ 代码评审内容
      - ◌ 控制流分析：非结构化的代码、死代码等
      - ◌ 数据流分析：未定义的数据的使用、未使用的数据等
      - ◌ 信息流分析
      - ◌ 断言分析

3.3 软件的代码检查

- **软件代码检查方法**
  - 代码评审 (续)
    - ✧ 代码评审组人员构成
      - ◌ 评审小组由组长、资深程序员、程序编写者与专职测试人员等组成；组长不能是被测程序的编写者；组长负责分配资料、安排计划、主持会议、记录并保存被发现的差错。
    - ✧ 工作方式
      - ◌ 采用讲解、提问结合检查表的方式审查代码，寻找问题和失误。
      - ◌ 执行流程：计划准备、程序阅读、会议、结果报告。
    - ✧ 代码评审清单
      - ◌ 数据引用错误、数据声明错误、计算错误、子程序参数错误、静态结构问题以及控制流错误、比较错误、输入/输出错误等。

3.3 软件的代码检查

- **软件代码检查方法**
  - 技术审查 Inspection
    - ✧ 技术审查是最正式的评审类型，具有高度的组织化。
    - ✧ 技术审查由开发组、测试组和相关人员 (QA、产品经理等) 联合进行，要求每一个参与者训练有素。
    - ✧ 技术审查综合运用走查和代码评审技术，逐行、逐段检查软件。
    - ✧ 表述者不是原来编写代码的程序员 (与走查和代码评审不同)，这就要求表述者学习和了解所要表述的材料，从而有可能对程序提出不同的看法和解释。
    - ✧ 技术审查的检查要点是设计需求、代码标准/规范/风格和文档的完整性与一致性。

3.3 软件的代码检查

- **软件编程规范检查**
  - 编码规范
    - ✧ 编码规范又称编程风格。
    - ✧ 编码规范是对程序代码的格式、注释、标识符命名、语句使用、函数、类、程序组织、公共变量等方面的具体要求。
    - ✧ 合适的编码规范是提高代码质量最有效、最直接的手段 。
      - ◌ 有助于提高代码的健壮性、安全性和可靠性。
      - ◌ 有助于提高代码的可读性，使代码易于查看、易于理解和维护。
  - 编码规范的分级
    - ✧ 编码规范分为两个级别：规则和建议
      - ◌ 软件开发人员必须严格遵守规则级的编码规范。
      - ◌ 软件开发人员应该尽量遵守建议级的编码规范。

3.3 软件的代码检查

- **软件编程规范检查**
  - 格式
    - ✧ 格式规范是对程序代码书写格式的要求，如：
      - ◌ 空行、空格的使用
      - ◌ 缩进：对程序语句要按其逻辑进行水平缩进
      - ◌ 长语句的书写格式
      - ◌ 清晰划分控制语句的语句块
      - ◌ 一行只写一条语句，一次只声明、定义一个变量
      - ◌ 在表达式中使用括号
      - ◌ 将操作符 “*”、“&” 和类型写在一起

3.3 软件的代码检查

- **软件编程规范检查**
  - 注释
    - ✧ 程序注释是程序与后续阶段的程序阅读者之间沟通的重要手段。
      - ◌ 良好的注释能够帮助阅读者理解程序，为后续阶段的测试和维护提供明确的指导。
    - ✧ 注释的基本原则：
      - ◌ 注释内容要清晰明了，含义准确，防止出现二义性。
      - ◌ 编写代码的同时编写注释，修改代码的同时修改相应的注释，保证代码与注释一致。
    - ✧ 注释的内容：
      - ◌ 对函数、类、文件、空循环体、多个 case 语句共用一个出口等进行注释。
      - ◌ 行末注释尽量对齐。
    - ✧ 注释量：注释行的数量不得少于程序行数量的 1/3 。

3.3 软件的代码检查

- **软件编程规范检查**
  - 命名
    - ✧ 命名是对标识符和文件的命名要求。
    - ✧ 在程序中声明、定义的变量、常量、宏、类型、函数，在对其命名时应该遵守统一的命名规范。
    - ✧ 标识符长度要求：
      - ◌ 在程序中声明、定义的变量、常量、宏、类型、函数，它们的名字长度要在4至25个字符之内。
    - ✧ 文件命名要求：
      - ◌ 代码文件的名字要与文件中声明、定义的类的名字基本保持一致，使类名与类文件名建立联系。

3.3 软件的代码检查

- **软件编程规范检查**
  - 语句
    - ✧ 一条程序语句中只包含一个赋值操作符
    - ✧ 不要在控制语句的条件表达式中使用赋值操作符
    - ✧ 赋值表达式要满足相关规定
    - ✧ 使用正规格式的布尔表达式
    - ✧ 限制/禁用 goto 语句 (内核程序除外)
    - ✧ 限制/禁用 break、continue 语句
    - ✧ 字符串的赋值应采用 _T("") 模式
    - ✧ 避免对浮点数值类型做精确比较
    - ✧ new 和 delete 要成对出现 (局部)
    - ✧ 对 switch 语句中每个分支要以 break 语句结尾
    - ✧ switch 语句中的 default 分支的使用约定

3.3 软件的代码检查

- **软件编程规范检查**
  - 语句 (续)
    - ✧ 指针初始化；释放内存后的指针变量赋 NULL 值
    - ✧ 指针指向的数据成员的访问方式，如在代码中用 ptr->fld 的形式代替 (*ptr).fld 的形式

3.3 软件的代码检查

- **软件编程规范检查**
  - 函数
    - ✧ 明确函数功能
      - ◌ 函数体代码长度不得超过100行 (不包括注释行)
    - ✧ 将重复使用的代码编写成函数
    - ✧ 尽量保持函数只有唯一的出口 (结构化要求)
    - ✧ 函数声明和定义的格式要求
      - ◌ 在函数参数列表中为各参数指定类型和名称
    - ✧ 为函数指定返回值
      - ◌ 要为每一个函数指定它的返回值。如果函数没有返回值，则要定义返回类型为 void 。
    - ✧ 在函数调用语句中不要使用赋值操作符
      - ◌ 在函数的参数列表中不要使用赋值操作符
    - ✧ 保护可重入函数中的全局变量

3.3 软件的代码检查

- **软件编程规范检查**
  - 类
    - ✧ 缺省构造函数：为每一个类显式定义缺省构造函数
    - ✧ 拷贝构造函数
      - ◌ 当类中包含指针类型的数据成员时，必须显式定义拷贝构造函数；建议为每个类都显式定义拷贝构造函数
    - ✧ 为类重载“=”操作符
      - ◌ 当类中包含指针类型的数据成员时，必须显式重载“=”操作符
      - ◌ 建议为每个类都显式重载“=”操作符
    - ✧ 析构函数：为每一个类显式定义析构函数
    - ✧ 虚拟析构函数：基类的析构函数一定要为虚拟函数
    - ✧ 在派生类中不要对基类中的非虚函数重新进行定义。
      - ◌ 如果确实需要，那么应该在基类中将该函数声明为虚函数

3.3 软件的代码检查

- **软件编程规范检查**
  - 类 (续)
    - ✧ 用内联函数 (inline) 代替宏函数
      - ◌ 相比之下，内联函数具有宏函数的效率，而且使用起来更安全。
    - ✧ 如果重载了操作符 "new"，也应该重载操作符 "delete“
      - ◌ 操作符 new 和操作符 delete 需要配对。
    - ✧ 类数据成员的访问控制
      - ◌ 类对外的接口应该是完全功能化的，类中可以定义 public 的成员函数，但不应该有 public 的数据成员。
    - ✧ 限制类继承的层数 (不超过5层)
    - ✧ 慎用多继承
      - ◌ 多继承会显著增加代码的复杂性，还会带来潜在的混淆。
    - ✧ 考虑类的复用

3.3 软件的代码检查

- **软件编程规范检查**
  - 程序组织
    - ✧ 一个头文件中只声明一个类。
    - ✧ 一个源文件中只实现一个类。
    - ✧ 头文件中只包含声明，不应包含定义或实现。
    - ✧ 源文件中不要有类的声明。
    - ✧ 可被包含的文件：
      - ◌ 只允许头文件被包含到其它的代码文件中去。
    - ✧ 避免头文件的重复包含。

3.3 软件的代码检查

- **软件编程规范检查**
  - 公共变量
    - 严格限制公共变量的使用。
      - 公共变量会增大模块间的耦合度。
    - 明确公共变量的定义。
      - 决定使用公共变量时，要仔细定义并明确公共变量的含义、作用、取值范围、与其它变量间的关系。
      - 明确公共变量与操作此公共变量的函数之间的关系，如访问、修改和创建等。
    - 防止公共变量与局部变量重名。

3.3 软件的代码检查

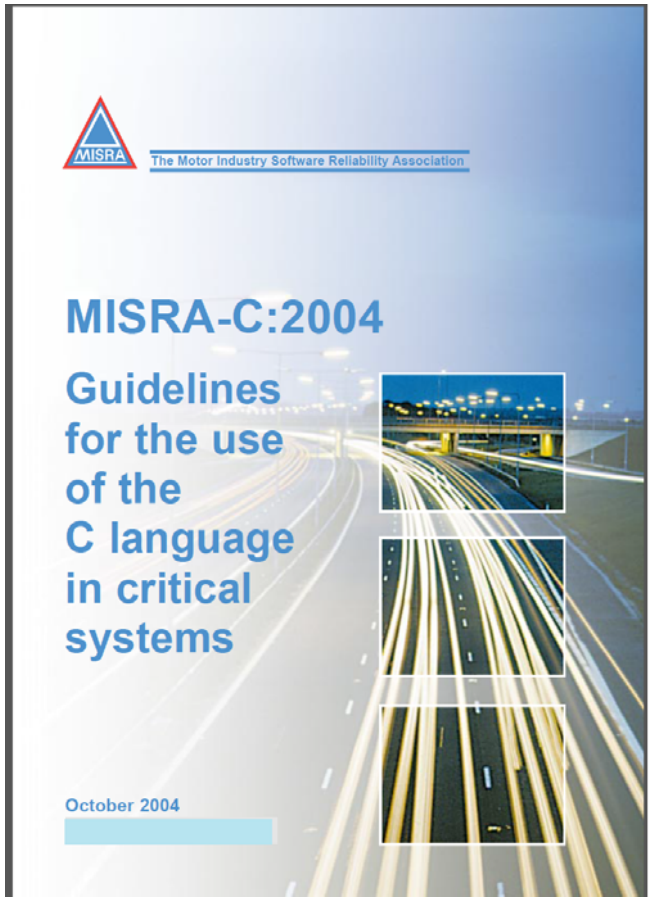
- **软件编程规范检查**
  - 其它
    - ✧ 慎用结构体和联合体
      - ◌ 以符合面向对象的思想
    - ✧ 用常量代替宏
      - ◌ 在不损失效率的同时，使用 const 常量比宏更加安全。
    - ✧ 括号在宏中的使用
      - ◌ 对于宏的展开部分，在宏的参数出现的地方要加括号“( )”
      - ◌ 保证宏替换的安全，同时提高代码的可读性
    - ✧ 尽量使用 C++ 风格的类型转换
      - ◌ 用 C++ 提供的类型转换操作符 static_cast，const_cast，dynamic_cast 和 reinterpret_cast 代替 C 风格的类型转换符
    - ✧ 将不再使用的代码删除

3.3 软件的代码检查

- **MISRA C 编程规范**
  - MISRA
    - ✧ MISRA, Motor Industry Software Reliability Association，汽车工业软件可靠性协会，*http://www.misra.org.uk/*
    - ✧ MISRA 于1998年发布了一个针对汽车工业软件安全性的 C 语言编程规范《汽车专用软件的 C 语言编程指南》(MISRA-C, a set of software development guidelines for the C programming language) 。
    - ✧ MISRA-C:1998 共有127条规则；MISRA-C:2004 共有强制规则121条，推荐规则20条：
      - ◌ 任何符合 MISRA-C:2004 编程规范的代码都应该严格的遵循121条强制规则的要求，并应该在条件允许的情况下尽可能符合20条推荐规则。
      - ◌ MISRA-C:2004 将其141条规则分为21个类别，每一条规则对应一条编程准则。

3.3 软件的代码检查

- **MISRA C 编程规范**
  - MISRA-C:2004
    - ✧ MISRA-C 规范不仅在汽车工业普及使用，也同时影响到嵌入式开发的其他方向。MISRA 得到广泛应用的工业控制领域包括铁路、航空航天、军事工程、医疗等重要领域。
    - ✧ 符合 MISRA-C 编程规范的 C 程序，不仅需要程序员按照规范编程，编译器也需要对所编译的代码进行规则检查。



3.3 软件的代码检查

Contents

Contents (continued)

3.3 软件的代码检查

- **MISRA C 编程规范**
  - MISRA-C:2004 的21个类别

分类	强制规则	推荐规则	分类	强制规则	推荐规则
开发环境	4	1	表达式	9	4
语言外延	3	1	控制表达式	6	1
注释	5	1	控制流	10	0
字符集	2	0	Switch 语句	5	0
标识符	4	3	函数	9	1
类型	4	1	指针和数组	5	1
常量	1	0	结构体和联合体	4	0
声明和定义	12	0	预处理命令	13	4
初始化	3	0	标准库	12	0
算术类型转换	6	0	运行失败	1	0
指针类型转换	3	2			

3.3 软件的代码检查

- **MISRA C 编程规范**
  - MISRA-C:2012 (March 2013) / 第三代 MISRA-C (MC3)
    - ✧ Support is provided for C99 as well as C90.
    - ✧ MISRA C3 includes 16 directives & 143 rules. Compliance (遵循) with a rule can be determined solely from analysis of the source code. Compliance with a directive may be open to some measure of interpretation or may, for example, require reference to design or requirements documents.
    - ✧ Each directive or rule is classified as either Mandatory, Required or Advisory. It is typically defined with sections devoted to Amplification, Rationale, Exceptions and Examples.
    - ✧ Each rule is classified as either a Single Translation Unit rule or a System rule, reflecting the scope of analysis which must be undertaken in order to claim compliance.
    - ✧ The concept of decidability (可判定性) has been introduced in order to expose the unavoidable uncertainty which exists in claiming compliance with certain rules.

3.3 软件的代码检查

- **代码的自动分析**
  - 代码自动分析利用代码分析工具，对照需求和设计文档以及程序编码进行检查，包括：
    - ✧ 程序逻辑和编码检查
    - ✧ 一致性检查
    - ✧ 接口分析
    - ✧ I/O 规格说明分析
    - ✧ 数据流
    - ✧ 变量类型检查
    - ✧ 模块分析等
  - 代码分析的结果可以为动态测试和其他测试做必要的准备。

3.3 软件的代码检查

- **代码的自动分析**
  - 代码自动分析的主要内容：
    - 生成引用表
      - 标号交叉引用表、变量交叉引用表、子程序、宏和函数表、等价表、常数表等
    - 进行程序错误分析
      - 变量类型和单位分析、引用分析以及表达式分析
    - 进行接口分析
      - 检查形参与实参在类型、数量、维数、顺序、使用上的一致性
      - 检查全局变量和公共数据区在使用上的一致性

3.3 软件的代码检查

- **代码的结构分析**
  - 代码的结构形式是白盒测试的主要依据。
    - ✧ 研究表明程序员38%的时间花费在理解软件系统结构上。
    - ✧ 代码以文本格式被写入多重文件中，审查人员在阅读理解和建立代码模块之间的联系上存在困难。
  - 代码结构分析的内容：
    - ✧ 测试者通过使用测试工具分析程序源代码的系统结构、数据结构以及内部控制逻辑等内部结构，生成函数调用关系图、模块控制流图、模块数据流图、内部文件调用关系图、子程序表、宏和函数参数表等各类图形图表。
      - ◌ 这些图表被用于检查软件的缺陷或错误。
  - 代码结构分析的结果可以清晰地标识整个软件系统的组成结构，使其便于阅读和理解。

3.3 软件的代码检查

- **代码的结构分析**
  - 程序流程图
    - ✧ 以描述程序控制的流动情况为目的，表示程序中的操作顺序。
      - ◌ 指明实际处理操作的处理符号，它包括根据逻辑条件确定要执行路径的符号
      - ◌ 指明控制流的流线符号
      - ◌ 便于读、写程序流程图的特殊符号
  - 调用关系图
    - ✧ 函数调用关系图或程序调用关系图 (被调用)
      - ◌ 调用关系图是对源程度中函数调用关系的一种静态描述。
      - ◌ 调用关系图中，节点表示函数，边表示函数之间的调用关系。
      - ◌ 函数调用关系图在软件工作领域有广泛的应用，如编译优化，过程间数据流分析，回归测试，程序理解等。

3.3 软件的代码检查

- **代码的结构分析**
  - 数据流图
    - ✧ 在单元测试中，数据仅仅在一个模块或者一个函数中流动。
      - ◌ 数据流的通路往往涉及多个集成模块，甚至整个软件
      - ◌ 数据流分析是必要的，尽管它非常耗时。
    - ✧ 数据流分析技术最早被用于编译优化，目前在程序测试、程序理解、程序验证、程序调试以及程序分片等许多领域，数据流都有着广泛的应用。
      - ◌ 例如：查找如引用未定义变量等程序错误；查找对以前未曾使用的变量再次赋值等数据流异常的情况。
        - 找出这些错误是很重要的，因为这常常是常见程序错误的表现形式。
  - 代码的结构分析不能确认运行时序图。

3.3 软件的代码检查

- **代码安全性检查**
  - 代码安全性
    - ✧ 代码安全性指代码运行或被调用时产生错误的容易程度。
    - ✧ 例如： C++ 规定了严格的语法，然而又有其灵活性。
      - ◌ 这种灵活性增加了程序的不可预见性，有可能导致故障的发生，致使代码的安全性变差。
      - ◌ 指针的指向发生错误，则直接导致结果错误。
      - ◌ 指针的指向越界，可能导致缓冲区溢出，被病毒利用。
      - ◌ C++ 提供的很多函数没有对参数范围进行限制和检查，这也很容易导致错误的发生。
    - ✧ 代码安全性检查或静态错误分析主要用于确定在源程序中是否有某类错误或 “危险/不安全” 的结构。
      - ◌ 一般可借助工具实施代码的安全性检查。

3.3 软件的代码检查

- **代码安全性检查**
  - 代码安全性检查的主要方法
    - ✧ 变量类型和单位的检查
    - ✧ 变量引用的检查
    - ✧ 表达式运算的检查
    - ✧ 接口分析

3.3 软件的代码检查

- **代码安全性检查**
  - 代码安全性检查关注的错误
    - ✧ 不安全的存储分配
    - ✧ 内存泄漏
    - ✧ 指针引用
    - ✧ 约束检查
    - ✧ 变量未初始化
    - ✧ 错误逻辑结构
    - ✧ 其他错误或异常
      - ◌ 缓冲区溢出、非法类型转换、非法的算数运算 (如除以零错误、负数开方)、整数和浮点数的上溢出/下溢出、多线程对未保护数据的访问冲突等 。

Thank you!