SOFTWARE TESTING

Chapter 3

Software Static Testing

School of Data & Computer Science
Sun Yat-sen University

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
  - 代码检查的内容和基本方法
  - 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
- 3.3 软件复杂性分析
- 3.4 软件质量度量
- 3.5 软件静态分析工具

软件代码检查概述

- 软件代码检查 (Code Review) 通常以小组为单位阅读程序源代码，是一系列评审规程和错误检查技术的集合，也是发现软件缺陷最有效的方法之一。
  - 代码检查研究分析程序结构和代码，而不是实际执行代码，一般采用静态测试的方法，包括：
    - 桌面检查、代码走查、代码审查和技术评审。
  - 代码检查的目的是发现错误，而非解决错误 (to test, not debug)。
  - 高级别的代码检查通常将注意力集中在审核代码的质量，如可读性、可维护性，以及程序的逻辑和对需求和设计的实现。
- 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。
    - 大部分的程序性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。

软件代码检查概述

- 软件代码检查的组织约束
  - 评审人员的程序语言能力和经验可以胜任
  - 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；编程规则的执行情况；程序流图和调用图的分析。
- 代码的度量
  - 代码复杂性度量
    - *Halstead* 复杂度、*McCabe* 复杂度、嵌套级别 (最大/平均) 等。
  - 代码规格度量
    - 行数、语句数、注释数、声明数等。

软件代码检查概述

- 软件代码检查内容
  - 软件代码检查的11项内容：
    - 完整性检查
    - 一致性检查
    - 正确性检查
    - 可修改性检查
    - 可预测性检查
    - 健壮性检查
    - 可理解性检查
    - 可验证性检查
    - 结构性检查
    - 可追溯性检查
    - 代码标准符合性检查

软件代码检查概述

软件代码检查内容 (续)

(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中的定义。
- 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。

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。

软件代码检查概述

软件代码检查内容 (续)

(7) 可理解性检查

- 注释是否足够清晰地描述每个子程序，是否删除了没用的代码注释。
- 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。

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建立相互对应。
- 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。

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。

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、代码评审和代码审查等类型
  - 桌面检查 (Desk Checking/Self Inspection) 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进行集体讨论或举行会议，找出程序的错误或遗漏，但不解决问题。
  - 代码审查 (Code Inspections) 是最正式的检查类型。
- 可选择的方法
  - 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 (有必要时) 举行代码审查会议。

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文档对源程序代码进行分析、检验，以期发现程序错误的过程。
      - 桌查用作自我检查以期发现一些较明显的错误和漏洞。
  - 桌查的主要内容：
    - 代码和设计是否一致；
    - 代码是否遵循语言标准、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。

软件代码检查方法

- 桌面检查 (续)
  - 桌查的工作细分：
    - 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。

- 软件代码检查方法
  - 桌面检查 (续)
    - 桌面检查的缺陷：
      - 检查效率不高；
      - 程序员没有发现自己的程序错误的心理欲望；
      - 程序员发现本身的一些习惯性错误比较困难；
      - 难以 (或无法) 纠正对程序功能的理解错误。

软件代码检查方法

- 代码走查 Walkthroughs
  - 代码走查是一类非正式的软件代码评审方法。
  - 代码走查的组织方法：
    - 编写代码的程序员 (作者) 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。
    - 作者只是小组的成员之一，大部分的检查过程由其它成员主导，符合 An individual is usually ineffective in testing his or her own program 的测试原理。
    - 测试人员应是具有经验或精通程序设计的程序设计人员。
    - 在这个小组中至少有一位资深程序员。
    - 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。

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；
    - 会议期间由作者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，交给作者。

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

- 软件代码检查方法
  - 代码评审 Reviews
    - 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

软件代码检查方法

- 代码评审 (续)
  - 代码评审的组织方法：
    - 评审小组由组长、资深程序员、作者与专职测试人员等组成；作者不能担任组长。
    - 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。
    - 采用作者讲解、评审员提问结合检查表的方式审查代码，寻找可能存在的问题和失误。
    - 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。

软件代码检查方法

- 代码审查 Code Inspections
  - 代码审查是最正式的评审类型，具有高度的组织化。
  - 代码审查由开发组、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。
  - 代码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 (与代码走查和代码评审不同)，这就要求表述者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。
  - 代码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。

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing.

职务	职责	备注
协调人	1. 为代码检查分发资料、安排进程； 2. 在代码检查中起主导作用； 3. 记录发现的所有错误； 4. 确保所有错误随后得到改正。	非被测程序作者的熟练程序员，不需要特别了解程序细节。
程序编码人员	朗读程序并逐行讲解	被测程序作者，也是被检查者。
其他成员	提出质疑，并讨论	被测程序的设计人员，非被测程序的编码人员。
测试专家	提出质疑，并讨论	具备软件测试经验，熟悉大部分的常见编码错误。

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - 检查议程与注意事项
    - Step1：会议之前，协调人分发程序清单和设计规范给小组成员，所有成员在检查之前进行熟悉。
    - Step2：会议中，程序编码人员逐条语句讲述逻辑结构，过程中其他成员结合常见的编码错误列表分析程序，提出质疑、讨论，确认是否存在错误，协调人记录讨论过程以及错误清单。
    - Step3：会议结束后，协调人将错误清单提交给程序编码人员进行确认，若发现错误过多，或者某个错误涉及对程序做根本性的改动，协调人需要在错误修正后，重新安排会议再次检查。

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - 时间控制
    - 最佳时间为90-120分钟，审查速度大约每小时150程序行。因此建议每次会议处理一个或几个模块或子程序。
  - 错误列表 (Inspection Error Checklist)：

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **Data Reference**
      1. Unset variable used?
      2. Subscripts within bounds?
      3. Non integer subscripts?
      4. Dangling (悬空) references?
      5. Correct attributes when aliasing?
      6. Record and structure attributes match?
      7. Computing addresses of bit strings? Passing bit-string arguments?
      8. Based storage attributes correct?
      9. Structure definitions match across procedures?
      10. Off-by-one errors in indexing or subscripting operations?
      11. Are inheritance requirements met?

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **数据引用错误**
      1. 是否存在引用的变量未赋值或未初始化？
      2. 对于所有的数组引用，是否每一个下标的值都在相应维数所规定的界限之内？
      3. 对于所有的数组引用，是否每一个下标的值都是整数？
      4. 对于所有通过指针的变量引用，当前引用的内存单元是否已经分配？(虚调用)
      5. 如果一个内存区域具有不同属性的别名，当通过别名进行引用时，内存区域中的数据值是否具有正确的属性？
      6. 变量值的类型或属性是否与编译器所预期的一致？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **数据引用错误 (续)**

7. 在使用的计算机上，当内存分配的单元数小于内存可寻址的单元数时，是否存在显示或隐含的寻址错误？
8. 当使用指针或引用变量时，被引用的内存的属性是否与编译器所预期的一致？
9. 假如一个数据结构在多个过程或子过程中被引用，那么每个过程或子过程对该结构的定义是否都相同？
10. 如果字符串有索引，进行索引操作或对数组进行下标引用时字符串是否有“仅差一个”的错误？
11. 对于面向对象的语言，是否所有的继承需求都在实现类中得到了满足？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **Data Declaration**
      1. All variables declared?
      2. Default attributes understood?
      3. Arrays and strings initialized properly?
      4. Correct lengths, types, and storage classes assigned?
      5. Initialization consistent with storage class?
      6. Any variables with similar names?

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **数据声明错误**
      1. 是否所有的变量都进行了声明？
      2. 如果变量所有的属性在声明中没有明确说明，那么默认的属性能否被正确理解？
      3. 数组和串是否正确初始化？
      4. 是否每个变量都被赋予了正确的长度和数据类型？
      5. 变量的初始化是否与其存储空间的数据类型一致？
      6. 是否存在名称相似的变量？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **Computation**
      1. Computations on non-arithmetic variables?
      2. Mixed-mode computations?
      3. Computations on variables of different lengths?
      4. Target size less than size of assigned value?
      5. Intermediate result overflow or underflow?
      6. Division by zero?
      7. Base-2 inaccuracies?
      8. Variable's value outside of meaningful range?
      9. Operator precedence understood?
      10. Integer divisions correct?

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **运算错误**
      1. 是否存在非算术变量间 (如非运算符) 的运算？
      2. 是否存在混合模式的运算？
      3. 是否存在相同数据结构、不同字长变量间的运算？
      4. 赋值语句的目标变量的数据类型 (长度) 是否小于右边表达式的数据类型或结果？
      5. 在表达式的运算中是否存在表达式向上或向下溢出的情况？
      6. 除法运算中除数是否为0？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **运算错误**
      7. 如果计算机表达变量的基本方式是基于二进制的，那么运算结果是否不精确？例如10*0.1，在二进制计算机中，很少会等于1.0.
      8. 在特定场合，变量的值是否超过了有意义的范围？
      9. 对于包含一个以上的操作符的表达式，赋值顺序和操作符的优先顺序是否正确？
      10. 整数的运算是否有使用不当的情况，尤其是除法？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **Comparison**
      1. Comparisons between inconsistent variables?
      2. Mixed-mode comparisons?
      3. Comparison relationships correct?
      4. Boolean expressions correct?
      5. Comparison and Boolean expressions mixed?
      6. Comparisons of base-2 fractional values?
      7. Operator precedence understood?
      8. Compiler evaluation of Boolean expressions understood?

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **比较错误**
      1. 是否有不同数据类型的变量之间的比较。例如将字符串与地址、日期或数字相比较？
      2. 是否有混合模式的比较运算，或不同长度的变量间的比较运算？
      3. 比较运算符是否正确？程序员经常混淆至多、至少、大于、不小于、小于和等于等比较关系。
      4. 每个布尔表达式所叙述的内容是否都正确？例如涉及与、或、非表达式时。
      5. 布尔运算符的操作数是否为布尔类型？比较运算符和布尔运算符是否错误地混淆在一起？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **比较错误**
      6. 在二进制的计算机中，是否有用二进制表示的小数或浮点数的比较运算？
      7. 对于那些包含一个以上的布尔运算符的表达式，赋值顺序以及运算符的优先顺序是否正确？
      8. 编译器计算布尔表达式的方式是否对程序产生影响？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **Control-Flow**
      1. Multiway branches exceeded?
      2. Will each loop terminate?
      3. Will program terminate?
      4. Any loop bypasses because of entry conditions?
      5. Are possible loop fall-throughs correct?
      6. Off-by-one iteration errors?
      7. DO/END statements match?
      8. Any nonexhaustive decisions?
      9. Any textual or grammatical errors in output information?

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **控制流程错误**
      1. 如果程序中包含多路分支，比如有计算 GO TO 语句，指示变量的值是否会超过可能的分支数量？
         例如：GO TO (200,300,400), i
      2. 是否所有的循环最终都终止了？
      3. 程序、模块或子程序是否最终都停止了？
      4. 是否存在入口条件不能满足而从未执行过的循环体？
      5. 如果循环同时由迭代次数和一个布尔条件所控制，循环是否可能越界？
      6. 是否存在“仅差一个”的错误，如迭代数量恰恰是多一次或者少一次？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **控制流程错误**
      7. 如果编程语言中有语句块或代码块的概念 (例如 do……while 或 {...})，是否每一组语句都有一个明确的 while 语句，并且具有相应的 do 语句？或者是否每一个左括号对应一个右括号？
      8. 是否存在不能穷尽的判断？
      9. 输出信息中是否存在文字或语法错误？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **Interfaces**
      1. Number of input parameters equal to number of arguments?
      2. Parameter and argument attributes match?
      3. Parameter and argument units system match?
      4. Number of arguments transmitted to called modules equal to number of parameters?
      5. Attributes of arguments transmitted to called modules equal to attributes of parameters?
      6. Units system of arguments transmitted to called modules equal to units system of parameters?

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **Interfaces**
      7. Number, attributes, and order of arguments to built-in functions correct?
      8. Any references to parameters not associated with current point of entry?
      9. Input-only arguments altered?
      10. Global variable definitions consistent across modules?
      11. Constants passed as arguments?

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **接口错误**
      1. 被调用模块接收到的形参数量是否等于调用模块发送的实参数量？顺序是否正确？
      2. 实参的属性 (如数据类型和大小) 是否与相应形参的属性相匹配？
      3. 实参的量纲是否与对应形参的量纲相匹配？
      4. 此模块传递给彼模块的实参数量，是否等于彼模块期望的形参数量？
      5. 此模块传递给彼模块的实参的属性，是否与彼模块相应形参的属性相匹配？
      6. 此模块传递给彼模块的实参的量纲，是否与彼模块相应形参的量纲相匹配？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **接口错误**
      7. 如果调用了内置函数，实参的数量、属性、顺序是否正确？
      8. 如果某个模块或类有多个入口点，是否引用了与当前入口点无关的形参？
      9. 是否有子程序改变了某个原本仅为输入值的形参？
      10. 如果存在全局变量，在所有引用它们的模块中，它们的定义和属性是否相同？
      11. 常数是否以实参形式传递过？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **Input/Output**
      1. File attributes correct?
      2. OPEN statements correct?
      3. Format specification matches I/O statement?
      4. Buffer size matches record size?
      5. Files opened before use?
      6. Files closed after use?
      7. End-of-file conditions handled?
      8. I/O errors handled?

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **输入/输出错误**
      1. 如果对文件明确声明过，其属性是否正确？
      2. 打开文件的语句中各项属性的设置是否正确？
      3. I\O 语句中的格式规范是否与信息相吻合？
      4. 是否有足够的可用内存空间，来保留程序将读取的文件？
      5. 是否所有的文件在使用之前都打开了？
      6. 是否所有的文件在使用之后都关闭了？
      7. 是否判断文件结束的条件，并正确处理？
      8. 对 I/O 出错情况处理是否正确？
      9. 任何打印或显示的文本信息中是否存在拼写或语法错误？

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **Other Checks**
      1. Any unreferenced variables in cross-reference listing?
      2. Attribute list what was expected?
      3. Any warning or informational messages?
      4. Input checked for validity?
      5. Missing function?

软件代码检查方法

- 代码评审：*G. J. Myers* – The Art of Software Testing
  - Inspection Error Checklist
    - **其他检查**
      1. 如果编译器建立了一个标识符交叉引用列表，那么对该列表进行检查，查看是否有变量从未引用过，或仅被引用过一次。
      2. 如果编译器建立了一个属性列表，那么对每个变量的属性进行检查，确保没有赋予过不希望的默认属性值。
      3. 即使程序编译通过了，也要逐一认真检查编译器提供的“警告”或“提示”信息。
      4. 程序或模块是否具有足够的鲁棒性？即是否对输入的合法性进行了检查？
      5. 程序是否遗漏了某个功能？

软件代码检查方法

- 匿名评审 Anonymous Rating/Peer Rating
  - 依据程序整体质量、可维护性、可扩展性、易用性和清晰性对匿名程序进行评价，目的不是为了发现程序错误，而是在自我评价和相互学习中提高代码质量。
  - Step1：选出一名程序员作为评审过程的管理员，然后由管理员挑选6-20名具备相似技术背景的参与者，组成评审团。
  - Step2：每个评审员提供两个自己编写的程序，其中至少有一个是代表自身能力的好作品。所有作品汇集成评审资料。
  - Step3：将评审作品随机分发给评审员 (非作者)，每名评审员阅读分配到的4个程序，评出两个“最好”，两个“较差”，填写评价表，全过程保密。每个程序的平均评审时间在30分钟左右。

软件代码检查方法

- 匿名评审 Anonymous Rating/Peer Rating
  - Step4：评审员对自己分配到的4个程序进行相对质量分级，分值为1-10分，并给出总的评价和建议的改进意见。评分参考如下标准：
    - 程序是否易于理解？
    - 高层次的设计是否可见且合理？
    - 低层次的设计是否可见且合理？
    - 修改此程序对评审者而言是否容易？
    - 评审者是否会以编写出改程序而骄傲？
  - Step5：会议结束，每个参与者会收到自己提供的两个程序的匿名评价表，以及一个带统计结果的总结报告，说明在所有程序中参与者程序的整体和具体得分情况，以及该参与人对他人程序的评价与其他评审人对同一程序打分的比较分析情况。

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的最直接、最有效的手段 。
    - 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
    - 编程人员必须严格遵守规则级的编码规范。
    - 编程人员应该尽量遵守建议级的编码规范。

- 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，例如：
      - 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；
      - 双目运算符两边各留空格；
      - 单目运算符 "*"、"&" 和变量之间不留空格。

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。

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。
    - 按规范采用匈牙利法、驼峰法、Pascal 法、下划线法或其它规定形式。
  - 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。

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 (switch 语句除外) 、continue 语句；
  - 字符串的赋值应采用如 _T(“hello world”) 模式；
    - ANSI 环境下为 ASCII 码串；Unicode 环境下将自动解释为双字节的 Unicode 编码字符串。
  - 避免对浮点数值类型做精确比较；
  - new 和 delete 要成对出现 (局部)；

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

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。

软件编程规范检查

- 类
  - 定义缺省构造函数：为每一个类显式定义缺省构造函数；
  - 定义拷贝构造函数：
    - 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。
  - 定义析构函数：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，那么应在基类中将该函数声明为虚函数。

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。

- 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。

- 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。

- 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 风格类型转换符。
    - 将不再使用的代码删除。

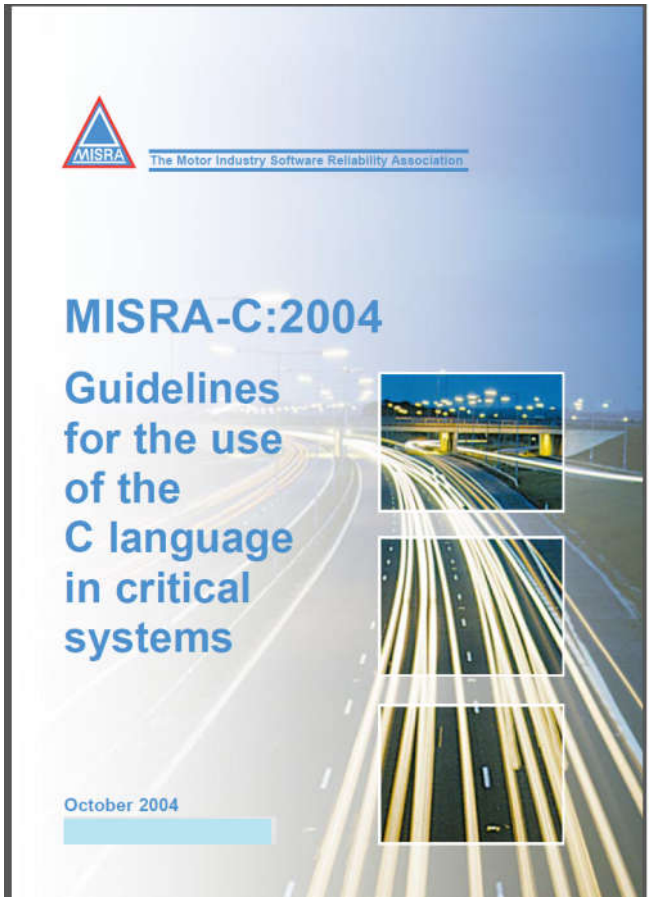
MISRA C 编程规范

- MISRA
  - MISRA, Motor Industry Software Reliability Association，英国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。

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。



MISRA C 编程规范

MISRA-C:2004

Contents

Contents (continued)

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			

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 (指令) and 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.

- 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。

- 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。

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。

- 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。

- 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。

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。
    - C++ 语法的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。
    - 代码的安全性检查一般可借助工具实施。

- 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

- 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 (Memory Leak)
    - 指针引用
    - 约束检查
    - 变量未初始化
    - 逻辑结构错误
    - 其他错误或异常
      - 如：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!