旋极科技
WATERTEK

嵌入式软件测试技术

（基础篇）

皮永辉

2012年8月

内容提要

- 嵌入式软件测试基础
  - 所需预备知识
  - 关于软件测试
  - 嵌入式软件测试的特点
    - 嵌入式系统与嵌入式软件
    - 嵌入式软件测试的特点
  - 嵌入式软件测试的基本方法
    - 交叉测试
    - 可用于交叉测试的调试方式
    - 基于需求的测试
    - 源代码分析

嵌入式软件测试基础：

预备知识

- 本课程所需基础知识
  - C语言
    - C程序设计
    - 开发环境使用
  - 嵌入式软件开发基础
    - 嵌入式系统的组成及特点
    - 嵌入式软件的开发方式
    - 嵌入式开发环境使用
  - 软件测试基础知识
    - 软件测试基本概念
    - 软件测试基本技术

关于软件测试

- 回顾软件测试
  - 软件测试基本概念
    - 定义、原则、过程、分类、模型
    - 测试用例、插装、打桩
    - 静态测试、动态测试
    - 白盒测试、黑盒测试
    - 策略与管理
  - 通过软件测试，我们要解决什么问题？
    - 软件功能——对不对？
    - 代码质量——高不高？
    - 测试过程——得唔得？

关于软件测试

- 软件测试的三个基本问题
  - 软件功能——对不对？
    - 需求准确
    - 功能正确、完整
    - 性能可靠
  - 代码质量——高不高？
    - 规范（可维护、易理解）
    - 少缺陷
    - 健壮（容错、结构化）
  - 测试过程——得唔得？
    - 有效
    - 有利
    - 有序

关于软件测试

❖ 软件测试的基本策略

“两条腿走路”

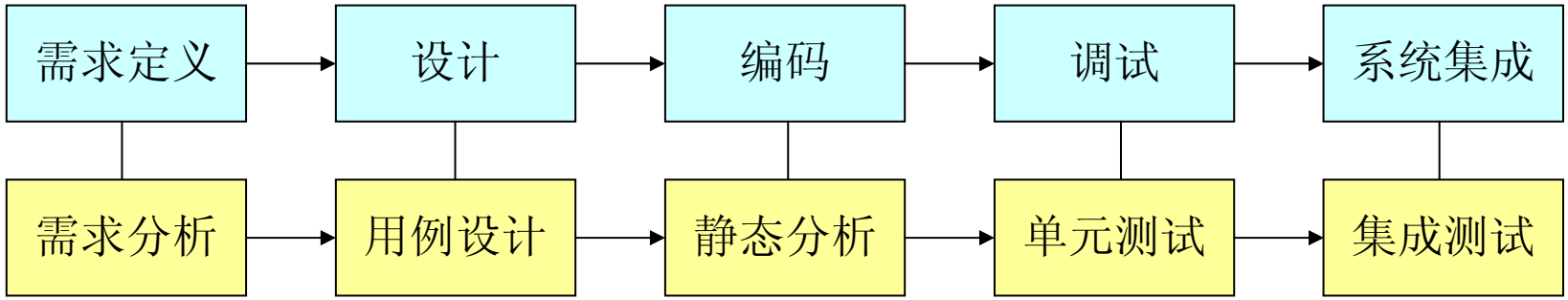
- ❑ 技术是基础
  - ➢ 静态测试
  - ➢ 动态测试
- ❑ 管理是保障
  - ➢ 计划
  - ➢ 策略
  - ➢ 资源配置
  - ➢ 过程管理
  - ➢ 缺陷追踪

关于软件测试

❖ 软件测试的基本策略

现代软件工程的测试策略

- ❑ 测试贯穿于开发的全过程
- ❑ 不形成专门的“测试阶段”



关于软件测试

- 软件测试的基本策略

  测试从需求开始

  - 需求定义是软件生命的开始
  - 准确的需求是软件测试的前提

  质量从代码抓起

  - 软件质量的基础是“优质”的代码
  - 如何获得高质量的代码
    - “预防”
    - “治病”
    - “强身”

关于软件测试

软件测试的基本策略

先静后动——先静态分析，再动态测试

- 多数缺陷源于编程语言使用不当
  - 静态分析能有效地发现之
- 静态分析能大大减轻后续测试的工作量，明显提升动态测试的效果
- 静态分析能及早地发现问题，改正问题
- 静态分析实施容易，操作简单

由小到大——先单元测试，再集成测试

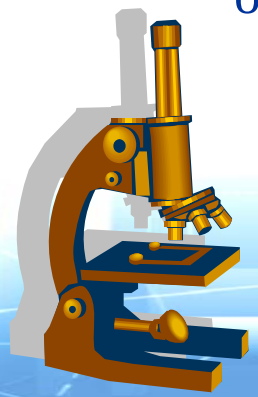
- 单元测试完成了，集成测试才有意义

关于软件测试

- 软件测试的基本策略

测试工具选择

- 工欲善其事，必先利其器
- 通常工具是必需的，但不是万能的。不要期望“一招鲜，吃遍天”
- 根据自己的需要和工具的特长进行选择
  - 明确自己要做什么？期望达到怎样的目标？
  - 测试工具的关键特性？
  - 其它因素
    - 人员、资源、价格等



关于软件测试

软件测试的基本策略

加强软件测试的管理

测试过程管理

- 计划、进度
- 测试文档管理
- 需求的可追溯
- 节点监控
- 资源配置

缺陷追踪与管理

- 测试过程中，会有各式各样的错误或缺陷（bug/defect）出现。需要建立有效的机制：报告、存储、分配、修复、追踪...
- 避免混乱、丢失、重复

变更管理——软件配置管理

嵌入式软件测试基础：

嵌入式软件测试的特点

❖嵌入式系统与嵌入式软件

❑什么是嵌入式系统

- ➢以应用为中心、以计算机技术为基础、适应特殊环境要求的专用计算机系统
- ➢嵌入式系统通常都是实时系统，即有一定时间约束的计算机系统

❑嵌入式系统的组成

- ➢嵌入式微处理器、外围硬件
- ➢嵌入式操作系统、应用软件

❑什么是嵌入式软件

- ➢嵌入式系统或产品中的软件
- ➢“非嵌入式”软件：通用计算机软件

嵌入式系统的典型特征：

硬件
软件
专用

嵌入式软件测试基础：

嵌入式软件测试的特点

❖嵌入式系统与嵌入式软件

❑嵌入式软件的特点

- ➢大部分软件用高级语言（C、C++等）编写
- ➢依赖于特定硬件环境，无统一的平台
- ➢与硬件密切相关，交互工作
- ➢实时性
  - o 实时约束
  - o 实时控制
- ➢交叉式开发
  - o 需要专门的环境及工具
  - o 目标软件与开发环境运行在不同的平台
- ➢资源受限

嵌入式软件测试基础：

嵌入式软件测试的特点

嵌入式软件测试的难点

- 实时性——要求测试工具准确测试软件性能
- 资源有限——要求对被测软件不能附加太多代码冗余
- 软硬结合——软件与硬件紧密相关，硬件可能成为测试的瓶颈
- 交叉开发——嵌入式软件与测试工具运行在不同的平台，载入目标系统执行需要特定的硬件测试工具配套
- 多样性——没有统一的硬件平台，需要“专款专用”
- 实验测试环境与真实运行环境存在差异
- 测试工具与目标系统的连接方式影响测试的可靠性
- 汇编语言难以测试
- 成本较高

嵌入式软件测试基础：

嵌入式软件测试的特点

嵌入式软件测试的基本思路

- 软件测试发展至今已形成了较为完整的理论、技术和策略，虽然它们大都针对计算机软件，但幸运的是其基本原理和典型方法同样适用于嵌入式软件。

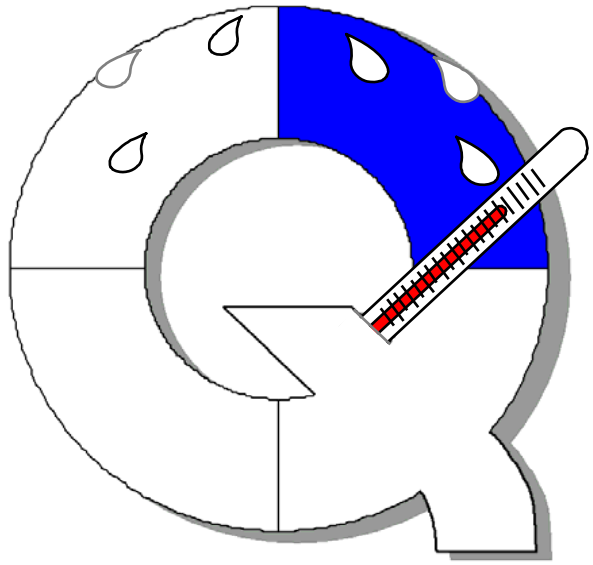
- 基于两种软件的共性——同为高级语言，我们首先利用通用计算机软件的测试技术，即“拿来主义”

- 然后从嵌入式软件的特点出发，采取一些有针对性的方法，比如“交叉测试”。

嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 嵌入式软件测试的基本方法
  - 动态测试方法
    - 交叉测试
    - 基于需求的测试
  - 静态测试方法
    - 源代码分析
    - 结构化测试
  - 系统测试方法
    - 真实环境下系统测试
    - 全数字模拟测试
    - 实时在线测试
    - 故障注入测试



嵌入式软件测试的基本方法

交叉测试（Host/Target测试）

- 利用交叉开发环境的测试方法
  - 测试工具需要支持目标环境
- 利用高级语言的特性，使用“常规的”软件测试方法
- 测试是在“主机”和“目标系统”中分别进行的：
  - 与硬件无关的大部分测试在“Host”上完成
  - 与硬件密切相关的小部分在“Target”上完成
  - 再根据需要，将“Host”上的测试在“Target”上验证

嵌入式软件测试基础：

嵌入式软件测试的基本方法

交叉测试的特点

- 将大部分工作转移到PC平台上，在硬件环境未建好或调试工具缺乏时就可以开展
- 适用于高级语言，如C，C++
- 主要用于动态测试，如单元测试（功能测试）
  - 测试用例设计是关键
- 操作方便，测试成本较低。
- 实时性受调试环境的制约
- 目标环境中测试时要占用一定的目标资源
- 注意目标环境和主机环境的差异
  - 目标编译器的影响
  - 内存资源

嵌入式软件测试的基本方法

为什么不把所有测试都放在目标上进行

- 在Target上测试软件，可能会造成与开发者争夺目标平台或使用时间。要避免这种矛盾只有提供更多的目标平台。
- 目标平台可能还不可行，或者主机与目标的连接不方便。
- 比起主机平台环境，目标平台通常是不精密的和不方便的。
- 成本问题。提供给开发者的目标平台和开发环境通常是很昂贵的。
- 开发和测试工作可能会妨碍目标上已经存在持续的应用。

嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 如何开展交叉测试
  - 选用带有支持目标环境的软件测试工具
  - 确定哪些模块与硬件无关，哪些与硬件相关
  - 配置相应的调试环境和目标环境
  - 设计测试用例
  - 分别进行Host和Target测试

嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 交叉测试的条件
  - 测试工具要支持目标系统
  - Target测试需要合适的开发环境配合
    - 编译环境（编译器/链接器）
    - 调试环境（仿真器、调试器）
- 可用于交叉测试的调试环境

嵌入式软件的调试的基本方法：

- 模拟器（Simulator）
- 调试器（Debugger）
- 仿真器（Emulator）

嵌入式软件测试基础：

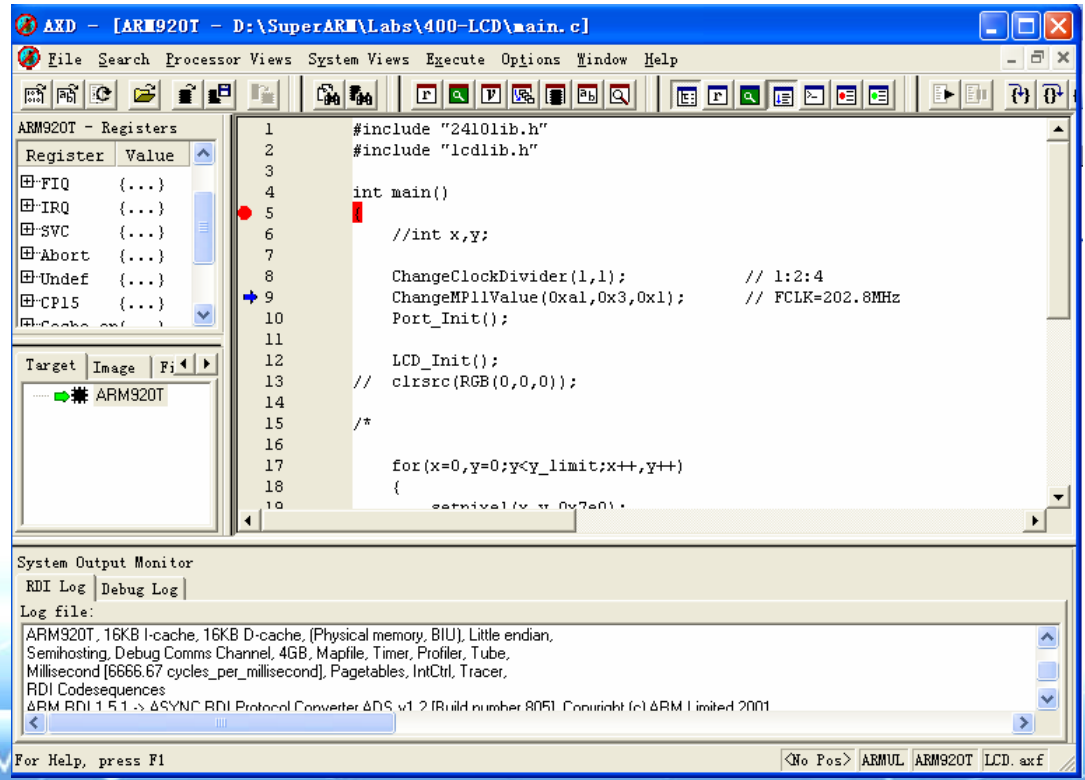
嵌入式软件测试的基本方法

- 模拟器（Simulator）

大多数集成开发环境（IDE）都具有调试功能，“模拟器”就是其中的一种。模拟器和待调试的嵌入式软件都在主机上运行，由主机提供一个模拟的目标运行环境，可以进行“指令集”调试，包括算法和流程等。

- 纯软件，简单方便，无需目标板，成本低
- 功能有限，无法体现硬件特性

大多数开发环境都提供Simulator功能,如ARM开发环境ADS1.2中的AXD(ARMulator)



嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 调试器（Debugger）

  即在线调试器，有两种形式

  - Monitor调试
  - JTAG调试

- Monitor调试器
  - 主机和目标平台通过某种接口（如串口、网口、USB等）连接，主机上运行调试器，代码“下载”到目标板上运行。
  - 调试之前要在Host和Target之间建立起通信联系，目标板上需要先期烧录“监控程序（Monitor或bootloader）”
  - 纯软件，价格较低，简单，有一定的硬件调试能力
  - 制作**Monitor**可能成为瓶颈（需要经验，且要求目标板工作正常），调试功能、实时性有限。

例如，**Linux**和**WinCE**开发环境就提供这种调试方式

嵌入式软件测试基础：

嵌入式软件测试的基本方法

- JTAG调试器
  - 基于集成在芯片上的调试接口（调试和跟踪逻辑）的一种调试方式。常见的调试接口有JTAG（边界扫描）和BDM（背景调试模式）等
  - 通过一个硬件调试体（俗称JTAG仿真器或调试器），连接主机和目标系统。
    - 主机端：串口、USB、网口等
    - 目标板：调试接口（JTAG、BDM等）
  - 调试软件运行在主机上，待调试代码则由JTAG仿真器控制，通过调试接口下载到目标板上运行。

嵌入式软件测试的基本方法

- JTAG调试器的特点
  - 软硬件结合，试用方便，无须制作Monitor，软硬件均可调试。是当今最常用的嵌入式调试方式。
  - 需要目标板，且目标板工作基本正常。适用于有调试接口的微处理器
  - 许多流行的嵌入式微处理器都支持这种方式。常见的JTAG调试器有：
    - ARM系列：Trace-ICP、Probe-ICE、MAJIC、Hitex等
    - MIPS系列：MAJIC
    - PowerPC系列：iSystem

嵌入式软件测试基础：
嵌入式软件测试的基本方法

❖仿真器（Emulator）

即传统意义上的“全仿真调试”方式。

- ❑仿真器具备目标系统微处理器的全部功能，并配有必要的存储器、外设等硬件资源。使用时仿真器取代目标板微处理器，开发者就可以全盘控制和调试目标系统
- ❑仿真器与目标板通过“仿真头”连接，与主机有串口、并口、网口或USB口等连接方式
- ❑仿真器一般自成体系，调试时既可以连接目标板，也可以不连接目标板（Stand alone）。
- ❑功能强大，实时性强，软硬件均可调试
- ❑价格昂贵，通用性差。一般用在低速、低位数处理器中

嵌入式软件测试基础：

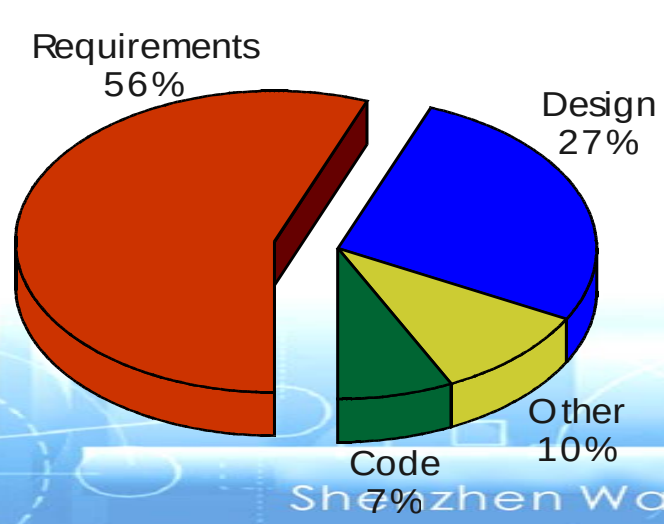
嵌入式软件测试的基本方法

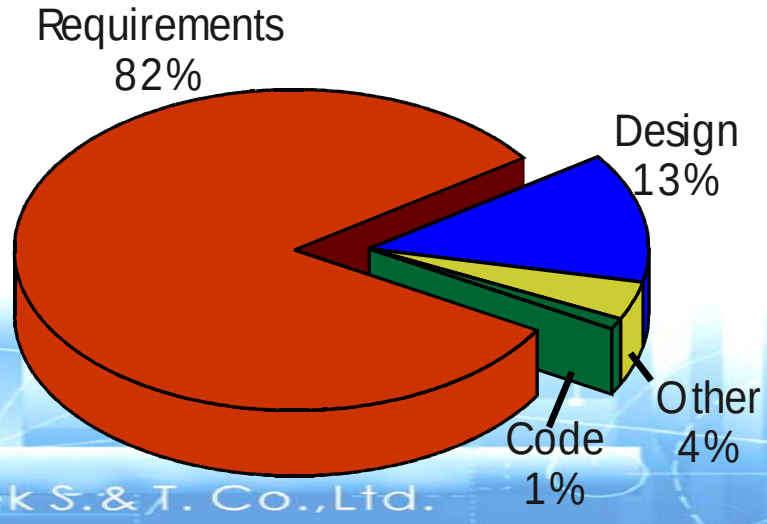
- 调试方式对软件测试的影响
  - 模拟器（Simulator）
    - 最方便，成本低，使用资源无限制
    - 实时性、硬件特性差
  - 调试器（debugger）
    - 最常用，成本较低，使用资源受目标板硬件限制
    - 硬件特性较好，对monitor调试器来说，测试结果输出可能受限
  - 仿真器（emulator）
    - 最真实，成本较高，使用资源受目标板或仿真器限制
    - 实时性、硬件特性好

嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 基于需求的测试
  - 什么是需求（Requirement）？
    - 需求就是用以说明软件“是什么”的文档，即产品的“规格说明书（**specification**）”
    - 需求是软件生命周期的起点和源头
    - 需求定义决定了软件的功能
  - 统计表明，软件测试中超过半数的错误可以追溯到“不良”的需求或缺少需求上
  - 测试所付出的成本中，超过80%消耗在追踪需求的错误上





嵌入式软件测试的基本方法

- 基于需求的测试
  - 没有需求，就没有真正的测试
    - 好的测试需要好的需求
    - 好的需求支撑好的测试
    - 准确的需求是功能测试的基础
  - 需求测试能够帮助您面对功能测试中的挑战：
    - 如何确保功能覆盖，即功能的完整性？
    - 在保证功能覆盖的前提下，如何减少测试的数量？
      - 穷尽是不现实的
      - 随意是不负责任的
    - 如何确认测试的正确性，即是由于“正确的原因”得到了“正确的结果”？
      - 避免“歪打正着”

嵌入式软件测试的基本方法

基于需求的测试

软件测试，始于需求——基于需求的测试

- 从分析需求定义入手，找出存在的错误和逻辑冲突
  - 二义性、逻辑一致、冗余、完整、可追踪…
- 依据需求设计测试用例，为后面的功能测试做准备
  - 在设计和编码之前就准备好了测试，缩短开发时间

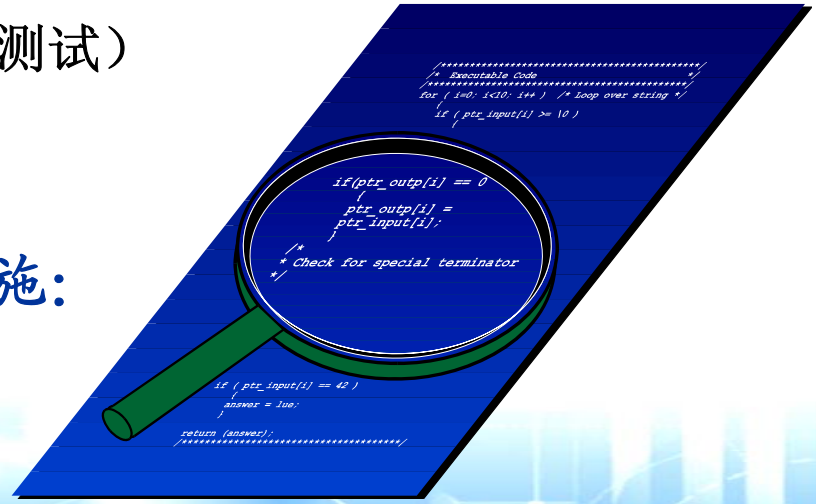
需求测试的优势（Bender-RBT）

- 在软件开发的初期优化并且验证需求，令其“准确”
- 设计测试用例用于代码功能测试，使其“正确”
  - 与编程语言无关
- “敏化路径法”：
  - 确保正确的“激励”得到正确的“响应”
- “最少的测试，最大的覆盖”：
  - 不冗余、不遗漏，既充分，又必要。
  - 功能覆盖率达100%，代码覆盖率70-90%

嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 源代码分析
  - 属于软件静态分析技术。从最初的“自动语法检查”发展而来
  - 不运行被测代码，利用先进的算法对源代码进行分析
  - 主要技术
    - 基于标准的规则检查
    - 基于结构的质量分析（结构化测试）
    - 基于算法的缺陷和漏洞分析
  - 无需测试用例
  - 在开发周期的早期就可以实施：
    - 简单
    - 高效
    - 成本低



嵌入式软件测试基础：
嵌入式软件测试的基本方法

- 源代码分析——代码规则检查
  - 以编程规则为准绳，检查代码的“违规”情况
  - 什么是规则
    - 规则是指导我们写出高质量代码的标准
    - 符合规则是代码高质量、少缺陷的基础
  - 规则检查的好处
    - 通过找出“违规”代码剔除潜在的错误和缺陷
    - 提高产品的可靠性、可移植性、可读性
    - 重在“预防”，影响深远
  - 著名嵌入式软件编程标准
    - **MISRA**（ = The Motor Industry Software Reliability Association ）
    - MISRA C
    - MISRA C++

嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 源代码分析——MISRA标准
  - 为什么要 MISRA C?
    - 标准C语言并不是针对代码安全需求的
    - 标准C语言不是专门为嵌入式应用设计的
    - 标准C语言太庞大了，很难操作
  - 是什么导致了C的不安全?
    - 程序员的失误——错误发生时，C往往能容忍
      - 笔误、算法理解错误等
    - 程序员对语言的误解
    - 程序员对编译器的误解
      - C当中有许多是未经完善定义的
    - 编译器的错误
    - 操作平台的差异
    - 运行时错误

嵌入式软件测试基础：
嵌入式软件测试的基本方法

- 源代码分析——MISRA标准
  - MISRA C的内容一览
    - 21个类别，147条“军规”
  - 从细节做起
    - 数据类型——正确、规范的数据类型及其转换
  - 清晰无误的表达
    - 函数和表达式——表达式、函数的声明和定义问题
  - 安全正确的指向
    - 指针和数组——安全、高效地应用指针
  - 别犯路线错误
    - 流程控制——程序流程的规范做法
  - 求人不如求己
    - 编译器的考虑——避免来自编译器的隐患

嵌入式软件测试基础：

嵌入式软件测试的基本方法

源代码分析——代码缺陷检测

典型的缺陷检测方法

- 动态测试
  - 可以在代码执行过程中发现问题
- 静态源代码分析
  - 通过算法检查代码的错误

随着技术的发展，静态源代码分析已远非当初的语法检查器可比，在缺陷检测方面的能力越来越强

- 强大的缺陷、架构、质量分析工具
- 可以检查运行时错误（**runtime error**）
- 可以用来发现在大规模代码库的复杂交互中所产生的缺陷。
- 可以用静态方法查动态错误！

嵌入式软件测试基础：

嵌入式软件测试的基本方法

源代码分析——代码缺陷检测

代码缺陷分析技术的发展

第一代——桌面工具

- 与编译、链接工具紧密结合
- 重点在编程风格、语法错误
- 单兵作战

第二代——集中式分析

- 部分新技术，包括跨过程的控制流、数据流分析等
- 依赖于对整个代码流的集中视图，分析结果准确
- 在集成构建阶段

第三代——分布式应用

- 新突破，保留了第一代、第二代的优点
- 允许控制分析过程，既可以集中式，也可以分布式
- 分析结果准确，可以在编码初期就应用，直至集成构建阶段

嵌入式软件测试基础：

嵌入式软件测试的基本方法

源代码分析——代码缺陷分析

代码缺陷分析的特点

- 使用静态技术对程序执行时状态和行为的集合进行预测，从而识别源代码中的错误，并标明问题区域
- 无需使用测试用例，算法本身决定了分析工具发现错误的效率
- 存在识别失误的可能：
  - 程序分析，包括运行时错误是不可能完全预测和判定的，理论上没有固定、可靠的方法能够真正回答程序是否有运行时错误
  - “误报率”成为衡量缺陷分析技术或测试工具的重要指标

代码缺陷分析技术要点

- 抽象语法树
- 数据流分析
- 路径分析
- 基于约束的分析
- 基于故障模型的分析
- …

嵌入式软件测试基础：
嵌入式软件测试的基本方法

源代码分析——代码缺陷检测

代码缺陷分析能发现哪些问题
（以Klocwork为例）

- 典型编码缺陷（C/C++）
  - 空指针引用
  - 内存管理问题（内存泄漏、释放未分配内存…）
  - 数组越界
  - 堆栈溢出
  - 未初始化的变量
  - 不匹配的返回类型
  - 编码风格
  - ……
- 典型安全漏洞
  - 访问控制问题
  - 资源泄漏
  - 缓冲区溢出
  - **DNS欺骗**
  - **SQL注入攻击**
  - 忽略返回值
  - ……
- 能检测安全漏洞是新一代源代码分析技术的一个优势

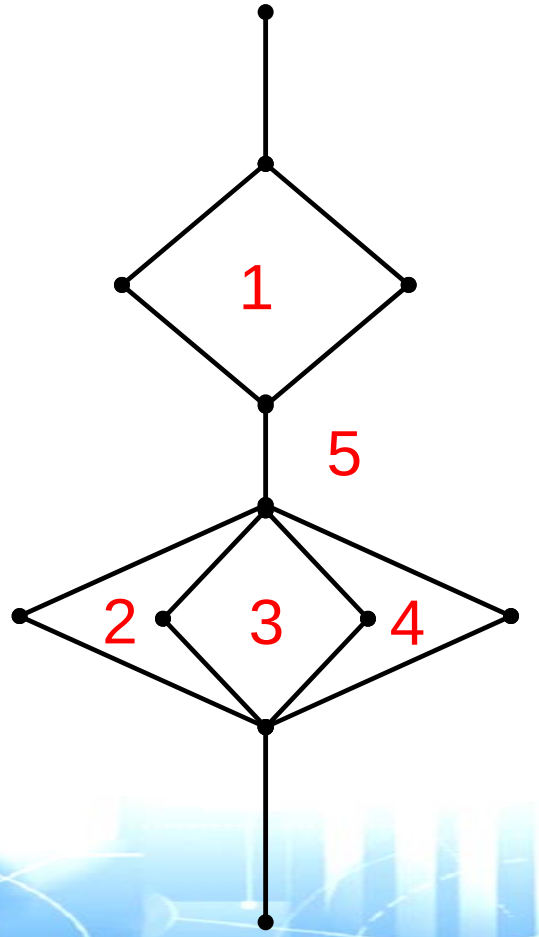
嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 源代码分析——代码质量分析
  - 使用圈复杂度的代码质量分析技术
    - 又称结构化测试，**Thomas McCabe**所创
    - 被美国国家标准技术学会（**NIST**）采用
    - 基于严格的数学计算
    - 客观反映代码的质量
  - 两个重要指标
    - 圈复杂度　——代表代码的“量”
    - 基本复杂度——代表代码的“质”
  - 结构化测试能够很好地预测
    - 故障的可能性
    - 理解的难易度
    - 维护的工作量
  - 为软件质量的改进指明方向

嵌入式软件测试基础：
嵌入式软件测试的基本方法

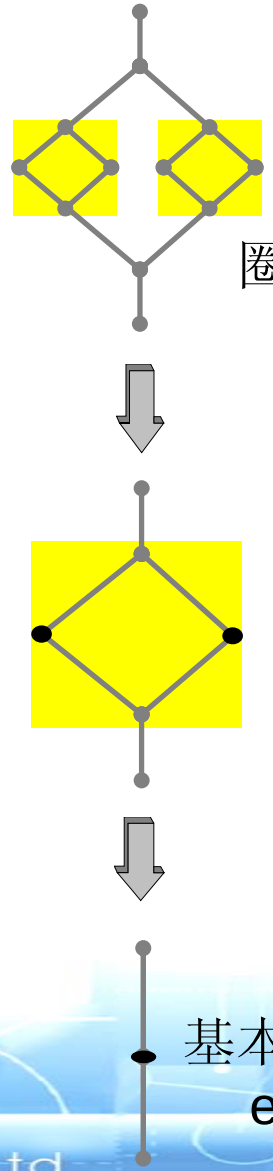
- 源代码分析——McCabe复杂度是如何分析代码质量的
  - 圈复杂度：代码结构的“复杂”程度
    - =模块内独立的线性路径数
    - =模块结构图中独立的“圈”数
  - 优势
    - 结构“复杂”程度的量化
    - 反映了代码的可靠性，出错的几率
    - 帮助查找及其复杂、需要分解的模块
    - 指导测试：
      - 圈复杂度 = 基本路径数 = 最小测试数
    - 独立于编程语言
    - 客观、易理解



嵌入式软件测试基础：

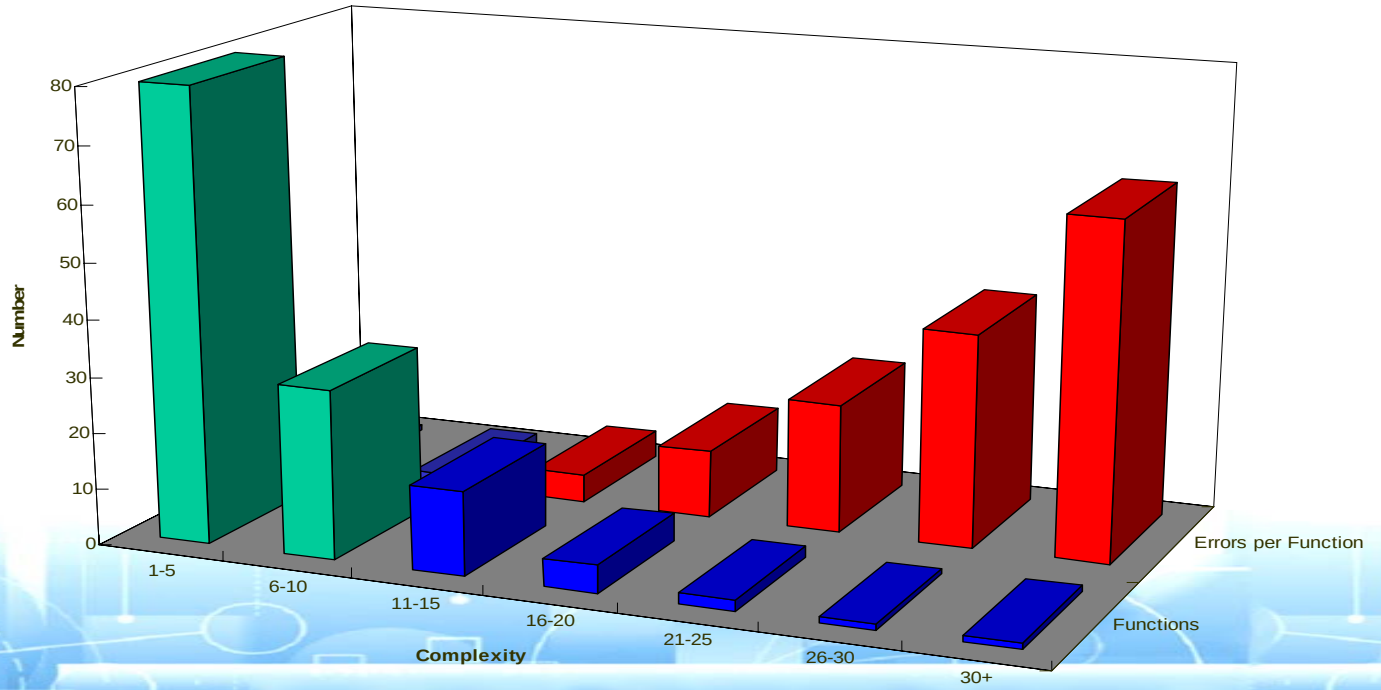
嵌入式软件测试的基本方法

- 源代码分析——McCabe复杂度是如何分析代码质量的
  - 基本复杂度：代码结构的“良好”程度
    - 模块结构依据“结构化”方法简化后的复杂程度
    - 即：模块内“非结构化”逻辑的圈复杂度
  - 优势
    - 结构 “良好”程度的量化
    - 反映了代码的可维护性，可理解程度
      - 预测维护代码和分解代码所需的工作量
    - 揭示了代码的质量
    - 独立于编程语言
    - 客观、易理解



嵌入式软件测试基础：嵌入式软件测试的基本方法

- 源代码分析——代码质量分析
  - 推荐使用标准
    - 圈复杂度 v(G) ≤10
    - 基本复杂度 ev(G) ≤ 4



嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 系统测试方法

真实环境系统测试

- 直接将整个系统（包括硬件平台和嵌入式软件）和其交联的物理设备真实地建立连接，形成闭环，进行测试
- 特点
  - 真实、实时
  - 不仅测试软件，也测试硬件和系统
  - 适用于功能和性能测试
- 局限性
  - 测试成本高昂
    - 构建成本
    - 运行及维护成本
  - 容错性问题
  - 通用性及重复利用问题

嵌入式软件测试基础：

嵌入式软件测试的基本方法

系统测试方法

替代方法：仿真环境测试

- 仿真环境：能对嵌入式软件进行测试的，自动的、实时的、非侵入性的闭环测试系统。
- 能够逼真地模拟被测软件运行所需的真实物理环境的输入和输出，并且能够组织被测软件的输入，来驱动被测软件运行，同时接收被测软件的输出结果。
- 仿真测试环境能够有效地保证测试的：
  - 可重复性
  - 完整性
  - 可扩展性

嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 系统测试方法

  仿真测试环境的必要性

  - 安全及经济上的考虑
  - 相关系统还不可用
  - 打造特殊条件或恶劣环境

- 仿真测试环境的优势

  - 仿真环境的可控性更强，测试更容易，更灵活，评估被测系统的行为特性，如时间特性等变得更加可行；
  - 仿真环境下，更容易获取被测系统的响应和中间结果
  - 仿真环境可以在线的生成大量的测试输入数据，有利于测试结果的分析和评估

嵌入式软件测试基础：
嵌入式软件测试的基本方法

- 系统测试方法

全数字模拟测试

- 采用数学平台的方法，将嵌入式软件从系统中剥离出来，通过开发微处理器指令，常用芯片、I/O、中断、时钟等模拟器在host上实现嵌入式软件的测试
- 主要特点
  - 与嵌入式硬件平台脱钩
  - 操作简单，可以借鉴常规的软件测试方法
  - 适用于功能测试
- 局限性
  - 通用性差
  - 实时性与准确性与真实情况有差异
  - 时序与同步问题
  - 成本高

嵌入式软件测试基础：

嵌入式软件测试的基本方法

实时在线测试

- 嵌入式软件测试的最大难点是对硬件的掌控
- 在硬件平台完全受控的情况下，真实地测试
- 通过支持软件测试的全仿真调试工具（Emulator）实现
- 实时、准确反映软件的运行状况
- 不占用目标资源
- 主要用于功能验证、性能测试及覆盖率分析

仿真器带来的好处

- 无需改变源代码
- 可以实现测试与调试的互动
- 可以实现汇编语言的测试
- 但支持软件测试的仿真器不多

嵌入式软件测试基础：

嵌入式软件测试的基本方法

- 故障注入测试
  - 通过引入故障，测试软件的故障处理能力以及系统的容错能力，验证其可靠性和安全性
  - 方法：通过某种方法向系统刻意注入故障，并观察系统在存在故障时的行为，看看“是否经得起考验”
  - 故障注入方式
    - 基于硬件
    - 基于软件
  - 往往和系统仿真测试联合进行

嵌入式软件测试基础：

嵌入式软件测试的基本方法

故障注入测试

基于硬件的故障注入

- 物理方法实现
- 通过改变环境参数或直接干扰硬件
  - 辐射、电磁干扰
  - 电源干扰
  - 芯片引脚上的信号
- 通过改变接口上的信息
  - 物理层
  - 电器层
  - 协议层
  - 应用层

基于软件的故障注入

- 软件方法生成
- 通过在软件级产生错误，从而造成系统级故障
  - 修改内存数据
  - 修改总线地址
  - 通过应用软件生成故障
  - 通过底层软件（如操作系统）生成故障
- 优点：不易造成硬件损害

```
int sum_two_numbers(int a, int b);
__xception__ = 4294967295;
sum_two_numbers(__xception__ , 10);
```

嵌入式软件测试基础：总结

本篇小结

- 软件测试基础知识
  - 基本概念
  - 基本方法
    - 静态测试、动态测试
    - 白盒测试、黑盒测试
  - 软件测试的“三个问题”
    - 软件功能——对不对？
    - 代码质量——高不高？
    - 测试过程——得唔得？
- 嵌入式软件测试的特点
  - 软硬结合
  - 交叉开发
- 嵌入式软件测试的基本方法
  - 动态测试方法
    - 交叉测试
    - 交叉测试的调试方式
    - 基于需求的测试
  - 静态测试方法
    - 源代码分析
    - 代码质量分析
  - 系统测试方法
    - 仿真环境测试
    - 故障注入测试

谢谢大家