第四章 静态测试技术

hhli@bjtu.edu.cn

2020年11月

内容目录

4.1 静态测试技术概要

静态测试是基于期望属性、专业经验、通用标准来对软件工作产品的特征进行详细检查的一种测试方法。

静态测试的测试对象是不同种类的软件产品交付件，即一切项目过程文档，例如系统设计说明书、产品需求文档、开发设计文档（详细设计说明书、数据库设计说明书）、源代码以及测试文档。

4.1 静态测试技术概要

- 静态测试也称为静态白盒测试，包括各种对需求规约、分析与设计规约、代码及开发过程中的各种文档的检查、静态结构分析等。
- 软件产品可以通过不同的静态技术进行检查以评估软件产品的质量。
- 静态测试可由人工进行，充分发挥人的逻辑思维优势；也可利用工具自动进行。

4.1 静态测试技术概要

- 静态测试方法是指不运行被测程序本身，仅通过分析或检查源程序的语法、结构、过程、接口等来检查程序的正确性。对需求规格说明书、软件设计说明书、源程序做结构分析、流程图分析、符号执行来找错。
- 静态方法通过程序静态特性的分析，找出欠缺和可疑之处，例如不匹配的参数、不适当的循环嵌套和分支嵌套、不允许的递归、未使用过的变量、空指针的引用和可疑的计算等。
- 静态测试具有发现缺陷早、降低成本、覆盖重点和发现缺陷的概率高等优点，以及耗时长、不能测试依赖和技术能力要求高的缺点。

静态测试的特点

- 静态测试不必动态的执行程序，也就是不必进行测试用例设计和结果判断等工作；
- 静态测试可以由人手工方式进行，充分发挥人的优势，行之有效。
  - 解铃还须系铃人，由于人的思维及交流障碍而造成的逻辑错误，由人通过逻辑思维去解决，是一种非常有效的方法；
  - 特别是在充分利用人思维互补的情形，检验出错误的水平非常高。
- 静态测试实施不需要特别条件，容易开展。
- 静态测试结果可用于进一步的查错，并为测试用例选取提供指导。

静态测试方法

静态测试方法分为：代码检查和技术评审两大类。其中代码检查又包括桌面检查、代码审查和代码走查。

- 人工静态测试方法包括：
  ① 桌面检查
  ② 代码审查(Code Inspection)
  ③ 代码走查(Walkthrough)
  ④ 技术评审
- 工具静态测试：主要由软件工具自动进行的程序静态分析。
  - 广义的理解静态测试还包括软件需求分析和设计阶段的技术评审。

4.2 代码检查

代码检查包括代码审查、代码走查、桌面检查。

代码检查的主要任务是检查：

1）代码和设计的一致性

2）代码对标准的遵循

3）代码的可读性

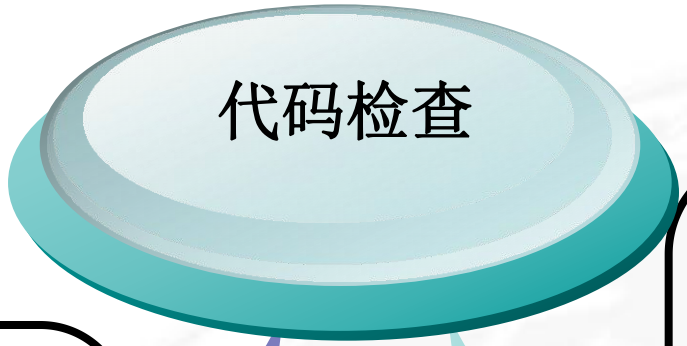
4）代码逻辑表达的正确性

5）代码结构的合理性等。

代码检查目的：

发现违背编程标准、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。

4.2 代码检查



检查类型

- ✓桌面检查
- ✓代码审查
- ✓代码走查

检查内容

- ✓检查变量的交叉引用表。
- ✓检查标号的交叉引用表。
- ✓检查子程序、宏、函数。
- ✓等价性检查、常量检查。
- ✓标准检查、风格检查。
- ✓比较控制流。
- ✓选择、激活路径。
- ✓对照程序与规格说明
- ✓补充文档。

4.2 代码检查

- 实际使用中，代码检查比动态测试更有效率，能快速找到缺陷，发现30%～70%的逻辑设计和编码缺陷；代码检查看到的是问题本身而非征兆。

- 代码检查非常耗费时间，而且代码检查需要知识和经验的积累。

- 代码检查通常应在动态测试之前进行。检查前，应准备好需求描述文档、程序设计文档、程序的源代码清单、代码编码标准和代码缺陷检查表（格）等。

针对代码的常见错误列表

① 数据引用错误

② 数据声明错误

③ 运算错误

④ 比较错误

⑤ 控制流程错误

⑥ 接口错误

⑦ 输入输出错误

⑧ 其它检查

4.2.1. 桌面检查

- **桌面检查**

由程序员在程序通过编译后，对自己的源程序代码进行分析、检验，并补充相关文档，以发现其中的错误或缺陷。

4.2.1桌面检查

- 程序员阅读自己所编的程序
- 缺点：
  - ✓ 第一，由于心理上的原因，容易对自己的程序的偏爱，没有发现错误的欲望（这和已经知道了程序错了读程序找错误所在极为不同）；
  - ✓ 第二，由于人的思维定势，有些习惯性的错误自己不易发现；
  - ✓ 第三，如果根本对功能理解错了，自己不易纠正
- ⑩ 所以这种方法效率不高，可用作为个人自我检查程序中明显的疏漏或笔误。

北京交通大学

4.2.2 代码审查

代码审查

由若干程序员和测试员组成一个审查小组，通过阅读、讨论和争议，对程序进行静态分析的过程。包括准备、发放评审依据文档（包括设计说明书、控制流程图、程序文本、规范标准、检查表），以及召开程序评审会（程序员讲解、大家讨论）。

4.2.2 代码审查

- 代码审查的测试内容：
  - 检查代码和设计的一致性
  - 检查代码对标准的遵循、可读性
  - 检查代码的逻辑表达的正确性
  - 检查代码结构的合理性

4.2.1 代码审查

- 代码审查的组织方式：
  - 代码审查由一组程序和错误检查技术组成。
  - 以代码审查组方式开展工作。

代码审查组

- **通常由四人组成，其中一人为组长。**

- 组长是关键，最好是一个称职的程序员，但不是被测试程序的编写者，也不需要对所检查的程序很熟悉，但需要较强的组织协调和语言能力。
- 组长的职责包括分配资料、安排计划、主持开会、记录并保存被发现的错误。
- 其余成员包括资深程序员、程序编写者与专职测试人员。
- 根据测试的组织方式（如内部测试和独立测试）不同，代码审查小组成员组成可以调节，但组长角色不能变动。

代码审查的步骤:

①准备

②程序阅读

③审查会议

④跟踪及报告

（1）准备

- 组长提前把程序目录表和设计说明书等材料分配给小组成员；
- 小组成员熟悉这些材料；
- 由被测程序的设计和编码人员向审查组详细说明所准备的材料，特别是代码的主要功能与功能间的关系。

(2)程序阅读

- 审查组人员仔细阅读代码和相关材料；
- 对照检查表标出明显缺陷及错误。

(3)审查会议

- 审查会由组长主持；
- 首先由程序员逐句阐明程序的逻辑，在此过程中可由程序员或其他小组成员提出问题，追踪错误是否存在；
  - 经验证明在上述阐述过程中，有很多错误由讲述程序者而不是其他小组成员发现。
- 大声地朗读程序给听众，这样简单的工作是有效的错误检测技术。
- 然后利用代码检查表（审查单）来分析讨论。
- 组长负责掌控讨论朝着建设性的方向前进，而其他人则集中注意力发现错误，但不去纠正错误。

(4)跟踪和报告

- 会后把发现的错误登记造表并交给程序开发人员。
- 如果发现错误较多或发现重大错误，那么在改正之后，组长要再次组织审查会议。
- 为了改进以后的审查工作，对错误登记表也要分析，归类和精炼。
- 如果以第三方测试的方式进行代码审查，则应就发现的缺陷及错误与软件开发人员讨论，避免由于理解不一致产生问题，形成共同认可的审查结果。

- **审查会议的时间：**
- 大约以1.5~2小时为宜
- 审查会需要高度集中注意力，时间太长反而容易使效率降低。
- 每次会议可能处理一个或几个模块。

代码检查表（审查单）

- 代码检查表是代码审查过程所用的主要技术；
- 通常是把程序设计及编码中可能发生的各种错误进行分类，对每一类列举出尽可能多的典型错误，然后制成表格；
- 其它测试中发现的错误也要及时归入代码检查表，形成某一类型软件针对性的代码检查表，以供审查时使用。

代码检查表内容举例（1）

- 数据引用错误：
  - ✓ 是否引用了未赋值或者未初始化的变量？
  - ✓ 所有的数组引用，其下标值是否都在各自的相应维数定义界内？
  - ✓ 所有的数组引用，每一个下表是否是整数值？
  - ✓ 所有引用的指针或变量当前是否已经分配储存了？（即是否存在“悬挂引用”的问题）
  - ✓ 在检索操作或下标引用数组时，是否存在“差1”的错误？

代码检查表内容举例（2）

- 数据说明错误：
  - ✓所有变量是否都显示地说明了？
  - ✓是否每个变量都赋予正常的长度、类型和存储分类？
  - ✓变量的初始化和它的存储类型是否有矛盾？

To be continue……

代码检查表内容举例（3）

- 计算错误：
  - ✓ 是否使用过非一致的数据类型的变量进行运算？
  - ✓ 是否存在混合运算？
  - ✓ 赋值语句的目标变量是否比其右边的表达式小？

代码检查表的其它内容

- 还可以包括编程风格、标准、规范的符合性方面的内容。
- 在代码检查表的错误登记表中，应写明所查出的错误的类型、错误类别、错误的严重程度、错误的位置、错误的原因等信息。

文档的阅读方法

- 要仔细阅读需求设计等文档，特别是了解软件的整体物理意义、应用背景以及在大系统中的地位。
- 对大型软件而言，这些信息会在阅读程序时有效地帮助读者从一定的高度审视，而不是停留在逐行扫描代码。
- 有些错误要有整体观才能发现。

阅读结构化代码的两种方法

1)追踪通过每个子程序的主要逻辑行，主要逻辑行全部跟踪完，然后开始跟踪第二条路径

- 相当于深度优先遍历。

2)按排列顺序追踪代码，从主要行开始，然后检查较低层的程序段

- 相当于广度优先遍历

两种阅读方法的综合使用

- 这两种阅读方式的差别在于何时进入下层模块，选择那种方式要视具体程序特点而定。
  - 广度优先遍历有助于很快地了解程序的全貌；
  - 深度优先适于详细查阅功能处理步骤。
- 应综合使用上述两种方法，在头一两遍阅读时，采用广度优先，然后用深度优先。

程序阅读的次数

- 现代软件测试的奠基人之一Boris Beizer提出至少要读程序4次，分别针对印刷错误、数据结构、控制流和处理
- 4次阅读要比读一次能更快、更容易、更可靠的完成任务。

代码审查的辅助工具

- 利用汇编或编译器生成的交叉引用表（变量、标号、子程序）；
- 利用逆向工程工具（例如从源代码生成流程图）；
- 使用带有快速查找功能的编辑器。

4.2.3代码走查

- **代码走查：**
- 与代码审查相似，先成立走查组，发放相关材料，认真研究，再开评审会。**但会议程序不同**。该方法中，测试组先准备一批有代表性的测试用例，小组开会集体扮演计算机角色，沿着程序逻辑运行一遍，随时记录程序的踪迹，供分析讨论用。

4.2.3代码走查

- 代码走查与代码审查相似，它也是由一组程序和错误检查技术组成，只是检查程序和错误的检查技术不完全相同。

代码走查组

- 代码走查以小组方式进行。
- 代码走查组包括：
  - ✓ 组长：类似代码审查组长
  - ✓ 秘书：负责记录发现的错误，要有一定水平。
  - ✓ 测试人员应是具有经验的程序设计人员，或精通程序设计语言的人员，或从未介入被测试程序的设计工作的技术人员（这样的人没有被已有的设计框住），没有约束，比较容易发现问题。

代码走查的过程

- 与代码审查过程相似
- 先把材料交给每个小组人员，让他们认真研究程序，然后再召开代码走查会议。

代码走查会议的内容

- 与代码审查不同，不是读程序和使用检查表，而是由被指定的作为测试员的小组成员提供若干测试用例（程序的输入数据和期望的输出结果）；然后，让参加会的成员当计算机，在会议上对每个测试用例用头脑来执行程序，也就是用测试用例沿程序逻辑走一遍，并由测试人员讲述程序执行过程，在纸上或黑板上监视程序状态（变量的值）。

代码走查会议的内容

- 每次开会时间以1－2小时为宜，但不允许中断；
- 如果发现问题由秘书记下来，中间不讨论任何纠错问题，主要是发现错误。

测试用例在代码走查中的作用

① 代码走查中，测试用例并不是关键，也并不是仅想验证这几个测试用例运行是否正确，人脑毕竟比计算机慢太多。

② 这里测试用例是作为怀疑程序逻辑与计算错误的启发点，在随测试实例游历程序逻辑时，在怀疑程序的过程中发现错误。

③ 这样比几个测试用例本身直接发现的错误要多得多。

代码走查中的缺点

- 代码走查使用测试用例启发以发现错误，人们注意力会相对集中在随测试用例游历的程序逻辑路径上，不如代码审查检查的范围广，错误覆盖面全。

代码审查与代码走查的比较

① 二者都是由若干程序员与测试员组成一个小组，集体阅读并讨论程序，或者用“脑”执行并检查程序的过程。

② 分两步完成：

- 预先作一定的准备工作；
- 然后举行会议进行讨论；

③ 会议的主题是发现错误而不是纠正错误。

代码审查与代码走查的优点

- 二者不仅比桌面检查优越得多，而且与动态测试的方法相比也有很多优点：
  - ① 使用这种方法测试，一旦发现错误，就知道错误的性质和位置，因而调试所花费的代价低；
  - ② 使用这种方法一次能揭示一批错误，而不是一次只揭示一个错误。
- 如果使用动态测试，通常仅揭示错误的征兆。而对错误的性质和位置还得逐个查找。

代码审查与代码走查的效果

- 经验表明，使用这种方法能够优先的发现30~70%的逻辑设计和编码错误。
- IBM使用代码审查方法经验表明，错误的检测效率高达全部查出错误的80%。
- Myers的研究发现，代码审查和代码走查平均查出全部错误的70%。

代码审查、代码走查与动态测试相互补充

- 研究表明：使用代码审查和代码走查发现某类错误比用动态测试更有效，而对另一类错误情况正好相反；
- 因此，代码审查和代码走查方法与动态测试结合，测试效果更佳。

4.3 (正规)技术评审

- 技术评审是一种审查技术，其特点是由一组评审者按照规范的步骤，综合运用走查和审查技术，对软件的需求，设计、代码或其它技术文档进行仔细的检查，以找出和消除其中的错误或缺陷。
- 评审也被当作一种管理工具，经过评审不仅可以提高各阶段软件产品的质量，还可以收集到一些有关该软件产品质量的数据。

4.3 (正规)技术评审

- 技术评审属于广义的测试范畴，也是一种质量保证手段。
- 软件开发过程中每个阶段的评审都必须十分正规的，严格的加以定义，并根据规程实施。
- 技术评审的目的：
  - 发现软件在功能、逻辑、实现上的错误或缺陷；
  - 验证软件是否符合它的需求；
  - 确认软件符合预先定义的开发规范和标准；
  - 保证软件在统一的模式下进行开发；
  - 便于项目管理等5大目的。

4.3 (正规)技术评审

- 技术评审不仅为新手提供软件分析、设计和实现的培训途径，而且为相关的开发、测试人员熟悉他人软件提供机会。
- 评审小组至少由3人组成（包括被审材料作者），一般为4-7人。包括评审员、主持人、宣读员、记录员、作者等角色。

4.3 (正规)技术评审

- 评审小组成员：
- 1）评审员：负责检查被审材料，找出错误或缺陷。一般为被审材料生命周期前一阶段、本阶段、下一阶段的相关开发人员。
- 2）主持人：负责会议计划、主持和问题复核。
- 3）宣读员：在评审会上朗读和分段引导评审组不偏离被审材料。除代码评审之外可以选择作者担任，其它评审最好选择直接参与后续开发的人员担任。
- 4）记录员：负责将发现的软件问题记录在“技术评审问题记录表”中。
- 5）作者：被审材料的作者负责在评审会上回答评审员提出的问题。并负责修正评审会发现的问题。

4.3 (正规)技术评审

- 技术评审活动过程（1）：
- 1）计划：主持人检查文档完整性，确定是否满足评审条件（如，评审前代码必须通过编译），确定评审小组、会议时间、地点，发放评审材料。
- 2）预备会：如果评审组不熟悉项目背景，可通过预备会由作者介绍评审目的、材料功能、用途及开发技术。
- 3）会前准备（自评审）：评审员根据检查要点检查被审材料，对发现的问题做好记录；主持人了解每位评审员情况并掌握发现的普遍问题。

4.3 (正规)技术评审

- 技术评审活动过程（2）：
- 4）评审会：全体评审员共同对材料进行检查。宣读员朗读或讲解材料，评审员随时提出发现的问题或疑问，记录员将问题写入"记录表"。评审会结束是，全体评审员做出最后的评审结论。主持人会后对问题进行分类。

4.3 (正规)技术评审

- 技术评审活动过程：
- **5**）修正错误：作者在会后对评审会提出的问题根据评审意见进行修正。
- **6**）复审：如果评审材料存在较多或较复杂的问题，主持人可以决定对修正后的材料再次举行评审会。
- **7**）复核：主持人或其委托人对修正后的材料进行复核，检查评审会提出的并需要修订的问题是否得到解决，主持人完成"技术评审总结报告"。

*4.4 静态测试的内容

- 针对不同的软件中间产品，静态测试的内容也不尽相同；
- 对不同的文档进行静态测试的内容可以体现在对特定文档的测试对照条例中。
- 下面以软件开发过程中的几个有代表性的主要文档和代码，列举静态测试的对照条例，以说明静态测试的内容：

4.4.1 需求定义的静态测试对照条例

- 对需求定义的测试着重于测试对用户需求的描述和解释是否完整、准确；
- 根据有关标准整理而成的对需求定义进行静态测试的对照条例。
- 我们可以将这些条例按照所测试的软件质量因素分成10组：

To be continue.....

需求定义的静态测试对照条例

- 兼容性：

  界面需求是否要求软硬件系统具有兼容性？

需求定义的静态测试对照条例

- 完备性：

① 需求定义是否包含了有关文件（指质量手册、质量计划以及其它有关文件）中所规定的需求定义所应该包含的所有内容？

② 需求定义是否包含了有关功能、性能、限制、目标、质量等方面的所有需求？

③ 功能性需求是否覆盖了所有非正常情况的处理？

④ 是否对各种操作模式（如正常、非正常、有干扰等）下的环境条件都作了规定？

⑤ 是否对所有功能与时间有关的方面都作了考虑？

⑥ 是否识别出了所有与时间因素有关的功能？它们的时间准则是否都说明了？

⑦ 时间准则的最大、最小执行时间是否都定义了？

⑧ 是否识别并定义了在将来可能会变化的需求？

需求定义的静态测试对照条例

- 完备性（续）

⑨ 是否定义了系统所有的输入？

⑩ 是否标识清楚了系统输入的来源？

⑪ 是否识别出了系统的输出？

⑫ 是否说明了系统输入、输出的类型？

⑬ 是否说明了系统输入、输出的值域、单位、格式等等？

⑭ 是否说明了如何进行系统输入的合法性检查？

⑮ 是否定义了系统输入、输出的精度？

⑯ 是否定义了系统性能的各个方面？

⑰ 在不同负载情况下，系统的生产率如何？

⑱ 在不同的情况下，系统的响应时间如何？

⑲ 系统对软件、硬件或电源故障必须作什么样的反应？

⑳ 是否充分地定义了关于人机界面的需求？

需求定义的静态测试对照条例

- 一致性：

① 各个需求之间是否一致？是否有冲突和矛盾？

② 所规定的模型、算法和数值方法是否相容？

③ 是否使用了标准的术语和定义形式？

④ 需求是否与其软硬件操作环境相容？

⑤ 是否说明了软件对其系统和环境的影响？

⑥ 是否说明了环境对软件的影响？

需求定义的静态测试对照条例

- 正确性：
  ① 需求定义是否满足标准的要求？
  ② 算法和规则是否有科技文献或其它文献作为基础？
  ③ 有哪些证据说明用户提供的规则或规定是正确的？
  ④ 是否定义了对在错误、危险分析中所识别出的各种故障模式和错误类型所需的反应？
  ⑤ 是否参照了有关的标准？
  ⑥ 是否对每一个需求都给出了理由？理由是否充分？
  ⑦ 对设计和实现的限制是否都有论证？

需求定义的静态测试对照条例

- 可行性：

① 需求定义是否使软件的设计、实现、操作和维护都可行？

② 所规定的模型、数值方法和算法是否对待解问题合适？是否能够在相应的限制条件下实现？

③ 是否能够达到关于质量的要求？

需求定义的静态测试对照条例

- 易修改性：

① 对需求定义的描述是否易于修改（例如，是否采用良好的结构和交叉引用表等等）？

② 是否有冗余的信息？是否一个需求被定义多次？

需求定义的静态测试对照条例

- 健壮性：
  - ✓是否有容错的需求？

需求定义的静态测试对照条例

- 易追溯性：

① 是否可从上一阶段的文档查找到需求定义中的相应内容？

② 需求定义是否明确的表明前阶段中提出的有关需求和设计限制都已经被覆盖了？（例如，使用覆盖矩阵或交叉引用表）？

③ 需求定义是否便于向后继续开发阶段查找信息？

需求定义的静态测试对照条例

易理解性：

① 是否每一个需求都只有一种解释？

② 功能性需求是不是以模块方式描述的，是否明确的标识出了其功能？

③ 是否有术语定义一览表？

④ 是否使用了形式化或半形式化的语言？

⑤ 语言是否有歧义性？

⑥ 需求定义中是否只包含了必要的实现细节而不包含不必要的实现细节？是否过分细致了？

⑦ 需求定义是否足够清楚和明确使其能够作为开发设计规约和功能性测试数据的基础？

⑧ 需求定义的描述是否将对程序的需求和所提供的其它信息分离开来了？

需求定义的静态测试对照条例。

- 易测试性和可验证性：

① 需求是否可以验证？（即是否可以检验软件是否满足了需求？）

② 是否对每一个需求都指定了验证过程？

③ 数学函数的定义是否使用了精确定义的语法和语义符号？

4.4.2 设计文档的静态测试对照条例

- 对设计文档的静态测试：
- 着重于分析设计是否与需求定义一致，所采用的数值方法和算法是否适用于待解问题，程序的设计中对程序的划分是否与待解问题相适应，需求是否都被满足了等等。

设计文档的静态测试对照条例

- --完备性

① 软件设计规约是否包含了有关文件（指质量手册、质量计划及其它有关文件）中所规定的所有内容？

② 是否有充分的数据（如逻辑结构图、算法、存储分配图等）来保证设计的完整性？

③ 算法、公式等是否充分、准确、完善？

④ 是否在设计中明确地说明了产品的开发中所需要的软件、硬件和测试所需要的软硬件？

⑤ 对于每一个程序界面，设计是否都实现了所要求的行为？

To be continue.....

设计文档的静态测试对照条例

- --完备性（续）

⑥ 是否标识出了程序的每一个输入、输出和数据库成分？其描述是否详细到了可以编码的程度了？

⑦ 是否说明了程序的操作环境？

⑧ 是否包含了所有的处理步骤？

⑨ 是否给出了每一个决策点的所有出口转向？

⑩ 设计是否考虑到了所有可能的情况和条件？

⑪ 设计是否指明了在出现异常情况和不正当输入的情况下的行为？

⑫ 设计是否参照了有关的设计标准？

设计文档的静态测试对照条例

- ——一致性

① 在设计文档中，是否始终使用标准的术语和定义？文档的风格和详细程度是否前后始终一致？

② 界面之间是否相容？

③ 设计是否包含内在矛盾？

④ 模型、算法和数值方法之间是否在数学上是相容的？

⑤ 输入 / 输出的格式是否一致？

⑥ 类似的功能和相关的功能的设计是否一致？

⑦ 计算中的输入、输出和数据库成分的计量单位和计算精度是否一致？逻辑表达式是否一致？

设计文档的静态测试对照条例

- —正确性

① 设计文档是否满足有关标准的要求？

② 设计是否只完成需求定义中要求的功能？若包含额外的功能，则这些功能的必要性是否经过论证？

③ 测试文档是否准确？

④ 设计逻辑是否正确？即，程序是否会完成所需的功能？

⑤ 设计是否与所描述的操作环境相一致？

⑥ 界面的设计是否与文档所描述的界面部分一致？

⑦ 对于设计者不能选择的输入输出和数据库成分的数据格式、内容和数据，设计是否正确地给予了安排？

设计文档的静态测试对照条例

- —可行性

①所设计的模型、算法和数值方法对于应用领域来说是否可以接受？

②设计能否在所规定的限制条件下和开发代价下实现？

③在可用的资源条件下，所设计的功能能实现吗？

设计文档的静态测试对照条例

—易修改性

① 设计是否使用了信息隐藏技术？

② 模块的组织使对一条需求的改变只影响较少的模块

③ 对于可能改变的数据与函数的结构的设计使它们的界面对于单个函数的改变不敏感

④ 将数据库和I/O的接口从应用程序中分离开来，并使用专门的程序来完成之，不使用全局可取接的数据

⑤ 功能分解成一系列子程序，使每一个子程序都具有最大限度的内部紧密联系和最低限度的互相依赖

⑥ 高内聚、低耦合

⑦ 每一个子程序都只有一个功能吗？

设计文档的静态测试对照条例

- ——模块性

① 是否采用了模块化的机制？

② 设计是否使软件系统由一系列相对来说较小的、以层次结构相互联系的子程序组成？是否每一个子程序都只完成一个特定的功能？

③ 设计是否使用了特殊的规则来限制子程序的大小？

设计文档的静态测试对照条例

- —可预测性

① 设计是否包含了子程序来提供在已经标识出的出错情况下所需的反应？

② 所设计的计算机资源调度方式是否是确定的和可预测的，而不是动态的？

③ 设计是否使用了尽量少的中断和事件驱动，对使用这样的功能是否进行了论证？

④ 是否设计了在程序的运行过程中进行正确性检查来发现运行时刻的错误和违反运行许可的情况？

设计文档的静态测试对照条例

- —结构化

①设计是否使用了层次式的逻辑控制结构？

设计文档的静态测试对照条例

—易追溯性

① 设计文档中是否包含设计与需求定义中的需求、设计限制等内容的对应关系？

② 设计是否标识出了设计中所包含的需求定义之外的功能？

③ 是否对所有函数都进行了适当的标识使代码能够唯一地引用该标识？

④ 设计规约是否包含修改历史记录，并使所有的对设计的修改和修改理由都记录在内并赋以编号了？

⑤ 设计规约是否包含了设计备案文档并记录了与设计有关的决策？

设计文档的静态测试对照条例

- —易理解性

①设计是否避免了不必要的成分和表达形式？

②设计文档是否不致造成歧义性解释？

设计文档的静态测试对照条例。

- —可验证性/易测试性

① 设计中对每一个函数的描述是否都使用了良好的术语和符号？是否可以验证它与需求定义相一致？

② 是否定量地说明了使用条件、限制等内容？是否可以由此产生测试数据？

4.4.3 源代码的静态测试对照条例

- 对源代码的静态测试着重于分析实现是否正确、完备。

源代码的静态测试对照条例

—完备性

①代码是否完全、准确地实现了设计规约中所规定的内容？

②代码是否满足设计要求？

③代码是否创建了所需的数据库或其它初始化数据？

④是否有未引用的或未定义的变量、常量或数据类型？

源代码的静态测试对照条例

- ——一致性

① 代码在逻辑上与设计规约一致吗？

② 是否自始至终使用了相同的格式、调用约定、结构等等？

源代码的静态测试对照条例

- —正确性

① 代码是否符合标准？

② 变量的定义和使用都正确吗？

③ 注释是否正确？

④ 子程序调用的参数个数正确吗？

源代码的静态测试对照条例

- **—易修改性**

① 代码中对常数的使用是否都通过符号来进行，使其便于修改？

② 是否有交叉引用表或数据字典来表明程序对常量和变量的取接？

③ 代码是否由单入口、单出口的子程序构成？（错误处理除外）

④ 代码是否避免了直接使用地址，而采用标号和符号常量？

源代码的静态测试对照条例

- —可预测性

① 是否避免了使用自我修改的代码？

② 是否避免了依赖于程序设计语言中的缺省值的代码？

③ 代码是否包含无穷循环？

④ 是否避免了递归？

源代码的静态测试对照条例

- —健壮性

① 代码是否防止可以发现的运行时刻的错误？如下标变量越界，除数为零，变量值越界，栈溢出等等

源代码的静态测试对照条例

- —结构化

①程序的每一个功能是否都可以作为一块代码而识别出？

②循环是否只有一个入口？

源代码的静态测试对照条例

- —易追溯性

①是否有一个交叉引用表，通过它可以便捷地从代码找到相应的设计？

②是否有修改历史记录，它记录对代码的所有修改历史和修改原因？

源代码的静态测试对照条例

- **—易理解性**

① 注释是否使用简洁明了的语言对每一个子程序都作了充分的描述？

② 是否有不必要地复杂的代码？若有，是否使用了注释对其进行解释？

③ 是否使用了一致的格式（如缩进和空格的使用）？

④ 是否使用了便于记忆的命名约定？命名是否反映了变量的类型？

⑤ 变量的有效值域是否定义了？

⑥ 代码中的公式是否使用了设计规约中相应数学模型的公式？

源代码的静态测试对照条例。

- —可验证性

①实现是否避免了使用测试难度大的技术和方法？

对照条例的说明

- 对照条例并不是一成不变的。
- 在静态测试的实践中，对照条例可根据实际情况进行适当的增减。
- 特定的应用领域和特定的开发方法都会对静态测试提出一些特定的要求，这些要求可以正确地反映在静态测试的对照条例之中。

练习题

① 主要由人工进行的静态测试方法有哪几种？

② 代码审查与代码走查有什么区别？

本章完