软件工程
第二部分 结构化方法学
软件测试

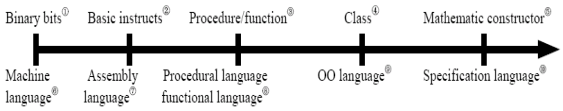
6.1编码
6.2软件测试基础
6.3软件测试的策略(单元/集成/确认)
6.4软件测试技术(白盒/黑盒)
6.5程序调试
6.6软件可靠性

6.1 编码

- 作为软件工程过程的一个阶段，实现是对设计的进一步具体化。通常把编码和测试统称为实现。
- 编码就是把软件设计结果翻译成用某种程序设计语言书写的程序。
- 程序的质量主要取决于软件设计的质量。所选用的程序设计语言的特点及编码风格也将对程序的可靠性、可读性、可测试性和可维护性产生深远的影响。

选择程序设计语言

- 程序设计语言是人机通信的最基本的工具，影响解题方式、实现的效率、程序的测试与维护等。
- 选择程序设计语言的主要依据:
  - ✓系统用户的要求
  - ✓可以使用的编译程序
  - ✓可以得到的软件工具
  - ✓工程规模
  - ✓程序员的知识
  - ✓软件可移植性要求
  - ✓软件的应用领域



①二进制位,②基本指令,③过程行代码,④类代码行,⑤数学构造子,⑥机器语言,⑦汇编语言,⑧过程式程序设计语言,⑨面向对象程序语言,⑩规格说明语言.

Fig.1 Illustration for evolution of computer language and promotion of reuse

图 1 计算机语言的变迁与复用的提升图示

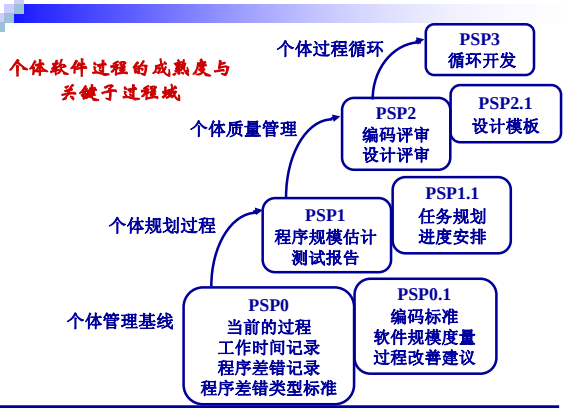
编码风格

- 源程序代码应该逻辑简明清晰、易读易懂。为此，应该遵循下述规则:
  - ✓程序内部的文档: 包括恰当的标识符、适当的注解和程序的视觉组织等等。
  - ✓数据说明: 数据说明的次序应该标准化。
  - ✓语句构造: 每个语句都应该简单而直接，不能为了提高效率而使程序变得过分复杂。
  - ✓输入输出: 输入/输出数据的格式简单、必要的合法性检验等。
  - ✓效率: 包括时间与存储两个方面。注意不要牺牲程序的清晰性和可读性来不必要地提高效率。

个体软件过程

- PSP是一种可用于控制、管理和改进个人工作方式的自我持续改进过程，是一个包括软件开发表格、指南和规程的结构化框架 。
- 由美国CMU/SEI 的 Watts Humphrey 领导开发的。他对104位参与PSP培训的软件人员的统计数据表明(程序规模从50至5000行不等)，在应用PSP后:
  - 总的差错减少58.0%
  - 在测试阶段发现的差错减少71.9%
  - 生产效率提高20.8%。

- PSP的结果表明，软件缺陷绝大多数是由于对问题的错误理解或简单的失误所造成，只有很少一部分是由于复杂的逻辑关系所产生。
- PSP的结果还表明，在编程阶段的缺陷引入率是设计阶段的3.5倍。因此，保障软件产品质量的一个重要途径是提高设计质量。
- 在软件设计阶段，PSP的着眼点在于软件缺陷的预防。具体的途径是强化设计结束准则，而不是设计方法的选择。

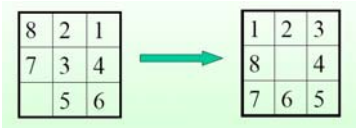


个体软件过程实践

■作业题目

① 输入n个数（实数或整数），输出其最大最小值。
② 输入n个数和一个整数K，输出n个数中第K个最大最小值。要求有输入错误判断及相应错误信息。
③ 输入n个数，用任意算法对其进行排序并按从小到大顺序输出。
④ 对给定一个英文文本文件，统计并输出26个英文字母（不区分大小写）的出现频率，并统计定冠词'the'的出现次数及频率。

⑤ 任意输入一个带有+, -, x, /和（ ）的算术表达式，输出表达式的值。
⑥ 给定一九格图，1-8八个数字随机被放入其中8个格内，编程序在方格内移动数字，使8个数字在九格图周围8个格内顺序（逆时针，顺时针均可）排列。任一数字，只有当其周围四连通的格之一为空格时，与空格交换位置。



时间、编码、缺陷记录与管理

时间记录日志
学生：xxx　日期：11.28
教员：xxx 课程：个体软件过程

日期	开始时间	结束时间	中断时间	净时间	活动	备注	C	U
11.17	9：00	10：10	10	60	编程	第一二题	X	1
	13：15	15：30	20	115	上网查资料	关于堆栈的资料，网上聊天		0
	17：30	20：10	40	140	编程序	第三四五题，	X	3
11.28	18：20	21：33	12	121	编程	做第六题，上网找资料，看算法书	X	1

表3.1　作业编号日志

作业号	日期	过程	估计数据		实际数据			累计数据				
			时间	单元	时间	单元	平均值	时间	单元	平均值	最大值	最小值
	描述：											
	监话：											

表3.2　程序规模估计表

程序	代码行	以前的功能	估计的功能	最小	平均	最大

表3.4 每周活动预算

任务 / 日期	阅读图书	上网	BBS	查询资料			日总计
日							
一	360	20	20	30			430
二	360	20	20	30			430
三	360	20	20	30			430
四	360	20	20	30			430
五	360	20	20	30			430
六							
周总计	1800	100	100	150			2150

表4.1 缺陷分类

类型编号	类型名称	描述
10	文档	注释，信息
20	语法	拼写，标点符号，打字，指令格式
30	联编打包	变更管理，库，版本控制
40	赋值	说明，重名，作用域，限制
50	接口	过程调用和引用，输入输出，用户格式
60	检查	出错信息，不合适的检查
70	数据	结构，内容
80	函数	逻辑，指针，循环，递归，计算，函数缺陷
90	系统	配置，记时，内存
100	环境	设计，编译，测试，其他支持系统问题
备注：		

表4.3 代码复查脚本

入口条件	在复查前，检查下列产品是否已经准备好： ◆ 需求规格说明； ◆ 程序设计文档； ◆ 程序的源代码清单； ◆ 编码标准； ◆ 代码复查检查表；
一般性说明	使用代码复查检查表； 在复查时遵照代码复查检查表的使用说明； 在复查结束时，填写累计、累计百分比和总结栏目；
1 复查规程	首先，完成源程序编码； 然后，在进行编译和测试之前，打印一份源程序清单； 下一步，进行代码复查； 进行代码复查时，仔细检查每一行源程序，以尽可能多地发现和修复缺陷；
2 修复缺陷	修复所发现的每一个缺陷； 确保所作的修复正确无误； 将缺陷登入在缺陷记录日志。
3 覆盖率复查	验证程序设计覆盖了需求规格说明中描述的每一个功能； 验证程序代码实现了所有的设计。
4 程序逻辑复查	验证程序设计在逻辑上是正确的； 验证程序代码正确的实现了设计中的逻辑。
5 命名和类型检查	验证所有的名字和类型已经正确的声明； 检查整型、长整型和浮点型是否正确声明。
6 变量检查	确保每个变量已初始化； 检查上溢、下溢或越界问题。
7 程序语法检查	验证程序代码符合编程语言的规格说明
出口条件	在复查结束时，应该有： 完整的、修复过的源程序清单； 填写完整的时间记录日志； 填写完成的缺陷记录日志。

6.2 软件测试基础

1、什么是软件测试?

简单地说，软件测试就是为了发现错误而执行程序的过程。

- **定义**：

软件测试是一种**软件质量保证活动**，其动机是通过一些经济有效的方法，发现软件中存在的缺陷，从而**保证软件质量**。

强调软件测试是一种**软件质量保证手段**。

2、软件测试的目的

- 以最少的人力、物力和时间找出软件中潜在的各种错误和缺陷，通过修正各种错误和缺陷，提高软件质量。
  - ✓发现软件的错误
  - ✓检验软件是否满足需求
  - ✓提高软件的质量

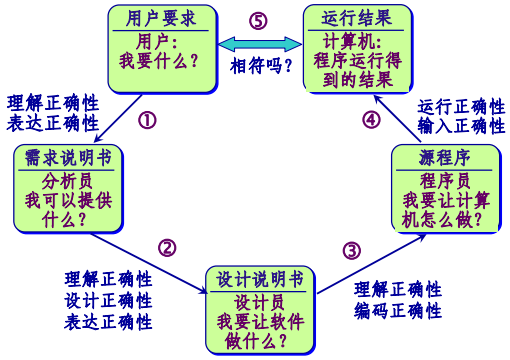
3、软件测试的原则

- ✓ 所有的测试都应当追溯到用户要求
- ✓ 把“尽早地和不断地进行软件测试”作为座右铭
- ✓ 应该由独立的第三方从事测试工作
- ✓ Pareto原则
- ✓ 从“小规模”测试开始，并逐步进行“大规模”测试
- ✓ 测试用例应由输入数据和预期输出结果组成，应包括合理的输入条件和不合理的输入条件
- ✓ 穷举测试是不可能的

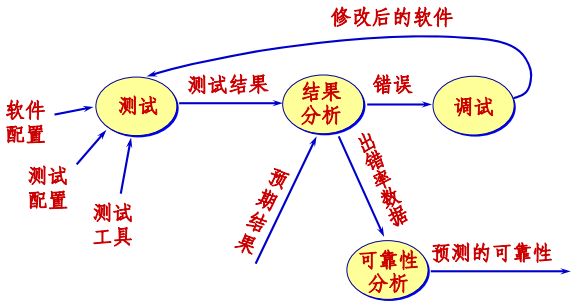
4、软件测试的对象

- 软件测试并不等于程序测试。软件测试应贯穿于软件定义与开发的整个期间。
- 需求分析、概要设计、详细设计以及程序编码等各阶段所得到的文档，包括需求规格说明、概要设计规格说明、详细设计规格说明以及源程序，都应成为软件测试的对象。
- 测试的两个方面：
  - 缺陷测试
  - V & V（验证和确认）

- 验证(Verification)，检查软件生存期各个阶段过程活动的结果是否满足规格说明的描述，证实各阶段和阶段之间的逻辑协调性、完备性和正确性。
- 确认(Validation)，是比验证更广泛的过程活动。目的是想证实在一个给定的外部环境中软件的逻辑正确性，即是否满足用户的要求。
- Boehm给出两者的区分：
  - 验证：我们是否在正确地建造一个产品
  - 确认：我们是否在建成一个正确的产品



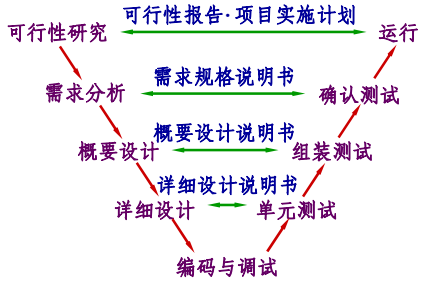
5、测试信息流



6、测试与软件开发各阶段的关系

- 软件开发过程是一个自顶向下，逐步细化的过程
- 软件计划阶段定义软件范围（作用域）
- 软件需求分析阶段建立软件信息域、功能和性能需求、约束等
- 软件设计阶段建立软件体系结构、用户接口、数据结构和过程设计
- 程序编码阶段把设计用某种程序设计语言转换成程序代码

- 测试过程是依相反顺序安排的自底向上，逐步集成的过程。



6.3 软件测试的阶段与策略

- 测试过程按5个步骤进行:
  - ✓单元测试集中对用源代码实现的每一个程序单元进行测试，检查各个程序模块是否正确地实现了规定的功能。
  - ✓集成测试把已测试过的模块组装起来，主要对与设计相关的软件体系结构的构造进行测试。

- ✓确认测试则是要检查已实现的软件是否满足了需求规格说明中确定了的各种需求，以及软件配置是否完全、正确。
- ✓系统测试把已经经过确认的软件纳入实际运行环境中，与其他系统成份组合在一起进行测试。是否完全、正确。
- ✓验收测试是向用户表明所开发的软件系统能够像用户所预定的那样工作。

6.4 软件测试技术

1. 黑盒测试
   - 等价类划分
   - 边界值分析
   - 错误推测法[*]
   - 因果图[*]
   - 功能图[*]
   - 接口测试[*]
2. 白盒测试
   - 逻辑覆盖
   - 基本路经测试
   - 控制结构测试[*]
   - 数据流测试[*]

- 黑盒测试方法主要是为了发现以下错误:
  1) 是否有不正确或遗漏了的功能?
  2) 在接口上，输入能否正确地接受? 能否输出正确的结果?
  3) 是否有数据结构错误或外部信息 (例如数据文件) 访问错误?
  4) 性能上是否能够满足要求?
  5) 是否有初始化或终止性错误?

- 软件人员使用白盒测试方法，主要想对程序模块进行如下的检查:
  1) 对所有的逻辑判定，取“真”与取“假”的两种情况都至少测试一次 — 逻辑覆盖测试;
  2) 对程序模块的所有独立的执行路径至少测试一次 — 路径覆盖测试;
  3) 在循环的边界和运行界限内执行循环体 — 控制流测试;
  4) 测试内部数据结构的有效性 — 数据流测试。

6.5 调试

- 软件调试是在进行了成功的测试之后才开始的工作。与软件测试不同，调试的任务是进一步诊断和改正程序中潜在的错误。
- 调试活动由两部分组成：
  1. 确定程序中可疑错误的确切性质和位置。
  2. 对程序（设计，编码）进行修改，排除这个错误。
- 调试工作是一个具有很强技巧性的工作。

- 软件运行失效或出现问题，往往只是潜在错误的外部表现，而外部表现与内在原因之间常常没有明显的联系。如果要找出真正的原因，排除潜在的错误，不是一件易事。
- 调试是通过现象，找出原因的一个思维分析的过程。

6.5.1 调试的步骤

① 从错误的外部表现形式入手，确定程序中出错位置；
② 研究有关部分的程序，找出错误的内在原因；
③ 修改设计和代码，以排除这个错误；
④ 重复进行暴露了这个错误的原始测试或某些有关测试。

- 从技术角度来看，查找错误的难度在于：
  - ✓现象与原因所处的位置可能相距甚远。
  - ✓当其他错误得到纠正时，这一错误所表现出的现象可能会暂时消失，但并未实际排除。
  - ✓现象实际上是由一些非错误原因（例如，舍入不精确）引起的。

- ✓现象可能是由于一些不容易发现的人为错误引起的。
- ✓错误是由于时序问题引起的，与处理过程无关。
- ✓现象是由于难于精确再现的输入状态（例如，实时应用中输入顺序不确定）引起。
- ✓现象可能是周期出现的。在软、硬件结合的嵌入式系统中常常遇到。

6.5.2 几种主要的调试方法

- 调试的关键在于推断程序内部的错误位置及原因。可以采用以下方法：

1)强行排错

这种调试方法目前使用较多，效率较低。它不需要过多的思考，比较省脑筋。例如：

a.通过内存全部打印来调试，在这大量的数据中寻找出错的位置。

b.在程序特定部位设置打印语句，把打印语句插在出错源程序的各个关键变量改变部位、

重要分支部位、子程序调用部位，跟踪程序的执行，监视重要变量的变化。

- ◆ 自动调试工具。利用某些程序语言的调试功能或专门的交互式调试工具，分析程序的动态过程，而不必修改程序。
- ✓ 应用以上任一种方法之前，都应当对错误的征兆进行全面彻底的分析，得出对出错位置及错误性质的推测，再使用一种适当的调试方法来检验推测的正确性。

2) 回溯法调试

- ✓ 这是用于小程序一种有效的调试方法。
- ✓ 一旦发现了错误，人们先分析错误征兆，确定最先发现“症状”的位置。然后，人工沿程序的控制流程，向回追踪源程序代码，直到找到错误根源或确定错误产生的范围。
- ✓ 例如，程序中发现错误处是某个打印语句。通过输出值可推断程序在这一点上变量的值。再从这一点出发，回溯程序的执行过程，…,直到找到错误的位置。

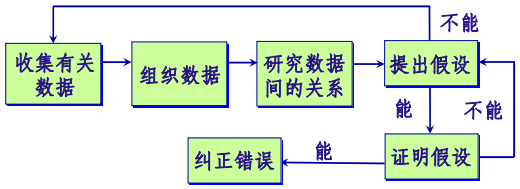
3) 对分查找法

基本思路：如果已经知道每个变量在程序内若干个关键点的正确值，则可以用赋值语句或输入语句在程序中点附近“注入”这些变量的正确值，然后运行程序并检查所得到的输出。如果输出结果是正确的，则错误原因在程序的后半部分；反之，错误原因在程序的前半部分。

对错误原因所在的那部分再重复使用这个方法，直到把出错范围缩小到容易诊断的程度为止。

4) 归纳法调试

- ✓ 归纳法是一种从特殊推断一般的系统化思考方法。归纳法调试的基本思想是：从一些线索（错误征兆）着手，通过分析它们之间的关系来找出错误。



a) 收集有关的数据　列出所有已知的测试用例和程序执行结果。看哪些输入数据的运行结果正确，哪些输入数据的运行结果有错误。

b) 组织数据　归纳法是从特殊到一般的推断过程，所以需要组织整理数据以发现规律。

常以3W1H形式组织可用的数据：

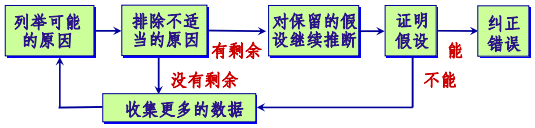
- ➢“what”　列出一般现象；
- ➢“where” 说明发现现象的地点；
- ➢“when” 列出现象发生时所有已知情况；
- ➢“how”　说明现象的范围和量级；

c) 提出假设　分析线索之间的关系，利用在线索结构中观察到的矛盾现象，设计一个或多个关于出错原因的假设。如果一个假设也提不出来，归纳过程就需要收集更多的数据。此时，应当再设计与执行一些测试用例，以获得更多的数据。

d) 证明假设　把假设与原始线索或数据进行比较，若它能完全解释一切现象，则假设得到证明；否则，就认为假设不合理，或不完全，或是存在多个错误，以致只能消除部分错误。

5) 演绎法调试

- ✓演绎法是一种从一般原理或前提出发，经过排除和精化的过程来推导出结论的思考方法。
- ✓首先根据已有的测试用例，设想出所有可能出错的原因；然后再用原始测试数据或新的测试，从中逐个排除不可能正确的假设；最后，再用测试数据验证余下的假设确是出错的原因。



a) 列举所有可能出错原因的假设 把所有可能的错误原因列成表。通过它们，可以组织、分析现有数据。

b) 利用已有的测试数据，排除不正确的假设 仔细分析已有的数据，寻找矛盾，力求排除前一步列出所有原因。如果所有原因都被排除了，则需要补充一些数据(测试用例)，以建立新的假设。

c) 改进余下的假设 利用已知的线索，进一步改进余下的假设，使之更具体化，以便可以精确地确定出错位置。

d) 证明余下的假设

6.5.3 调试原则

- 在调试方面，许多原则本质上是心理学方面的问题。调试原则也分成两组。
- 确定错误的性质和位置的原则
  - ✓分析思考与错误征兆有关的信息。
  - ✓避开死胡同。只把调试工具当做辅助手段来使用。利用调试工具，可以帮助思考，但不能代替思考。
  - ✓避免用试探法，最多把它当做最后手段。

- 修改错误的原则
  - ✓ 在出现错误的地方很可能还有别的错误。
  - ✓ 修改错误的一个常见失误是只修改了这个错误的征兆或这个错误的表现，而没有修改错误的本身。
  - ✓ 当心修正一个错误的同时有可能会引入新的错误。
  - ✓ 修改错误的过程将迫使人们暂时回到程序设计阶段。
  - ✓ 修改源代码程序，不要改变目标代码。

■基于程序频普的故障定位技术Tarantula

- totalFailed=4
- totalPassed=4
- failed(5)=2
- passed(5)=4
- suspiciousness(5)=0.33

$$suspiciousness(e) = \frac{\frac{failed(e)}{totalfailed}}{\frac{passed(e)}{totalpassed} + \frac{failed(e)}{totalfailed}}$$

	coc a# 0	coc a# 12	nul k# 0	nul k# 12	coc a# 10 00	coc a# 12 00	nul k# 10 00	nul k# 12 00		rank
1	•	•	•	•	•	•	•	•	0.5	3
2	•	•	•	•	•	•	•	•	0.5	3
3	•	•	•	•	•	•	•	•	0.5	3
4	•	•	•	•	•	•	•	•	0.5	3
5	•	•			•	•	•	•	0.33	8
6	•	•			•	•	•	•	0.33	8
7	•	•			•	•	•	•	0.33	8
8	•	•			•	•	•	•	0.33	8
9			•	•					1.0	1
10			•	•					1.0	1
11	•	•	•	•	•	•	•	•	0.5	3
Pass/Fail Status	F	F	F	F	T	T	T	T		

6.6 软件可靠性

1. 基本概念

- 测试阶段的根本目标是消除错误，保证软件可靠性。用户通常关注软件可以使用的程度。
- 软件可靠性的一般定义(MTTF)：软件可靠性是程序在给定的时间间隔内，按照规格说明书的规定成功地运行的概率。
- 软件可用性定义：程序在给定的时间点，按照规格说明书的规定，成功地运行的概率。

- 随着运行时间的增加，运行时出现程序故障的概率也将增加，即可靠性随着给定的时间间隔的加大而减少。
- 区分“故障”与“错误”：
  - “错误”的含义是由开发人员造成的软件差错
  - “故障”的含义是由错误引起的软件的不正确行为。
- 可靠性和可用性之间的主要差别：可靠性意味着在0到t这段时间间隔内系统没有失效；可用性只意味着在时刻t，系统是正常运行的。

2. 估算软件可靠性(MTTF)

- 软件的平均无故障时间MTTF (Mean Time To Failure)是一个重要的质量指标。
- 程序中潜藏的错误的数目是一个十分重要的量，它既直接标志软件的可靠程度，又是计算软件平均无故障时间的重要参数。
- 程序中的错误总数E_T与程序规模I_T、类型、开发环境、开发方法论、开发人员的技术水平和管理水平等都有密切关系。

估算错误总数的方法

(1) 植入错误法

- 基本思路：在测试之前由专人在程序中随机地植入一些错误，测试之后，根据测试小组发现的错误中原有的和植入的两种错误的比例，来估计程序中原有错误的总数E_T。

- 人为地植入的错误数为N_s，经过一段时间的测试之后发现n_s个植入的错误，此外还发现了n个原有的错误。如果可以认为测试方案发现植入错误和发现原有错误的能力相同，则能够估计出程序中原有错误的总数：

$$n/N\hat{} = n_s/N_s \Rightarrow N\hat{} = n/n_s \times N_s$$

$N\hat{}$为错误总数E_T的估计值。

(2) 分别错误法

- 基本思路：为了消除植入错误法的基本假定带来的误差，对程序中的原有错误加标记，然后根据测试过程中发现的有标记错误和无标记错误的比例，来估计程序中原有错误的总数E_T。
- 具体说来，为了随机地给一部分错误加标记，两个测试员彼此独立地测试同一个程序的两个副本，把其中一个测试员发现的错误作为有标记的错误。

- 用 τ 表示测试时间，假设
  - τ =0时错误总数为B_0;
  - $\tau = \tau_1$时测试员甲发现的错误数为B_1;
  - $\tau = \tau_1$时测试员乙发现的错误数为B_2;
  - $\tau = \tau_1$时两个测试员发现的相同错误数为b_c。
- 如果将甲发现的错误认为是有标记的，并且乙发现有标记与无标记错误的概率相同，则能估计原有错误的总数为：

$$b_c/B_1 = B_2 / B_0\hat{} \Rightarrow B_0\hat{} = B_2/b_cB_1$$

MTTF的计算公式

- 经验表明，平均无故障时间MTTF与单位长度程序中剩余的错误数成反比，即

$$MTTF=1/[K(E_T/I_T-E_c(\tau)/I_T)]$$

其中：

- ✓ K为常数，典型值是200
- ✓ E_T：测试之前程序中错误总数
- ✓ I_T：程序长度(机器指令总数)
- ✓ τ：测试(包括调试)时间
- ✓ $E_d(\tau)$：在0至τ期间发现的错误数
- ✓ $E_c(\tau)$：在0至τ期间改正的错误数

小结

- 实现包括编码和测试两个阶段。编码是把软件设计的结果翻译成用某种程序设计语言书写的程序。编码使用的语言，特别是写程序的风格，对程序质量有相当大的影响。
- 软件测试是保证软件可靠性的主要手段。测试阶段的根本任务是发现并改正软件中的错误。
- 大型软件的测试应该分阶段地进行，分为单元测试、集成测试和确认测试3个基本阶段。
- 白盒测试和黑盒测试是软件测试的两类基本方法。通常，在测试过程的早期阶段主要使用白盒方法，在测试过程的后期阶段主要使用黑盒方法。

- 调试的主要任务是改正测试过程中发现的软件错误。调试包括确定错误的位置与性质和排除错误两个步骤。
- 软件可靠性与可用性是评价软件系统的重要指标。程序中潜藏的错误的数目，直接决定了软件的可靠性。通过测试可以估算出程序中剩余的错误数。根据测试和调试过程中已经发现和改正的错误数，可以估算软件的平均无故障时间。