第3章

软件静态测试技术

“软件测试”在线课程

本章导学

内容提要

本章介绍软件静态测试（分析）的概念、静态测试技术、静态分析方法。静态测试技术包括代码检查、静态结构分析（包括程序数据流分析方法和程序控制流分析方法）、代码质量度量、评审与检查等。本章还详细介绍了静态分析的另一主要策略——软件评审，阐述了评审的概念、评审的组织、评审过程、评审的类型。此外，本章较细致、全面地介绍了静态测试工具 IBM Logiscope 的主要功能和在静态测试方面的应用。

学习目标

- 正确理解软件静态测试的概念
- 正确理解和掌握软件静态分析方法
- 认识通过测试工具进行静态测试分析的方法及应用过程
- 认识和理解关于软件评审的概念、作用及评审的内容、过程、类型

3.1 软件静态测试

在测试计划制订完后，实施软件测试的最基本、最主要的问题是如何设计测试用例。没有测试用例，则无法执行测试，自然也就无法找出软件缺陷或错误。为此，需要进一步明确什么是测试用例。

一个测试用例用于执行一项测试。

测试环境需要有输入和输出。

测试关联的事项：数据、路径。

测试需要考虑成本的效应、所包含的风险、时间的限制、测试工具的支撑程度。

3.1.1 静态测试技术概述

1. 静态测试的基本概念

静态测试是软件测试的主要技术手段之一，包括手工评审和静态分析两种技术方法，静态测

试的组成如图 3-1 所示。

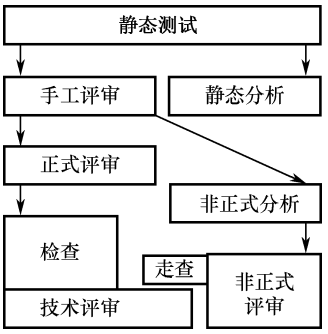

图 3-1　静态测试的组成

静态测试技术在软件生命周期的各级测试中均有应用，但常应用于软件的早期测试过程，如需求分析阶段、项目概要设计阶段、详细设计阶段及组件测试阶段。

与软件动态测试不同，静态测试不是测试用例的执行，而是一个静态分析过程。这个分析过程可通过人工审查的方式来完成，也可使用特定的测试分析工具来进行。

分析与检查的主要目的是从已有规格说明、已定义标准甚至项目中发现缺陷和偏差。检查结果可用于优化开发过程。静态分析与检查的基本思想是预防缺陷。在运行软件前，越早发现缺陷并纠正与消除，越能降低软件风险和开发成本。

静态测试主要以人工方式进行，充分发挥人的逻辑思维优势，同时可借助测试专用工具进行自动化测试。通常，在执行测试时根据测试的不同阶段需采用不同的静态测试方法或综合使用多种测试方法。

对软件系统（产品）开发中所有项目技术文档的审查，是静态测试的主要内容之一。目前在业界的实际操作中，基本上都采用人工评审的方式进行。应用测试工具进行的静态分析，只能用于能被自动检查的文档，如程序文档等。

静态分析有别于程序的编译。编译过程虽能发现某些程序的错误，但这些错误基本为程序中的语法错误和违反编译规则性的问题。编译系统通常不能检查与判定软件或程序中的逻辑错误，更无法达到寻找缺陷或故障的目的，因此，静态分析是编译所不能替代的。

2．静态测试的内容及过程

静态测试的内容及过程主要有测试需求分析、测试概要设计、测试详细设计、测试执行与结果分析。

（1）测试需求分析。测试需求分析是静态测试过程的首要阶段，该阶段主要完成静态测试的需求分析工作。测试需求分析主要依据软件开发计划、需求文档，确定测试的需求，建立测试基础与评审基础，制订标准测试计划，进行细节的设计、数据库的测试。

（2）测试概要设计。测试概要设计是在需求分析的基础上，完成测试方案的制定，包括确定测试内容、测试策略、测试方法、测试目标。该阶段还将建立测试详细设计的基础与测试评审的基础。

（3）测试详细设计。测试详细设计是在完成了测试的需求分析和概要设计，并通过静态测试评审后进行的。静态测试的详细设计的主要任务是完成测试过程的各项具体安排和确定测试实施的具体细节，包括测试用例设计、测试环境搭建、测试工具选用、测试人员组织及测试进度安排等。

（4）测试执行与结果分析。根据已制订的静态测试计划进行静态测试，落实和完成各项测试的具体任务，并提交测试工作的结果。

3.1.2　静态测试技术

静态测试技术主要包括代码检查、静态结构分析、代码质量度量、评审与检查。

1．代码检查

代码检查，主要检查代码和设计的一致性，代码对标准的遵循、可读性，代码的逻辑表达的正确性，代码结构的合理性等方面。

通过代码检查，可发现其中存在的违背程序编写标准的问题，程序中不安全、不明确和逻辑混乱、错误的部分，可以找出程序中不可移植的部分和违背程序编程规范（或行业、企业制定的技术风格）的问题。代码检查的内容包括变量检查、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。

在实际测试中，静态测试的代码检查比动态测试更为有效，能快速并较准确地找到缺陷，发现 30%～60%的逻辑设计和编码缺陷。

代码检查看到的是程序问题本身。因此，代码检查非常耗时，而且代码检查人员需要有比较深厚的专业技术知识与大量的编程经验。代码检查一般在编译与动态测试之前进行。检查前，应准备好需求分析文档、程序设计文档、程序源代码清单、代码编写规范及代码缺陷检查表等资料。

2．静态结构分析

静态结构分析主要以图形方式表现程序的内部结构，如函数调用关系图、函数内部控制流图。其中，函数调用关系图以直观的图形方式描述一个应用程序中各个函数的调用和被调用关系；函数内部控制流图显示一个函数的逻辑结构，其由许多节点组成，一个节点代表一条语句或数条语句，连接节点的叫边，边表示节点间的控制流向。

检查项：代码风格和规则审核，程序设计和结构的审核，业务逻辑的审核，走查、审查与技术复审手册。

3．代码质量度量

目前在软件工程中，针对软件或代码的复杂度的度量测试主要存在三种度量方式（或称为度量参数）：Line（行）复杂度度量，Halstead（运算符与运算元）复杂度度量、McCabe 复杂度度量。

（1）Line 复杂度：以代码的行数作为度量计算的基准。

（2）Halstead 复杂度：以程序中出现的操作符和操作数（运算符和运算元）作为计算对象，直接测量指标，并据此计算出程序长度和程序容量，描述程序的最小实现和实际实现之间的关系，据此度量程序复杂程度。

任意程序 P，总是由操作符和操作数通过有限次的组合而形成。

P 的符号表词汇量 $N=n_1+n_2$（n_1 表示唯一操作数数量，n_2 表示唯一操作符数量）。

设 n_1 是 P 中出现的所有操作数，n_2 是 P 中出现的所有操作符。

度量指标计算：程序长度 $N=n_1+n_2$，程序容量 $V=N\times\log_2 N$。

在编程时，代码体积会因程序实现方式和编码习惯有所差异，还会因所采用的程序设计语言（如保留字数量、语句结构等）而有所不同。因此，又有如下标准。

程序语言等级：$L=V_{min}/V$（V_{min} 是程序实现时可能的最小代码容量）。

编程效率：$E=V/L$。

Halstead 的度量分析指出，在软件开发中，把系统划分为单独的模块的好处是：短代码的设计、编码及测试的难度都比长代码低。

（3）McCabe 复杂度：完整的 McCabe 复杂度包括圈复杂度、基本复杂度、模块设计复杂度和集成复杂度等。圈复杂度：将程序流程图结构转化为有向图结构，然后以图论方法来衡量软件的

质量。圈复杂度在数量上表现为独立的程序现行路径条数，这是合理地预防缺陷所需进行测试的最少路径条数。一个程序的圈复杂度相当于至少需要多少个测试用例才能对这个程序实现全路径覆盖。

程序圈复杂度（度量值）越高，程序代码质量可能越低。据经验，程序存在的可能错误及缺陷与高的圈复杂度度量值有很大关系，这将影响设计测试用例及进行后期的软件维护。

目前，业界广泛使用的一些测试工具的代码复杂度度量功能，就是依据某种代码质量度量的原理及算法设计的。

4．评审与检查

1）静态测试评审

静态测试评审是指对需求分析和概要设计进行评审。静态测试评审在需求分析和概要分析阶段的评审基础上开展，包括手工评审和静态技术分析两个步骤。手工评审分为正式评审和非正式评审。正式评审为执行检查过程（技术评审），非正式评审主要为走查过程。

2）静态测试检查

对静态测试的每个过程都要进行检查，以确保静态测试的有效性和测试的质量。静态测试检查的内容是：从所指定的测试计划开始，检查测试的初始工作和测试的准备情况。检查以会议的形式进行，根据检查结果决定是否需要重新制订计划或确定其后的实现环节及工作。若检查通过，则继续测试过程。

3）静态测试的具体细节

- 检查算法的逻辑正确性，确定算法是否实现所要求的功能。
- 检查模块接口的正确性，确定形参的个数、数据类型及顺序是否正确，确定返回值类型及返回值的正确性。
- 检查输入参数是否有合法性检查。若没有合法性检查，应确定该参数是否不需要合法性检查，否则应进行参数的合法性检查。
- 检查调用其他模块的接口是否正确，检查实参类型、实参个数是否正确，返回值是否正确。检查当被调用模块出现异常或错误时，程序是否有适当的出错处理代码。
- 检查是否设置了适当的出错处理措施，以便在程序出错时，能对出错部分进行处理，以保证其逻辑的正确性。
- 检查表达式、语句是否正确，是否含二义性。
- 检查常量或全局变量的使用是否正确。
- 检查标识符的使用是否规范、一致，变量命名能否顾名思义，是否简洁、规范和容易被理解、识记。
- 检查程序风格的一致性、规范性，代码是否符合行业（企业）规范，所有模块的代码风格是否与规定的相一致，且符合规范。
- 检查代码是否能优化，算法效率是否最高。
- 检查代码注释是否完整，是否对算法做了正确说明。

4）静态测试可发现的错误

静态测试可发现程序的下列错误或提供程序缺陷的间接信息。

- 错用局部变量和全局变量。
- 未定义的变量、不匹配的参数。

- 不适当的循环嵌套或分支嵌套、死循环、不允许的递归。
- 调用不存在的子程序或函数，遗漏标号或代码。
- 从未使用过的变量、不会执行到的死代码、从未使用过的标号。
- 标识符的使用错误和过程调用错误，潜在的死循环。

采用静态测试可发现 1/3～2/3 的软件逻辑设计和编码方面的错误，但软件代码中仍会有隐藏的故障无法通过静态测试方法发现。除了静态测试方法，在实际测试中，还必须通过动态测试对程序运行中的缺陷与错误进行细致、深入的分析。

3.2　程序数据流分析方法

3.2.1　数据流测试

数据流测试是另一种通过静态测试技术来发现缺陷的手段与方法。数据流测试是关注变量的赋值与使用（或引用）位置的结构性测试方法，可认为这是基于路径测试的一种改良方案，用于路径测试的“真实性”检查。因此，数据流测试重点关注变量的定义与使用的检查。实际上，在修改缺陷或错误时，常会使用此方法。例如，在一段程序代码中搜索某个变量所有的定义、引用位置，并考察在程序运行时，该变量的值会如何发生变化，从而找出 Bug 产生的原因。

数据流测试将测试方法进行了形式化，以便于构造算法，从而实现自动化分析。

现做下述定义：P 代表程序，$G(P)$为程序图，V 为变量集合，P 的所有路径集合为 PATH(P)。

节点 n 是变量 v 的定义节点，记作 DEF(v,n)。输入语句、赋值语句、循环控制语句和过程调用，都是定义节点语句的例子。如果执行定义这种语句的节点，那么与该变量关联的存储单元的内容就会改变。

节点 n 是变量 v 的使用节点，记作 USE(v,n)。输出语句、赋值语句、条件语句、循环控制语句、过程调用，都是使用节点语句的例子。如果执行对应这种语句的节点，那么与该变量关联的存储单元的内容就会保持不变。

如果 USE(v, n)作为一个谓词使用（条件判断语句中），则记为 P-use；如果 USE(v, n)作为一个计算使用（计算表达式中），则记为 C-use。

变量 v 的定义-使用路径记为 du-path。如果对于 PATH 中的某个路径，定义节点 DEF(v,m)为该路径的起始节点，使用节点 USE(v, n)为该路径的终止节点，则该路径是变量 v 的定义-使用路径。

变量 v 的定义-清除路径记为 dc-path，如果变量 v 的某个定义-使用路径，除起始节点外没有其他定义节点，则该路径是变量 v 的定义-清除路径。

数据流覆盖指标（拉普斯-韦约克）层次结构图（见图 3-2）描述数据“定义-使用”对，找出所有变量的定义-使用路径，考察测试用例对这些路径的覆盖程度，就可作为衡量测试效果的参考。

数据流测试方法主要是为了发现定义/引用异常的缺陷。这里所说的异常是指可能会导致程序失效的情形，异常可能会触发运行风险。例如，发现数据流异常：没有初始化就读取了变量的值，或根本没有使用变量的值。在测试中，检查每个变量的使用情况，对每种类型的变量的用法或变量的状态进行区分。

数据流异常的现象，有时不一定很明显，也并非每个异常都会导致错误的行为。对已经发现数据流问题的那部分程序（片），需要做更多的检查，进一步发现定义/引用的问题。

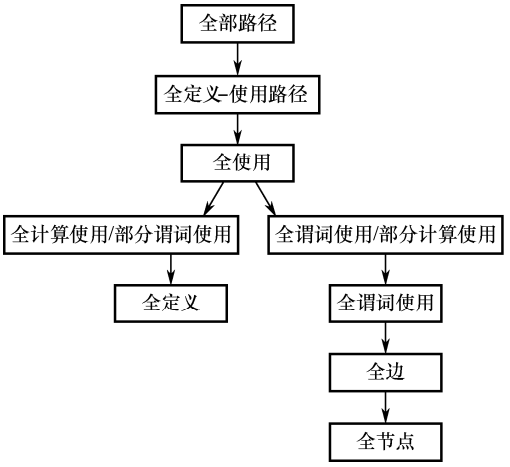

图 3-2　数据流覆盖指标（拉普斯-韦约克）层次结构图

3.2.2　数据流测试的应用举例

1．变量的定义和使用

例如，a=b。

- DEF(1)={a}
- USES(1)={b}

又如，a=a+b。

- DEF(1)={a}
- USES(1)={a,b}

2．测试举例

程序如下：

```
1  a=5;                    //定义 a
2  while(c1)
3    {
4      if(c2)
5        {
6          b=a*a;          //使用 a
7          a=a-1;          //定义且使用 a
8      }
9  print(a);               //使用 a
10  print(b);              //使用 b
    }
```

数据流测试使用数据流覆盖指标（拉普斯-韦约克）层次结构图来描述上述程序的数据“定义-使用”对，并找出所有变量的定义-使用路径，可得到程序图（见图 3-3），使用路径 du-path 与清除路径 dc-path、定义/使用节点（见表 3-1）、定义/使用路径（见表 3-2）、路径图（见图 3-4）。

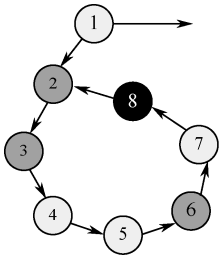

图 3-3　程序图

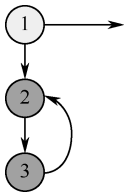

图 3-4　路径图

表 3-1　定义/使用节点

变量	定义节点	使用节点
a	1，5	4，5，7
b	4	8

表 3-2　定义/使用路径

变量	路径（开始、结束）节点	是否定义清除
a	1，4	是
	1，5	否
	1，7	否
	5，7	是
b	4，8	是

3.3　程序控制流分析方法

3.3.1　程序的控制流图

1．控制流图

程序结构就是一幅控制流图。为了清晰地说明控制流的作用，首先需要对有关控制流图、圈复杂度和图形矩阵等基本概念进行说明。

在进行软件测试的设计时，为了能更加突出程序控制流的结构，可对程序的流程图进行简化，简化之后所得的图形称为程序控制流图，控制流图简称流图。

一个程序控制流图中涉及的图形符号只有两种：判断节点和控制流线。

（1）**判断节点：**由带有标号的圆圈表示，它可代表一个或多个语句、一个处理框的序列和一个条件判断框（不包含复合条件）。

（2）**控制流线：**由带有箭头的弧线及直线表示，称为边。它代表程序中的控制流。常见程序语句的控制流图如图 3-5 所示。包含条件的节点称为判断节点（简称节点），由判断节点出发的边必须终止于某一个节点，由边和节点所限定的范围（包围的区域）称为区域。

2．矩阵图

矩阵图是控制流图的矩阵表示形式，其矩阵的维数等于控制流图的节点数。列与行对应于标识的节点，矩阵每个元素与节点连接的边对应，控制流图的矩阵图表示如表 3-3 所示，与其对应的控制流图如图 3-6 所示。

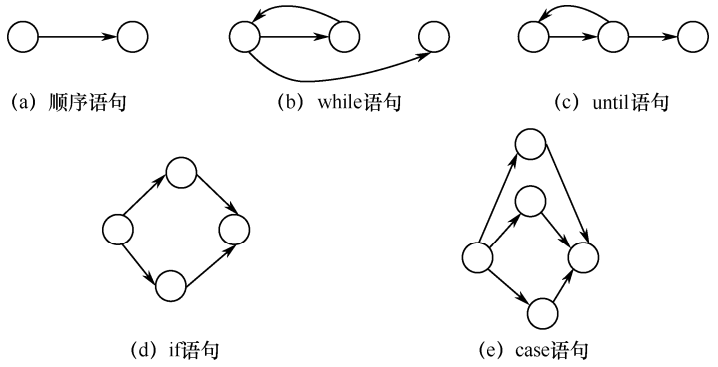

图 3-5　常见程序语句的控制流图

表 3-3　控制流图的矩阵图表示

节点	1	2	3	4
1		a		
2			b	
3				c
4	d			

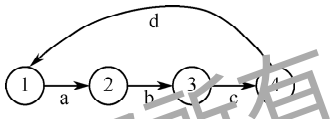

图 3-6　与表 3-3 对应的控制流图

控制流图的节点一般用数字表示，边可以用字母表示。在图 3-6 所示的例子中，若矩阵记为 M，则 M(1,2)= “a”，表示边 a 连接节点 1 和节点 2。需要注意的是，边 a 的方向是从节点 1 到节点 2。同样 M(4,1)= “d”，表示边 d 连接节点 4 和节点 1，边 d 的方向是从节点 4 到节点 1。

3．控制流异常

通过控制流图，很容易理解程序的结构，同时可发现一些可能的异常情况。例如，程序异常地跳出循环体，或程序结构有多个出口。某些异常情况并不一定会导致程序失效，但可能不符合结构化的编程原则或面向对象的程序规范。通常，控制流图的生成并不一定要采用手工方式，特别是在程序较大或复杂时，更多的是使用分析工具映射产生控制流图。

若控制流图的某些部分或整个图都很复杂，事件发生的顺序和相互关系很难被理解，则需修改程序的内容，降低程序的复杂性，因为复杂的程序结构常常意味着更大的发生错误的风险。

另外，有些静态测试分析工具还能产生前驱后继表，来表示每个语句之间的相互关系。若某个语句没有前驱，则这个语句是不可达的（为死代码）。这样，可发现一个缺陷或至少一个异常的情况。正常的情况是，只有程序的第一个语句没有前驱，最后一个语句没有后继。对于多入口和多出口的程序，这个原则同样适用。

3.3.2　将程序流程图转换为控制流图

将程序流程图转换为控制流图，主要是为了进行软件复杂度的度量，并能方便地设计测试用例。转换的方法如下（关键是对程序分支的处理）。

（1）将程序流程图中的每个分支转换为一个独立的节点。

（2）分支前的顺序块（不论有几个）均可合并入节点。

（3）对所有的节点及程序控制的流向进行编号。

下面依照上述转换原则，将一个典型的程序流程图转换为程序控制流图，如图 3-7 所示。