第42卷第6期 原 子 能 科 学 技 术 Vol. 42, No. 6
2008年6月 Atomic Energy Science and Technology June 2008

安全软件验证与确认中的单元模块测试技术

李 铎,张良驹,冯俊婷
(清华大学 核能与新能源技术研究院,北京 100084)

摘要:核动力厂保护系统实现数字化必须解决的一项关键技术是如何完成安全软件的验证与确认(V&V),以证明和确认执行安全功能的软件自身的安全性和可靠性。软件单元测试是V&V过程中的重要环节,主要目的是验证和确认软件代码开发过程中,软件的设计转变为软件代码是适当、正确和完整的。本工作初步研究了安全软件的单元模块测试技术,着重讨论如何保证测试的完整性、建立测试环境、建立测试用例及实施单元模块测试等方面,并以某数字化保护系统安全软件单元模块测试实例说明单元模块测试的具体过程。

关键词:安全软件;单元测试;验证与确认
中图分类号:TL36 **文献标志码**:A **文章编号**:1000-6931(2008)06-0551-006

Technique for Unit Testing of Safety Software Verification and Validation

LI Duo, ZHANG Liang-ju, FENG Jun-ting
(*Institute of Nuclear and New Energy Technology, Tsinghua University, Beijing* 100084, *China*)

Abstract: The key issue arising from digitalization of the reactor protection system for nuclear power plant is how to carry out verification and validation (V&V), to demonstrate and confirm the software that performs reactor safety functions is safe and reliable. One of the most important processes for software V&V is unit testing, which verifies and validates the software coding based on concept design for consistency, correctness and completeness during software development. The paper shows a preliminary study on the technique for unit testing of safety software V&V, focusing on such aspects as how to confirm test completeness, how to establish test platform, how to develop test cases and how to carry out unit testing. The technique discussed here was successfully used in the work of unit testing on safety software of a digital reactor protection system.

Key words: safety software; unit testing; verification and validation

收稿日期:2007-01-30;**修回日期**:2007-03-23
基金项目:国家自然科学基金资助项目(10376013)
作者简介:李 铎(1971—),男,辽宁抚顺人,副研究员,博士,反应堆仪表与控制专业

核动力厂仪表控制系统进入了数字化时代，如何有效地保证并证明执行安全功能的软件自身的安全性和可靠性是实现数字化必须解决的关键问题。按照有关核安全法规的要求，必须通过独立的验证与确认(V&V)过程，这是保证安全软件质量的重要举措和必要步骤，是对安全软件的强制要求。安全软件只有通过了独立 V&V 过程，才能被允许应用于核电站的安全功能。

安全软件的 V&V 是一个按严格的步骤评定软件产品的过程，贯穿于软件产品的整个生命周期。按照软件工程的定义，软件产品的生命周期包括从其构想开始到退出使用为止的整个过程，通常分为软件需求分析阶段、设计阶段、实现阶段、系统集成与测试阶段、安装与调试阶段、运行与维护阶段，在软件生命周期的每个阶段均有相应的 V&V 活动。其中，一个重要步骤是软件实现(Implementation)阶段或称编码(Coding)阶段的 V&V。在软件实现阶段，软件的设计转变为软件代码及相关的机器执行码，软件实现阶段 V&V 的任务是验证和确认这种转变是正确、完整地，保证软件设计说明书中定义的各个软件模块的功能得到了正确实现，未在编码中引入错误，要求的编程规范、习惯和约定在编码中得到了正确贯彻和落实。软件实现阶段 V&V 的主要目的是：跟踪软件设计和代码的实现，验证产生的代码是与设计相符合的；发现设计和需求阶段的问题(错误或含混的描述)导致的软件错误；发现在编码过程中引入的错误。软件实现阶段 V&V 的活动包括对软件单元模块静态分析和动态测试两部分内容。单元测试是对软件基本组成单元逐个进行的测试，单元测试必须执行白盒测试，关注的是软件单元的具体实现、内部的逻辑结构、数据流向等，不能以仅关注输入、输出特性的黑盒测试代替。

本工作着重讨论安全软件单元模块的测试技术，包括如何保证测试的完整性、建立测试环境、建立测试用例、实施单元模块测试的过程。

1 单元测试的完整性

安全软件 V&V 的单元测试必须执行白盒测试，要求对软件单元的逻辑结构实现完整的测试。度量测试完整性的主要指标是测试覆盖率，对安全软件的单元模块测试必须达到 100%覆盖率。

覆盖率的种类很多，一种覆盖率只针对软件代码的一个方面，各种覆盖率是不能互相代替或包含的[1]。对安全软件单元模块逻辑结构的测试需要选择哪几种覆盖率测试才能保证达到完整的测试覆盖要求，是需首先解决的重要问题。对这个问题没有简单、统一的答案，须结合具体的软件代码设计特点分析确定。

本工作在安全软件设计中采取了以下措施：1) 整个软件系统采用模块化设计；2) 单元模块内采用简单结构，没有软件嵌套；3) 不使用操作系统支持，软件源代码完全透明；4) 不使用中断，软件代码的执行流程是完全确定的。在这种前提下，为满足测试完整性要求，在单元测试中选择了语句覆盖、分支覆盖和更改条件判定覆盖。

语句覆盖(Statement Coverage)的含义是，在测试时设计若干测试用例，使程序中的每个可执行语句至少执行 1 次。当测试用例使语句覆盖率达到 100%时，可保证每个可执行语句均得到测试。语句覆盖存在 1 个缺点：软件的设计不可能逐条语句顺序执行，包含很多语句分支，语句覆盖未反映各个语句分支的执行是否正确，因此，需增加分支覆盖。

分支覆盖(Branch Coverage)也称判定覆盖(Decision Coverage)，它的含义是，在测试时设计若干测试用例，使程序中的每个判断条件的真、假值均曾被满足，从而使每个判断至少取真分支和假分支各执行 1 次。当测试用例使分支覆盖率达到 100%时，可保证软件中的每个语句分支均得到测试。分支覆盖还存在以下不足：当复合条件用于控制分支时，虽能保证每个分支均被测试，但由于一个特定分支的执行是由两个或多个条件项的组合逻辑值决定的，分支覆盖可能只测试了其中的 1 种组合，而产生同一逻辑值的其他组合可能未被测试到。

可使用判定条件覆盖(Decision Condition Coverage)实现复合条件的完全测试，其含义是，设计足够多的测试用例，使得判断中每个条件的所有可能值(为真为假)至少出现 1 次，且每个判断本身的判定结果(为真为假)也至少出

现1次。判定条件覆盖理论上可实现复合条件的完全测试,但在组合的条件项增加时,测试用例的增加将呈指数上升,实际上很难实现测试的100%覆盖,因此,在安全软件测试中采用更改条件判定覆盖。

更改条件判定覆盖(Modified Conditions/Decision Coverage,MC/DC)是判定条件覆盖的一个变体,它要求:1)被测试程序每个软件分支的入口点和出口点必须至少被走1次,且每一个程序判定的结果至少被覆盖1次;2)通过分解逻辑操作,程序的判定被分解为基本的布尔条件表达式,每个条件独立地作用于判定的结果,覆盖所有条件的可能结果。应用分解逻辑操作可防止测试用例的指数增长,使软件分支判断式中的复合条件测试实现100%覆盖成为可能。

在安全软件采用模块化、简单结构、不使用中断等条件下,软件的行为具有确定性,在单元模块测试中使用了语句覆盖、分支覆盖和更改条件判定覆盖3个覆盖率指标,即可实现一个安全软件单元完整的逻辑结构测试。

2　单元测试环境的建立

待测试的软件单元本身不是一独立的程序,并未构成一个完整的可运行的软件系统,因此,需为它建立一个测试环境。建立单元测试环境的工作实际就是开发驱动模块和桩模块。在大多数应用中,驱动模块只是一个接收测试数据,并把测试数据传给待测试模块,然后输出相关结果的"主程序"。桩模块的功能是替代那些隶属于待测模块(被调用)的子模块,这种桩模块可能要使用子模块的接口,才能做一些数据操作,并验证输出结果的信息,然后返回。单元测试环境示于图1。

为一个单元模块建立测试环境的主要工作有:1)建立驱动模块,构造最小运行调度系统,用以模拟被测模块的上一级模块;2)建立桩模块,模拟实现单元接口,实现被测单元需调用的其他函数接口;3)模拟生成测试数据或状态,为单元运行准备动态环境。然后,即可开展测试并收集测试结果数据,形成测试报告。

在安全软件的单元测试中使用了软件自动

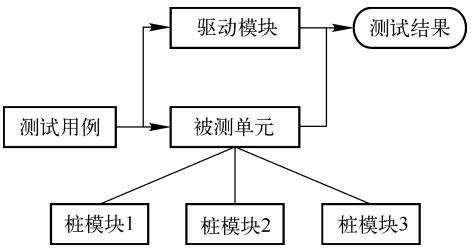

图1　单元测试环境

Fig. 1　Unit testing scheme

测试工具VectorCAST,运行界面示于图2,它提供的功能可辅助建立1个单元测试环境,包括:1)分析需要增加的测试用例,以提高测试结果的语句覆盖、分支覆盖和更改条件判定覆盖的覆盖率;2)通过用户配置,生成必要的驱动模块和桩模块,并连接成一个完整的程序,以实现测试用例的运行;3)自动运行测试用例并采集测试结果,整理测试报告。

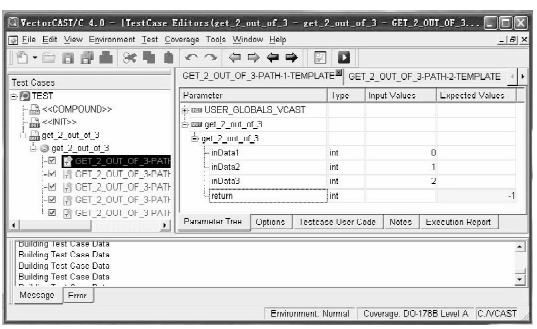

图2　VectorCAST的运行界面

Fig. 2　User interface of VectorCAST

3　测试用例的建立

单元模块测试的主要工作是为每个待测单元模块建立测试用例,包括编辑测试输入值和期望输出值,以判断程序是否正确执行了预定的功能;同时,分析需要设计哪些测试用例,以实现测试结果中语句覆盖、分支覆盖和更改条件判定覆盖的100%覆盖率,达到安全软件逻辑结构要求的测试目标。

每个测试用例的建立是针对具体的软件单元模块进行的,一般需经过以下步骤:1)分析软件的设计说明书,清楚待测软件单元需要完成的功能、测试输入值和期望输出值的取值范围;2)建立第1个测试用例,按软件正常运行时的期望值设计测试输入值,运行测试用例,以

验证待测软件可实现设计功能;3) 结合软件流程图和软件源码分析已建立测试用例的测试结果,建立新的测试用例,以增加测试覆盖率,新的测试用例通过修改测试输入值实现,使其执行待测软件中某个期望的语句、分支,或1个分支中某种期望的组合条件;4) 因每个测试用例的设计目的是为了增加测试覆盖率,在每个新的测试用例编辑并执行测试后,检查当前的测试结果,验证是否增加了测试覆盖率,如果测试覆盖率无变化,说明当前的测试用例是无效的,需进行编辑修改;5) 重复上述3)、4)步骤,直至建立全部需要的测试用例。

测试用例的建立可使用辅助编辑工具,如图2所示为采用VectorCAST编辑测试用例时的界面。

4 单元测试的实现

应用上述讨论的软件单元测试技术,在一个数字化保护系统安全软件V&V的过程中完成了单元测试,下面结合1个示例单元模块的测试过程说明安全软件单元测试的实现。

示例程序"get_2_out_of_3.c"是保护系统逻辑符合单元执行"2/3"(三取二)符合运算的函数,软件的源代码为:

```
const   ERROR=-1;
/* get get_2_out_of_3 of three input data,
   the acceptable value of the result is 0 or 1,
   with the meanings of logic trip or normal,
   respectively */
int   get_2_out_of_3(int inData1, int inData2,
   int inData3)
{int mData[3];
int reData;
/* initiate loval parameters with input data */
mData[0]=inData1;
mData[1]=inData2;
mData[2]=inData3;
/* 2_out_of_3 calculation, if there are no
   any pair data are equal among input data,
   return ERROR message */
if ( mData[0] == mData[1])
reData = mData[0];
else
if ( mData[1] == mData[2] )
reData = mData[1];
else
if ( mData[2] == mData[0] )
reData = mData[2];
else
return ERROR;
/* if return value is out of acceptable values,
   return ERROR message */
if (reData == 0 || reData == 1 )
return reData;
else
return ERROR;}
```

程序流程框图示于图3,程序完成的功能为:1) 首先根据输入变量初始化3个本地变量mData[0,1,2];2) 对mData[0,1,2]中的每两个变量逐一比较,如果有两个变量值相等,则存入待返回变量reData,否则,返回ERROR("-1");3) 判断reData结果是否合适(值为"0"或"1"),如果合适,则返回reData,否则,返回ERROR("-1")。

为实现函数"get_2_out_of_3"测试,应用VectorCAST工具辅助建立软件测试环境,工作步骤如下:1) 设置待测函数的编译环境,使它与软件"get_2_out_of_3.c"在生成最终产品时应用的编译环境一致;2) 函数"get_2_out_of_3"没有对其它函数的调用,不需要建立桩模块;3) VectorCAST能自动生成驱动模块,并与待测函数"get_2_out_of_3"连接,准备执行测试用例;4) 建立并编辑测试用例,由VectorCAST自动执行,记录测试结果。软件测试的主要工作是逐个编辑测试用例。在本示例程序中,测试输入值是3个整数,期望输出值是3个数的"2/3"结果("0"或"1"),遇到错误时输出ERROR("-1")。

下面分析需要为示例程序设计哪些测试用例。如图3所示,示例程序的代码流程中有4个分支点,需要设计测试用例使每个分支点取真(T)、取假(F)各1次;同时,分支点④是复合条件判断,需要使每个判断条件(reData==0和reData==1)取真(T)、取假(F)各1次。因此,设计了5个测试用例,每个测试用例的内容和测试的分支点总结列于表1。

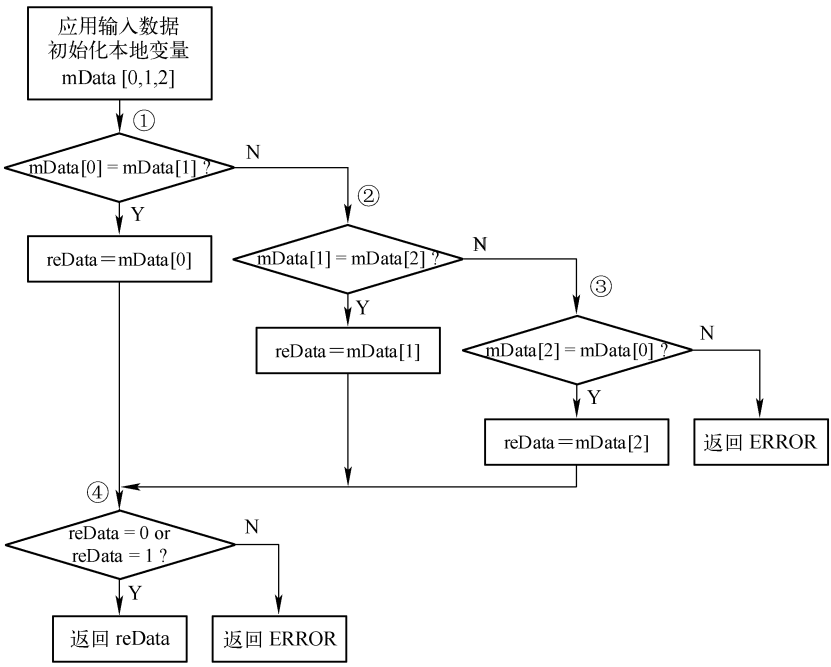

图 3　示例程序流程框图

Fig. 3　Flow diagram of example program

测试用例建立并编辑完成后(编辑界面见图 2),VectorCAST 可自动依次执行每个测试用例。在测试用例执行过程中,VectorCAST 检查单元模块的实际输出结果是否与期望值一致,统计执行每个测试用例时的软件覆盖情况,并生成 html 格式的测试结果报告。如果测试用例中存在设计输出值与期望值不符,则说明该单元模块代码存在功能错误;如果测试结果未实现 100%覆盖,则说明对该单元模块的逻辑结构没有实现完全测试。

表 1　示例程序的测试用例设计

Table 1　Testing case design for example program

序号	测试用例				测试的分支点					
	Data[0]	Data[1]	Data[2]	返回	①	②	③	④	reData==0	reData==1
1	0	1	2	−1	F	F	F	—	—	—
2	1	0	1	1	F	F	T	T	F	T
3	2	1	2	−1	F	F	T	F	F	F
4	0	1	1	1	F	T	—	T	F	T
5	0	0	1	0	T	—	—	T	T	F

执行上述测试用例的测试报告为:

```
Code Coverage for Unit: get_2_out_of_3
Coverage Type: level_a
Test Case: Aggregate
const  ERROR=-1;
/* get_2_out_of_3  of three input data, the
    acceptable value of the result is 0 or 1,
    with the meanings of logic trip or normal,
    respectively */
int get_2_out_of_3(int inData1, int inData2,
    int inData3)
{int mData[3];
```

```
int reData;
/* initiate loval parameters with input data */
10    (T)  get_2_out_of_3
11     *     mData[0]=inData1;
12     *     mData[1]=inData2;
13     *     mData[2]=inData3;
/* 2_out_of_3 calculation, if there are no any
   pair data are equal among input data,
   return ERROR message */
14    (T)(F)    if ((
14.1    (T)(F)  mData[0]== mData[1]))
15    {*      reData = mData[0];}
16    (T)(F)     else
       if ((
16.1   (T)(F)  mData[1] == mData[2]))
17   {*   reData = mData[1];}
18    (T)(F)  else
      if ((
18.1   (T)(F)  mData[2] == mData[0]))
19     {*     reData = mData[2];}
      else {
111      *      return ERROR;}
/* if return value is out of acceptable values,
   return ERROR message */
113      (T)(F)         if ((
113.1   (T)(F)  reData = 0 ||
113.2   (T)(F)  reData = 1 ))
114      {*     return reData;}
          else {
116      *      return ERROR;}}
```

其中：* 表示该语句通过测试，(T)和(F)表示分支点取真和取假经过测试。

设计的测试用例实现了 3 个覆盖(语句覆盖、分支覆盖和更改条件判定覆盖)的 100%覆盖率。在实际的软件测试中，一般情况下，不可能 1 次编写测试用例就能完全实现 100%覆盖率，需根据测试报告的统计结果增加新的测试用例。

5 结论

软件单元测试是安全软件 V&V 过程中的重要环节。本工作对安全软件单元测试的实现技术进行了初步研究，着重讨论了如何保证测试的完整性、建立测试环境及测试用例、实施单元模块测试的过程等方面。本文所讨论的技术已在某工程数字化保护系统安全软件 V&V 过程中应用于单元测试工作中。

参考文献：

[1] 古乐，史九林. 软件测试技术概论[M]. 北京：清华大学出版社，2004.