单元测试手册

(PATAC 版本)

Version1.1

Table of Contents

1.单元测试简介

1.1 单元测试定义

单元测试是软件开发过程中的一种测试方法，用于验证软件中最小单元（即：最小可测试部分，如函数、方法等）的输入和输出是否符合预期，通常会涵盖各种情况，包括正常情况、边界情况和异常情况，以确保软件在各种情况下都能正确运行。

相比于后道的软件集成测试和黑盒功能测试，单元测试可以通过桩函数等方式隔离特定代码，更容易模拟验证异常或特定工况下异常值/无效值/极限值处理逻辑，以尽早发现并修复代码中的错误，减少在后期阶段排查和修复问题的时间及成本，确保代码的质量和可靠性。

其中，需要重点关注客户强感知或实时性及安全性要求较高的功能模块，确保车载软件在各种复杂场景下都能正常工作，从而提升客户体验感及驾乘安全性。

1.2 单元测试范围及要求

单元测试包括所有自研的手写代码（包括平台代码，以及基于第三方 API 函数上更新的内容）和创建的模型，不包括 OSS 和纯第三方代码。模型须进行 MIL （Model-In-the-Loop）模型在环测试、SIL（Software-In-the-Loop）软件在环测试、PIL（Processor-In-the-Loop）处理器在环测试。

对于 ASIL **非 C&D** 的零件，语句覆盖率（Statement Coverage）和分支覆盖率（Branch Coverage）须达到 100%。对于 ASIL C & D 的零件，语句覆盖率（Statement Coverage）、分支覆盖率（Branch Coverage）和修改条件判定覆盖率（Modified Condition/Decision Coverage, MC/DC）须达到 100%。

特例说明:

（1）如在 OSS/协议栈等第三方函数基础上新增自研代码，仅验证增加的自研代码，无需验证第三方函数已有的代码，语句/分支覆盖率可不达 100%（P.S. 为避免歧义，如后续有类似情况，请作必要的备注说明）。

（2）如因工具限制性导致单元测试的语句/分支覆盖率未达 100%，可通过 code review 等方式确认工具未验证部分的代码逻辑正确性。P.S. 请和工具供应商沟通确认非工具使用问题，确实是工具限制性。

1.3 单元测试交付物

单元测试交付物包括详细单元测试结果和覆盖率统计结果，且为工具原始生成的PDF 或 HTML 等不可更改格式的测试结果。

注：如涉及 1.2 中的特例情况（即在 OSS/协议栈等第三方函数基础上新增自研代码 和工具限制性），请附加必要的覆盖率说明文档 及 未覆盖代码的 code review 结果等。

1.4 单元测试建议

（1）单元测试实现方式

建议使用单元测试工具实现自动化单元测试。单元测试工具可以基于导入的被测代码自动生成单元测试用例框架/默认测试脚本，并通过丰富的桩函数辅助加速测试过程，减少人工操作，提高测试覆盖率及测试效率。

（2）单元测试用例设计

对于单元测试用例设计，为了避免遗漏需求 或 未按需求正确实现，应根据原始需求 或 基于原始需求细化的详细设计文档 设置具体的输出期望值），并以更多的负向思维设计单元测试用例，即输入无效值/边界值/异常值去验证函数的逻辑完整性。

（3）单元测试脚本维护

应在测试用例名称中概要总结测试逻辑或测试点，便于内部传承及维护单元测试脚本。同时，根据代码迭代，周期更新单元测试脚本，避免脚本因缺失维护而废弃，导致无法对后续软件版本进行持续的自动化单元测试。

2.单元测试设计方法

2.1 等价类划分

等价类划分是将数据划分为不同分区（也称为等价类），包含有效值和无效值。有效值是组件或系统应接受的值，包含有效值的等价分区称为“有效等价类”。无效值是组件或系统应拒绝的值，包含无效值的等价分区称为“无效等价类”。

当测试用例中使用无效等价类时，应单独测试，即不能与其他无效等价类组合，以避免无效值之间互相影响，导致无法发现其他失效。

举例：成绩统计函数，输入变量是：分数，输出变量：等级（优秀、合格、不合格）。

分数	等级
0-59	不合格
60-79	合格
80-100	优秀

对该函数进行单元测试时，可应用如下等价类设计输入变量值及输出变量期望值：

等价类	输入变量值	输出变量期望值
有效等价类	32	不合格
	65	合格
	88	优秀
无效等价类	-1	无效成绩
	101	无效成绩

2.2 边界值分析

边界值分析是等价类划分的扩展，但仅适用于由数字或顺序数据组成的有序的等价类。等价类的最小和最大值是其边界值。在区域边界上的行为往往比区域内的行为更容易出现错误（比如边界值被忽略 或 归到其他等价分区），需在开发过程中重点确认。

举例：成绩统计函数，输入变量是：分数，输出变量：等级（优秀、合格、不合格）。

分数	等级
0-59	不合格
60-79	合格
80-100	优秀

对该函数进行单元测试时，可应用如下边界值设计输入变量值及输出变量期望值：

边界值 1		边界值 2		边界值 3		边界值 4	
输入变量	输出变量期望值	输入变量	输出变量期望值	输入变量	输出变量期望值	输入变量	输出变量期望值
-1	无效成绩	59	不合格	79	合格	99	优秀
0	不合格	60	合格	80	优秀	100	优秀
1	不合格	61	合格	81	优秀	101	无效成绩

注：也可根据边界值分析思路，在原等价类设计上进行优化补充：

等价类	输入变量值	输出变量期望值
有效等价类	0	不合格
	59	不合格
	60	合格
	79	合格
	80	优秀
	100	优秀
无效等价类	-1	无效成绩
	101	无效成绩

2.3 判定表（决策表）

判定表通常应用于无序的输入（比如枚举类型） 或 输入组合。当建立判定表时，需要罗列输入和输出之间的对应关系。其优点在于能够有效地处理多条件判断的复杂性，减少测试用例数量的同时增加覆盖度，适合于需要考虑多种组合情况的决策逻辑，帮助发现可能存在的逻辑错误 或 逻辑漏洞。

举例：星期输入函数，输入变量是：第一个字母 & 第二个字母，输出变量：星期几。当输入第一字母为：m / w / f 时，会直接输出打印 Monday / Wednesday / Friday；当输入第一字母为：t / s 时，需要再输入第二个字母，即：

Input：m，print：Monday

Input：w，print：Wednesday

Input：f，print：Friday

Input：tu，print：Tuesday

Input：th，print：Thursday

Input：sa，print：Saturday

Input：su，print：Sunday

在设计单元测试用例时，除了上述有效输入，还需考虑无效输入，以确认函数的逻辑完整性，如下表：

输入变量		**输出变量**
第一个字母 i	**第二个字母 j**	**星期**
m	/	Monday
w	/	Wednesday
f	/	Friday
t	u	Tuesday
t	h	Thursday
s	a	Saturday
s	u	Sunday
a	/	无效输入（print: Error）
t	c	无效输入（print: Error）
s	k	无效输入（print: Error）

2.4 真值表

如果每个输入变量 或 原子条件都是一个离散的布尔值，可采用真值表来设计单元测试用例。为了简化测试用例数量，可使用改进的条件/判定 (MC/DC) 思路进行优化。即：假设有 N 个唯一的、独立的原子条件，改进的条件/判定测试通常可通过 N+1 个不同的测试用例来实现，且均为成对的测试用例，以表明单个原子条件可以独立影像判定的结果。

举例：if (A or B) and C then…，在列出完整真值表后，可筛选出如下能表明单个原子条件独立影像判定的测试用例，用于统计 MC/DC 覆盖率，且有效简化了测试用例数量。

	A	B	C	(A or B) and C
测试用例 1	真	假	真	真
测试用例 2	假	真	真	真
测试用例 3	假	假	真	假
测试用例 4	真	假	假	假

A 独立影像判定：测试用例 1 VS 测试用例 3

B 独立影像判定：测试用例 2 VS 测试用例 3

C 独立影像判定：测试用例 1 VS 测试用例 4

2.5 错误推测

错误推测是指基于测试开发人员的专业知识和技能经验来识别错误、缺陷和失效，包括：编程语言的语义语法；应用软件/代码常规的运行逻辑；软件开发过程经常引入的错误类型；在其他应用软件中发生的失效等。

最典型的例子是**空指针问题**。编程语言中指针初始化是指给所定义的指针变量赋初值。指针变量在被创建后，如果不被赋值，它的缺省值是随机的。如果指针变量取到的是空值，就会导致软件异常。因此，对于入参为指针变量的函数进行单元测试时，需将入参赋 NULL 进行空指针相关的 negative test，以验证函数是否已有必要的判空逻辑。

3. 总结

设计单元测试用例的方法有很多，并不仅限于第 2 章节提到的内容，但主要的宗旨是需要进行**必要的负向测试**（即 negative test，包括无效值、边界值、空值等），以尽早发现代码逻辑漏洞，避免溢出卡死、未预期的功能异常等问题。

因此，以负向思维设计单元测试用例，能够覆盖验证更多的边界及异常 case，在发现问题的时间上早于后道的软件集成测试和黑盒功能测试；而且可以通过软件层面的桩函数等模拟异常输入去发现最小单元/基本组件中的逻辑错误，实现难度上相对较低，运行所需资源较少，且速度较快（P.S. 一个单元测试用例可在毫秒级执行完毕），能够更高效地发现代码问题，有效降低问题修复成本，并提升软件的稳定性及鲁棒性。

Appendix A：单元测试手册更新记录

Version	Revision Date	Author	Description of Change
1.0	6/28/2024	Gu Yan	Initial Release
1.1	7/3/2024	Gu Yan	Update chapter1.4 & chapter2.3, add chapter3