avionicstech@avic.com

DOI:10.12175/j.issn.1006-141X.2023.01.07

基于 TestBed 和 Visual Studio 对 QT 程序单元测试的研究与实现

马慧芳，费自强，范孝彬

（中国航空无线电电子研究所，上海 200233）

[摘　要] 针对测试工具 TestBed 无法动态执行 QT 程序单元测试用例的问题，文章通过介绍单元测试定义、测试原理以及 TestBed 工具的使用原理、并指出了 TestBed 工具对 QT 程序进行单元测试存在的问题，最终提出了一种结合 TestBed 和 Visual Studio 对 QT 程序进行单元测试的方法，并给出了方法的详细设计与实现步骤，为 QT 程序动态执行单元测试提供一种简单、便捷的实现方式。实际结果表明，结合 TestBed 与 Visual Studio 对 QT 程序的单元测试方法显著提高了单元测试的效率，减少了单元测试的成本。

[关键词] QT；单元测试；TestBed；Visual Studio

[中图分类号] TP311.52　　[文献标识码] A　　[文章编号]1006-141X(2023)01-0042-05

Research and Implementation of Unit Testing of QT Program Based on TestBed and Visual Studio

MA Hui-fang，FEI Zi-qiang，FAN Xiao-bin

（China National Aeronautical Radio Electronics Research Institute, Shanghai 200233, China）

Abstract：Aiming at the problem that test tool TestBed cannot dynamically implement unit testing of QT program, introduces the definition of unit testing and the principle of unit testing and TestBed tool are introduced, and the problems of unit testing of QT program by TestBed tool are pointed out, then a method that combines TestBed and Visual Studio to do unit testing of QT program is proposed, and detailed steps of design and implementation and a simple implementation method for unit testing of QT program are described. As the result of practical application, the method combining TestBed and Visual Studio in unit testing of QT program improves the efficiency of unit testing, decreases cost of unit testing.

Key words：QT; unit testing; TestBed; Visual Studio

QT 作为图形用户界面的 C++ 开发框架，在界面显示领域得到了广泛的应用。其直观、强大的应用程序编程接口（API：Application Programming Interface）以及跨平台等特性也深受软件开发者的厚爱，目前大多数的地面站软件采用了基于 QT 进行开发，随着 QT 市场的不断扩大，则针对 QT 程序的测试验证工作也更显得至关重要。

收稿日期：2021-12-14

引用格式：马慧芳，费自强，范孝彬 . 基于 TestBed 和 Visual Studio 对 QT 程序单元测试的研究与实现 [J]. 航空电子技术 ,2023,54(1)： 42-46.

在软件测试阶段，一般最先开始的是单元测试，跟踪阶段以详细设计说明为基础，借助相应的单元测试工具（例如 TestBed、Cantata、GJSTest 等，目前在航空软件测试领域中单元测试的主要工具是 TestBed），通过设计单元测试用例、执行待测程序来跟踪比较实际结果与预期结果的差异。单元测试要求对语句、分支的覆盖达到 100%，极大的降低了缺陷出现率，缺陷越早被发现，后期的测试和维护成本越低；而且单元测试可以平行开展，多人可以同时开展。进行充分的单元测试，是提高软件质量、降低开发成本的必由之路。

本文介绍了单元测试的定义、测试原理、以及 TestBed 的使用原理，然后根据 TestBed 作 QT 程序单元测试时出现的问题，提出了一种结合 TestBed 和 IDE 工具 Visual Studio，对 QT 程序进行单元测试的方法，并给出详细的实现步骤。该方法已经在多个型号的地面站软件测试中得以应用，明显地提高了 QT 程序单元测试的效率，同时也为 QT 程序的单元测试领域提供了一种研究的方向。

1　概述

1.1 单元测试

单元测试是对软件基本组成单元进行的测试，它主要通过对代码的逻辑结构进行分析来设计测试用例 [1]，GJBZ141 明确指出软件单元测试的目的是检验每个软件单元能否正确地实现设计说明中的功能、性能、接口和其他设计约束等要求，发现单元内可能存在的各种错误。

实际测试过程中，单元函数不能独立运行，需要构造一个运行环境才能完成测试。单元测试环境由驱动函数、被测试单元函数和桩函数组成 [2]，如图 1 所示。

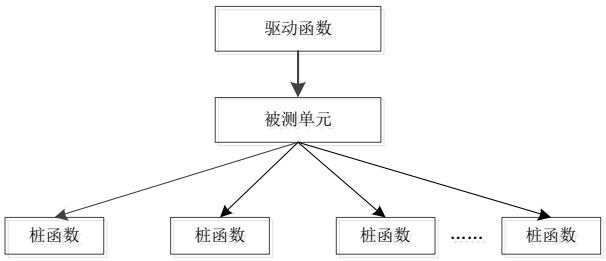

图 1 单元测试环境组成框图

驱动函数是一个主函数，其作用是将测试用例数据传送给被测试单元函数，并引导被测试单元函数运行，保存运行结果；桩函数是一个构造子函数，是用来替代被测试单元函数调用的子函数，是为了隔离子函数对被测试单元函数的影响而构造的，它可以是一个“空”子函数或是具有对接口做少量操作的子函数（少量操作是指根据桩函数的输入输出要求对其进行输入设置、输入检查、输出赋值、输出检查 [3-4]）。

1.2 TestBed 工具

TestBed 是包含一系列静态测试、动态测试功能的工具集，可以进行代码规则检查、单元测试、覆盖率分析等，其执行单元测试的原理图如图 2 所示。首先，TestBed 加载被测单元源文件，通过静态分析后生成插桩文件（插桩文件是给源文件所有分支的路径打个标记，并将已执行的路径的标记输出到执行数据 .exh 文件）；其次，通过单元测试指令，TBRun 自动加载插桩文件，并利用 TestBed 自动生成的驱动函数、桩函数来一起将单元测试用例动态执行起来，并自动生成测试结果文件；然后，TestBed 自动查找相应目录下的执行数据 .exh 文件并进行覆盖率分析；最后，自动输出覆盖分析报告。

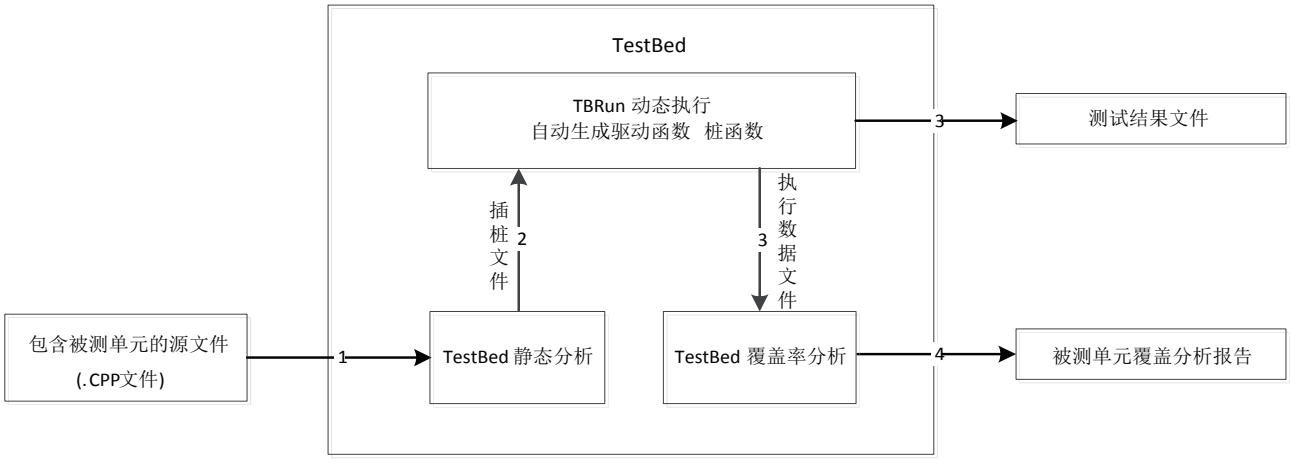

图 2 TestBed 单元测试原理图

2 存在的问题以及解决方法

TestBed 是一款功能齐全的单元测试工具，但是对 QT 程序的支持并不是很好，主要存在以下问题：

（1）QT 框架本身封装很多层的相互调用，其嵌套调用的层级比较深，而 TestBed 无法很好的支持多层嵌套调用，经常出现运行卡死的情况；

（2）QT 程序通过 TestBed 工具生成的插桩文件较为复杂，不便于对程序的调试。

Visual Studio 是一个集成开发工具环境（IDE：Integrated Development Environment），QT 的工程可以在 Visual Studio 中编译链接和执行，同时 Visual Studio 为 QT 程序提供了强大的调试环境，让运行 QT 程序变得简洁明了。

综上所述，文章提出一种结合 TestBed 与 Visual Studio 来实现对 QT 程序进行单元测试的方法，使用 Visual Studio 作为 QT 程序的编译器，替代 TestBed 自带的编译器。即选择使用 TestBed 生成插桩文件和覆盖分析报告，而选用 Visual Studio 来作为执行单元动态测试用例的开发环境，很好地解决了 TestBed 工具无法支持 QT 程序单元动态调试的问题。其原理分析如图 3 所示。

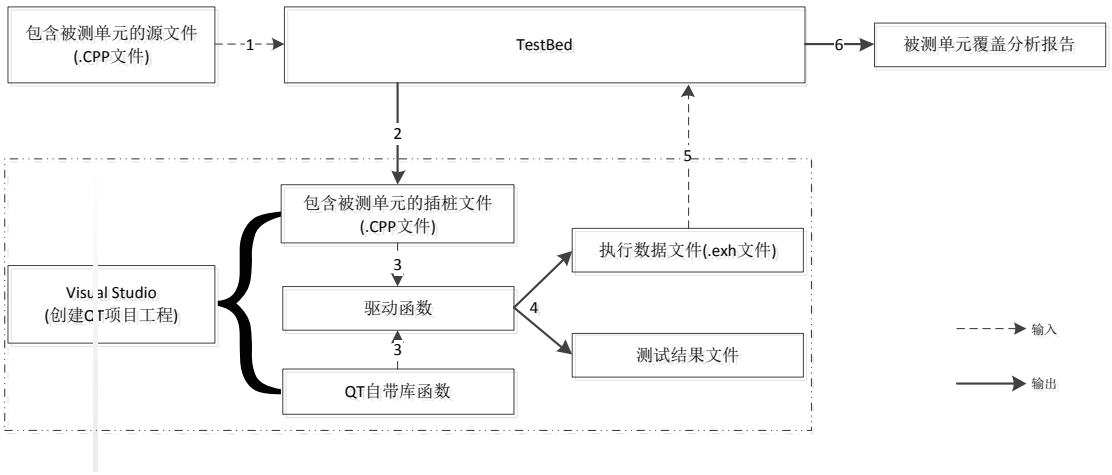

图 3 QT 程序单元测试的原理图

第一步：利用 TestBed 加载源文件，生成包含被测单元路径标记的插桩文件；第二步：在 Visual Studio 中创建 QT 项目工程，将插桩文件包含进来；第三步：在 QT 项目工程中编写驱动函数，在驱动函数中调用被测单元，并在驱动函数中引入插桩文件、QT 自带的库函数，使整个 QT 项目工程编译链接通过；第四步：运行驱动函数，直接生成执行数据 .exh 文件和测试结果文件；第五步：由 TestBed 加载执行数据 .exh 文件进行覆盖率分析；第六步：TestBed 输出被测单元的覆盖分析报告。

3 设计与实现的详细过程

（1）在本地电脑配置好 Visual Studio 和 QT 的环境变量 (配置过程需要注意 QT、Visual Studio 的版本必须与实际开发环境保持一致，文章以 QT 5.0、Visual Studio 2010 版本为例)；将 QT 安装目录 msvc2010 下的 bin 文件夹、以及 Visual Studio 安装目录下的 Common7\IDE 路径添加到 Path 环境变量中；

（2）在 TestBed 中的配置文件 Testbed.ini 中增加 QT 的编译器的配置；

（3）TestBed 加载包含被测单元的源文件，编译器选择 QT，点击“Unit Test only”按钮进行分析生成插桩文件，插桩文件中每一个被测单元前都存在唯一的标识；

（4）在 Visual Studio 中创建 QT 工程，主函数 main() 所在文件默认是 main.cpp，将生成的插桩文件拷贝至工程目录中，main.cpp 将插桩文件 include 进来；

（5）使用 define 定义被测单元对应的唯一标识，这样被测单元才可以被识别和调用；

（6）main.cpp 中定义全局变量 g_tcNumber 表示当前正在执行的用例序号；

（7）main.cpp 中编写驱动函数如下：

第一步：利用 TestBed 加载源文件，生成包含

被测单元路径标记的插桩文件；第二步：在 Visual Studio 中创建 QT 项目工程，将插桩文件包含进来；第三步：在 QT 项目工程中编写驱动函数，在驱动函数中调用被测单元，并在驱动函数中引入插桩文件、QT 自带的库函数，使整个 QT 项目工程编译链接通过；第四步：运行驱动函数，直接生成执行数据 .exh 文件和测试结果文件；第五步：由 TestBed 加载执行数据 .exh 文件进行覆盖率分析；第六步：TestBed 输出被测单元的覆盖分析报告。

3　设计与实现的详细过程

（1）在本地电脑配置好 Visual Studio 和 QT 的环境变量 (配置过程需要注意 QT、Visual Studio 的版本必须与实际开发环境保持一致，文章以 QT 5.0、Visual Studio 2010 版本为例)；将 QT 安装目录 msvc2010 下的 bin 文件夹、以及 Visual Studio 安装目录下的 Common7\IDE 路径添加到 Path 环境变量中；

（2）在 TestBed 中的配置文件 Testbed.ini 中增加 QT 的编译器的配置；

（3）TestBed 加载包含被测单元的源文件，编译器选择 QT，点击“Unit Test only”按钮进行分析生成插桩文件，插桩文件中每一个被测单元前都存在唯一的标识；

（4）在 Visual Studio 中创建 QT 工程，主函数 main() 所在文件默认是 main.cpp，将生成的插桩文件拷贝至工程目录中，main.cpp 将插桩文件 include 进来；

（5）使用 define 定义被测单元对应的唯一标识，这样被测单元才可以被识别和调用；

（6）main.cpp 中定义全局变量 g_tcNumber 表示当前正在执行的用例序号；

（7）main.cpp 中编写驱动函数如下：

```
int TC_1() // 第 1 个测试用例
{
        g_tcNumber = 1;
设置被测单元测试的输入数据 ;
调用被测单元函数 ;
对函数返回值、全局变量、成员变量等进行判断，全部和预期结果一致则返回 1，否则返回 0;
}
int TC_2()  // 第 2 个测试用例
{
        g_tcNumber = 2;
设置被测单元测试的输入数据 ;
调用被测单元函数 ;
对函数返回值、全局变量、成员变量等进行判断，全部和预期结果一致则返回 1，否则返回 0;
}
int TC_3(){……}  // 第 3 个测试用例
int TC_4(){……}  // 第 4 个测试用例
……
int main(int argc, char *argv[])  // 驱动函数
{
调用 TC_1()、TC_2()、TC_3()、TC_4() 等函数，并记录成功失败的结果；
打印输出用例总数、失败用例个数、失败用例编号等。
}
```

实际在调试 main 函数时会遇到很多问题，典型的问题则是提示很多 QT 底层库函数未实现的错误，解决措施则是在 main.cpp 中重写相应库函数，函数体为空即可。

（8）运行代码，成功执行后会在工程目录下生成相应的执行数据 .exh 文件，使用 TestBed 工具将 .exh 文件加载并进行覆盖率分析，最终生成被测单元的覆盖分析报告；

（9）若被测单元的语句 / 分支覆盖未完全，则查找原因，若是因为缺少测试用例导致的则在第 7 步中增加，然后编译运行，重复第 8 步重新生成覆盖分析报告。

经过上述操作步骤，对某型地面站分布式执行平台软件实施单元测试：该软件使用 QT 程序开发，代码规模十万余行，涉及近千个单元。抽样其中一百个单元进行量化统计，每个单元对应的代码规模、单独使用 TestBed 工具进行单元测试消耗的时间（即传统方法单元测试消耗时间 / 小时）、使用本方法进行单元测试消耗的时间（即改进方法单元测试消耗时间 / 小时）如表 1 所示。

表 1 各单元执行单元测试消耗时间对照表

单元名称	代码规模 / 行	传统方法单元测试消耗时间 / 小时	改进方法单元测试消耗时间 / 小时
单元 1	84	1.17	0.70
单元 2	473	12.36	8.65
单元 3	62	1.83	1.10
单元 4	612	15.14	10.60
单元 5	373	11.64	8.15
单元 6	8	0.42	0.25
单元 7	135	2.75	1.65
单元 8	123	2.83	1.70
单元 9	369	5.83	3.50
单元 10	456	12.64	7.95
		……	
单元 91	210	3.42	2.05
单元 92	16	0.28	0.21
单元 93	230	4.33	2.60
单元 94	343	6.42	3.85
单元 95	12	0.38	0.28
单元 96	329	4.83	2.90
单元 97	623	15.00	10.50
单元 98	378	7.58	4.55
单元 99	23	0.67	0.44
单元 100	468	10.79	7.55
合计	24589	283.4	191.9

若单独使用 TestBed 工具进行单元测试，在调试阶段由于无法准确定位错误的代码行，需要不断在被测单元中增加输出打印语句，导致被测单元不断被重复编译链接，消耗了大量的调试时间，故而完成该软件单元测试消耗的时间大大增加。而应用本方法可以很快定位出现编译错误的代码行，进而修正，根据表 1 抽样数据统计得知应用改进后的方法时间成本降低了 32.3%。

4 结论

单元测试是整个软件测试验证过程中的一个重要环节，确保了软件的实际功能与详细设计说明的一致性 [5]。而基于目前面对 QT 程序单元测试复杂性、难以调试等问题，文章研究并实现了 TestBed 结合 Visual Studio 的测试方法，很好的解决了对 QT 程序的单元测试：使用 TestBed 工具生成插桩文件和覆盖分析报告的功能，完美的避开了 TestBed 动态执行单元测试引起的编译调试问题；利用 Visual Studio 工具编写驱动函数，将被测单元动态的调用起来，而 Visual Studio 与 QT 之间是无缝衔接，编译调试过程极为方便，可以通过单步调试代码定位编译或运行时的错误，使得整个单元测试过程变得极为简单、易操作。

该单元测试的方法已经在多个型号地面站软件测试阶段应用，极大地提高了单元测试效率、节约了单元测试的成本，为今后 QT 程序的单元测试奠定了良好的基础。

参 考 文 献

[1] 陈站华 . 软件单元测试 [J]. 无线电通信技术 , 2003, 29(5): 50-51.

[2] 张巍 , 尹海波 , 孙立财 , 等 . 软件的单元测试方法 [J]. 光电技术应用 , 2006, 21(2): 36-38.

[3] 郭荣 . 基于 Testbed 的 C++ 单元测试 (动态测试) 方法 [J]. 网络安全技术与应用 , 2014, (3): 55-59.

[4] 任俊 . 软件单元测试及测试用例设计 [J]. 科技与企业 , 2013, (4): 293-295.

[5] 陈静 . 单元测试在软件开发过程中的作用 [J]. 舰船电子对抗 , 2006, 29(3): 63-65.