实施白盒测试的几个误区

2006-8-30

白盒测试作为软件质量保证中的重要一环，对产品稳定性起到至关重要的影响，不幸的是，由于实施白盒测试有较高技术难度，该软件过程常被嵌入式厂商忽略，因为难于实施，所以容易失败，失败后产生畏惧心理，就更不愿意进一步去尝试，如此形成恶性循环。更令人担忧的是：产品进度很少有不紧张的，大家习惯于在产品发布前补做测试，甚至把测试留给用户，成天陷于紧张的救火工作。研发进度总会被许多意外打断，在最终交付日要严防死守的前提下，白盒测试自然最先被喀嚓掉了。

本篇总结实施白盒测试的几个主要误区，我们先从认识上端正对白盒测试的看法。

误区之一：白盒测试太耗时间，不值得一做

这是救火式团队对白盒测试的最典型看法。

评估白盒测试值不值得去做，不只要看白盒测试能发现多少问题，还要看白盒方式下发现问题并解决它的工作效率。另外，在确定的质量标准下，还要分析不做白盒测试，以其它测试方式（如系统测试）代替是否能达到目标，也即：让产品达到能满足市场的稳定程度，只做系统测试需要多少时长，若改成一半时间做白盒测试，另一半做系统测试，看看这两种方式的测试总投入差别有多大？

依据实际经验，成功的白盒测试与不做白盒测试相比，测试投入应节约 1/3 以上。当然，这个对比是产品要保证较好发布质量的前提下才成立的，如果不做测试，产品一调通就发布，那没得比了，这样测试投入是最节约的。

以上观点的详细论述请参考《为什么要做白盒测试》。

误区之二：系统测试可以发现所有问题，不必做白盒测试

从理论上讲，系统测试是可以代替白盒测试的，但现实操作中，让系统测试代替白盒测试的代价太高。白盒测试直接面对函数内的各个分支，如果在系统测试阶段设计用例，也让每个分支情况都能覆盖到，恐怕要付出数万乃至数百万倍的测试投入，现实情况是不可能这么操作的。

白盒测试不可或缺，不仅因为白盒测试的发现与解决问题效率很高，也因为白盒测试独具特点，能发现其它测试手段很难发现的问题，比如逻辑问题、边界条件、变量未初始化、内存越界等问题。

更详细内容请参考《为什么要做白盒测试》一文。

误区之三：白盒测试必须在真实环境下进行

近代量子力学有一个海森堡测不准原理，讲的是某个粒子的位置与动量不能同时被测量出来，由测量存在干扰，对其中一个参数测量越准确，另一个参数就越不准确。测不准原理在我们日常生活中很常见，比如要测试某物质的导电性，我们串联接上一个安培计来观察电流，但是，安培计本身也带电阻，导致测不准，测量值会偏小。

在软件测试领域也存在类似情况，比如要测试系统的 CPU 占用，于是添加代码统计 CPU 占用率，但添加的代码运行时，它本身也会消耗 CPU。再如，为了实施白盒测试，必然追加测试代码，比如：构造特定运行环境、替换桩函数使之在特定情况下返回特定值，这些都改变了被测对象自身的特性，追求完全准确的测试非常困难。所以，我们并不是一定要追求在真实环境下做测试，而是要评估非真实环境（或仿真环境）下的测试对最终结果能产生多大偏离。

本人曾辅导过一个白盒测试项目，该项目是一个中间件系统，支持 Windows 与 RT-Linux 跨平台运行，当时本人竭力推荐在 VC 环境下做测试，但产品经理断然否决了这个提议，原因是他们有重要客户要跑 RT-linux，测试就必须在 RT-linux 下进行。这下可好，也怪他们的调测环境不争气，且不论 RT-Linux 下缺乏测试手段，单单下载程序到驻留实时 linux 系统的单板，一次就要 5 分钟，如此测试完全可以想见最终结果是什么！后来我们统计该产品与 RT-Linux 平台相关的 API 总共不到 15 个，基本上都与任务调试、内存申请分配相关，完全可以改用 Windows 测嘛，整整两万多行源码，仅仅担心几个 API 不真实，而最终导致白盒测试做不下去，可悲呀。

根据我们多年实践经验，嵌入式产品的白盒测试大可不必非得在真实环境下进行，而且，绝大部分情况下，只有移到仿真环境下白盒测试才能做成功。这方面我们有太多经验教训了，众多坚持上单板做单元测试的尝试都失败，而在仿真平台下的白盒测试，成功率接近百分之百。

我们先看看上单板做测试的几个致命问题：

1. 测试成本高

   上单板做测试，搭建测试环境的成本比较高，调试与测试都很麻烦，上面提到的每次测试都要花 5 分钟下载程序就是一个例子。

2. 面对初始代码，导致工作效率低下

   因为白盒测试经常要面对初始代码，尤其是单元测试，刚写完的代码很不稳定，就要做单元测试，测试中代码跑飞是常事，更加剧了测试成本飞涨。而且，目前多数在单板运行的实时系统，都不具备像个人 PC 那样拥有完善的异常捕获与处理的能力，测试支撑手段必然不够稳健，经常要复位重起，严重影响测试效率。

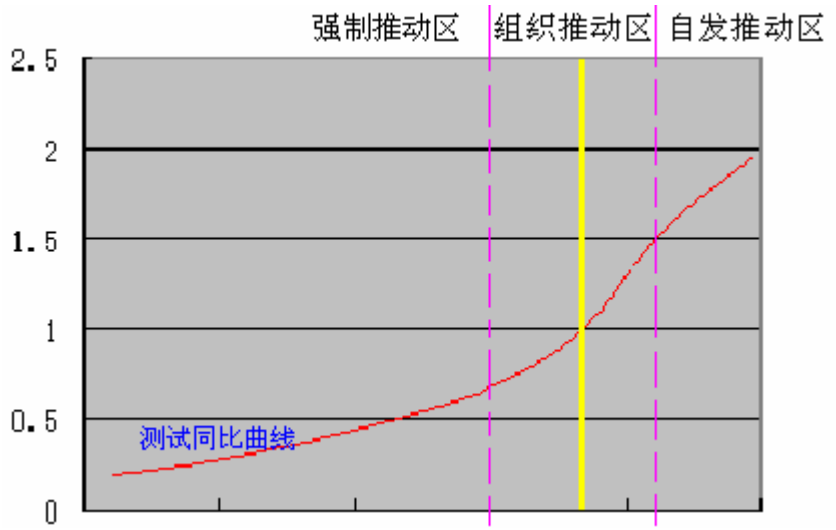
3. 上单板做测试的起点是多任务集成测试

   要上单板能做测试，至少加载任务要起来，下数据配置的任务也起来，另外任务调度模块也要正常启动，这已经是复杂的运行环境了。让这种环境稳定运行就需要一个测试过程，这加大了上单板做测试的难度。而且，许多不依赖多任务调度环境的测试项，容易受单板任务调度影响，增加了调测难度，若改在仿真环境下在单机桌面系统中做测试，孤立的一个代码片断是很容易把单元测试做起来的。

4. 单板实时系统下，缺乏测试手段

与实时操作系统配套的测试手段或测试工具缺乏，是众所周知的事实，这种情况也很难在数年内改善，所以，若追求高效率的测试，有必要将测试的工作平台转移到桌面系统，如 Windows 与 Linux。

在《企业如何推行白盒测试》一文中，我们介绍了“测试同比”的概念，测试同比实际反映了测试工作效率，尽管测试效率不是推动白盒测试的唯一因素，但在嵌入式白盒测试领域，该因素无疑是决定性因素。如下图：



由测试效率的高低，决定了白盒测试处于强制推动区，或是组织推动区，或是自发推动区，对于绝大多数企业，测试效率处于强制推动区的白盒测试最终都会以失败告终，当然也有例外，例外情况是那些不计成本追求质量的行业，如军工、航天等，他们可以为一个人做开发，配备数十人做测试。而测试效率处于组织推动区的，只有遵循较严格的研发流程，并拥有充分执行能力的企业才能成功。只有测试效率提高到自发推动区，推动白盒测试时，流程因素就降为次要条件，白盒测试也就能从组织的强制性行为，过渡到员工的自发与自愿行为。

上单板做白盒测试的最大问题是，测试效率太低，远没达到组织推动区要求的范围。强调在嵌入式实时系统下做白盒测试，其实是一个误区，业界有一些商用工具，宣称能支持各种实时系统下的白盒测试，但据我们试点过的四、五种主流工具来分析，这些都不足以真正推动一个组织把嵌入式白盒测试做起来，而且，不只做不起来，实际效果离预期目标还差很远。

误区之四：个人能力强的不必做白盒测试

能力强者不必做白盒测试，这显然是谬论。一个人能力再强，软件开发中都会犯错误，只不过能力强的，犯错误少些而已。

所以，这里要讨论的不是能力强的该不该做测试，而是他既然能力很强，如何才能更高效的做测试，能力强的人编码能以一当十，但测试呢？如何也让他以一当十。

首先是测试粒度，能力强者可以按粗粒度展开测试设计，让同一测试场景服务于多个函数同时测试，测试工具要支持被测函数在调用前嵌入小段脚本，用于微调测试场景，比如修改传入参数或某全局变量，让被测函数的特定路径能被覆盖。其次，尽量使调用过程能自

动转化成测试过程，调试需要让程序在特定场景下运转，测试也如此，也要在特定场景下运行，这两者是可以重用的。最后还有关于测试评估的问题，希望测试工具能适应普通人员按部就班方式一个函数接一个函数的测试，也同时适应多个函数批量测试的情况，测试覆盖率与测试程度的通过标准可以自主定制。

误区之五：白盒测试仅证明该怎么跑的代码是这样跑了

在嵌入式产品实施白盒测试主要有两大困境，一是没效率，白盒测试做不起来，二是没效果，白盒测试发现不了多少问题。前者更多是因为没找到合适的测试工具，而后者则是测试方法的问题，其典型表现就是经常有人说：白盒测试仅证明该怎么跑的代码是这样跑了，测不出什么问题！比如测试“1+1”，设计用例让它跑起来，1 加上 1 最终验证结果肯定是 2，但如此设计用例价值何在？代码中深层次的问题仍旧暴露不出来。

实际上，上述观点陷入了“机械测试”的陷阱，因为白盒测试不仅证明该怎么跑的代码是这样跑了，还尝试证明不该这么跑的代码也这么跑了。换一句话说，白盒测试针对的是规格，而非可见代码，通过可见代码做测试只是手段，最终目的是要测规格，所以我们白盒测试不仅要测可见代码，也测试不可见代码，如缺失的 else 分支、default 分支，甚至漏写的处理过程都在测试范围之内。

强调按规格做测试是一种理念，单靠个人自觉是很难在组织中实施的，我们把它提高到规范的软件过程，就是第 4 代白盒测试方法中所提倡的测试先行（TDF，Test Design First），该实践能有效保证每位员工，不管他的能力很强还是很弱，都按接口规格去设计用例，更详细请参考《企业如何推行白盒测试》。

上面列举了嵌入式白盒测试的五大误区，理解了产生这些误区的原因，不必再畏惧“白盒测试”这只拦路虎，只要方法得当，选用合适的工具，白盒测试还是比较容易做起来的。目前业界有众多白盒测试做成功的案例，比如 java 开发中的 JUnit 实践，C#开发中的 NUnit 实践，嵌入式领域的开发场景特殊一些，只要我们从中找出差距，想办法克服了，嵌入式白盒测试自然能推行起来。

第 4 代白盒测试方法就是这种基于实践的总结，JUnit 用 java 写测试用例，NUnit 用 C#写用例，这两者开发效率都很高，我们同样为嵌入式测试找一个高效的脚本语言，测试设计就不会成为瓶颈。另外，JUnit 或 NUnit 主运行平台是桌面系统，我们要求嵌入式产品的调测转移到仿真平台，测试效率不再是瓶颈。还有其它困难（测试评估等）更容易克服，嵌入式白盒测试没多少理由做不起来。

参考文献：

1. IPL, "Why Bother to Unit Test? "
2. ezTester, 《为什么要做白盒测试》,《企业如何推行白盒测试》
3. Wayne Chan, 《第 4 代白盒测试方法概述》理论篇与 VcTester 实践篇