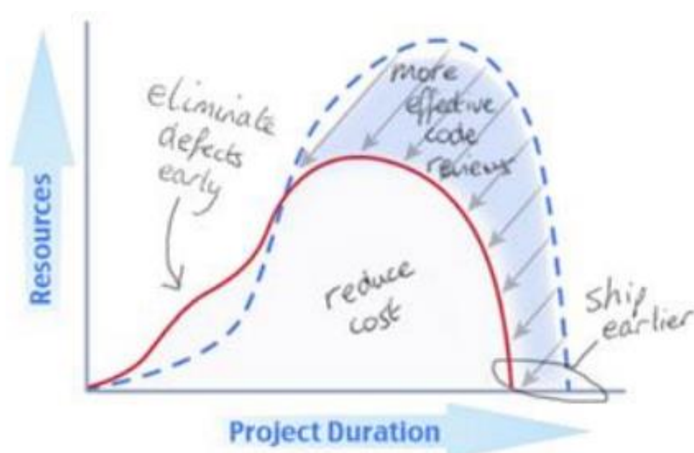
PRQA Programming Research

Trinity®

投资回报率

静态分析的商业案例

2015年12月



尽量在开发早期
清除所有问题

因为计算投资回报率时会涉及很多参数,而且参数性质都不尽相同,所以计算常常比较复杂。有些驱动因素很容易就能找到，但是有些却很难获取，而要确定数量就更难了。在这份白皮书中，我们将确定并讨论10个影响投资回报率优化的主要驱动因素。我们也会根据开源项目中的分析代码生成场景，来演示并量化这些驱动因素对投资回报率的影响。

引言

要详尽地计算投资回报率是非常复杂的，因为需要考虑很多因素，如：成本/收益流，所研究的时段，预测的准确性，预计到的风险，资本成本，等等。而简化的投资回报率则可归结为**两大主要组成部分：成本和收益**。

本白皮书主要着眼于软件开发生命周期（SDLC），研究**通用 SDLC 参数**（尤其是与静态分析相关的参数）是如何影响投资回报率的。首先我们有必要知道这里讨论的 SDLC 与投资回报率方程式中的**成本部分**关系十分紧密。但是，这些驱动最终也很有可能会产生后续的增量收益流。

首先，我们先确定 **10 个主要驱动因素**，并研究它们是怎样影响投资回报率的。然后，再检查开源项目的静态分析结果，利用这些数据来帮助证实这些驱动函数对投资回报率的影响。

A）主要的驱动函数

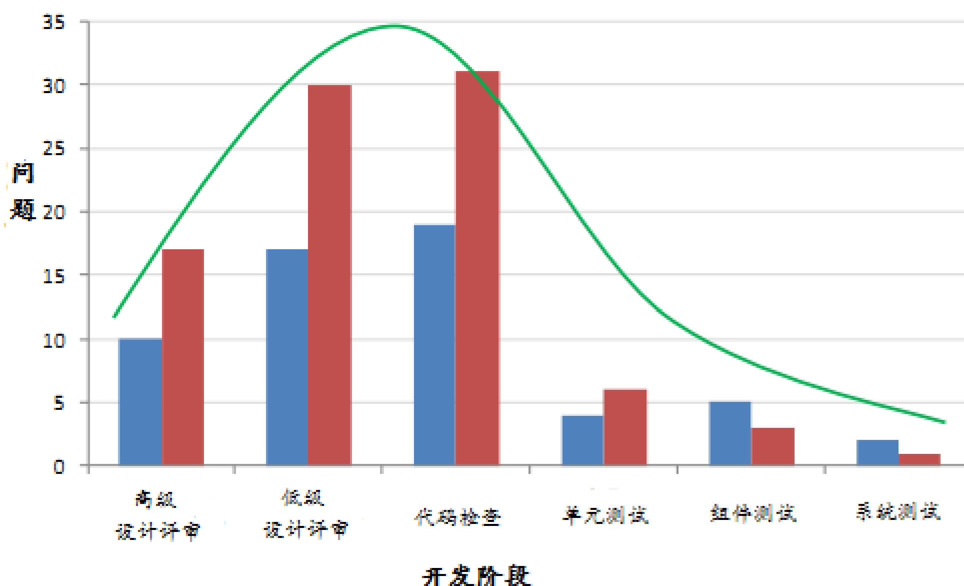
1. 缺陷生命周期

问题的引入及随后问题的解决是影响投资回报率的一个最明显、最基本的驱动因素。

关于这个问题，已经有了很多研究【1】【2】【3】。比如，针对很多以服务和产品为基础的组织机构中的项目所做的实证研究揭示了一个典型分布情况：

软件开发生命周期阶段	引入问题的比例
需求	50%-60%
设计	15%-30%
实施	10%-20%
其它（如：问题修复失败）	高达 20%

问题的引进和解决通常会表现为瑞利分布模式（Rayleigh distribution）【4】【5】:



一个广为接受的理念是：如果问题拖延到软件开发生命周期的后期才解决，那么解决问题所花费的时间（因而成本也是）会大大增加【1】【6】【7】。下表是最近所作的研究【6】得出的结论，该综合性研究机构经研究发现的这些数据，代表了随着问题的拖延，成本不断上涨情况：

软件开发生命周期阶段	解决问题所花费的成本
需求	1x
设计	3-6x
编码	10x
开发	15-40x
验收	30-70x

简单来说，**最有效的方法是**：

a) **首先尽量避免引进问题**。
b) **想办法尽早解决所发现的问题**，使得瑞利曲线（Rayleigh curve）的顶点尽量分布在左下方。

很多时候，问题的引进和解决情况是影响投资回报率的主要驱动因素。但是，一般都要通过宏观分析才能得出这个结论。我们还要深入挖掘潜在的驱动因素和根源，从而真正了解这些

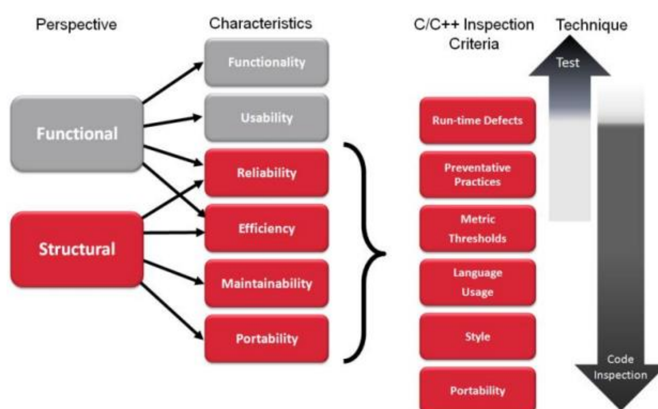
因素是如何影响投资回报率的。

2. 测试

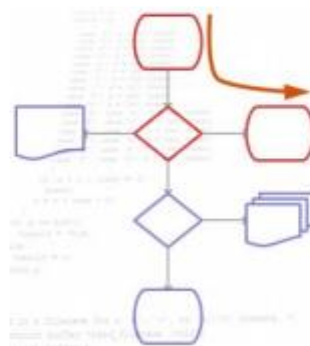
如果真的有问题没有发现（尤其是将问题遗留到了发布版本中），人们第一反应就是埋怨测试做得不够精确，效率不高，不够全面。因为大家普遍误认为，单凭测试就足以实现高度集成并能够保证代码的高质量。

当然，测试是非常重要的，但是很多人忘记了其实测试并不是万能药：

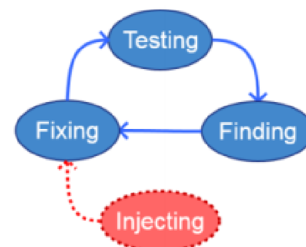
- **测试与功能性之间的关系十分紧密。**软件行业有一个广泛共识：如果不将非功能性需求或结构需求（非功能性需求（NFR）——确定处理系统内部集成的属性）考虑在内，就无法实现软件系统所期待的功能性需求（功能性需求（FR）—— 描述为实现用户需求所要做的事）。ISO/IEC-9126 【8】提出了一个非常有用的模型，来描述两者之间的不同之处。功能性需求和结构性需求之间有一些重合的地方，比如：功能性需求也会涉及用户希望实现其它功能时，在可靠性和效率方面应该要达到的特定目标【9】。这些重合的地方与投资回报率之间关系与这两个因素有关：1） 不仅仅只需要考虑功能性需求， 2） 某些独立的功能性需求和非功能性需求，可以在软件开发生命周期的不同阶段对其进行测试/验证。所以，将一些工作从测试阶段提前到开发阶段是非常有可能实现的。



- **要达到 100%的测试覆盖率是不太现实的**。这有几个原因：第一，要生成测试用例来覆盖所有的可能性，并执行每个特定语句、分支、条件和边界情况是很不切实际的【10】。在越来越细化的测试用例上进行投资，注定是会减少收益的。第二，如果有不可行路径、死码或防御性程序就更不可能达到 100%的覆盖率了——在这种情况下，复杂度和所需的测试用例的数量都会大大增加【10】。需要注意的是，静态分析与之不同，**静态分析可以达到 100%的代码覆盖率**。**数据流分析**会模拟程序的运行时行为，从而**提供一个非常有效的机制，来发现不同级别的问题**（如：不变式/重复运算，空指针的解引用，溢出/概括条件，等）



- **测试的成本非常高**：修复软件开发后期发现的软件问题所产生的成本在开发成本中所占的比例最高【11】【12】。在检测问题这个过程中，总是需要不断发现/解决问题，然后再重新



集成（**这个过程，反过来又可能会引进新问题**），如果检测问题这一步花费的时间太长，延迟进入生命周期中的下一阶段，那么成本就会增加。

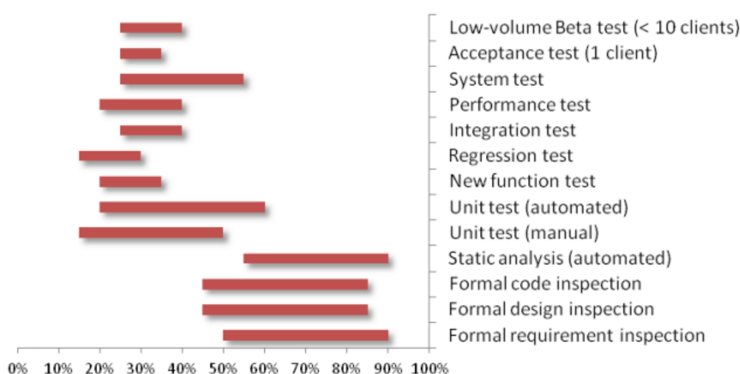
- **测试会增加交付的次数**。最简单的场景是：在测试过程中发现了问题，然后需要返回到开发阶段，对问题进行修复，再重新集成，重新测试。这个循环会**严重影响交付次数**。而且这样也会引进**风险**和**不确定因素**——我们都遇到过这种情况：本来打算在“下星期”交付项目，但是突然又将时间改到了“下个月”。在实际情况下，这样会显得过于简化了，所以通常还需要不断重复进行额外的“测试-修复-测试”工作，比如在这些情况下：
  - 开发阶段的初次修复工作没能解决应该解决的问题
  - 回归测试过程中发现进行初次修复时引进了新问题

在对高安全性的嵌入式系统进行研究时发现：修复在软件开发后期发现的软件问题所产生的成本在开发成本中所占的比例最高，所耗费的时间在整个项目中的占比也是最高的【11】【12】。

The structural quality of the code entering testing is important. Thus, for example:

- **进入测试阶段的代码的结构质量是非常重要的**。
  - 当代码变得越来越复杂的时候，测试就会越来越困难。复杂的代码中所增加的线性会大大增加测试的负担。所以，控制设计/编码阶段的复杂度（比如：通过圈复杂度度量，功能结构图，等）对测试的范围/成本会有重大影响。
  - 如果测试者主要着眼于功能性，他们就不希望有潜在的结构性问题来分散精力，或掩盖功能性缺陷。
  - 修复结构良好的代码中所存在的问题，与修复杂乱无章的代码或异构代码库中存在的问题，是完全不同的情况。

- **使用关键的、经验丰富的资源进行测试**——进行这类测试，需要测试者经过适当的培训，并且有丰富的经验，以便他们能够有效地安排和执行测试；即使是在自动化测试情境下，进行功能性验证也是一项很费力的工作；如果代码出现严重的结构性问题，那么随后相关的修复工作也会非常混乱。【9】



所以，虽然测试依然是极为重要的，但是也有严重的局限性。左边的表很好地总结了在评审、检查、静态分析，以及几类测试阶段，对进行问题修复的效率。从表中可以看出，大多数形式的测试在发现问题或缺陷方面的效率都低于35%。

测试当然不能凭空进行，它与开发和静态分析之间有着很深的依赖关系。我们**不能单独分析静态分析的投资回报率**，要将软件开发生命周期中的其它阶段都考虑在内。在优化静态分析的投资回报率时，尤其要考虑到与测试相关的两大挑战：

- **将检测问题这一工作从测试阶段提前到编码阶段**（这个过程有时候被称作**开发测试**），

同时要注意的是:这个范例是将保证最终的软件质量的部分责任从品质保证阶段转移到了开发阶段。

- **提高测试效率**。通过保持功能性测试和非功能性测试之间的平衡,和保证代码库的稳定性、连贯性和较好的结构性,来提高测试效率。(意大利面式代码——结构紊乱、复杂的代码,通常是因为多年来一直对旧有代码进行修改造成的——在维护方面需要花更多的时间,而且这种代码更容易崩溃,更容易产生意料之外的影响,包括:由附加的回归测试产生的延误)。

3. 代码的重复使用

通常情况下,在大多数商业软件的发布版本中,新创建的、内部的、从头开始编写的代码所占的比例非常小——实际上,大多数代码都是原有的,都是在重复使用。

有很多合理的理由可以解释为什么要对代码进行重复使用。很明显,最主要的动机就是提高投资回报率。但是,我们也注意到,“已有的代码”也分很多种,我们一定要知道它们各有各的优点和缺点。当公司开始依赖在核心控制开发过程之外产生的代码时,一定要意识到这些代码的不同特征,以及可能由此产生的更高的成本。我们重点说明以下 5 类代码:

a) 旧有代码

基本上,旧有代码都存在于公司的核心开发程序中。但是,根据实际经验可以发现,随着时间的推移,旧有代码会渐渐与现在的开发程序和最佳范例不相匹配。也会不再支持特定版本的主要开发工具,而曾经编写这些代码的开发人员通常都已经离开该项目了(参与到别的项目中了,或者进入别的公司了)。

人们以为旧有代码是“在实际应用中得以验证的”,所以认为在重复使用这些代码时,它们会自动适用于新场景中,(比如:用于不同的目标硬件上,或者用于一个不同的系统中),其实这一观点是需要被质疑的,尤其是在与安全相关的系统中使用旧有代码的时候。

无疑,静态分析工具和编码规范在帮助开发团队了解旧有代码的结构完整性方面起着很大的作用,同时还有检验机制来帮助更新代码,使代码遵循现在的编码规范和最佳范例。它对投资回报率(以及减少风险)的影响力是显而易见的。

b) 开源软件(免费的&商业的)

开源代码和库与旧有代码之间有一个非常重要的相似点,就是开发团队可以获得源代码。虽然,人们重复使用代码的商业动机很明确,但是,代码的出处(后续支持)不明却是一个大问题。在开发能够利用这类代码的框架的同时,不同的垂直市场甚至自己创造了术语(如:医学领域 IEC 62304 确定了“软件的未知血统”(SOUP),A&D 认可了“商用现成品或技术”(COTS))。

能够再次使用源代码使得静态分析和编码规范有机会帮助提高投资回报率。

c) 自动生成的代码

验证自动生成的代码需要利用代码自动生成器的可靠性,而且根据之前部署的代码库可以在使用中使这些自动生成的代码符合规范要求。这里也需要使用静态分析，工具在发现和有选择性地评估手动编写的代码方面的能力对投资回报率有决定性影响。

有些团队使用的代码库相对比较小,他们所采用的方法很有意思,就是使用 MDD 进行设计，实现可视化，完成建模——但是不会自动生成代码。他们认为手动生成并验证这些相对较小的代码段，更能节约成本【14】。

d) 外包

软件外包的趋势仍在继续发展。这有很多原因，包括【15】:

- 解决人员短缺的问题
- 减少人力成本
- 没有专业的技术
- 连续不断的工作流程

外包有很多种形式。有时候外包只局限于软件本身，有时候则是将软件和硬件捆绑在一起。举例来说，汽车行业的原始设备制造商（OEM）的供应链很深广。在这种情况下，OEM 就特别需要供应商开发的代码能够符合 MISRA 规范,同时还要求供应商能够提供合规证明(通常还会要求证明是使用 QA·C 进行合规检查的)。这就等于是为 OEM 提供了软件结构完整性的保障。

外包商完成开发工作这一时间点是一个重要关键的里程碑，届时双方会完成签字核准，并将开发完成的代码交给供应商，或者交给任何一个负责维护工作的人员。如果维护合同是外包的（如：客户付钱——比如——每年支付 20%的费用进行技术支持、帮助修复问题、做一些简单的改进)，那么**代码的质量越高，越易于理解和维护，供应商能够获得的利润就越高**。同样的，如果供应商延续使用旧有的代码，而这些代码结构性很差（意大利面式的代码)，那么维护以及解决问题所耗费的成本就会非常高（风险也会很高)。

4. 生命周期模型

据我们所知，静态分析的基本原则就是：在出现的问题之后，尽早发现并解决，并且这一过程要尽量在软件开发生命周期的早期完成。这对于投资回报率有非常重大的影响。**“尽早”和“经常性”的原则对于增量/迭代开发**（如：敏捷和持续集成/持续交付）**而言也是最为基本的。**

敏捷开发推崇的是短期（主要是 4-8 周）冲刺（sprint)，它的另外一个目标是在每个冲刺阶段结束的时候,所开发好的代码都适合发布。所以,这就要求一直保持代码处于良好的状态，因而每次在问题引进之后，都要尽快解决，绝对不可以拖延。

同样地，持续交付的目标也是保持代码始终处于良好的状态，每次有新代码检入的时候，都会对代码进行自动检测。所以，静态分析能够很好地与这些步骤相结合就不奇怪了——**这些操作要求能够尽早、并经常性地进行静态分析，以提高静态分析的投资回报率**。

相比之下，传统的循序开发模式，如瀑布式或 V 模型模式，**更容易延缓发现问题的时间**。而且如果在编码阶段已经注入了很多问题，而这些问题直到测试阶段才能被发现的话，这种延误情况就更为严重了。在循序开发过程中，编码和测试两个阶段之间会相隔很长时间。在这种情况下，正如之前讨论过的，要实现静态分析投资回报率的最大化，意味着要将修复问题这一步从测试阶段提前到开发阶段。

5. 自动化

自动化无疑是提高效率的关键因素，在软件开发生命周期的每一步以及整个开发工具链中，都要考虑到自动化的问题。

静态分析的自动化与以下两个方面关系最为密切。首先，与融入软件开发生命周期中的不同环节（上面已经讨论过）密切相关；其次，与源代码分析工具的效率紧密关联（这一点会在下一节进行详细讨论）。

但是，自动化——从其自身角度来说——会为提高投资回报率带来很多帮助，比如：

- **可扩展性**——能够轻松地增加分析量
- **速度**——可大幅提高分析的持续时间
- **一致性和可预测性**——自动化工具每次都可以获得相同的结果。但是，专业水平不同的个人，不可能每次都能获得相同的结果。
- **独立性**——工具输出的结果是客观的，而不同的人则会有不同的观点。
- **精确度&准确性**——通常会给出质量较高的结果
- **可追溯性**——自动化处理容易留下有记录的（稽核）踪迹，能够作为执行或验证的证据
- **关键的专业资源**——可以摆脱日常的重复工作，从事具有附加值的任务。

6. 静态分析工具的有效性

不幸的是，**如果选择了效率不高的静态分析工具的话，就会严重损害采用自动化处理过程带来的优势**。

误报

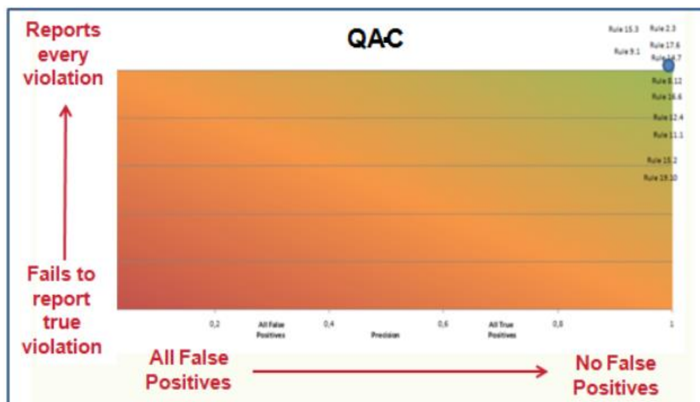
分析结果的误报率高造成的影响很明显，就是开发人员要耗费更多的时间和成本来评估、发现并排除这些误报信息【16】【17】。

如果只是在一个小示例代码上执行少量的规则，那么问题可能也不是很大。但是，随着规则和代码的增多，使用干扰性大的工具带来的影响（额外成本和时间）就会很明显了。而且，很多误报信息会造成的真正危害是，最终将损害工具的信誉，这样就很少有开发团队会使用该工具了。

漏报

虽然误报会影响运营商的整体生成力，**但是漏报产生的危害更大，因为它会让人过度自信**【18】。很多标准都明确强调了这种风险，包括 DO-178，IEC 61508（以及同类标准），MISRA。

这里要引入查准率和正确率的概念。**查准率**表示的是，发现的所有问题中，判断为真的正确率比例。**正确率（查全率）**表示的是，正确分类（判断为真的正确率和判断为假的正确率）的比例。最好的静态分析工具拥有最高的查准率和正确率【16】。TERA 实验室独立研究了一系列静态分析工具在执行 MISRA C 中的 11 条主要规则方面的效率，所得出的结果【19】【20】，已经就这个论题给出了一个很好的结论。研究发现 QA·C 的性能堪称典范——它发现了所有的违规行为，并且没有误报。



需要进一步说明的是——与漏报这个话题有些相关——要清楚了解工具的适用范围，以及该工具所发现的问题的类别。比如，如果与编码规范相关的话，就要考虑规则覆盖是什么。现代的编码规范，如 MISRA C:2012，都是专门设计的，能够清楚分辨出哪些部分是可以通过静态分析工具（规则）来验证的，哪些部分需要手动交互（指令）来完成。要特别留意那些声称能使覆盖率达到 100%的工具!

7. 证书要求

现在的高安全项目特别需要遵循现代的质量标准。无论是在汽车、航空电子设备、国防、医疗、轨道交通行业，还是在核能等行业，软件必须遵循的行业公认的标准（ISO 26262, DO-178B/C, IEC 62304, EN 50128, IEC 60880）。这些项目中使用的工具要能帮助用户遵循这些标准。有些情况下——比如 DO-178B，补充了 DO330 的最新版 DO-178C——会正式要求使用合格的工具。

显然，投资回报率会受到可用的认证/合规工具的影响，如果要用户来承担这个重任（或者用户选择自己完成认证工作）,这可能会大大增加项目的成本和不确定性。

8. 支持编码规范

编码规范背后一个主要目的就是为所选语言定义一个更安全、更具确定性的子集，鼓励遵循最佳范例，排除可能出现未定义行为的情况，帮助识别编码错误（比如：初始化，名称藏匿，不可达的代码等问题）。

无论采用了**广泛认可的**标准（如：MISRA, JSF or HIC++），还是一个**内部标准**，（内部标准可能是由广泛认可的标准衍生而来，并结合了一些**自定义的规则**，如：专用的命名约定），静态分析工具必须能够完全支持编码规范的执行；工具的可用性和可定制性是工具的核心价值所在，能使其从其它工具中脱颖而出。

大多数开发者，无论是新手还是有经验的，都发现编码规范都是由对编程语言理解得非常深刻的知名专家编写的，因为他们知道可能存在的风险和陷阱。而且，大多数编码规范都很便宜，或者是免费的。大部分成本（可能节约的成本）都与这些最佳规范的有效实施有关。最

后，从技术角度以及成本角度来看，利用这些规则集比尝试从头开始“创造”新规则更有效率。实际上，评审员的反应值得留意：当他们看到代码采纳了公认的编码规范的时候，他们就立刻放心了。他们非常热衷于深究违反编码规范的理由，以及详察内部创建的“替代”规则。

9. 开发角度——不只考虑成本问题

正如引言中所指出的，上面的例子主要都着眼于**提高效率和节约成本**，而且主要涉及的是开发活动和软件开发生命周期的过程。但是，工程团队的优势并不仅仅局限于节约成本方面。从传统的项目管理三角形来看，也有与交付次数、质量（和可预测性）相关的改善，比如：

- **加快交付频率**，缩短连续交付之间的时间间隔
- 提高对**代码质量**的信心——**内部和外部**
- 更**安全**地使用**开源**模块
- 提高接受**外包**任务的门槛
- 简化技术支持和维护工作
- 提高交付的**可预测性**

10. 商业角度——收入方面

从商业角度看，提高工程管理的效率有很多优势——花的每一分钱都会有更大的收益。但是，增量**收益**——**增长的单元产品销售额和/或更高的价格**——可能会对投资回报率（和损益）产生更大的影响。很明显，增量收益是由开发成果产生的。

本白皮书主要是讨论软件开发生命周期/开发方面的影响。所以，我们就不深入讨论商业和收益方面的情况了。但是，通过下表可以发现一些具体时机，以进一步降低成本，提高整个商业运营中的收益。

与成本相关的方面：

- 降低**开发**成本
- 降低技术**支持**和**维护**成本
- 降低**质保**成本
- 降低产品召回的成本
- 降低法律责任/成本

与收益相关的方面：

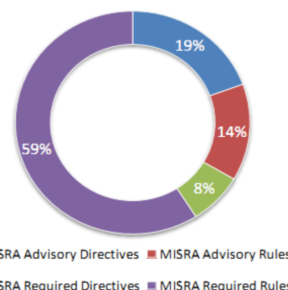
- 缩短**上市时间**
- 提高**销售量**
- 提高**质量**
- 提交**价格**
- 加快对**市场机遇**的反应速度
- 加快回应**竞争**的速度
- 提高品牌**价值/信誉**
- 提高**可预测性**，降低风险

B．投资回报率分析——举例

为了验证我们已讨论过的观点，我们选了一个小型开源项目，并希望能解决所有与 MISRA C:2012 相关的问题。所选的项目是**时间-1.7**(用于评估项目资源使用情况的实用程序的 GNU 版本，比如时间和 CPU)。该版本由 **6 个头文件和 5 个源文件**组成，大概共有 **2000 行代码**。

使用**带有 MISRA C:2012 合规模块的 PRQA QA·C v.8.2** 对该代码进行分析。使用该配置 QA·C， 一共诊断出了 **520** 个问题。

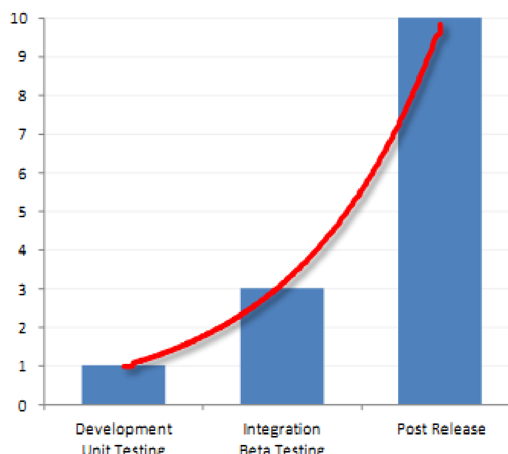
在下面的场景中，我们假设我们解决了所有的违规行为。



我们将生命周期简单地划分为 3 个阶段:

- 开发/单元测试
- 集成/ Beta 测试
- 发布之后

我们假设，在开发阶段，发现和修复问题的平均时间为 1 小时，在集成阶段增加到 3 小时，在发布后这个时间则会增加到 10 小时。注意：在比较从各个场景中所得出的结果时，比例（1:3:10）比绝对值更有意义。在下面所有的场景中，发现和修复问题时间都保持不变。



案例

我们研究了 **5 个不同场景**，主要研究在每个阶段，随着问题修复比例的变化，投资回报率所受到的影响。

场景 1——参考案例

在参考案例中，我们特意根据 NIST【21】中的数据大致分布了发现和修复问题的比例。在开发阶段修复 30%的问题，集成阶段修复 60%，产品发布之后再修复 10%。

场景 2——好工具/开发测试

这个场景反映的是使用理想化的高质量静态分析工具的情况（没有误报和漏报）。发现问题这一步从测试阶段提前到了开发阶段，而且开发人员在发现问题之后，立刻就对问题进行了修复，并且这些都是在开发阶段完成的。

场景 3——好工具/品质保证阶段的合规检测

在这个场景中，使用的是同样的高质量静态分析工具，但这次静态分析执行得比较晚，在集成/ beta 测试阶段才进行静态分析。这反映的是在开发工作完成之后（如：为“保证品质”的“验收测试”），才开始评估代码是否符合规范的情况。

场景 4——糟糕的工具/漏报

这里使用的静态分析工具并不是一流的，甚至有些问题它还发现不了（漏报）。假设有 20%的问题该工具没有发现，这些问题就遗留在了正式发布的产品中。为了简单一点，我们只考虑在发布后期对问题进行修复和改进产生的额外费用。（当然，如果正式发布的产品中有未发现的问题，也会影响收益。而且如果该产品是投放在高安全性市场的话，甚至还可能引来

诉讼，成本会大大增加。）

场景 5——糟糕的工具/误报

假设还是使用质量较低的静态分析工具。这次，工具可以找出所有的违规行为，但同时也产生了误报信息。所以，假设该工具多报告了 20% 的问题，找出了多于上面所说的 520 个问题。这里我们也假定修复真正的违规行为所花的时间和排除误报所花的时间一样多。在场景 5 中，发现和修复问题都是在开发阶段完成的。

每个场景中所用的主要参数如下：

Scenario		SDLC phase			Total
		Development Unit Testing	Integration Beta Testing	Post Release	
1 - Reference	%	30%	60%	10%	*100%*
	diagnostics	156	312	52	*520*
2 - Development Testing	%	100%	0%	0%	*100%*
	diagnostics	520	0	0	*520*
3 - Compliancy in QA	%	0%	100%	0%	*100%*
	diagnostics	0	520	0	*520*
4 - False negatives	%	80%	0%	20%	*100%*
	diagnostics	416	0	104	*520*
5 - False positives	%	120%	0%	0%	*120%*
	diagnostics	624	0	0	*624*
All scenarios	Time to find and fix	1x	3x	10x	

结果

5 个场景中的成本（从时间方面看）已经计算出来了。和上面说的一样，在简化的模型中个，修复每个问题所要花费的时间仅仅与修复工作发生在软件开发生命周期的哪个阶段有关。

Scenario	Time cost to fix F&F defects @ SDLC phase			Total Time (/cost)
	Development Unit Testing	Integration Beta Testing	Post Release	
1 - Reference	156	936	520	*1.612*
2 - Development Testing	520	0	0	*520*
3 - Compliancy in QA	0	1560	0	*1.560*
4 - False Negatives	416	0	1.040	*1.456*
5 - False Positives	624	0	0	*624*

场景 1 所花费的成本最高。主要原因在于，大部分修复工作都延误到了生命周期的后期（越晚成本越高）。

和预计的一样，**场景 2 所取得的成绩最好，因为这个场景中使用的是准确率非常高的工具，而且尽量在软件开发生命周期的早期就解决了所有的问题**。

场景 3 说明的是，在开发阶段没有使用工具进行静态分析，而是拖延到集成/beta 测试阶段才开始进行静态分析所产生的负面影响——这里的想法是：先写好代码库，然后再使用工具来验证代码是否符合规范要求。这种情况下，虽然使用了最好的工具，但是因为没有在最佳时间使用该工具，所以还是大大降低了投资回报率。

为什么场景 4 中的投资回报率很低，是显而易见的。需要再次强调的是，这个简单的模型没有考虑严重的问题遗留到正式发布的产品中（尤其是高安全性系统中），可能产生其它危害或诉讼费用。

最后，和预计的一样，**误报所产生的额外费用随着误报的比例相应地增长（场景 5）**。所以，与场景 2 相比，额外 20%的误诊率会导致成本增加 20%。

结论

在本白皮书中，我们发现了会对静态分析投资回报率产生实际影响的 10 大主要驱动函数。同时，我们还在操作层面（比如：在哪个阶段下，可以开始那些有意义的操作）对其做了深入研究。

分析开源项目所获得的结果，有助于更好地了解 5 个不同场景对投资回报率的潜在影响。

我们发现，在实际情况下，要计算投资回报率可能是非常复杂的，尤其是因为有些成本和收益很难准确估算出来。举个例子，工具的价格很容易就能知道，但是开发人员持续重新编写代码及花时间排除误报所产生的成本就没那么好计算了。为了能更加客观地选择静态分析工具，需要先利用自己的样本代码来评估该工具在以下两个方面的性能：

1. 工具的技术能力（有效性）
2. 所产生的投资回报率

当我们的用户评估 PRQA 的解决方案时，这些方面我们都会涉及。

参考

[1] e. a. L. Lazić, “Estimating Cost and Defect Removal Effectiveness in,” in INFOTEH-JAHORINA, vol. 12, 2013.

[2] G. N. T. Suma V, “Defect Management Strategies in Software Development,” in Recent Advances in Technologies, 2011.

[3] V. S. T.R. Gopalakrishnan Nair, “The Pattern of Software Defects Spanning,” International Journal of Software Engineering (IJSE), July 2010.

[4] A. Vladu, “Software Reliability Prediction Model Using Rayleigh Function,” in U.P.B. Sci. Bull., Series C, Vol. 73, Iss. 4, 2011, 2011.

[5] S. H. Kan, “Metrics and Models in Software Quality Engineering,” 2003.

[6] M. C. A.A. Frost, “Advancing Defect Containment to Quantitative Defect Management,” CrossTalk, no. December, 2007.

[7] W. Humphrey, “A Personal Commitment to Software Quality,” in 5th European Software Engineering Conference, 1995.

[8] ISO/IEC, “9126-1:2001: Software engineering -- Product quality -- Part 1: Quality model”.

[9] F. B. C. Weimer, “Continuous Code Inspection,” PRQA White Paper, 2013.

[10] M. Baluda, “Automatic Structural Testing with Abstraction Refinement and Coarsening,” in European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2011.

[11] R. Bartholomew, “Using Combinatorial Testing to Reduce Software Rework,” CrossTalk, no. January/February, 2014.

[12] A. P. S. Sulistyo, “PMG-Pro: A Model-Driver Development Method of Service-Based Applications,” in 15th International SDL Fourm, 2011.

[13] C. Jones, “Software Quality and Software Economics,” Software Quality, Reliability, and Error Prediction, vol. 13, no. 1, 2010.

[14] C. Study, Static Analysis - for Manual Code and Auto-generated Code, PRQA, 2014.

[15] L. Rierson, Developing Safety Critical Software – A Practical Guide for Aviation Software and DO-178C Compliance, CRC Press, 2013.

[16] L. W. S. Heckman, “On Establishing a Benchmark for Evaluating Static Analysis Alert Priorization and Classification Techniques,” in Empirical Software Engineering and Measurement (ESEM), 2008.

[17] D. A. J. M. B. F. Wedyan, “The Effectiveness of Automated Static Analysis Tools for Fault Detection and Refactoring Prediction,” in International Conference on Software Testing, Verification and Validation (ICST 2009) pp. 141-150, 2009.

[18] G. M. B. Chess, “Static Analysis for Security,” IEEE Security & Privacy, no. November/December 2004 Issue.

[19] H. V. H. K. B. M. Temmerman, “KriCode Research Report I: Comparative Study Of MISRA-C Compliancy Checking Tools,” TERALabs, 2010.

[20] PRQA, “Comparative Study Of MISRA-C Compliancy Checking Tools,” [Online]. Available: http://www.programmingresearch.com/content/white-papers/prqa-white-paper-tera-labs-static-analysis-tool.pdf.

[21] R. f. NIST, “The Economic Impact of Inadequate Infrastructure for Software Testing,” 2002. [Online]. Available: http://www.nist.gov/director/planning/upload/report02-3.pdf.
[22] H. Gall, ReUse: Challenges andBusiness Success, Advanced Software Engineering, FS 12 - University of Zurich, 2010.
[23] D. M. Mehta, “Effective Software Security Management,” OWASP - Open Web Application Security Project, 2007.
[24] S. Goldfarb, “Industry Metrics for Outsourcing and Vendor Management,” Q/P Management Group Inc., 2008.
[25] R. P. M. S. A. Dautovic, “Automated Quality Defect Detection in Software Development Documents,” in SQM 2011 - ifth International Workshop on Software Quality and Maintainability, 2011.
[26] C. P. J. Zou, “Control Cases during the Software Development Life-Cycle,” in IEEE Congress on Services - Part I, 2008.
[27] J. C. L. Amar, “Measuring the Benefits of Software Reuse,” [Online]. Available: http://www.drdobbs.com/measuring-the-benefits-of-software-reuse/184406111.
[28] C. C. Weber, “Assessing Security Risk In Legacy Systems,” [Online]. Available: https://buildsecurityin.us-cert.gov/articles/best-practices/legacy-systems/assessing-security-risk-in-legacy-systems.
[29] F. Bolger, “Controlling automotive software deviations in a MISRA compliance environment,” 2014. [Online]. Available: http://embedded-computing.com/articles/controlling-misra-compliance-environment/#.
[30] F. Bolger, “The Best Coding Standards Eliminate Bugs,” PRQA White Paper, 2011.
[31] V. L. d. Mendonça, “Static Analysis Techniques and Tools: A Systematic Mapping Study,” in ICSEA 2013 : The Eighth International

创提信息科技（上海）有限公司 – Trinity Technologies

专注于嵌入式软件研发质量和自动化测试的方案和咨询服务，提供覆盖软件测试整个流程的完整的解决方案，包括从研发前期的代码级测试到后期的系统级测试，从静态分析到动态测试，从编码检查，单元测试、集成测试到性能测试和测试覆盖率分析等。

公司通过专业的自动化工具（如 DT10, VectorCAST, PRQA, SQUORE 等）和服务满足不同客户对软件质量和测试的需求，持续协助客户改进软件研发质量和效率。客户主要集中在高安全和高可靠性领域，如国防和航空航天、轨道交通、汽车电子、医疗器械、工业控制、通讯和电力电子等行业。公司提供的领先的解决方案不仅为数以百计的客户提高产品质量，还协助客户遵循高安全和高可靠性行业的合规性要求，如 DO-178B/C，IEC61508, EN50128, ISO26262, IEC62304 和 MISRA 等行业标准，并获得相关机构认可和认证。

创提信息科技（上海）有限公司
www.trinitytec.com.cn

上海市浦东新区张江高科张衡路
200 号 3 幢 3207E 室，201204
T: 21 - 3126 8126
F: 21 - 5132 8526
E: info@trinitytec.net