Matlab与信号处理

周治国

2018.9

作业2

要求

- 1. 每人都需要做，形式为PPT+支撑材料（如论文原文，相关背景文档，程序代码等）

- 2. 自行分组，建议6人一组；每组准备一份PPT ，用于课堂讲解（演示），时间15分钟左右； PPT内容可以选择作业题目的一条或几条，范围可以有所扩展，但要把握思路主线，展开详细论述

- 3. 做好的两份PPT发到邮箱3220180517@bit.edu.cn，个人PPT命名 '班号-学号-姓名'，小组PPT里面要写上'组员姓名学号'

作业2

- Matlab与信号处理系统（设备）
  - ✓ 是工具？产品
  - ✓ 代码质量？可靠性？

作业2-OBE1

- Matlab - Hardware
  - 数据采集
  - 数据处理
  - 数据测试

按 产品 查找

MATLAB	14
Simulink	13
Audio System Toolbox	1
Communications System Toolbox	5
Computer Vision System Toolbox	2

按 Product Family and Category 查找

MATLAB®产品家族	98
Simulink Product Family	116

按 供应商 查找

3D Robotics	1
ADLINK	3
ARM	18
AUTOSAR Development Partnership	1
Adimec	3

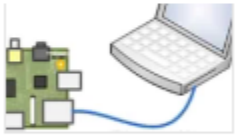
按 应用 查找

FPGA 设计和协同设计	19
图像处理与计算机视觉	58
嵌入式系统	62

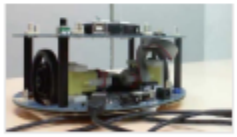
結果 1 - 25 of 172

Data Acquisition Toolbox 提供的 NI-DAQmx 支持
从 NI-DAQmx 设备采集数据并进行分析
Vendors: National Instruments
Tags: Support Package Installer Enabled, MathWorks Supported

MATLAB 提供的 Arduino 支持
从 MATLAB 连接并控制 Arduino 输入和输出
Vendors: Arduino
Tags: Support Package Installer Enabled, Project-Based Learning, MathWorks Supported



MATLAB 提供的 Raspberry Pi 支持
从已连接的 Raspberry Pi 采集传感器和图像数据
Vendors: Raspberry Pi
Tags: Support Package Installer Enabled, Project-Based Learning, MathWorks Supported



Simulink 提供的 Arduino 支持
在 Arduino 开发板上创建和运行 Simulink 模型
Vendors: Arduino
Tags: Support Package Installer Enabled, Run on Target Hardware, C/C++ Code Generation, Project-Based Learning, MathWorks Supported

Simulink 提供的 Raspberry Pi 支持
在 Raspberry Pi 上运行模型
Vendors: Raspberry Pi
Tags: Support Package Installer Enabled, Run on Target Hardware, C/C++ Code Generation, Project-Based Learning, MathWorks Supported

作业2-OBE2

- 代码管控对自动化工具的需求
- Matlab自动代码生成质量如何？
- Matlab能否作为产品级代码生产平台？尤其是针对嵌入式这种资源有限的硬件载体

知乎

代码质量管控的四个阶段

张鑫
软件工程师

背景

本文讨论的代码质量指的是代码本身的质量，包括复杂度、重复率、代码风格等要素。代码是团队的共同财产，代码质量是团队技术水平和管理水平的直接体现。

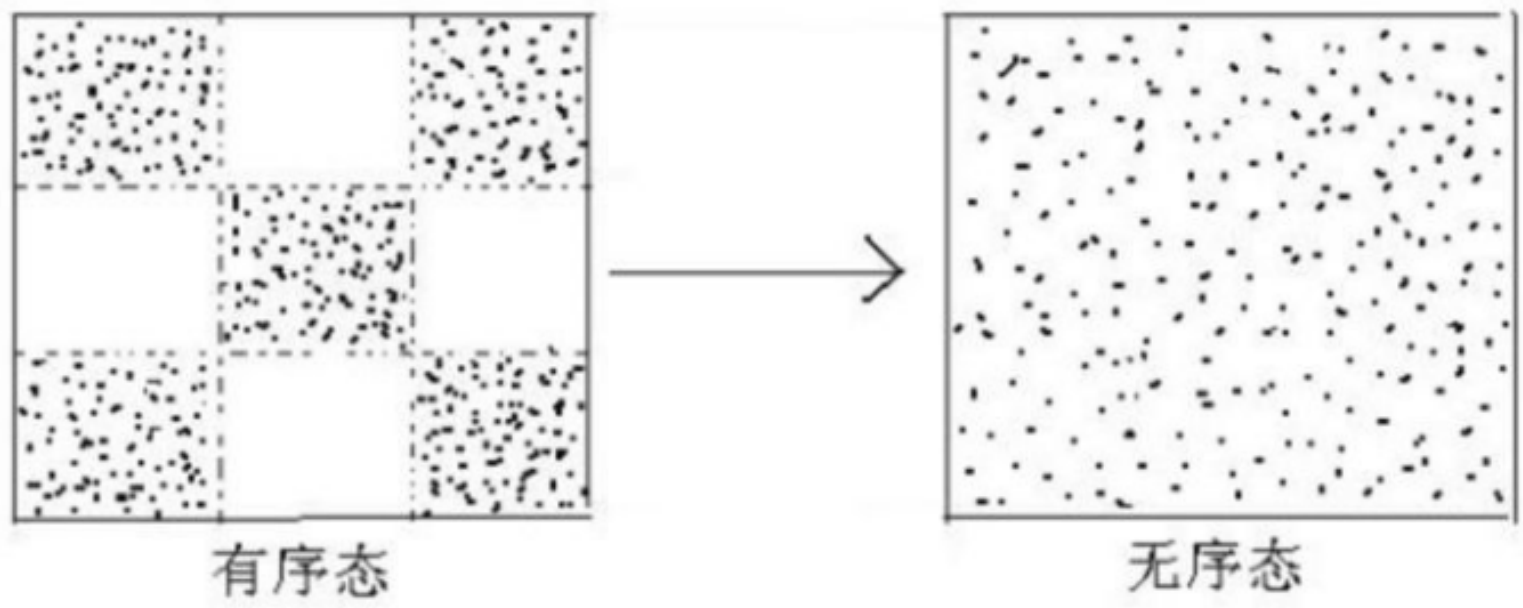
代码质量下降通常会自成因果，导致恶性循环：

- 破窗效应：在烂代码上继续生产烂代码的心理负担小很多
- 传染性：烂代码传递着一种不在意质量，只看业务成果的负面信息，会伤害团队的技术热情和工作氛围，导致更多烂代码出现

本文会分析代码质量下降的内在机制，并分享在代码质量管控方面的一些实践经验。

熵增定律与代码质量

熵增定律告诉我们，一个封闭系统总是趋向于熵增，也就是系统的无序程度只会不断增加。

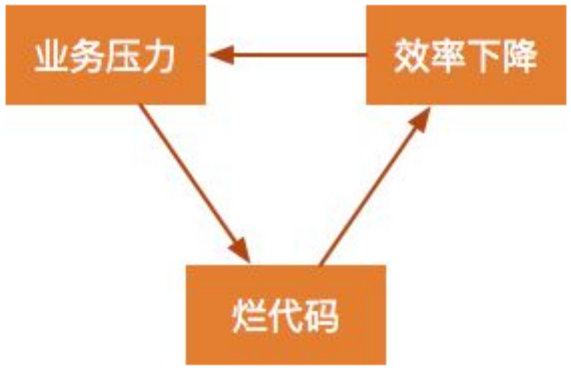


对于软件项目来说，代码质量代表着系统的有序程度，烂代码增加就是系统无序性上升的体现。在无外力影响的情况下，烂代码只会原来越多。

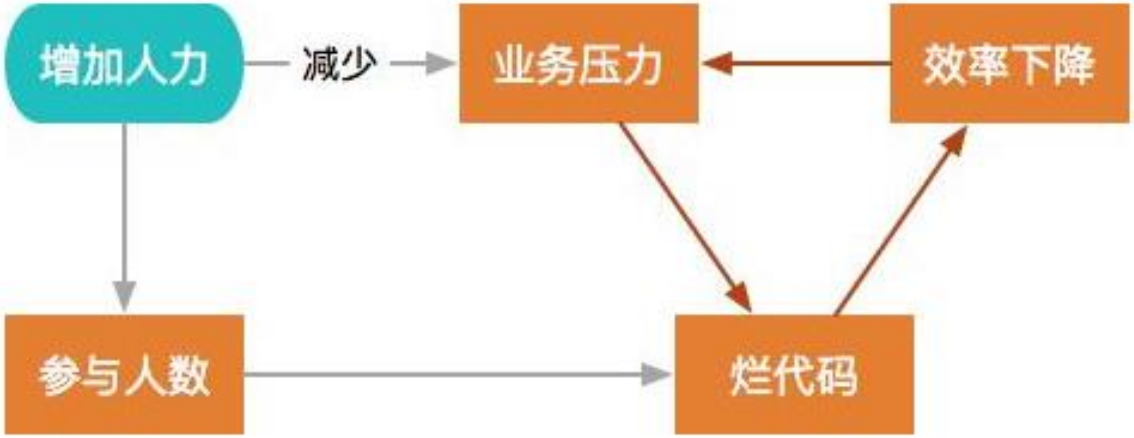
为了维持系统有序，需要外界向系统不断输入能量。对于代码质量，我们需要主动投入资源，来有意识地抑制烂代码越来越多的自然趋势。

经典循环

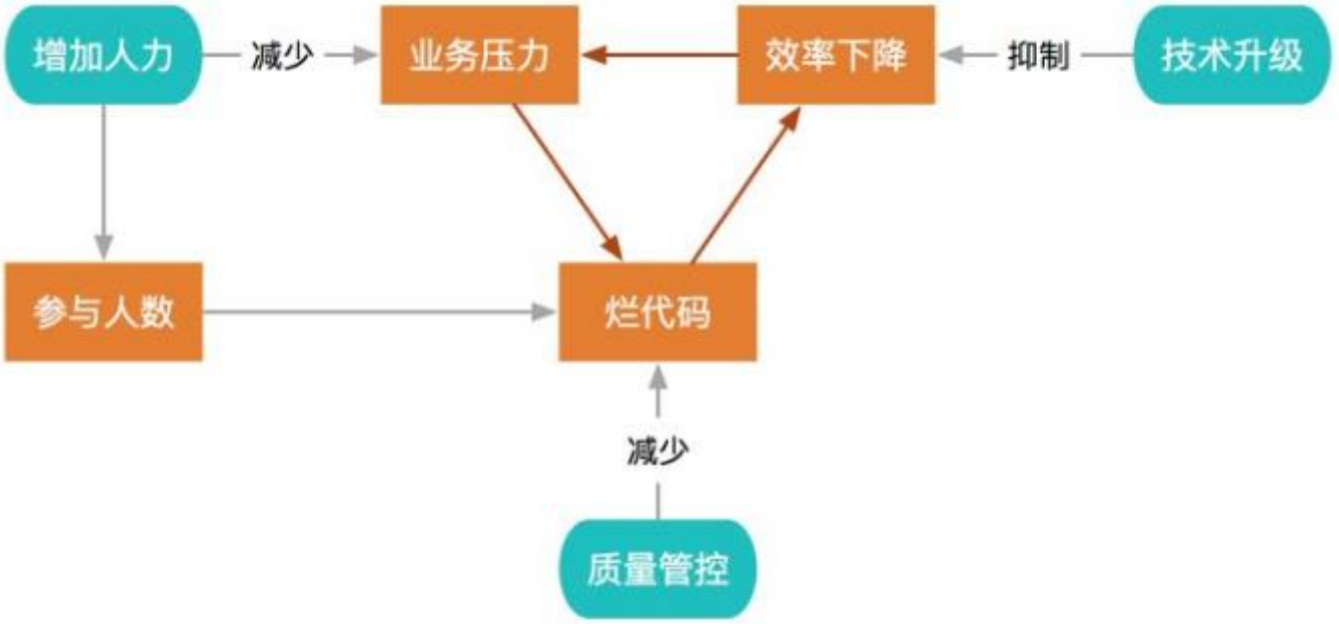
烂代码产生的常见原因是业务压力大，导致没有时间或意愿讲究代码质量。因为向业务压力妥协而生产烂代码之后，开发效率会随之下降，导致业务压力更大，形成一种典型的恶性循环。



为了应对业务压力，常见的做法就是向项目中增加人力，但是单纯地增加人力的话，会因为风格不一致、沟通成本上升等原因导致烂代码更多。



要遏制这种恶性循环，需要多管齐下，主动对代码质量进行管控，并且持续进行技术升级，系统性地解决问题。



不过质量管控和技术升级需要长期投入才能产生效果。通常情况下人们还是倾向于通过增加人力快速地解决业务压力的问题，而忽略了对于代码质量的负面影响，导致代码质量越来越差。

四个阶段

我把代码质量管控通常需要经历的四个阶段，称之为“四个现代化”：

- 规范化 - 建立代码规范与Code Review制度
- 自动化 - 使用工具自动检查代码质量
- 流程化 - 将代码质量检查与代码流动过程绑定
- 中心化 - 以团队整体为视角，集中管理代码规范，并实现质量状况透明化

阶段一：规范化

保障代码质量的基础是建立团队的代码规范，通常包括：

- 风格规范 - 缩进、换行、大小写等风格问题
- 实践规范 - 规避一些常见的隐患，或者针对特定问题的最佳实践
- 业务规范 - 与业务有关的特殊要求，比如文案中的关键词

团队的代码规范通常以文档的形式存在，供新人们学习。但文档这种形式常见的情况就是新人看过之后就不再回顾了，也很难对实际写代码形成真正的约束。

在规范的基础上，要通过Code Review将规范落地。Code Review中大家可以对代码质量问题进行交流，并且相互监督，形成团队重视代码的习惯。

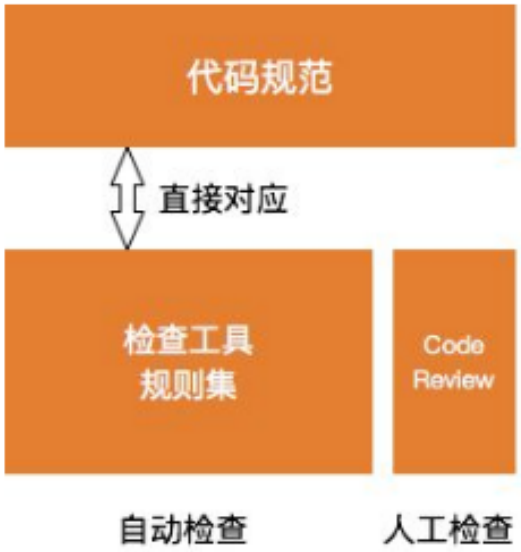
关于Code Review可以参考另一篇文章：Code Review体系与团队文化

阶段二：自动化

自动化是指在代码规范的基础上，使用自动化工具进行质量检查，通常包括：

- 代码规范检查 - 包括风格规范、实践规范、业务规范
- 重复率 - 重复出现的代码区块占比，通常要求在5%以下
- 复杂度 - 总行数，模块大小，循环复杂度等
- 检查覆盖度 - 经过检查的行数占代码库总行数的比例

自动化质量检查可以覆盖多数常见问题，能够提升开发效率，也可以降低人工Code Review的成本。



自动化检查工具的规则集与代码规范直接对应。通过编辑器插件，在写代码的时候直接给出检查结果。到这个阶段，团队的代码规范文档已经不再需要陈述各种细节，开发者可以直接通过查看自动化工具的规则集来了解代码规范。

阶段三：流程化

流程化的意思是将自动化代码质量检查和Code Review与代码流动的过程绑定，从而保证所有上线的代码都经过机器与人工多个环节的检查。

代码流动过程：

执行自动化代码质量检查的时机：

- 编辑时 - 使用编辑器插件，实时运行质量检查
- 构建时 - 在本地或者开发机的构建脚本中运行质量检查
- 提交时 - 利用Git Hooks，提交代码或者生成Pull Request时运行质量检查
- 发布时 - 在发布脚本中再做一次质量检查，通常与自动化测试放在一起

质量检查与代码流动绑定后的效果：

除了人工的Code Review之外，各个环节的代码质量检查都是机器自动运行的，不会给开发者带来额外的成本。

阶段四：中心化

当团队规模越来越大，项目越来越多时，代码质量管控就会面临以下问题：

- 不同项目使用的代码规范不一样
- 部分项目由于放松要求，没有接入质量检查，或者存在大量未修复的缺陷
- 无法从团队整体层面上体现各个项目的质量状况对比

为了应对以上问题，需要建设中心化的代码质量管控体系，要点包括：

- 代码规范统一管理。使用Git或者NPM包管理自动化代码质量检查的规则集，自动安装，不在本地写规则。一个团队、一类项目、一套规则。
- 使用统一的持续集成服务。质量检查不通过的项目不能上线。
- 建立代码质量评分制度。让项目与项目之间能够横向对比，项目自身能够纵向对比，并且进行汇总反馈。

总结

在面临业务压力时，人们通常会倾向于通过增加人力来缓解业务压力。但从系统整体的角度来看，人力增加会造成代码质量变差，开发效率下降，从而再度增大业务压力。这种代码质量越来越差的循环，是熵增定律在软件工程领域的生动体现。

为了抑制这种循环，我们需要有意识地投入资源来建设代码质量的管控体系。这个过程分为四个阶段：规范化，自动化，流程化，中心化。每个阶段都有不同关注的要点。

目前我们团队正在建设代码质量检测中心，是中心化质量管控的一种实践，已经接入几十个项目在进行试点。关于中心化建设的细节，我会在后续的文章中继续阐述。

转 如何评价代码质量

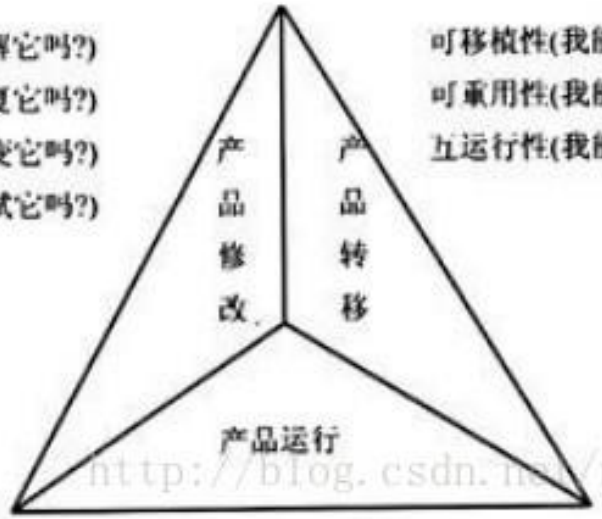
2017年07月01日 13:34:14 阅读数：692 标签： 产品质量 软件 标准 更多

我们平时买东西的时候，要看一看东西的质量怎么样，如颜色好看否、样式时尚否、经久耐用否，然后再决定买不买。

软件作为一种商品，也存在质量高低之分，从哪些方面来评价软件的质量状况呢？

代码是软件的元素，软件是产品的灵魂，所以评价代码质量的标准源于评价产品质量的标准。 产品服务于用户，用户的评价体现了代码的质量，用户在使用软件产品时的三种不同倾向或观点：产品运行、产品修改和产品转移。

可理解性(我能理解它吗?)
可维修性(我能修复它吗?)
灵活性 (我能改变它吗?)
可测试性(我能测试它吗?)



可移植性(我能在另一台机器上使用它吗?)
可重用性(我能再用它的某些部分吗?)
互运行性(我能把它和另一个系统结合吗?)

正确性(它按我的需要工作吗?)
健壮性(对意外环境它能适当地响应吗?)
效率(完成预定功能时它需要的计算机资源多吗?)
完整性(它是安全的吗?)
可用性(我能使用它吗?)
风险性(能按预定计划完成它吗?)

由图可见，评价代码质量的主要指标：

1. 正确性（Correctness） 系统满足规格说明和用户目标的程度，即在预定环境下能正确地完成预期功能的程度。 如软件有没有按照需求规格来完成，计算出的结果是否正确，计算结果是否精确。

2.健壮性/鲁棒性(Robustness) 健壮性是指在异常情况下，如硬件发生故障、输入的数据无效或操作错误等软件能够正常运行的能力。 健壮性有两层含义：一是容错能力，二是恢复能力。 容错是指发生异常情况时系统不出错误的能力，对于应用于航空航天、武器、金融等领域的这类高风险 系统，容错设计非常重要。而恢复则是指软件发生错误后（不论死活）重新运行时，能否恢复到没有发生错误前的状态的能力。 例如：因输入数据不正确，引起系统异常，这是容错能力不高引起的健壮性问题；操作系统死机了，重启后能够正常使用，说明具有一定恢复能力，具有一定的健壮性；数据库发生故障后，再次启动时一般能 够恢复到正常的状态，恢复能力比较好。

3. 可靠性（Reliability） 软件系统在一定的时间内无故障运行的能力。 可靠性是一个与时间相关的属性，指的是在一定环境下，在一定的时间段内，程序不出现故障的概率，因此是一个统计量，通常用平均无故障时间（MTTF, mean-time to fault）来衡量。 可靠性不同于正确性和健壮性，软件可靠性问题通常是由于设计中没有料到的异常和测试中没有暴露的代码缺陷引起的。 例：由于某个地方数据库连接没有释放，在长时间运行的时候，出现活动的数据库连接数过多，造成系统越来越慢，甚至停止服务。健壮性是指在异常情况下（如硬件发生故障、输入的数据无效或操作错误等），软件能够正常运行的能力。

4. 性能（Performance） 性能是指软件及时提供相应服务的能力。 具体而言， 性能包括速度、 吞吐量和持续高速性三方面的要求：速度往往通过平均响应时间来度量； 吞吐量通过单位时间处理的交易数来度量； 持续高速性是指保持高度处理速度的能力。 效率（Efficiency）指软件对 CPU 处理能力和存储能力这两大类计算机资源的使用效率。效率和性能反映了同一问题的“表”、“里”，性能为“表”，效率为“里”。 如系统运算一个报表，需要很长时间，这就是性能问题。

5. 安全性（Security） 指软件同时兼顾向合法用户提供服务，以及阻止非授权使用软件及资源的能力。 安全性既属于技术问题又属于管理问题。一般地，如果黑客为非法入侵花费的代价（考虑时间、费用、风险等多种因素）高于得到的好处，那么这样的系统就可以认为是安全的。 例：有人可以访问非授权的资源，这就是安全性问题。

6. 易用性（Usability） 易用性是指用户使用软件的容易程度。软件的易用性要让用户来评价。例：对于一般用户而言，Windows 的易用性比 Linux 的高。

7. 可用性（Availability） 指的是产品对用户来说有效、易学、高效、好记、少错和令人满意的程度，即用户能否用软件完成他的任务，效率如何，主观感受怎样。ISO 9241-11 国际标准对可用性作了如下定义：产品在特定使用环境下为特定用户用于特定用途时所具有的有效性（effectiveness）、效率（efficiency）和用户主观满意度（satisfaction）。其中：有效性：用户完成特定任务和达到特定目标时所具有的正确和完整程度；效率：用户完成任务的正确和完整程度与所使用资源（如时间）之间的比率；满意度：用户在使用产品过程中所感受到的主观满意和接受程度。

8. 互操作性（Interoperability） 指本软件与其他系统交换数据和相互调用服务用以协同运作的难易程度。例：利用 Web Service 增加软件的互操作性。

7. 可用性（Availability） 指的是产品对用户来说有效、易学、高效、好记、少错和令人满意的程度，即用户能否用软件完成他的任务，效率如何，主观感受怎样。ISO 9241-11 国际标准对可用性作了如下定义：产品在特定使用环境下为特定用户用于特定用途时所具有的有效性（effectiveness）、效率（efficiency）和用户主观满意度（satisfaction）。其中：有效性：用户完成特定任务和达到特定目标时所具有的正确和完整程度；效率：用户完成任务的正确和完整程度与所使用资源（如时间）之间的比率；满意度：用户在使用产品过程中所感受到的主观满意和接受程度。

8. 互操作性（Interoperability） 指本软件与其他系统交换数据和相互调用服务用以协同运作的难易程度。例：利用 Web Service 增加软件的互操作性。

9. 易理解性（Understandability） 理解和使用系统的难易程度。

10. 可扩展性（Extensibility）/灵活性（Flexibility）/适应性（Adaptability）/可伸缩性（Scalability） 反映软件适应"变化"的能力。调整、修改或改进正在运行的软件系统以适应新需求、变化了的需求的难易程度。例如报销系统原来不需要总经理审批，现在要改为总经理审批，可扩展性强的系统不需要作太多调整；如用户和数据量增加时，通过增加服务器来提高系统性能，这样可伸缩性比较强。

11. 可重用性（Resuability） 重用软件或其中一部分的难易程度。

12. 可测试性（Testability） 对软件测试以证明其满足需求规约的难易程度。

13. 可维护性（Maintainability） 为修改 Bug、增加功能、提高质量而诊断并修改软件的难易程度。

14. 可移植性（Portability） 软件不经修改或稍加修改就可以运行于不同软硬件环境的难易程度，主要体现为代码的可移植性。