质量管理:软件质量之路-软件质量框架 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/260/2021_2022__E8_B4_A8_E9_87_8F_E7_AE_A1_E7_c41_260164.htm 软件质量的重要性是不言而喻的，但是当所有人都意识到它的重要性的时候，却很少有人能够清晰的描述出如何才能够提高软件质量。软件质量框架的目的就在于提出一个评价的原型，帮助我们分析一种方法和技术是否能够提高软件质量。本系列文章分日构建、测试驱动开发、建立核心框架、面向组件的大规模软件架构来进行深入分析。 什么才是一个高质量的软件 在讨论软件质量原型之前，我们先回答第一个问题。一个软件之所以被认定为质量优秀，并不是因为它获得了一个省级或部级奖，而是它的内在具备了这样一些特性： 满足用户的需求。这是最重要的一点，一个软件如果不能够满足用户的需要，设计的再好，采用的技术再先进，也没有任何的意义。所以这一点非常的朴实，但却是软件质量的第一个评判标准。 合理进度、成本、功能关系。软件开发中所有的管理都是围绕着这几个要素在做文章的，如何在特定的时间内，以特定的成本，开发出特定功能的软件。三者之间存在一种微妙的平衡。在Planning XP一书中，专门有一个章节讨论它们。一个高质量的软件的开发过程中，项目成员一定能够客观的对待这三个因素，并通过有效的计划、管理、控制，使得三者之间达成一种平衡，保证产出的最大化。 具备扩展性和灵活性，能够适应一定程度的需求变化。当今的社会已经变成一种变化速度极快的设计了。变化就会对软件产生冲击，所以一个质量优秀的软件，应该能够在一定程度上适应这种变化，并

保持软件的稳定。 能够有效的处理例外的情况。写过软件的人都知道，实现主体功能的工作量其实不大，真正的工作量都在处理各种例外。所以，一个软件如果能够足够的强壮、足够的鲁棒，能够承受各种的非法情况的冲击，这个软件就是高质量的。 保持成本和性能的平衡。性能往往来源于客户的非功能需求，是软件质量的一个重要的评价因素。但是性能问题在任何地方都存在，所以需要客观的看待它。例如，一段性能不错的代码可能可读性很差，这就需要进行平衡，如果这段代码的性能是整个软件的关键，那么取高性能而舍弃可读性，反之则取可读性而舍弃高性能。一个优秀的软件能够保持成本和性能之间的平衡。 能够可持续的发展。很少有软件组织只开发一个软件的，所以，一个优秀的软件在开发完成后，可以形成知识沉淀，为软件组织的长期发展贡献力量。这是一个优秀的软件应该要能够做到的。 软件质量框架的组成 软件质量框架不是理论，而是优秀软件开发思想的一个应用，是对软件开发过程的有效管理实践。它以敏捷方法论为基础，并将先进的软件开发技术融入其中。您可能在之前听说过，学习过，尝试过各种软件技术，但是缺少一个统一整体的认识。所以，软件质量框架的目的是将您原先在脑海中就存在的思路进一步的整理，将一副完整的图像（big picture）展现在你面前。软件质量框架偏重应用，所以不会涉及太多的理论，但是，它是基于理论的，所以，在需要理论支持的地方，我们会简单的描述理论，并给出必要的链接，供有兴趣的读者进一步阅读。 软件质量框架并不复杂，它由几个部分组成，第一部分是前提，说明了软件框架的适用范围，以及适合的环境，和方法学一样，没有泛之四海皆准

的方法学，所以软件质量框架也需要一个上下文环境。第二部分是价值观，价值观说明了软件质量框架中强调的价值，在软件框架的结构和实践中，都将充分的的表现出一开始我们定义的价值。第三部分是结构。结构定义了软件质量框架的组成部分，以及软件质量框架和开发过程之间的关系。第四部分是文章中着墨最多的部分，即优秀实践。优秀实践通过具体、实际的分析、举例，深入阐述了软件质量框架的价值观和结构。 在本文剩下的篇幅中，将会对前三个部分进行阐述，并对软件质量开发的实践进行简单的描述。在剩余的篇章中，将会针对这些实践进行细致的分析。 软件质量框架的前提 平台前提：由于软件质量框架的实践将会涉及具体的技术和代码，所以我们首先为软件质量框架定义了平台。软件质量框架将会运行在J2EE平台上，使用对象分析技术（并不一定是面向对象技术，我们可以采用以数据为中心的技术）。 组织前提：执行软件质量框架需要投入，需要付出，软件质量框架最难的地方不是学习，而是执行。在一个组织中，需要评估应用软件质量框架需要多少的投入，对目前的开发过程有多大的助益。一般来说，组织的规模越大、其开发过程和产品越复杂，就越适合采用软件质量框架。 方法学前提：在敏捷方法学中，对规则和秩序有两种不同的观点，一种是强调规则和秩序，以XP为代表，它对代码都有要求；另一种则不那么强调，以自适应软件开发为代表，它不要求程序员的具体行为。软件质量框架采用第一种观点，要求组织中存在严谨的规则和秩序。 软件质量框架的价值观 明确具体：对软件的管理必须是明确具体的。软件开发是工程、也是艺术，需要紧密的协作和沟通，任何一个含糊的指令都可能

导致软件开发中出现错误，所以，在软件开发中，任何一个指令都应该是相对明确的。为什么说是相对呢？是和成本相对，指令越明确，成本就越高。例如，你可以把需求文档写的非常的具体，但是你需要付出制作和维护的代价。所以我们的明确性是一个考虑成本前提下的特性。 明确具体要从综合上考量。怎么理解呢？例如，XP中的用户故事是非常不精确的，按道理说它是不明确，也是不具体的。但是在整个开发周期中，将会有迭代、测试、现场用户等多种手段使得用户故事明确具体起来，所以从整体上看，它并不违反我们的价值观。产品质量是一个系统工程，决不仅仅是QA部门的工作，这个道理适用于制造业，也适用于软件开发业。 容错：软件开发是人的工作，人是无法避免错误的。所以，软件质量框架中允许犯错。因为不犯错是天方夜谭。你就算做了这方面的强制规定也无法避免它的出现，反而会引发其它的问题，例如隐瞒错误，或为了隐瞒错误而导致的额外成本。所以正确的态度是允许发生错误，并建立一套监测、管理、反馈、修改错误的体制。 规范：在前提中，我们已经提到了，规范是软件质量框架的基本态度。所以，软件质量框架中强调规范，并使用规范来推动框架的运作。 测试：软件质量框架非常强调测试，测试是保证质量的必由之路。测试要尽可能的多，尽可能的频繁、测试结果要尽可能快的反馈。这是软件质量框架对测试的基本态度。测试是综合性的，软件开发过程中的所有工件，都需要伴随着相应的测试工件。这是基于一个简单的理念，如果你不能够为你的工作制定一个完成的标准，你又该如何开展你的工作呢？ 上图表现了软件质量框架的结构。处于结构核心的是技术架构和管理架构。软

件质量框架既不是方法学，也不是一个软件，更像是两者的结合体。技术架构和管理架构的融合体现了这一特性。软件质量框架并不关心单个开发人员的效率，它关注的是开发团队整体的效率。因此，管理架构在框架中的意义在于它定义了一套软件管理的方法，能够对开发人员及其他们的工作进行管理。从这一点来看，它的作用和软件工程方法学是一样的。但是，在现实中我们发现软件组织在迈向软件过程的途中往往因为现实的困难而止步不前。其中一个主要的原因是在引入方法学的过程中生产效率降低了，而引起组织成员对变革的怀疑和不满。 所以，除了管理架构之外，软件质量框架还提供了一个技术架构，其目的是明确的定义如何应用组织中涉及的软件技术，以及管理软件技术的方法。技术架构是具体的代码，相比起方法学来说，它更加的具体，更容易为开发人员所理解。而技术架构存在的目的，一方面是进行技术积累，另一方面也是为管理架构服务。 技术架构和管理架构的下一层是支撑框架。支撑框架包括代码、组件、文档，目的是为技术架构和管理架构提供底层的支持。 处于结构最顶层的是业务架构。这个部分对于任何一个软件组织来说都是不同的，因为不同的软件组织的业务不同。业务架构的目的是对业务进行建模和抽象，提取出可重用的部分，以提高软件组织的生产率。本文中不涉及该部分的内容。 软件质量框架的优秀实践 一个开发团队要提高效率，就需要思考目前的管理活动中有哪些要素是可以改进的：如何把一些事务性的操作变得自动化，从而节约人力；如何找到更好的方法，让开发过程更为合理，更注重软件的质量；如何在团队中传播优秀的思想，让团队成员不断的学习和进取，自发的改

进过程。这些美好的愿望几乎是所有方法论和各种认证的共同心声，但要完全做到可就太难了。在我们的文章中，提出了一些优秀的实践，优秀实践均是来源于软件开发界中的一些新思路和新理论，它们能够为以上愿望的达成起到正面的作用。在组织中引入用这些实践决不是一个容易的过程，但它们确实非常的有效。不论是在成本控制上，还是在质量的改进上。 每日创建：一个组织应当拥有一个有效的工作流程，这个工作流程能够指导软件开发的进行。这个流程应当是具体的、可操作的。随意的计划和从来不遵循的进度决不是一个有效的工作流程。日创建实践提出了一种对开发过程进行精细管理的方法，它是量化软件管理的基础。有了日创建，你会发现计划的制定和进度的监控是非常容易的一件事情。 测试驱动开发：软件质量的根源来源于测试，测试做好了，软件质量就会好。这是毫无疑问的。问题的关键在于怎么做测试，才能保证测试的投入能够带来软件质量的有效提升。测试驱动开发正是为了解决这个问题而出现的。它不是一个完整的方法论，可以和任何一种开发流程进行融合。测试驱动开发不但能够改善测试效果，还能够改进软件的设计。建立核心框架：框架是一种具有高度重用性的软件，这个特性决定了它非常适合成为软件组织积累知识的一种有效手段。传统的知识积累的方法是文档，但是文档容易产生歧异，开发人员往往也不愿意去阅读和理解文档。框架提供的是一种综合的手段，包括文档、模型和代码。更容易理解，更重要的是，开发人员必须在日常的工作中使用框架，这使得他们对框架中的知识非常的熟悉，并根据工作的需要来改进框架。 面向组件编程：有效的组织在于有效的分工。体力活动

容易进行分工，脑力劳动则比较难，而软件开发似乎就更难了。所以，长久以来我们都习惯采用以功能块为单位的粗粒度划分方式。面向组件编程采用更加细密的划分方式，并以服务作为组件之间相互依赖的契约，不但定义了组件和组件之间的关系，也规定了组件开发者、组件使用者、组件测试者的权利和义务。从而能够进行软件开发工作的分配、管理、QA等工作。 以上的几个优秀实践看起来似乎并没有多大的关系，他们的提出者也大都不同。但是有一点却是共同的，就是他们都能够对软件质量的改进起积极的作用。此外，他们为软件质量框架结构的实现提供了一个明确的实现方式。从软件结构的角度来看，日创建和测试驱动开发似乎偏向于管理架构，而建立核心框架和面向组件编程则偏向于技术架构。事实上，他们既包含了技术架构，也包含管理架构，彼此之间也有相互关联。例如，面向组件编程在合理划分组件之后，就需要一个有效的核心框架来集成组件，通过每个组件都需要采用测试驱动开发方法来保证质量，同时，日创建将会以组件为单位来进行每日的创建，从而为进度估算提供有效数据。 随着软件设计技术的发展，新的实践将会出现，取代旧的实践。因此，以上的实践也会落伍，当可以肯定的是，以上的实践和具体的技术并没有直接的关系，更侧重于开发思想，因此他们的生命力会很长。而随着新技术的出现，他们更可能将新的技术融合如自身，呈现出一种崭新的形态。例如，未来的一种可能性是UML2.0和MDA技术的普及，以上的几个实践从以代码为核心转变为以设计为核心，而另一种可能性是随着以AspectJ为代表的AOP技术的普及和J2SE1.5中引入的元数据机制，面向组件编程将把Aspect作为

组件的一种，而测试驱动开发也会加入测试Aspect的相关内容，在日创建中也会增加相应的处理AOP的步骤。