ICS 03.120.10
CCS A 00

DB32

江　苏　省　地　方　标　准

DB32/T 5215—2025

小微企业质量管理提升指南　软件企业

Guideline for improving quality management in small and micro enterprises—Software enterprise

2025-09-10发布　　2025-10-10实施

江苏省市场监督管理局　发布
中国标准出版社　出版

目　　次

前　言

本文件按照GB/T 1.1—2020《标准化工作导则　第1部分：标准化文件的结构和起草规则》的规定起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别专利的责任。

本文件由江苏省市场监督管理局提出、归口并组织实施。

本文件起草单位：南京市雨花台区市场监督管理局、方圆标志认证集团江苏有限公司、江苏省产品质量监督检验研究院、江苏省质量和标准化研究院。

本文件主要起草人：顾正、马雪龙、徐新宇、姚卓、王卿、鲁杰、蒋林辉。

小微企业质量管理提升指南　软件企业

1　范围

本文件确立了小微软件企业质量管理提升的总体原则，给出了需求分析过程管理、软件开发过程管理及软件测试过程管理的建议。

本文件适用于指导小微软件企业实施核心过程的质量管理提升行动。

2　规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 8566　系统与软件工程　软件生命周期过程

GB/T 11457　信息技术　软件工程术语

GB/T 19000　质量管理体系　基础和术语

3　术语和定义

GB/T 8566、GB/T 11457、GB/T 19000 界定的以及下列术语和定义适用于本文件。

3.1

小微企业　small and micro enterprise

小型企业、微型企业、家庭作坊式企业的统称。

注：符合国家统计局《统计上大中小微型企业划分办法（2017）》（国统字〔2017〕213 号）规定的小型、微型企业划分，具有独立法人以及法人分支机构资格的社会经济组织。

3.2

软件企业　software enterprise

在中国境内依法设立的从事软件产品开发销售（营业）及相关服务的企业。

4　总体原则

4.1　问题导向原则

以问题为导向，聚焦小微软件企业质量管理中存在的痛点、难点问题，建立符合行业特征和企业需求的质量管理体系。

4.2　针对性原则

聚焦小微软件企业的特定需求、问题和特点及其行业特殊性，准确识别小微软件企业在质量提升过程中面临的主要挑战，将有限的资源集中用于解决关键质量问题，提高资源利用效率。

4.3 适宜性原则

考虑小微软件企业在技术、经济、管理等多个方面的适宜性，标准所提出的方法和管理要求等既符合小微软件企业的技术能力和经济承受能力，又与企业的管理模式和组织架构相适配。

4.4 持续改进原则

建立持续改进的理念与机制，将质量提升作为长期目标和常态化工作，发现存在的问题和不足，积极寻求改进措施。

5 需求分析过程管理

5.1 控制要点

5.1.1 需求获取

5.1.1.1 与相关利益方充分沟通，对项目的预期成果有清晰、一致的理解，避免目标模糊或不明确导致需求获取的偏差。

5.1.1.2 需求分析宜准确、完整且清晰，避免因语言表达、文化背景或专业知识差异导致对需求的误解，避免项目后期的变更和风险，确保项目交付成果符合预期。

5.1.1.3 利益相关方的需求，不仅关注功能需求，还宜关注非功能需求。

5.1.2 任务分解

5.1.2.1 避免因误解导致任务分解方向偏差，宜涵盖需求的各个方面。功能需求到非功能需求，在任务分解宜有体现。

5.1.2.2 任务分解既不能过于粗放，导致工作内容不清晰、责任不明确，也不能过于细碎，增加管理成本和协调难度。

5.1.3 需求验证

5.1.3.1 对需求的技术可行性进行评估，确定是否能够实现所获取的需求，同时识别需求潜在的问题。为每个需求确定明确的验证方法和标准，以便在软件开发过程中及完成后能够对需求是否得到满足进行验证。

5.1.3.2 宜建立需求与后续设计、开发、测试等工作成果之间的追溯关系。当在测试或使用过程中发现问题时，能够准确地找到对应的需求来源，便于及时定位和解决问题。

5.1.4 需求输出

5.1.4.1 需求输出宜涵盖软件系统的所有功能需求。这包括对系统各个模块功能的详细描述，从用户界面的基本操作到后台复杂的业务逻辑处理。

5.1.4.2 输出的各个部分之间保持一致，避免出现架构无法支持功能实现或者功能需求与架构设计冲突的情况。

5.1.4.3 建立需求模型，采用结构化方式对需求基线进行描述、组织与管理，建立并维护可视化、数字化的表达，明确定义、标识和关联其核心要素及其依赖关系，消除非结构化需求描述导致的质量不可控风险，实现需求基线高效管理、自动化变更控制，并探索与人工智能大模型的深度集成以提升效能与质量。

5.2 实施

5.2.1 对所有相关方的需求和期望进行分类和优化，分类维度宜包含业务需求、用户需求、系统需求。采

用软件工程的语言结构对非形式的需求进行重新表述，形成软件需求的完整集。开展需求的动态行为分析和静态数据分析，将分析结果记录于需求文档和总体设计方案。建立可视化、可交互的模型，展示预期软件产品功能，收集利益相关方反馈建议，及时解决需求分析过程的潜在问题。形成明确的可验证条款与验收指标，建立需求－测试用例映射表。

5.2.2 对系统需求进行拆分，形成的独立功能点，功能点宜与利益相关方的需求一致，功能点间交互关系清晰。评估需求的技术可行性，建立需求优先级矩阵，宜设置明确的可验收指标以及责任人，作为最小化需求的跟踪。

5.2.3 采用静态和动态验证结合的方法对需求的完整性、一致性、可行性和清晰性进行评审验证，验证利益相关方的需求是否得到满足。针对不同类型的需求，宜分别设定详细的验证子目标。实施基线化管理，未经基线化的需求不宜进入下阶段工作，建立需求变更程序，任何变更须经过评审，基线化后的需求不宜擅自更改。

5.2.4 保持需求输出的完整性，对功能需求、性能需求、数据需求、界面需求、安全性需求宜进行详细描述，包括正常流程和异常情况下的功能操作。建立需求跟踪矩阵，随着需求的变更或项目的进展，及时更新需求跟踪矩阵。确保矩阵能够真实反映需求与开发工作的实际对应情况，以便有效地跟踪需求的变化和实现情况。建立需求变更机制，确保需求的变更受控，避免随意变更导致项目混乱。需求输出宜形成文档记录，保持文档记录的完整性和可追溯。

5.3 改进

5.3.1 设置指标并实施监控，指标宜包含需求评审缺陷密度、需求变更率和测试用例覆盖率等内容，开展需求实现偏差分析，制定偏差纠正措施。

5.3.2 宜评估所用需求管理工具的效能，及时优化响应流程。定期更新需求模型，并基于实际情况调整。

6 软件开发过程管理

6.1 控制要点

6.1.1 项目计划管理

6.1.1.1 制定覆盖项目全流程的质量管控方案，建立质量追溯机制，各阶段质量要素宜可验证、可追溯。

6.1.1.2 编制软件开发计划，宜包含功能点分解、进度节点、资源配置。输出经评审的软件设计说明书，明确系统架构和接口规范。

6.1.1.3 实施版本基线管理，确保开发过程可追溯。

6.1.2 业务协同管理

业务部门宜全程参与原型设计、功能验证等关键项目节点。建立业务需求与技术实现的联合评审机制，定期开展架构合理性评估，形成架构优化建议。

6.1.3 数据质量管理

制定数据完整性校验规则和异常数据处理机制，实施人工抽样和自动化脚本相结合的双重比对机制，建立数据质量监控体系，定期开展数据质量分析。

6.2 实施

6.2.1 开发团队宜依据经批准的软件需求说明书进行功能模块分解，采用结构化分析方法完成系统架构设计，输出概要设计说明书。概要设计说明书宜包括模块化功能组织结构图、系统全局数据结构定义、模

块间消息接口规范、分层数据流图、状态转换图和模块耦合度分析等内容。建立跨部门评审机制，组织评审，参与人员宜包括项目经理、系统架构师、质量保证工程师、测试人员和运维支持人员。评审宜输出评审会议纪要、问题跟踪表、架构合理性评估和风险项等记录。

6.2.2 对概要设计说明书细化分解，输出详细设计说明书。详细设计说明书宜包含接口参数校验规则、数据结构存储方案、错误码定义规范。实施双人互审机制，确保设计逻辑与需求规格双向追溯，接口定义与概要设计保持一致。

6.2.3 制定软件编码规范，明确命名规则、注释标准、安全编码要求，建立编码规范检查清单。实施代码版本基线管理，开展静态代码扫描、编码规范符合性检查和交叉代码走查。开发人员不宜擅自变更接口参数、全局变量、数据库结构，变更需求宜经评审后进行修改。变更实施后宜执行影响范围分析、回归测试用例更新和关联文档修订。

6.2.4 开发过程输出宜包括但不限于以下几个方面：

a） 各阶段节点的开发计划；

b） 输出设计说明书，包含概要设计与详细设计；

c） 设计方案评审会议纪要、问题跟踪表、架构合理性评估和风险项记录。

6.3 改进

6.3.1 软件质量保证计划

软件质量保证计划宜包括技术评审、软件测试、缺陷跟踪以及其他可能影响软件产品质量的技术要点。实施代码质量量化管理，量化指标宜包含设计文档评审缺陷密度、代码重复率和单元测试通过率。适宜时，建立代码审查缺陷跟踪机制和分类统计机制。

6.3.2 项目配置工具

建立配置管理体系，配置目录结构，设定目录访问和存取权限，对目录结构进行系统性策划。每个配置项宜标识作者、时间、版本号、状态等信息，定期编制配置状态报告，包含基线变更统计、版本发布记录、备份完整性验证。建立配置管理审计机制，定期检验不符合项关闭情况。

7 软件测试过程管理

7.1 控制要点

7.1.1 测试计划

7.1.1.1 测试范围宜覆盖软件的所有功能、特性、用户场景以及可能受到影响的接口等。

7.1.1.2 合理预估每个测试阶段的开始时间、结束时间和所需时长。测试阶段包括单元测试、集成测试、系统测试和验收测试等各个阶段。如果时间估算偏差过大，可能导致项目进度延迟或测试不充分。

7.1.1.3 根据测试任务的需求，合理分配人力、设备和工具等资源。例如，确定需要多少测试人员、是否需要专门的性能测试工具和环境等。

7.1.2 测试设计

7.1.2.1 建立分层测试策略体系，包含单元测试、集成测试和系统测试，适宜时，可采用组合测试方法。

7.1.2.2 测试用例能准确地发现软件中的缺陷，并且符合软件的需求和功能逻辑。测试用例覆盖尽可能多的代码路径、功能分支和用户操作场景。宜通过计算语句覆盖、分支覆盖等指标来衡量。测试数据宜真实、有效且具有代表性，能够模拟实际生产环境中的数据情况。

7.1.2.3　实施缺陷全流程跟踪，包含缺陷发现、定位、修复及验证，建立自动化测试框架，提升阶段集成测试效能。

7.1.3　测试执行

7.1.3.1　测试人员严格按照测试用例执行测试，记录测试结果，包括测试通过的情况、发现的缺陷以及测试过程中的异常情况等。

7.1.3.2　及时发现软件中的缺陷，准确地描述缺陷的症状、出现的条件、重现步骤等信息。同时，要对缺陷进行分类和优先级排序，确保严重的缺陷能够得到及时处理。

7.1.3.3　保证测试环境（包括硬件、软件、网络等）与实际生产环境尽可能相似，并且在测试过程中保持环境的稳定。

7.1.4　测试评估

7.1.4.1　测试结果能够真实反映软件的质量状况，包括缺陷数量、缺陷分布、测试覆盖率等指标的准确性。宜根据测试结果，对软件的质量进行客观评估，判断软件是否满足上线或交付的标准。避免主观臆断，要基于数据和事实进行评估。

7.1.4.2　对整个测试过程进行回顾，总结成功的经验和不足之处，为后续的测试项目提供参考。

7.2　实施

7.2.1　与开发团队、产品经理等相关人员进行沟通，对软件功能和特性的理解一致。组织多方人员对测试计划中的范围进行评审，从不同角度（如用户需求、业务逻辑、技术实现）检查是否完整。参考以往类似项目的测试时间和资源消耗情况，结合当前项目的特点进行调整。可邀请经验丰富的测试专家对时间估算和资源分配进行评估和指导。制定详细的测试计划模板，在模板中宜明确测试范围、时间、资源等关键部分的编写要求，确保计划的规范性和完整性。例如，规定测试范围要详细列出每个功能模块和特性，时间估算要细化到每个测试阶段和主要测试任务。

7.2.2　采用多种测试用例设计方法，如等价类划分、边界值分析、决策表、状态迁移等，以测试用例的全面性。结合代码结构，分析代码中的分支、循环等逻辑结构，设计能够覆盖这些结构的测试用例。同时，对测试用例进行同行审查，检查用例是否符合需求、是否有效且无冗余。建立测试数据生成机制，对于常规数据可使用工具自动生成，对于特殊数据则根据实际需求手动创建。对测试数据进行分类和管理，确保在测试过程中能够方便地获取和使用。

7.2.3　在测试执行前，宜对测试人员进行培训，使其熟悉测试流程、工具和规范。在测试过程中，定期检查测试人员的执行情况，及时发现和纠正不规范的行为。利用缺陷管理工具来记录和跟踪缺陷。在工具中设置缺陷状态（如新建、已分配、正在修复、已关闭等）、优先级（如高、中、低）等字段，方便对缺陷进行管理。在测试环境中安装监控工具，实时监测硬件资源使用情况、网络带宽等。定期对测试环境进行维护，如更新软件补丁、清理磁盘空间等，确保环境的稳定性。

7.2.4　对测试结果进行多次核对，尤其是缺陷数量和状态。对于重要的测试指标，宜采用不同的方法或工具进行验证，确保测试覆盖率数据的准确性。在项目初期定义好软件质量标准，如缺陷密度允许范围、关键功能的可靠性要求等。在测试评估阶段，将实际测试结果与质量标准进行对比，客观地评价软件质量。组织测试团队成员共同回顾测试过程中的各个环节，包括测试计划、设计、执行和缺陷管理等。分享经验教训，将这些内容整理成文档，形成知识库。

7.3　改进

7.3.1　根据测试计划检查中发现的缺失部分，补充测试计划中的内容，并制定相应的应对措施。优化测试计划中的资源分配和进度安排。如果发现资源不足，可申请增加资源或者调整测试策略以适应现有的

资源。建立严格的测试计划审批流程。测试计划完成后,经相关负责人的审批,确保测试计划的可行性和有效性。在审批过程中,宜充分考虑各方的意见,对测试计划进行必要的修改和完善。

7.3.2 对检查中发现的不准确或不完整的测试用例,进行修改和补充;对于未覆盖的测试场景,新增相应的测试用例。宜采用先进的测试用例设计方法,如边界值分析、等价类划分、状态转换等方法,提高测试用例的质量和覆盖率。制定测试用例的维护规范,明确在软件需求变更、缺陷修复等情况下如何更新测试用例。定期对测试用例进行评审和回顾。组织测试团队成员对测试用例进行评审,分享测试经验和教训,不断优化测试用例的设计和维护。

7.3.3 分析测试执行过程中导致效率低下的原因,对于测试用例执行顺序问题,可以按照功能模块或者测试场景的关联性重新排列执行顺序,减少测试环境的切换时间。可以制作测试环境搭建的脚本或者镜像,减少每次搭建环境的时间。

7.3.4 完善缺陷报告和跟踪系统。确保缺陷报告的内容完整、准确,并且能够方便地进行查询、统计和分析。建立缺陷管理的激励和考核机制,以提高整个团队对缺陷管理的重视程度。

7.3.5 建立测试进度的监控和预警机制。通过定期检查测试进度,与计划进行对比,当发现进度偏差超过一定范围时,及时发出预警,以便采取相应的措施进行调整。

7.3.6 建立硬件设备的维护和管理机制。定期对硬件设备进行维护,如清洁、检查硬件状态等,确保硬件设备的正常运行。同时,要对硬件资源的使用情况进行监控,避免资源的浪费。及时更新软件环境的版本,确保其稳定性和安全性。优化软件环境的配置,根据软件的运行要求和测试特点,调整软件环境的配置参数,如数据库的连接池大小、中间件的线程数等,以提高软件的性能和稳定性。

参 考 文 献

[1] GB/T 19001—2016 质量管理体系 要求
[2] T/SIA 002—2019 软件企业评估标准
[3] ISO/IEC/IEEE 12207:2017 Systems and software engineering—Software life cycle processes
[4] 统计上大中小微型企业划分办法(2017)(国统字〔2017〕213号)