Centro Universitário Católica de Santa Catarina

Curso Engenharia de software

Manual Técnico Suplementar

Qualidade de Software

Integrantes do Grupo:

Bruno Luis Pereira

Leonardo Raye de Aguiar

Luis Fillipe Venturini Quintiono

Ramires Silva Paes

Wellerson Kuan Meredyk

Professor(a): Diego Sauter Possamai

Joinville – SC
2025

Sumário

Introdução

Nos dias de hoje, os sistemas computacionais estão presentes em praticamente todos os aspectos da nossa vida. Essa realidade tornou a qualidade de software um elemento essencial para o sucesso de qualquer solução digital. Mais do que simplesmente funcionar, espera-se que o software seja confiável, seguro, acessível, eficiente, escalável e fácil de usar. Nesse cenário, qualidade deixou de ser apenas um diferencial e passou a ser uma necessidade estratégica.

Este manual foi pensado para ser um guia claro, atual e útil sobre os principais aspectos da qualidade de software. Trazemos aqui conceitos-chave, padrões internacionais, métodos, ferramentas práticas, exemplos do dia a dia e reflexões importantes para ajudar na construção de produtos digitais robustos e sustentáveis.

Organizamos o conteúdo em dez capítulos temáticos, cada um tratando de um ponto essencial no processo de planejar, desenvolver, testar e manter software de alta qualidade:

- **O Capítulo 1 apresenta os fundamentos da qualidade, com ideias de pensadores como Garvin, Deming e Juran, e também dos métodos ágeis e Lean.**
- **O Capítulo 2 mostra os modelos e normas mais usados no mercado, como ISO 25010, CMMI e ITIL.**
- **O Capítulo 3 trata da engenharia de requisitos, destacando a importância de especificações claras e bem definidas.**
- **O Capítulo 4 fala sobre testes mais avançados, como BDD, testes exploratórios e testes em diversas plataformas.**
- **O Capítulo 5 aborda o planejamento da qualidade com foco em garantias, mudanças e auditorias.**
- **O Capítulo 6 destaca o papel das pessoas, da cultura organizacional e da comunicação entre áreas.**
- **O Capítulo 7 traz práticas modernas aplicadas em times ágeis e DevOps.**
- **O Capítulo 8 trata da visualização de dados e da tomada de decisões baseadas em indicadores.**
- **O Capítulo 9 discute temas como segurança, ética e conformidade legal com leis como a LGPD.**

- **Por fim, o Capítulo 10 foca na experiência do usuário final e na percepção real de qualidade.**

Cada capítulo foi construído com base em fontes confiáveis da área de Engenharia de Software, seguindo as normas da ABNT para citação e bibliografia. A ideia é oferecer conteúdo que seja útil tanto para quem está estudando quanto para quem já trabalha na área de tecnologia.

Acreditamos que qualidade em software não se conquista apenas no fim do processo. Ela precisa estar presente desde o começo, de forma integrada, contínua e colaborativa. Esperamos que este material ajude você — estudante, profissional ou gestor — a criar soluções mais eficazes, humanas e centradas em quem vai usar o software.

Capítulo 1: Fundamentos e Filosofia da Qualidade

---

1. Introdução

A qualidade de software é um conceito fundamental na engenharia de software moderna, envolvendo tanto aspectos técnicos quanto percepções subjetivas dos usuários. Este capítulo apresenta os fundamentos da qualidade com base nos principais pensadores da área e nas abordagens contemporâneas aplicadas ao ciclo de vida do desenvolvimento de sistemas.

---

2. Conceitos Clássicos de Qualidade

2.1 Garvin e as 8 Dimensões da Qualidade

David A. Garvin definiu oito dimensões para avaliar a qualidade de um produto, adaptáveis ao software:

1. Desempenho
2. Conformidade
3. Características
4. Confiabilidade
5. Durabilidade
6. Serviçabilidade
7. Estética
8. Qualidade percebida

Exemplo: Um aplicativo de banco digital deve apresentar alta confiabilidade (sem travamentos) e serviçabilidade (fácil suporte e atualização).

2.2 W. Edwards Deming

Deming propôs 14 princípios para transformação da gestão com foco em melhoria contínua. Três deles têm aplicação direta em software:

- Eliminar métodos de inspeção em massa: adotar testes automatizados e integração contínua.
- Melhorar constantemente: aplicação de ciclos PDCA e feedback de usuário.
- Instituir liderança: gestão que facilita a qualidade em toda a equipe.

2.3 Joseph M. Juran

Conhecido pelo conceito de "trilogia da qualidade":

- **Planejamento da Qualidade**: identificar clientes e suas necessidades.

- **Controle da Qualidade**: monitorar performance do software.
- **Melhoria da Qualidade**: promover melhorias baseadas em métricas.

Aplicativo móvel que coleta métricas de uso e evolui com base no comportamento do usuário está seguindo o ciclo de Juran.

---

3. Custo da Não Qualidade (CNQ)

O custo da não qualidade é o valor gasto com correção de falhas, retrabalho, perda de imagem, entre outros. É dividido em:

- **Falhas internas:** erros detectados antes da entrega (bugs, retrabalho).
- **Falhas externas:** erros após entrega (suporte, reclamações, perda de clientes).

Uma falha crítica em uma API pode resultar em prejuízos financeiros e perda de credibilidade.

---

4. Qualidade de Produto vs. Qualidade de Processo

- **Qualidade de Produto:** Características observáveis do software (eficiência, usabilidade).
- **Qualidade de Processo:** Conformidade com boas práticas durante o desenvolvimento (modelo ágil, documentação, testes).

Um sistema pode ter boa qualidade de produto mesmo com um processo mal estruturado, mas isso raramente é sustentável.

---

5. Qualidade Percebida pelo Usuário

A percepção do usuário é subjetiva, mas essencial. Envolve:

- Velocidade de resposta
- Interface intuitiva
- Estabilidade
- Suporte eficaz

Ferramentas como **Hotjar**, **Google Analytics** e pesquisas de NPS ajudam a medir essa qualidade percebida.

---

6. Princípios Ágeis e Lean Aplicados à Qualidade

6.1 Lean Software Development

Baseado em eliminar desperdícios, entrega contínua de valor e melhoria constante.

6.2 Qualidade no Manifesto Ágil

- "Software funcionando mais que documentação abrangente"
- "Colaboração com o cliente mais que negociação de contratos"

Técnicas associadas:

- Testes automatizados
- Integração Contínua (CI/CD)
- Desenvolvimento orientado a testes (TDD)

---

7. Estudo de Caso

Sistema Web de Atendimento ao Cliente

Problema: Reclamações frequentes sobre lentidão e instabilidade.

Ações:

- Aplicar métricas de confiabilidade (Garvin)
- Levantar causas com ferramentas Lean (Ishikawa)
- Automatizar testes com Selenium (Deming)
- Medir NPS e aplicar melhorias iterativas (Juran)

Resultados:

- Redução de falhas em produção em 45%
- Aumento de NPS de 6,2 para 8,5

---

8. Referências

GARVIN, D. A. *Managing Quality: The Strategic and Competitive Edge*. Free Press, 1988.

DEMING, W. E. *Out of the Crisis*. MIT Press, 1986.

JURAN, J. M. *Juran on Quality by Design*. Free Press, 1992.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software*. 8. ed. McGraw Hill Brasil, 2016.

BECK, K. *Manifesto Ágil: Princípios*. Disponível em: https://agilemanifesto.org/

Poppendieck, M.; Poppendieck, T. *Lean Software Development: An Agile Toolkit*. Addison-Wesley, 2003.

Capítulo 2: Modelos e Normas de Qualidade

---

1. Introdução

A qualidade de software não é apenas uma meta subjetiva; ela pode e deve ser normatizada por meio de modelos e padrões reconhecidos internacionalmente. Este capítulo apresenta os principais modelos e normas que guiam o desenvolvimento e a avaliação de produtos e processos de software.

---

2. Normas de Qualidade de Produto

2.1 ISO/IEC 9126 e ISO/IEC 25010

A ISO 9126 foi uma norma pioneira para avaliação da qualidade do produto de software. Foi posteriormente substituída pela ISO/IEC 25010, que define um modelo de qualidade com 8 características principais:

1. Funcionalidade
2. Eficiência de desempenho
3. Compatibilidade
4. Usabilidade
5. Confiabilidade
6. Segurança
7. Manutenibilidade
8. Portabilidade

Exemplo: A usabilidade pode ser medida por testes com usuários e aplicação de heurísticas.

---

3. Normas de Qualidade de Processo

3.1 ISO/IEC 12207

Esta norma define o ciclo de vida do software, incluindo processos de:

- Aquisição
- Desenvolvimento
- Operação
- Manutenção

É útable para mapear responsabilidades e melhorar o controle sobre entregas.

3.2 ISO/IEC 29119

É um conjunto de padrões internacionais para **testes de software**, dividida em:

- Conceitos e definições (Parte 1)
- Processo de teste (Parte 2)
- Documentação (Parte 3)
- Técnicas de teste (Parte 4)
- Métricas de teste (Parte 5)

Uso comum: padronizar o plano de teste, casos de teste e relatórios.

---

4. Modelos de Maturidade e Avaliação de Processos

4.1 CMMI (Capability Maturity Model Integration)

Modelo de maturidade com 5 níveis:

1. Inicial
2. Gerenciado
3. Definido
4. Gerenciado quantitativamente
5. Em otimização

Organizações no nível 3 têm processos definidos e documentados.

4.2 MPS.BR (Melhoria de Processo do Software Brasileiro)

Iniciativa brasileira baseada no CMMI e adaptada à realidade nacional. Possui níveis de maturidade de G (inicial) até A (otimizado).

Empresas públicas frequentemente exigem MPS.BR em licitações de software.

4.3 ISO/IEC 15504 (SPICE)

Foca na **avaliação de processos** com base em atributos como:

- Performance
- Capacidade
- Cobertura

Usado como base para auditorias e certificações de processo.

---

5. ITIL: Qualidade em Serviços de TI

ITIL (Information Technology Infrastructure Library) é um conjunto de boas práticas voltadas para a gestão de serviços de TI.

É dividida em 5 etapas do ciclo de vida do serviço:

- Estratégia
- Desenho
- Transição
- Operação
- Melhoria Contínua

Ferramentas como ServiceNow e Jira Service Management aplicam princípios do ITIL.

---

6. Quadro Comparativo das Normas

Norma/Modelo	Foco	Aplicação Principal
ISO 25010	Produto	Qualidade do software entregue
ISO 12207	Processo	Ciclo de vida do software
ISO 29119	Testes	Teste e validação de software
CMMI	Maturidade	Nível de maturidade da organização
MPS.BR	Maturidade	Adaptação brasileira do CMMI
SPICE	Avaliação	Auditoria e melhoria de processo
ITIL	Serviço	Qualidade em serviços de TI

---

7. Estudo de Caso

Empresa de Desenvolvimento Web - Certificação MPS.BR

Problema: Alta rotatividade, bugs em produção e baixa previsibilidade de entregas.

Soluções:

- Adoção do MPS.BR nível F
- Mapeamento dos processos com base na ISO 12207

- Adoção de ITIL na operação de suporte

Resultados:

- Redução de 35% nos retrabalhos
- Melhora na comunicação entre equipes de QA, dev e suporte
- Reconhecimento em editais de órgãos públicos

---

8. Referências

ISO/IEC 25010:2011. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE)*.

ISO/IEC 12207:2017. *Systems and software engineering — Software life cycle processes*.

ISO/IEC 29119:2013. *Software and systems engineering — Software testing*.

ISO/IEC 15504:2004. *Information technology — Process assessment*.

CMMI Institute. *CMMI for Development, Version 1.3*.

SOFTEX. *Guia Geral MPS.BR*. Disponível em: https://softex.br/

AXELOS. *ITIL Foundation Handbook*. TSO, 2012.

Capítulo 3: Qualidade em Engenharia de Requisitos e Análise

---

1. Introdução

A engenharia de requisitos é uma etapa essencial no desenvolvimento de software. A qualidade dos requisitos impacta diretamente na confiabilidade, manutenibilidade e usabilidade do produto final. Este capítulo apresenta boas práticas, técnicas de análise, ferramentas e padrões para garantir requisitos claros, coerentes e testáveis.

---

2. Verificação e Validação de Requisitos

2.1 Verificação

Processo de revisão sistemática para garantir que os requisitos estejam:

- Documentados corretamente
- Consistentes com padrões e templates
- Aprovados pelas partes interessadas

2.2 Validação

Garante que os requisitos representem corretamente as necessidades do cliente.

Ferramentas: Reuniões de revisão, walkthroughs, protótipos e simulações.

---

3. Análise de Ambiguidade e Inconsistência

3.1 Ambiguidade

Palavras vagas ou abertas a interpretações diferentes.

Exemplo ruim: "O sistema deve ser rápido."
Exemplo bom: "O sistema deve responder em até 2 segundos para 95% das requisições."

3.2 Inconsistência

Conflitos lógicos entre requisitos. Por exemplo:

- Um requisito exige login obrigatório e outro permite navegação anônima.

3.3 Ferramentas para Detecção

- Revisões por pares
- Análise de causa raiz (ex: Diagrama de Ishikawa)
- Checklist de requisitos bem formados (IEEE 830)

---

4. Técnicas para Melhorar a Qualidade das Especificações

4.1 Casos de Uso

- Representam a interação entre usuários e o sistema
- Facilitam a compreensão do escopo funcional

4.2 User Stories (Histórias de Usuário)

Padrão ágil para definir requisitos de forma simples:

"Como [tipo de usuário], eu quero [funcionalidade] para [benefício]"

4.3 Protótipos e Wireframes

Visualizam requisitos e validam ideias antes da codificação.

4.4 Linguagens Especificadas

- BPMN (Business Process Model and Notation)
- UML (Diagramas de casos de uso, classes, sequência)

4.5 Ferramentas

- **Jira**: gestão de requisitos e backlog
- **Figma** / **Balsamiq**: prototipagem
- **Lucidchart**, **Draw.io**: diagramas

---

5. Rastreabilidade e Cobertura de Requisitos

5.1 Rastreabilidade

Permite acompanhar cada requisito desde sua origem até a implementação e testes.
Tipos:

- Rastreabilidade para frente: requisitos → design → código → testes
- Rastreabilidade para trás: testes → código → design → requisitos

5.2 Matriz de Rastreabilidade

Tabela que relaciona requisitos com testes, módulos e entregáveis.

Exemplo: Requirement ID RQ-007 está associado ao Test Case TC-007 e ao Módulo Financeiro.

5.3 Cobertura

Avalia o percentual de requisitos que já possuem testes definidos e implementados.

Ferramentas: TestRail, Zephyr, Xray (plugins do Jira)

---

6. Estudo de Caso

Projeto: Sistema de Agendamento Online para Clínica

Problemas Iniciais:

- Requisitos escritos com termos vagos ("interface intuitiva")
- Falta de rastreabilidade entre as funcionalidades e os testes

Soluções Aplicadas:

- Redefinição dos requisitos usando User Stories
- Protótipos validados com stakeholders
- Matriz de rastreabilidade entre funcionalidades e casos de teste

Resultados:

- Redução de 40% em solicitações de retrabalho
- Melhoria na satisfação dos usuários internos

---

7. Referências

SOMMERVILLE, I. *Engenharia de Software*. 10. ed. Pearson, 2019.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: Uma Abordagem Profissional*. 8. ed. McGraw Hill Brasil, 2016.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. *IEEE 830 - Recommended Practice for Software Requirements Specifications*, 1998.

COCKBURN, A. *Writing Effective Use Cases*. Addison-Wesley, 2000.

CSDM. *Guia para Boas Práticas em Requisitos de Software*. Conselho de Engenharia de Software, 2021.

Capítulo 4: Testes em Profundidade

---

1. Introdução

O teste de software é uma das principais atividades para assegurar a qualidade do produto final. Testes em profundidade não apenas verificam funcionalidades, mas também garantem robustez, segurança, performance e usabilidade em diferentes contextos. Este capítulo aborda técnicas modernas de teste, tipos especializados e ferramentas amplamente utilizadas no mercado.

---

2. Teste Baseado em Risco

Essa abordagem prioriza os testes com base nos riscos mais críticos ao negócio ou à estabilidade do sistema. Envolve:

- Identificação e classificação de riscos
- Planejamento de testes focado nas funcionalidades com maior impacto

Exemplo: Em um sistema bancário, testes priorizam transferências e autenticação.

Ferramentas:

- TestLink
- RiskStorming (Jogo de planejamento de risco para QA)

---

3. Teste Baseado em Modelo (Model-Based Testing - MBT)

O MBT utiliza modelos formais (como diagramas de estados ou fluxos de dados) para gerar casos de teste automaticamente.

Benefícios:

- Geração automatizada de cenários
- Cobertura sistemática
- Facilita manutenção dos testes com base no modelo

Ferramentas:

- GraphWalker
- Conformiq Creator
- Spec Explorer (Microsoft)

---

4. Testes Orientados por Comportamento (BDD)

BDD (Behavior-Driven Development) conecta os testes aos requisitos de negócio, com linguagem natural (Gherkin).

Formato:

Funcionalidade: Login seguro

Cenário: Login com senha incorreta

Dado que o usuário está na tela de login

Quando ele digitar a senha errada

Então deve ver uma mensagem de erro

Ferramentas:

- Cucumber (Java)
- SpecFlow (.NET)
- Behave (Python)

---

5. Testes Exploratórios

Não seguem scripts predefinidos. O testador explora livremente a aplicação com base na experiência e conhecimento do sistema.

Aplicados principalmente em:

- Ambientes desconhecidos
- Funcionalidades novas
- Descoberta de falhas inesperadas

Técnicas:

- Session-Based Testing (teste com sessões cronometradas)
- Charters (objetivos guias para sessão)

Ferramentas:

- Rapid Reporter
- TestBuddy
- Xray (Jira)

---

6. Testes em Plataformas Específicas

6.1 Testes em Aplicações Móveis

- Verificação de gestos, performance e compatibilidade
- Testes em diversos dispositivos reais ou emulados

Ferramentas:

- Appium
- Espresso (Android)
- XCTest (iOS)

6.2 Testes Web

- Testes de interface e responsividade
- Automatização com Selenium, Cypress

6.3 Testes para IoT e Sistemas Embarcados

- Simulação de sensores
- Validação em ambientes de hardware real

Ferramentas:

- TOSCA
- Ranorex
- Robot Framework

---

7. Estudo de Caso

Projeto: Aplicativo de Entregas sob Alta Carga de Usuários

Problemas Identificados:

- Falhas em geolocalização
- Travamentos em dispositivos Android
- Comportamento diferente em navegadores Web

Ações Aplicadas:

- Teste baseado em risco priorizou geolocalização e login
- Uso de Appium e BrowserStack para simulação multiplataforma

- BDD com Cucumber para cenários de cliente e entregador

Resultados:

- Redução de 60% em falhas de produção
- Melhoria significativa na experiência do usuário

---

8. Referências

AMBLER, S. W. *Agile Modeling and Testing*. Agile Data, 2010.

KOCHAN, G. *Exploratory Software Testing*. Microsoft Press, 2010.

WAKE, W. *Behavior-Driven Development*. Addison-Wesley, 2011.

COSTA, L. et al. *Testes de Software: Fundamentos e Práticas*. Novatec, 2020.

ISO/IEC/IEEE 29119-4:2015. *Software and systems engineering - Software testing - Part 4: Test techniques*.

Capítulo 5: Planejamento Estratégico da Qualidade

---

1. Introdução

O planejamento estratégico da qualidade é essencial para garantir que as atividades de desenvolvimento estejam alinhadas com os objetivos do projeto e as expectativas dos stakeholders. Neste capítulo, abordamos os elementos fundamentais para organizar, monitorar e controlar a qualidade ao longo do ciclo de vida do software, com ênfase em planos de garantia, responsabilidades, versões e auditorias.

---

2. Plano de Garantia da Qualidade (PQA)

O PQA é um documento formal que descreve:

- Objetivos da qualidade do projeto
- Critérios de aceitação
- Métricas e indicadores de qualidade
- Ações corretivas em caso de desvios

Componentes:

- Escopo do controle de qualidade
- Padrões adotados (ex: ISO 25010)
- Ferramentas utilizadas (SonarQube, TestRail)

Exemplo: Um PQA pode definir que 90% das funcionalidades devem passar em todos os testes automatizados antes de ir para staging.

---

3. Matriz de Responsabilidades (RACI)

A matriz RACI define quem é:

- **R**esponsável (Responsible): quem executa a tarefa
- **A**provador (Accountable): quem aprova e tem autoridade final
- **C**onsultado (Consulted): quem contribui com conhecimento
- **I**nformado (Informed): quem deve ser comunicado

Exemplo: No processo de revisão de código, o desenvolvedor é o R, o líder técnico é o A, o QA é o C e o PO é o I.

Ferramentas:

- Trello, ClickUp, Asana (para registrar responsabilidades)

---

4. Definição de Critérios de Aceitação

Critérios de aceite descrevem as condições mínimas para considerar uma funcionalidade "pronta".

Técnicas:

- Dê foco ao comportamento (ex: BDD)
- Use métricas objetivas (ex: tempo de resposta < 2s)
- Envolva stakeholders na definição

Exemplo: "A funcionalidade deve passar em todos os testes unitários e ser aprovada em revisão de código."

---

5. Gerenciamento de Configuração e Controle de Mudanças

5.1 Gerenciamento de Configuração

Controla os artefatos de software (código, documentos, builds). Atividades:

- Identificação de itens
- Controle de versão
- Auditoria de configuração

Ferramentas:

- Git, GitHub, GitLab
- Subversion (SVN)

5.2 Controle de Mudanças

Processo para analisar, aprovar e implementar alterações no projeto.

- Envolve Change Requests (CRs)
- Define impacto, custo, responsáveis

Exemplo: Uma mudança de layout no sistema exige aprovação do time de UX e comunicação com o cliente.

---

6. Controle de Versões e Auditoria de Builds

6.1 Controle de Versões

Garante que todas as alterações no código sejam rastreáveis e documentadas.

- Commits atômicos
- Branches nomeadas por funcionalidade
- Pull Requests com revisão obrigatória

6.2 Auditoria de Builds

Avalia se os pacotes gerados estão corretos, seguros e conforme os padrões.

Ferramentas:

- Jenkins, GitHub Actions, GitLab CI/CD
- SonarQube (qualidade do código)
- Nexus/Artifactory (repositórios de builds)

---

7. Estudo de Caso

Projeto: Plataforma de Ensino a Distância (EAD)

Desafios:

- Entregas com baixa previsibilidade
- Bugs em produção não rastreados

Ações:

- Elaboração de PQA com critérios claros
- Uso de matriz RACI para testes e revisões
- Controle de builds e releases com GitLab CI/CD
- Auditoria com SonarQube e testes automatizados

Resultados:

- Redução de 70% nos bugs em produção
- Releases quinzenais com maior estabilidade

---

8. Referências

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: Uma Abordagem Profissional*. 8. ed. McGraw Hill Brasil, 2016.

SOMMERVILLE, I. *Engenharia de Software*. 10. ed. Pearson, 2019.

ISO/IEC/IEEE 12207:2017. *Systems and software engineering — Software life cycle processes*.

COHEN, M. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley, 2010.

FOWLER, M. *Continuous Integration*. martinfowler.com.

Capítulo 6: Fatores Humanos e Organizacionais

---

1. Introdução

A qualidade de software vai muito além de processos e ferramentas: ela depende de pessoas, cultura organizacional e comunicação eficaz entre equipes. Este capítulo destaca os aspectos humanos e comportamentais que influenciam diretamente a qualidade do produto e do ambiente de desenvolvimento.

---

2. Cultura de Qualidade na Equipe

A cultura de qualidade se estabelece quando todos os membros do time assumem responsabilidade pela entrega de valor com confiabilidade e melhoria contínua.

Princípios:

- Qualidade como responsabilidade compartilhada
- Tolerância ao erro e aprendizagem
- Incentivo à melhoria contínua (Kaizen)

Práticas:

- Retrospectivas periódicas
- Celebração de pequenas conquistas
- Feedback construtivo

---

3. Comunicação entre Áreas (Dev, QA, Produto)

Comunicação eficaz evita retrabalho, melhora o entendimento dos requisitos e acelera a resolução de problemas.

Técnicas e ferramentas:

- Reuniões diárias (Daily Scrum)
- Documentação leve e colaborativa (Confluence, Notion)
- Ferramentas de chat (Slack, Microsoft Teams)

Exemplo: QA identifica inconsistência em requisito e consulta o PO antes de abrir bug, evitando retrabalho.

---

4. Papel do QA em Times Ágeis

O QA não atua apenas como testador, mas como facilitador da qualidade.

Atuações:

- Definição de critérios de aceite junto ao PO
- Planejamento de testes automatizados desde o início
- Apoio às squads com métricas e relatórios de falhas

Em times ágeis, a qualidade é construída diariamente, não apenas verificada ao final.

---

5. Gestão do Conhecimento e Documentação Viva

5.1 Documentação Viva

Documentações que evoluem junto com o software e refletem seu estado atual.

Ferramentas:

- Markdown em repositórios (Git)
- Notion, Confluence, Docusaurus

5.2 Repositório de Lições Aprendidas

- Registra erros recorrentes e soluções aplicadas
- Incentiva boas práticas e evita desperdício de conhecimento

---

6. Treinamento Contínuo de Times

Ambientes de aprendizado constante são fundamentais para manter a qualidade e a motivação da equipe.

Estratégias:

- PDI (Plano de Desenvolvimento Individual)
- Pares técnicos (pair programming, code reviews)
- Comunidades internas de prática

Ferramentas de apoio:

- Plataformas como Alura, Coursera, Udemy

- Treinamentos internos com especialistas da empresa

---

7. Estudo de Caso

Projeto: Migração de Sistema ERP para Nuvem

Desafios:

- Resistência de parte da equipe
- Falta de envolvimento dos testadores na fase inicial
- Documentação desatualizada

Ações:

- QA foi integrado desde a fase de levantamento
- Documentação migrada para Confluence com versões vivas
- Ciclos de treinamento interno com especialistas

Resultados:

- Redução de 50% no tempo de onboarding de novos devs
- Menos bugs em ambiente de produção

---

8. Referências

DINZEO, C. *Qualidade de Software com Foco em Pessoas*. Bookman, 2020.

SUTHERLAND, J. *Scrum: A Arte de Fazer o Dobro do Trabalho na Metade do Tempo*. Leya, 2016.

NONAKA, I.; TAKEUCHI, H. *The Knowledge-Creating Company*. Oxford University Press, 1995.

BECK, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2004.

PRESSMAN, R. S. *Engenharia de Software*. 8. ed. McGraw Hill, 2016.

Capítulo 7: Qualidade em Ambientes Ágeis e DevOps

---

1. Introdução

Ambientes ágeis e DevOps transformaram a forma como o software é desenvolvido, testado e entregue. A qualidade não é mais uma etapa ao final do processo, mas uma responsabilidade compartilhada em todo o ciclo de desenvolvimento. Este capítulo explora as práticas de qualidade integradas aos pipelines de entrega contínua, com foco em agilidade, automação e resiliência.

---

2. QA Contínua (Continuous Quality)

A qualidade é avaliada continuamente com base em testes automatizados, métricas de desempenho e feedback do usuário.

Práticas:

- Testes automatizados em todas as etapas (unitários, integração, e2e)
- Monitoramento de qualidade em tempo real (SonarQube, New Relic)
- Acompanhamento contínuo de KPIs de qualidade

Ferramentas:

- Jenkins, GitHub Actions, GitLab CI/CD
- Cypress, Selenium, JUnit, PyTest

---

3. Shift-left e Shift-right Testing

3.1 Shift-left

Antecipar os testes para o início do ciclo de desenvolvimento.

- Benefícios: detecta erros cedo, reduz custo de correção
- Exemplo: TDD, testes em PRs, validação de requisitos automatizada

3.2 Shift-right

Executar testes após a entrega, em ambiente real de produção ou staging.

- Benefícios: valida robustez, desempenho e experiência real
- Exemplo: monitoramento, testes A/B, coleta de logs e feedback

---

4. Integração da Qualidade nos Pipelines DevOps

O pipeline de CI/CD deve integrar:

- Lint e validação de código (Ex: ESLint, Prettier)
- Testes automatizados (unitários, integração, regressão)
- Análise estática de código (SonarQube)
- Geração de relatórios e artefatos

Exemplo: pipeline bloqueia o deploy se cobertura de testes for menor que 80%.

Ferramentas:

- Jenkins, CircleCI, GitHub Actions, Azure DevOps
- Nexus, Docker, Kubernetes

---

5. Feature Flags, Canary Releases e Rollback Controlado

5.1 Feature Flags

Permitem ativar ou desativar funcionalidades em tempo de execução sem novo deploy.

Exemplo: liberar nova página de checkout apenas para 5% dos usuários.

Ferramentas: LaunchDarkly, Unleash, ConfigCat

5.2 Canary Releases

Entregar uma nova funcionalidade para uma pequena parte dos usuários antes da liberação total.

5.3 Rollback Controlado

Permite desfazer releases com falha de forma segura e rápida.

- Ex: reversão de versão em ambiente Kubernetes via Helm

---

6. Chaos Engineering como Validação de Robustez

Técnica para simular falhas reais (latência, perda de conexão, indisponibilidade) e verificar a resiliência do sistema.

Princípios:

- Injetar falhas de forma controlada

- Medir impacto e tempo de recuperação
- Melhorar tolerância a falhas

Ferramentas:

- Chaos Monkey (Netflix)
- Gremlin
- LitmusChaos (Kubernetes)

---

7. Estudo de Caso

Projeto: Plataforma de Streaming com Implantação Contínua

Problemas:

- Falhas não detectadas nos ambientes de staging
- Reclamações de usuários após releases

Ações:

- Adotado shift-left com TDD e validação de requisitos
- Aplicado Chaos Engineering para simular instabilidades
- Releases com feature flags para testes A/B controlados
- Integração de qualidade com GitHub Actions e SonarQube

Resultados:

- Redução de 70% nos erros em produção
- Feedback positivo de usuários quanto à estabilidade

---

8. Referências

HUMBLE, J.; FARLEY, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010.

KNAUSS, E. et al. *Quality Assurance in DevOps: State of Practice and Challenges*. IEEE Software, 2020.

NETFLIX TECH BLOG. *The Netflix Simian Army*. Disponível em: https://netflixtechblog.com/

BASS, L.; WEBER, I.; ZHU, L. *DevOps: A Software Architect's Perspective*. Addison-Wesley, 2015.

BECK, K. *Test-Driven Development: By Example*. Addison-Wesley, 2002.

Capítulo 8: Visualização e Tomada de Decisão

---

1. Introdução

A visualização de dados desempenha um papel vital na tomada de decisão baseada em evidências. Em qualidade de software, dashboards, indicadores e mapas visuais ajudam equipes a entenderem rapidamente falhas, tendências e pontos de melhoria. Este capítulo explora ferramentas e boas práticas para transformar dados brutos em conhecimento acionável.

---

2. Dashboards de Qualidade

Painéis de controle visuais que reúnim dados de testes, desempenho, estabilidade e entregas.

Indicadores comuns:

- Bugs abertos por prioridade
- Cobertura de testes (% por módulo)
- Tempo médio de resolução de falhas

Ferramentas:

- Grafana, Kibana
- Power BI, Tableau
- Jira Dashboards, TestRail Reports

Exemplo: Dashboard exibe em tempo real o número de testes com falha em cada sprint.

---

3. Ferramentas de BI na Engenharia de Software

Business Intelligence (BI) permite consolidar dados de diferentes fontes para análise e previsão.

Casos de uso:

- Análise de produtividade de times
- Previsão de prazos e riscos
- Acompanhamento de KPIs de qualidade

Ferramentas:

- Power BI (Microsoft)
- Metabase (open-source)
- Google Data Studio

Times ágeis usam BI para entender velocidade e estabilidade entre sprints.

---

4. Heatmaps de Cobertura e Falhas

Mapas de calor (heatmaps) destacam visualmente as áreas do sistema mais cobertas por testes ou mais sujeitas a erros.

Tipos:

- Heatmap de cobertura de testes por módulo
- Heatmap de falhas por componente ou navegador

Ferramentas:

- SonarQube (cobertura e qualidade de código)
- Allure, Test Coverage Map
- Hotjar (comportamento do usuário em tela)

Exemplo: um heatmap revela que 70% das falhas ocorrem no checkout mobile.

---

5. Indicadores de Performance (KPI) e SLA de Testes

5.1 Indicadores de Qualidade (KPIs)

Indicadores ajudam a quantificar o desempenho e apontar melhorias.

- Taxa de defeitos por sprint
- % de automação de testes
- Tempo médio de ciclo de entrega (lead time)

5.2 Acordos de Nível de Serviço (SLA)

Definem metas mensuráveis de entrega e qualidade.

- SLA de resposta de bugs críticos em até 4h
- SLA de execução de regressão antes do deploy

Gestores usam KPIs e SLAs para avaliar fornecedores e liderar melhorias internas.

---

6. Estudo de Caso

Projeto: Plataforma de E-commerce Multicanal

Problemas:

- Difícil identificar onde estavam os principais gargalos de qualidade
- Equipe tomava decisões com base em percepções e não em dados

Ações:

- Implantação de dashboards com Power BI
- Uso de heatmaps de cobertura com SonarQube
- Definição de KPIs: % de testes automatizados, defeitos por sprint

Resultados:

- Melhor visibilidade para decisões rápidas
- Redução de 45% nas falhas em produção em 3 meses

---

7. Referências

FEW, S. *Information Dashboard Design*. O'Reilly, 2006.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software*. 8. ed. McGraw Hill, 2016.

SONARQUBE. *Documentation and Quality Gate Practices*. Disponível em: https://docs.sonarqube.org

TABLEAU SOFTWARE. *Visual Analytics Best Practices*. Disponível em: https://www.tableau.com

KIBANA DOCS. *Elastic Visualization Guide*. Disponível em: https://www.elastic.co/guide/en/kibana/

Capítulo 9: Conformidade, Ética e Segurança

---

1. Introdução

A qualidade de software também está associada ao cumprimento de normas legais, à segurança da informação e à responsabilidade ética no uso de dados e recursos. Este capítulo aborda como a conformidade, a segurança e a ética devem estar integradas desde o início do desenvolvimento, em uma abordagem conhecida como "qualidade com responsabilidade".

---

2. Qualidade e Conformidade Legal (LGPD, GDPR)

As leis de proteção de dados exigem que sistemas garantam:

- Consentimento do usuário para coleta de dados
- Armazenamento seguro e rastreável
- Possibilidade de exclusão dos dados mediante solicitação

2.1 LGPD (Lei Geral de Proteção de Dados - Brasil)

- Base legal para tratamento de dados pessoais
- Exige controle de acesso, criptografia e rastreabilidade

2.2 GDPR (General Data Protection Regulation - Europa)

- Influenciou diversas legislações mundiais
- Traz penalidades severas em caso de vazamento

Ferramentas:

- DataMasker, OneTrust, Controlle LGPD
- Mecanismos de "opt-in" e "opt-out"

---

3. Boas Práticas de Segurança (DevSecOps)

DevSecOps insere a segurança no ciclo DevOps desde o planejamento.

Práticas:

- Scans automatizados de vulnerabilidades (SAST, DAST)
- Gerenciamento seguro de secrets (ex: HashiCorp Vault)

- Monitoramento de logs e alertas (ex: Splunk, ELK)

Ferramentas:

- OWASP ZAP, Snyk, SonarQube Security
- GitHub Dependabot, Trivy (segurança em containers)

DevSecOps = segurança como código e cultura.

---

4. Ética em Testes de Software

4.1 Uso de Dados Reais

- Evitar uso de bases reais sem anonimização
- Preferir dados sintéticos ou mascarados

4.2 Testes Intrusivos

- Evitar testes que coloquem sistemas em risco em ambiente de produção
- Planejar autorizações e rollback seguros

4.3 Responsabilidade Social

- Garantir acessibilidade digital (WCAG)
- Evitar vieses em IA e algoritmos discriminatórios

---

5. Testes de Acessibilidade como Dimensão de Qualidade

A acessibilidade é parte da qualidade para todos os usuários.

Critérios (WCAG):

- Conteúdo perceptível (texto alternativo, contraste)
- Navegabilidade por teclado
- Compatibilidade com leitores de tela

Ferramentas:

- Axe DevTools, WAVE, Lighthouse
- NVDA (leitor de tela)

Software de qualidade deve ser inclusivo.

---

6. Estudo de Caso

Projeto: Portal de Benefícios Online para Servidores Públicos

Desafios:

- Reclamações de usuários com deficiência visual
- Vazamento de dados de e-mail marketing

Ações:

- Adoção da WCAG 2.1 e testes com NVDA
- Anonimização de dados reais nos ambientes de QA
- Implementação de DevSecOps com GitHub + Snyk

Resultados:

- Portal certificado como "acessível"
- Redução de 90% nos riscos de conformidade em auditoria externa

---

7. Referências

BRASIL. *Lei nº 13.709, de 14 de agosto de 2018 (LGPD)*.

UNIÃO EUROPEIA. *General Data Protection Regulation (GDPR)*. 2016.

OWASP. *Top 10 Security Risks*. Disponível em: https://owasp.org

W3C. *Web Content Accessibility Guidelines (WCAG)*. Disponível em: https://www.w3.org/WAI/standards-guidelines/wcag/

SNYK. *DevSecOps Essentials*. Disponível em: https://snyk.io

Capítulo 10: Qualidade do Produto Final

---

1. Introdução

A entrega de um software de qualidade é percebida pelo usuário final não apenas por sua funcionalidade, mas também por sua usabilidade, desempenho, suporte e evolutividade. Este capítulo trata da qualidade na entrega, uso e manutenção do produto em ambiente real.

---

2. Qualidade Percebida: UX, UI, Performance, Estabilidade

2.1 UX e UI (Experiência e Interface do Usuário)

- Interfaces intuitivas, responsivas e acessíveis
- Jornada fluida e consistente

Ferramentas:

- Figma, Adobe XD (prototipação)
- Hotjar, Crazy Egg (mapas de calor e cliques)

2.2 Performance

- Tempo de resposta ideal: abaixo de 2s para 90% das requisições
- Leveza em conexões móveis

2.3 Estabilidade

- Baixo número de falhas por versão
- Alta disponibilidade (uptime > 99%)

Usuários abandonam sistemas lentos ou instáveis mesmo que sejam funcionais.

---

3. Qualidade em Manutenção e Extensibilidade do Código

Critérios:

- Baixo acoplamento e alta coesão
- Modularização
- Cobertura de testes automatizados
- Comentários e documentação clara

Ferramentas:

- SonarQube (análise de métricas de manutenibilidade)
- ESLint, Prettier (padrões de código)

Código de qualidade permite correções rápidas e facilita evoluções futuras.

---

4. Suporte Técnico e Qualidade de Atendimento

Dimensões da qualidade de suporte:

- Tempo de resposta
- Cordialidade e empatia
- Precisão na resolução
- Base de conhecimento acessível

Ferramentas:

- Zendesk, Freshdesk, Jira Service Management
- FAQ e chatbots integrados

Métricas:

- CSAT (Customer Satisfaction Score)
- NPS (Net Promoter Score)

---

5. Feedback Loops com o Usuário

Coletar e incorporar feedback em ciclos curtos permite evoluir o produto continuamente.

Métodos:

- Pesquisas in-app (ex: NPS, 5 estrelas)
- Mapa de calor e gravação de sessão (Hotjar)
- Coleta de bugs e sugestões automatizadas

Ferramentas:

- Hotjar, FullStory, Usabilla
- Google Forms, Typeform

Feedbacks guiam melhorias centradas no usuário.

---

6. Testes A/B e Testes de Aceitação com Usuários Reais

6.1 Testes A/B

Comparar duas versões de uma funcionalidade para ver qual performa melhor.

- Ex: layout A vs. layout B do carrinho de compras

Ferramentas:

- Google Optimize, Optimizely, VWO

6.2 Testes de Aceitação com Usuários

- Convidar usuários reais para validar requisitos
- Observar comportamentos e dificuldades

Validação de aceito requer empatia e escuta ativa.

---

7. Estudo de Caso

Projeto: App de Agendamento de Consultas Médicas

Desafios:

- Usuários não completavam agendamentos
- Alta taxa de abandono na etapa de pagamento

Ações:

- Redesenho da UI com foco em UX simplificado
- Testes A/B para nova tela de checkout
- Inclusão de pesquisas de feedback ao final da consulta

Resultados:

- Conversão aumentou 35%
- Redução em 50% nas solicitações de suporte

---

8. Referências

GARRET, J. J. *The Elements of User Experience*. New Riders, 2010.

NIELSEN, J. *Usability Engineering*. Morgan Kaufmann, 1993.

KRUG, S. *Don't Make Me Think*. New Riders, 2014.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software*. 8. ed. McGraw Hill, 2016.

SONARQUBE DOCS. *Measuring Maintainability*. Disponível em: https://docs.sonarqube.org

Conclusão

A jornada pelos dez capítulos deste manual demonstrou que a qualidade de software não é uma etapa isolada, mas sim um valor contínuo que deve estar presente em todas as fases do desenvolvimento. A abordagem integrada — envolvendo processos, pessoas, ferramentas, segurança e feedback — é o que garante um produto confiável, sustentável e com alto valor percebido pelo usuário.

Desde os **fundamentos filosóficos e modelos normativos**, até os **testes avançados**, **fatores humanos**, **DevOps**, **indicadores de desempenho** e **conformidade legal**, fica evidente que a qualidade depende de **planejamento, disciplina, automação, colaboração entre áreas e responsabilidade ética**.

A entrega de software com qualidade envolve:

- Especificações bem definidas e validadas
- Testes contínuos, inteligentes e automatizados
- Equipes capacitadas, com comunicação eficaz
- Processos auditáveis, rastreáveis e em melhoria constante
- Respeito à legislação, à segurança da informação e à acessibilidade
- Respostas rápidas ao feedback do usuário e evolução contínua do produto

Ao adotar práticas e ferramentas modernas como BDD, DevSecOps, dashboards de qualidade, pipelines de CI/CD e testes de acessibilidade, as equipes se aproximam de um modelo de **qualidade real, mensurável e percebida**.

Mais do que entregar código funcional, o desafio é criar **soluções digitais confiáveis, éticas, úteis e sustentáveis**. E esse desafio exige compromisso coletivo e visão estratégica, como apresentado ao longo deste manual.