# Em busca da Qualidade de Software

Este artigo aborda alguns dos principais fatores que influenciam a qualidade de software. Ele faz uma compilação de pontos importantes que nos remetem a refletir sobre como incrementar e assegurar que a qualidade seja a principal engrenagem.



Artigo do tipo **Teórico**

Recursos especiais neste artigo:

Conteúdo sobre boas práticas.

**Em busca da Qualidade de Software**

Este artigo aborda alguns dos principais fatores que influenciam a qualidade de software. Ele faz uma compilação de pontos importantes que nos remetem a refletir sobre como incrementar e assegurar que a qualidade seja a principal engrenagem que conduz o desenvolvimento de software.

**Em que situação o tema útil**

Este tema é útil para todos os profissionais direta ou indiretamente envolvidos no desenvolvimento de software e que querem uma visão sobre pontos importantes relativos à qualidade.

Software tem um papel extremamente importante na sociedade contemporânea. Se olharmos ao nosso redor, podemos citar incontáveis dispositivos que direta ou indiretamente afetam nossa vida e que dependem de software: carros, aparelhos eletrônicos, eletrodomésticos, etc. Nossa dependência por software desenfreadamente cresce de maneira sutil e imperceptível. Dependemos de software para realizar nosso trabalho, para o nosso lazer, para realizar compras, para nos manter informados, e a lista não para.

Tomemos como exemplo para essa crescente dependência os dispositivos móveis. Vivenciamos sua diária proliferação, cada vez mais acessíveis, sustentados pela necessidade de constantemente estarmos conectados. Ele nos fornecem a capacidade de gerenciarmos a nossa vida, e de nos divertirmos, na palma de nossa mão. E software para isso é o que não falta.

Em 2012, Google Play e Apple Store registraram aproximadamente 25 bilhõe de downloads de aplicativos cada uma.

A quantidade de downloads realizados impressiona. Mas o que mais chama a atenção é a quantidade de aplicativos que cada um dos "mercados de aplicações" das duas gigantes possui. Números de janeiro de 2013 apontam que o Google Play conta com mais de 800.000 aplicativos, já a Apple Store oferece mais de 775.000 aplicativos para iPhone, iPad e iPod.

Este mar de aplicativos é suportado pelo crescimento constante do número de desenvolvedores, aliados ao fácil acesso a ferramentas poderosas e frameworks bem escritos que facilitam muito a criação de aplicações. Muitos destes são capazes de gerar conteúdo automaticamente, visando diminuir o *time to market* e aumentar a produtividade de quem desenvolve.

Quando olhamos para estes fatos sob o aspecto da evolução dos métodos e práticas de desenvolvimento de software, ficamos felizes em ver o quanto a engenharia de software tem evoluído de forma rápida. Porém, há um ponto muito importante que devemos observar e sempre levar em consideração. Um ponto subjetivo e que vem à tona quando vemos a quantidade de software sendo desenvolvido: qualidade. Quantos desses milhares de aplicativos são realmente confiáveis? Os resultados que eles nos apresentam e com os quais tomamos decisões refletem a realidade?

Qualidade não é obtida espontaneamente. Ela tem de ser construída. Assim, a qualidade do produto depende fortemente da qualidade de seu processo de desenvolvimento, das pessoas envolvidas – stakeholders, desenvolvedores, gestores – dos requisitos e de muitos outros fatores, não necessariamente tangíveis, mas que devem ser administrados constantemente para que se consiga atingir a qualidade esperada ao final do ciclo de desenvolvimento.

A definição de qualidade de software dada pela NBR 13596, em agosto de 1996, diz o seguinte: "A totalidade de características de um produto de software que lhe confere a capacidade de satisfazer necessidades explícitas implícitas." As necessidades explícitas são todas aquelas que surgem ainda no desenvolvimento, diagnosticadas pelos profissionais que trabalham no projeto, para atender as características solicitadas pelo cliente. As implícitas como a classificação já impõe, são aquelas que passam tanto pelos desenvolvedores quanto pelos usuários (ou clientes), enquanto utilizam o produto desenvolvido.



Com base nisso, este artigo fala sobre qualidade no desenvolvimento de software. Ele apresenta tópicos importantes que nos remetem a refletir, trazendo à tona o fato de que qualidade vai muito além do que podemos ver.

## Exemplos Históricos de Falta de Qualidade

Antes de caminharmos pelos fatores que influenciam a qualidade do desenvolvimento de software, vamos falar um pouco de história e visitar alguns fatos que marcaram o desenvolvimento de software e que trouxeram à tona suas falhas.



## 1985 – 1987: THERAC 25

Therac-25 foi uma máquina de radioterapia produzida pela empresa canadense Atomic Energy of Canada Limited (AECL). Entre os anos de 1985 e 1987, ela esteve envolvida em pelo menos seis acidentes onde pacientes receberam *overdoses* de radiação, aproximadamente cem vezes a dose real a ser aplicada. Durante as investigações, além dos erros explícitos de codificação encontrados, ficou claro que o software era praticamente impossível de ser testado de forma automatizada, não houve revisão e muito menos teste em ambiente de produção.

## 1990: Linhas da AT&T param de funcionar

Em 1990, uma linha de código implementada para acelerar chamadas de telefone gerou um efeito cascata que fez com que a AT&T – American Telephone & Telegraph (multinacional norte americana de telecomunicações) ficasse inativa por nove horas. Isto causou a perda de 75 milhões de ligações, além de perdas em reservas de passagens aéreas.

## 1993: Erro em divisão de ponto flutuante do Pentium da Intel

Uma falha na matéria prima dos processadores fez com que o processador Pentium da Intel cometesse erros quando executava a divisão de números de ponto flutuante. Por exemplo, dividindo 4195835.0 por 3145727.0, o resultado apresentado era 1.33374 ao invés de 1.33382. Um erro de 0.006%. Embora este *bug* atingisse um pequeno número de pessoas, ele se tornou um pesadelo. Com uma estimativa de 3 a 5 milhões de processadores defeituosos em circulação, primeiramente a Intel ofereceu a troca dos processadores de quem provasse precisar de alta precisão. Porém a companhia acabou substituindo os chips de qualquer cliente que reclamasse



## 1995/1996: The Ping of Death

Falta de testes e verificações relativas ao código de remontagem da fragmentação do protocolo IP tornou possível que vários sistemas operacionais falhassem devido ao envio de um pacote "ping" mal formado, vindo de qualquer lugar na internet. O mais notável dos sistemas operacionais afetados foi o Windows, que apresentava a famosa "tela azul" quando recebia este tipo de pacote.

## 1996: Foguete Ariane

Em 4 de junho de 1996, menos de um minuto após o lançamento, o foguete francês Ariane 501 se autodestruiu. Ele era um foguete não tripulado que transportava satélites que iriam auxiliar no estudo das interações do campo magnético da Terra com os raios solares. A explosão ocorreu durante a execução de uma conversão de dados de um número de 64 bits em ponto flutuante para um inteiro de 16 bits com sinal. O valor do número em ponto flutuante era maior do que poderia ser representado pelo inteiro de 16 bits com sinal. O resultado foi um operando inválido. A instrução de conversão de dados (em código ADA) não estava protegida contra erros de operando.

## 1999: Bug do Milênio

Para economizar espaço de armazenamento, em uma época em que cada byte economizado significava economia de dinheiro, softwares (hoje legados, mas não se espante se descobrir que muitos estão de pé por aí) utilizavam os dois últimos dígitos de um ano para representá-lo, como 99 para 1999. Como todas as datas eram representadas por somente dois dígitos, os programas assumiam o 19 na frente para formar o ano completo. Assim, quando o calendário mudasse de 1999 para 2000 o computador iria entender que estava no ano de 19 + 00, ou seja, 1900. Caso as datas realmente voltasse para 1900, clientes de bancos veriam suas aplicações dando juros negativos credores passariam a serem devedores, e boletos de cobrança para o próximo mês seriam emitidos com 100 anos de atraso. Surpreendentemente ocorreram poucas falhas decorrentes do Bug do milênio, que se revelou qua inofensivo apesar de ter gerado uma onda de pânico coletivo, especialmente nos países nos quais os computadores estavam mais popularizados.



## 2010: Problema na escala de trabalho dos tripulantes da Gol

Após um erro na atualização do software responsável pelo planejamento da escala dos tripulantes, foram gerados dados incorretos que culminaram no planejamento inadequado da malha aérea e da jornada de trabalho dos tripulantes. Consequentemente, houve atrasados e cancelamentos que prejudicaram diretamente os passageiros.

Os fatos supracitados demonstram o quão problemático pode ser não focar em qualidade durante todas as etapas do desenvolvimento de um software. Seja delineando um requisito, escrevendo código, testando ou colocando um software em produção, deve-se sempre colocar a constante qualidade em destaque, para que ela transpareça em todas as etapas de forma consistente.

## Subjetividade da Qualidade

Subjetividade é o que varia de acordo com o julgamento, os sentimentos, os hábitos de cada um. Qualidade é subjetiva. Cada pessoa, ou setor, tem a sua própria definição e seus padrões estabelecidos.

Considere como exemplo alguém que adquire um produto simples, como uma camiseta. Ninguém a compra pensando nas propriedades mecânicas do tecido da qual ela fora feita. Não é o fato de a camisa suportar 10 kg de tração que fará o consumidor levá-la para casa. Fatores mais sutis e algumas vezes intangíveis é que conduzirão a decisão final.

A qualidade de um produto tem um objetivo claro e bem definido: satisfazer superar as expectativas do cliente. E isto implica em abordar aspectos de u domínio nem sempre claro e bem definido, que pode ser instável e mutável, mas cujas características devem ser claramente delineadas e profundament exploradas a fim de se obter o melhor entendimento possível, provendo uma solução que se adéque aos requisitos de qualidade que se espera.



## Janelas quebradas

No livro "O Programador Pragmático", escrito por Andrew Hunt e David Thomas, os autores fazem uma correlação simples, mas muito feliz entre a Teoria da Janela Quebrada de James Q. Wilson e George Kelling e a qualidade no desenvolvimento de software.

Embora o desenvolvimento de software esteja imune a quase todas as leis da física, a entropia o afeta. Entropia é definida como um termo da física que se refere ao nível de desordem de um sistema. No desenvolvimento de software, quando a desordem aumenta, requisitos começam a serem deixados de lado, prazos estouram, testes não são adequadamente escritos (algumas vezes nem mesmo são feitos), não se sabe o porquê se está implementando determinada funcionalidade, etc. Muitos fatores podem contribuir para que isto ocorra. Não importa o tamanho da equipe, uma hora o projeto passará por algumas tempestades durante seu ciclo de vida. No entanto, isso não é regra. Há projetos que conseguem superar suas adversidades e combatem com sucesso a tendência à desordem, se saindo muito bem. Mas o que causa esta diferença? Vamos dar uma olhada em um fato relativo à segurança para ilustrar a justificativa.

Em algumas cidades, alguns prédios são imponentes obras de arte, muito bem cuidados, enquanto outros são estruturas deterioradas. Por quê? Pesquisadores da área criminal descobriram um fascinante mecanismo acionador, um mecanismo que torna muito rapidamente um prédio limpo, intacto e habitado, em uma construção quebrada e abandonada: uma simple janela quebrada.

Uma janela quebrada, deixada sem reparos por qualquer período de tempo, faz brotar nos moradores uma sensação de abandono – uma sensação de qu os responsáveis não se preocupam com o prédio. Portanto, é certo que outr janela será quebrada. As pessoas começam a acumular lixo na área externa Surgem grafites. Danos estruturais graves começam a aparecer. Em um período de tempo curto, o prédio fica danificado demais para o proprietário querer consertá-lo e a sensação de abandono se torna realidade.

A "Teoria da Janela Quebrada" inspirou os departamentos de polícia de Nova Iorque e outras cidades a atacar os problemas menores para evitar maiores. Ficar atento a janelas quebradas, grafites e outras pequenas infrações reduziu o alto índice criminal.

Portanto, não deixe "janelas quebradas" (projetos insatisfatórios, decisões erradas ou código inadequado) sem reparos. Conserte cada uma assim que for descoberta. Tome uma medida para evitar um dano maior e mostre que você está com a situação sob controle. A negligência acelera mais a deterioração do que qualquer outro fator. Há uma história de uma pessoa bastante rica, que ajudará a ilustrar esse texto. Sua casa era muito bonita, cheia de relíquias de grande valor e objetos de arte. Um dia, um tapete pendurado próximo à lareira da sala de estar pegou fogo. O corpo de bombeiros apressou-se para salvar a casa. Mas antes de arrastarem suas mangueiras porta adentro, eles pararam – com o fogo enfurecido – para desenrolar uma esteira entre a porta da frente e o ponto de origem do fogo. Eles não queriam estragar o carpete.

Este é um caso bastante extremo, mas é assim que deve ser com software. Uma janela quebrada – um trecho de código mal projetado, uma decisão insatisfatória da gerência que desmotive a equipe – é o ponto de início do declínio.

Se você perceber que está trabalhando em um projeto com algumas janelas quebradas, não vai ser difícil começar a pensar da forma padrão: "Todo o resto deste código está perdido. Vou me resignar e segui-lo". Não importa s o projeto vinha se saindo bem até esse ponto. Na experiência original que levou à "Teoria da Janela Quebrada", um carro abandonado permaneceu um semana intocado. Mas quando apenas uma janela foi quebrada, o carro foi destruído em pouco tempo.



Da mesma forma, se você perceber que está em uma equipe ou em um projeto no qual o código se encontra em perfeito estado – escrito apropriadamente, bem projetado, com testes unitários e comentários que deixam claro o que ele faz – provavelmente vai tomar um cuidado especial para não danificá-lo, assim como os bombeiros. Mesmo se tiver um incêndio ocorrendo (prazo, data de lançamento, demonstração para clientes), você não vai querer ser o primeiro a causar danos.

## Integridade conceitual

Um dos ensaios – um texto literário breve, situado entre o poético e o didático, expondo ideias, críticas e reflexões éticas e filosóficas a respeito de certo tema – presente no livro "O Mítico Homem-mês", de Frederick Brooks, tem como tema a integridade conceitual. Em seu texto, Brooks cita como exemplo a construção da Catedral de Reims (Catedral de Notre Dame), cuja integridade arquitetural só foi possível alcançar graças a sucessivas gerações de construtores que sacrificaram algumas de suas ideias para que o projeto final representasse o que havia sido inicialmente planejado. Estes construtores mantiveram-se fiéis aos requisitos iniciais, tomando-os como guia para conduzi-los durante o progresso do projeto, mantendo assim sua integridade conceitual. Mas o que esta história tem a ver com desenvolvimento de software? Como a integridade conceitual se relaciona com a qualidade de software? É simples. Alguns sistemas sofrem de uma acentuada falta de unidade conceitual – durante o ciclo de vida de desenvolvimento ocorrem muitas mudanças, repriorizações, rotatividade dos membros da equipe, requisitos dispersos e mal descritos (muitas vezes quem solicita não sabe o quer), etc., que desviam o caminho para algo fora do que inicialmente foi planejado. Isto acarreta em implementação que não condiz com o que o requisito "especificou", em cancelamentos, em aumento crescente de complexidade, entre outros fatores, que nos deixa sem rumo durante o desenvolvimento. Um sistema que omita certas funcionalidades que não foram bem definidas ou não estão bem claras e acrescente melhoria desnecessárias é melhor do que ter um projeto que contenha muitas ideias boas, mas independentes e descoordenadas.



## Qualidade dos requisitos

Muitos livros e tutoriais se referem à coleta de requisitos como uma fase inicial do projeto. "Coleta" implica que os requisitos já estão por aí – você só tem que encontrá-los, colocá-los em seu backlog e continuar feliz e sorridente pelo seu novo sprint. Infelizmente, não é assim que funciona. Requisitos emergem de necessidades explícitas. Eles são descrições dos serviços e funcionalidades que devem ser fornecidos pelo sistema, e também das limitações e restrições que influenciam diretamente o desenvolvimento. Isto deixa claro que a qualidade na definição de um requisito impacta diretamente na qualidade do desenvolvimento.

Não precisamos nos esforçar muito para encontrarmos algum exemplo de requisito que, inicialmente falho em sua descrição e compreensão, fez com que o produto final não fosse o esperado. Talvez a falha na integridade conceitual tenha contribuído para isso. Mesmo hoje, com a adoção massiva de métodos ágeis de desenvolvimento de software, que pregam a interação frequente dos stakeholders durante o ciclo de vida de desenvolvimento, alguns requisitos acabam não sendo bem compreendidos, nem por quem o propôs e muito menos por quem o irá implementar. Isso acaba levando para um ambiente instável, onde um dos lados tomará as rédeas e conduzirá o desenvolvimento baseado no que ele acredita ser a compreensão ideal do requisito. E este ponto é perigoso. Com a incerteza do que deve ser feito, não há como garantir qualidade, não há como mensurar o que deve ser feito, nem mesmo estabelecer quando tal requisito estará pronto. Entender muito bem o que deve ser feito é um dos mais importantes fatores quando nos referimos à qualidade. Devemos nos esforçar para que mitiguemos dúvidas e incertezas o mais previamente possível, focando em entregar uma solução que satisfaça as necessidades de quem a solicitou, no menor tempo possível.

## Qualidade de desenvolvimento



Quando falamos em qualidade no desenvolvimento de software estamos falando em qualidade do processo de desenvolvimento de software. Não basta termos os melhores profissionais, precisamos de um guia que nos conduza a entregar uma solução com a máxima qualidade e no menor tempo possível.

Processo é o alicerce da engenharia de software. Ele é complexo, e como todos os processos intelectuais e criativos, dependem de julgamento humano. A existência de um processo não garante que o software será entregue no prazo, de que ele irá satisfazer as necessidades do cliente, ou exibirá os atributos arquiteturais que manterão as características de qualidade ao longo do tempo. Ele deve ser acoplado a uma sólida prática de engenharia de software e deve ser avaliado para garantir que satisfaça a um conjunto de critérios que demonstrem ser essenciais para o desenvolvimento.

Processos são úteis porque imprimem consistência e estrutura a um conjunto de atividades. Estas características são importantes quando se sabe como fazer algo bem e se quer garantir que outras pessoas o façam da mesma maneira. E este é o ponto no qual um processo mostra sua força e impacta diretamente a qualidade. Quando se tem acesso a documentação de boas práticas e observações baseadas em experiências passadas, as famosas lições aprendidas, cria-se um arcabouço no qual novos projetos, novos softwares, se basearão, desenvolvendo-se e evoluindo de forma consistente.

## Fatores humanos de qualidade

Pessoas são o mais importante ativo de uma organização (seja ela qual for). Elas representam seu capital intelectual. A forma como trabalham, seja individualmente ou em equipe, impacta diretamente os resultados obtidos. Além disso, a forma como são gerenciadas, motivadas e remuneradas são alguns dos fatores que diretamente exercem influência e podem comprometer a qualidade de um projeto – para o bem ou para o mau.

Não é raro ouvirmos pelos corredores que alguém está insatisfeito com seu trabalho. Que alguma atitude a desmotivou, que a falta de reconhecimento a faz se sentir impotente e incompetente. De maneira involuntária, esse sentimento tem o poder de se espalhar rapidamente. E então se vê pessoas quietas, ansiosas, trabalhando sem propósito, fazendo uma atividade que antes era prazerosa e enriquecedora, e agora cansativa e monótona.



Se uma organização é capaz de manter a moral de um time alta, não há dúvidas de que este é um dos melhores benefícios que ela tem a oferecer. Pessoas motivadas produzem mais e melhor. Alguém dúvida disso?

## Conclusão

Do surgimento de um requisito até a entrega de software em ambiente de produção, deve-se ser capaz de assegurar a qualidade do que é desenvolvido. Manter a integridade conceitual, assegurando que o software evolua, mas não fuja de suas pretensões e objetivos iniciais, é a linha tênue que nos conduz a desenvolver software de forma oportuna e objetiva, dentro de prazos e custos definidos.

Software é desenvolvido por pessoas e por isto está suscetível a falhas. Exemplos não faltam, e a história está pronta para contá-los. Vimos que a falta de qualidade no passado tirou vidas e causou prejuízos financeiros. Apesar disto, prever novos erros é praticamente impossível. Novos erros surgirão e mais uma vez a história se encarregará de contá-los. Assim, devemos ficar atentos e nos disciplinarmos para sermos críticos quanto ao que e de que forma desenvolvemos, assegurando que em todo o ciclo de vida o foco em qualidade esteja em primeiro plano. Devemos evitar que as janelas quebradas prevaleçam em nossos projetos e os deteriorem. A qualidade não surge do dia para a noite, deve ser constantemente construída. Ela é parte de um processo ativo e contínuo ao qual temos que nos envolver, visando desenvolver softwares cada vez melhores.

**Referências**

**Brooks Jr., Frederick P., "No Silver Bullet – Essence and Accidents of Software Engineering", IEEE Computer, pp. 10-19, April 1987.**

**Juran JM, Gryna Fm. Juran's Quality Control Handbook. Mcgraw-Hil l, 1988.**

**Andrew Hunt, David Thomas. O Programador Pragmático: de apren diz a mestre. Primeira Edição. Bookman, 2010.**

**Brooks Jr., Frederick P. O Mítico Homem-mês: ensaios sobre engen haria de software. Rio de Janeiro, Elsevier 2009.**

**Bartié, Alexandre. Garantia da qualidade de software: adquirindo maturidade organizacional. Rio de Janeiro, Elsevier 2002.**

**Brinkkemper, S. Method Engineering: Engineering of Information S ystems Development Methods and Tools, In: Information and Soft ware Technology, Volume 38, 4th edition, 1996.**

**Omachonu, Vicent K. Principles of total quality, 3rd edition. CRC Pr ess LCC 2004.**

**Kenett, Ron; Baker, Emanuel (1999). Software Process Quality: Ma nagement and Control. CRC Press, p. 130 ff. ISBN 9780824717339.**

**Pall, Gabriel A. Quality Process Management. Englewood Cliffs, N. J.: Prentice Hall, 1987.**

**Links**

**Software Quality Engineering**
*http://www.sqe.com/*

**The Global Voice of Quality**
*http://asq.org/pub/sqp/*

**Consortium for IT Software Quality**
*http://it-cisq.org/*

**American Society for Quality Control, ASQC**
*http://www.asq.org*.

por Rafael Peria
Expert em Java e programação Web