Qualidade de Software: Teoria e Prática, Orgs. Rocha, Maldonado, Weber, Prentice-Hall, São Paulo, 2001
Capítulo 17.

# Gerenciando a Qualidade de Software com Base em Requistos

©Julio Cesar Sampaio do Prado Leite
http://www.inf.puc-rio.br/~julio

Obter qualidade nos processos e produtos de engenharia de software não é uma tarefa trivial. São vários os fatores que dificultam atingir os objetivos de qualidade. No entanto, nada mais decepcionante do que produzir software que não satisfaça as necessidades dos clientes. Grande volume de recursos são gastos, mas, em muitos casos, ocorre uma grande frustação por parte dos clientes, face a forma final apresentada pelo software encomendado. Nossa experiência indica que muitos desses problemas são derivados da falta de atenção para a tarefa de definir e acompanhar a evolução dos requisitos durante o processo de construção de software. Reportaremos resultados iniciais sobre o conceito de *baseline* de requisitos que toma por base as necessidades dos clientes, serve de referência para o desenvolvimento de software e está em constante evolução. Acreditamos que ao atacarmos a definição e gerência de requisitos estaremos colaborando de maneira fundamental para a qualidade geral do software.

## Introdução

Antes de mais nada, precisamos entender que ao falar-se de software e de sua engenharia estamos falando ao mesmo tempo de produtos e de processos. De nada adianta centrarmos nossa atenção só no produto ou só no processo. É necessário que as duas visões caminhem juntas. Portanto, para lidar com qualidade é necessário termos claro que o processo de produção deve ter qualidade e que o produto deve ter qualidade.

Durante muito tempo a engenharia de software centrou sua atenção na qualidade do produto. Nessa visão orientada a produto destacamos três aspectos que, acreditamos, dominaram a pesquisa e a prática da construção de software.

- Uma ênfase na qualidade das representações, isto é nas linguagens artificiais.
- A crença de que a qualidade do produto é função principalmente de teste do produto final.
- Que o processo de produção era centrado em fases caracterizadas por produtos bem definidos.

A qualidade das representações, sejam elas executáveis ou não é um importante aspecto da engenharia de software. Linguagens de programação mais robustas, de maior confiabilidade e de maior nível de abstração são, ainda hoje, um tema de discussão. Muitas vezes essas discussões saem do nível técnico e passam a ser discussões mais voltadas para gostos pessoais. Vale lembrar que na década de 80 a grande discussão era sobre ADA e que no final dos anos noventa passou a ser sobre JAVA, apenas para citar duas linguagens que geraram e ainda hoje geram muito debate. Sob o ponto de vista de representações mais abstratas, e a princípio, não executáveis, o ponto central são linguagens de especificação. Por uma lado temos a necessidade de linguagens formais e abstratas e por outro temos a necessidade de linguagens de fácil comunicação e também abstratas.

O papel do teste, fundamental na produção de qualquer produto, foi e, ainda hoje, é visto, principalmente, como teste de programas, isto é teste de descrições em linguagens de programação. Hoje dispõe-se de técnicas de teste, de ferramentas de teste e de facilidades embutidas nas linguagens de programação que possibilitam ao engenheiro de software o uso de um valioso arsenal para combater a presença de defeitos no produto final. No entanto, sabe-se

que esse processo é extremamente dispendioso e muitas vezes envolve uma grande número de recursos humanos para garantir um nível mínimo de qualidade.

A visão inicial do processo de produção, ou ciclo de vida do software, era centranda em produtos bem definidos e em regras de verificação entre produtos. Esses produtos, com características bem definidas eram vistos como passos ou fases num processo, que apesar da possível retroalimentação, fundamentava-se na visão de sequencialidade. Mesmo em propostas não sequênciais, como a prototipação, ainda assim a atenção principal é dirigida às representações dos produtos.

Hoje tem-se uma visão muito mais abrangente. Primeiro, sabe-se que o processo de produção é fundamental para obtenção de produtos de qualidade. Segundo, tem-se uma visão mais equilibrada dos aspectos essenciais e acidentais da engenharia de software. Terceiro, a sociedade passa a entender melhor os custos relacionados com a evolução do software.

Entendemos que a visão da construção de software como um processo onde normas, procedimentos e gerência devem estar bem definidos para garantir a qualidade dos produtos é diferente da visão do processo de produção centrando em fases/produtos. Nesta visão, o papel da gerência passa a ser visto como um aspecto técnico e não como um aspecto exterior ao conhecimento da engenharia de software. Procura-se centrar atenção na aquisição de dados sobre o processo e a transformação desses dados em conhecimento sobre o processo de produção [Humphrey 95]. Entender que o processo de produção de software é composto por diferentes subsistemas [Freeman 87], Figura 1, é um importante passo para entender o processo de produção de uma maneira mais ampla.

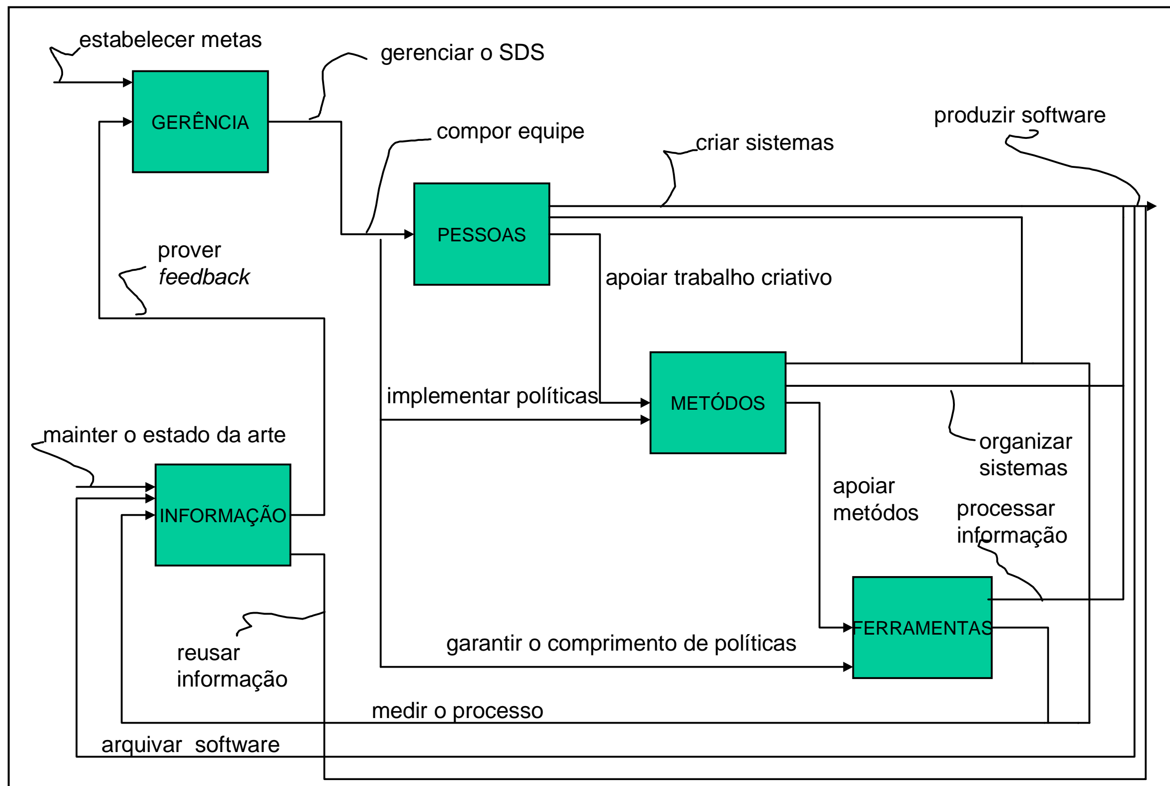


Figura 1 – Um modelo SADT [Ross 77] do Sistema de Desenvolvimento de Software

A importância dos aspectos essenciais foi salientada por Fred Brooks [Brooks 87] e a citação abaixo procura justamente passar essa idéia.

> *"The hardest part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interface to people, machines and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."*

É importante salientar que esse tipo de problema, considerado por Brooks, como o problema essencial, é aquele para o qual não existem soluções mágicas (*silver bullet*) e que propostas recentes, como por exemplo, orientação a objetos, lidam apenas com aspectos não essenciais ou seja, acidentais. No entanto, David Harel [Harel 92] acredita que a visão de Brooks seria pessimista, em função de alguns avanços importantes na área de modelagem e análise de modelos. O argumento de Harel é que se 40 anos antes trabalhassemos com a hipótese de que o principal era conceber um algorítmo, como teríamos alcançado progresso se não havia meio de expressar esse algorítmo? Portanto ele considera que os aspectos representacionais (acidentais) não devem ser desconsiderados, porque eles são decisivos em termos de produtividade, haja visto o progresso alcançado, como por exemplo: linguagens de especificação executáveis, tipos abstratos de dados, linguagens lógicas, editores gráficos, gerência de configuração, entre outros. Ou seja, os problemas de essência permanecem, mas temos hoje dispomos de ferramentas mais poderosas.

A recente crise do ano 2000 e as constantes notícias de sistemas de software com problemas [Levenson 93], [Breitman 99] ajudaram a que a sociedade passe a ver software como parte de suas vidas. Esse amadurecimento da sociedade passa a gerar uma maior demanda por qualidade. Também passa a ficar mais claro que a evolução do software, tendo em vista o caso do ano 2000, tem um custo. Se por um lado a sociedade passa cada vez mais a depender de software, exigir mais qualidade dos produtos de software e pressionar por mudanças, por outro fica mais claro que existe um custo associado e não completamente entendendido. Pelo fato de que a sociedade moderna passa a depender cada vez mais do software para sua própria evolução, maior será o desafio dos engenheiros de software para manter a qualidade desejada dentro dos recursos disponíveis.

## Qualidade

O produto software passa a ser, cada vez mais, um componente comum em uma série de outros produtos, desde carros, fornos de micro-ondas, elevadores, telefones, até sistemas de informação organizacionais. De um produto exige-se qualidade e preço. Portanto como produto, software tem que ter o nível de qualidade exigido e procurar ser desenvolvido no menor custo possível. Esta é exatamente a função da engenharia, procurar sistemas de melhor qualidade dentro de um custo compatível com essa qualidade, otimizando a redução de custos.

Produzir software de qualidade é uma meta básica da Engenharia de Software, que disponibiliza métodos, técnicas e ferramentas para este fim. Muito tem-se escrito sobre qualidade e seus vários adjetivos, no entanto, o fundamental é que o software seja confiável, isto é seja eficaz e siga os padrões exigidos pelo contexto onde irá atuar. Freeman [Freeman 87] faz uma distinção entre qualidade básica e qualidade extra. Em qualidade básica ele lista: funcionalidade, confiabilidade,

facilidade de uso, economia e segurança de uso. Em qualidade extra ele lista: flexibilidade, facilidade de reparo, adaptabilidade, facilidade de entendimento, boa documentação e facilidade de adicionar melhorias. Este tipo de classificação nos fornece uma idéia do que estamos falando, mas é importante ressaltar que dar prioridade à essas qualidades depende de cada caso e do custo de cada uma dessas qualidades. No entanto, cada vez mais a sociedade pressiona o setor de software para que a característica qualidade seja preponderante. Normas internacionais como a ISO e iniciativas como a da SEI (*Software Engineering Institute*) [Fiorini 98] são exemplos disso.

É importante ressaltar que a escolha de quais métodos serão utilizados no processo de produção é uma escolha gerencial ligada a qualidade, porque a qualidade resultante será função dos métodos de produção escolhidos e seguidos. É também importante não esquecer que a produção de software é um processo que envolve, como parte fundamental, seres humanos. As tecnologias de produção de software, nas quais estão incluídas as tecnologias de gerência, tem por objetivo automatizar ao máximo a produção de software, mas ainda são fundamentalmente dependentes da qualidade das equipes.

A qualidade deve estar presente não só nos produtos produzidos, como também nos processos utilizados para gerar esses produtos. Neste caso os processos de qualidade, ou de auditoria, são processos que se aplicam tanto aos produtos, como aos próprios processos (uma meta aplicação dos processos de qualidade). Assegurar a qualidade dos produtos e dos processos é responsabilidade do subsistema GERÊNCIA (Figura 1), que necessita dispor de MÉTODOS (Figura 1) e FERRAMENTAS ( Figura 1) compatíveis com a qualidade desejada. Atuar eficazmente na função de garantir a qualidade exige conhecimento técnico sobre os MÉTODOS e FERRAMENTAS, além do conhecimento de gerência para lidar com PESSOAL (Figura 1).

Como base para qualquer processo de gerência, deve existir um sistema de retroalimentação que forneça ao subsistema GERÊNCIA as informações sobre o funcionamento do processo para que eventuais problemas possam vir a ser corrigidos. Também é necessário organizar as informações coletadas, métricas, para que se possa conhecer melhor os produtos produzidos, bem como os processos utilizados na sua produção. A responsabilidade da guarda e distribuição dessas informações é do subsistema INFORMAÇÃO (Figura 1).

Cabe ressaltar que sem métricas não é possível comunicar-se de uma maneira precisa. Portanto, é fundamental para garantir qualidade, que os processos sejam baseados em métricas. Cabe aqui uma citação de Lord Kelvin [Thomson]:

> " *When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you can not measure it, and when you can not express it in numbers, your knowledge is of a meagre and unsatisfactory kind*."

Se por um lado métricas são essenciais para a garantir confiabilidade e transparência aos processos, por outro lado a engenharia de software ainda encontra dificuldade na definição e uso de métricas.

**Requisitos**

Engenharia de Requisitos, uma sub-área da Engenharia de Software, tem por objetivo tratar o processo de definição dos requisitos de software. Para isso estabelece um processo no qual o que

deve ser feito é elicitado, modelado e analisado. Este processo deve lidar com diferentes pontos de vista, e usar uma combinação de métodos, ferramentas e pessoal. O produto desse processo é um modelo, do qual um documento chamado requisitos é produzido. Esse processo é perene e acontece num contexto previamente definido a que chamamos de Universo de Informações.

Universo de Informações é o contexto no qual o software deverá ser desenvolvido e operado. O UdI inclui todas as fontes de informação e todas as pessoas relacionadas ao software. Essas pessoas são também conhecidas como os atores desse universo. O UdI é a realidade circunstanciada pelo conjunto de objetivos definidos pelos que demandam o software.

Entendemos requisitos de software como sentenças que expressam as necessidades dos clientes e que condicionam a qualidade do software. Em função disso classificamos requisitos como requisitos funcionais e requisitos não funcionais. O primeiro está diretamente ligado a funcionalidade do software, enquanto o segundo reflete os requisitos que expressam restrições que o software deve atender ou qualidade específicas que o software deve ter. As três sentenças abaixo exemplificam essa distinção:

- O sistema deve prover um formulário de entrada para o a entrada dos resultados dos testes clínicos de um paciente. (RF)
- Dependendo do resultado do teste, somente o Supervisor pode efetuar a entrada do resultado do teste de um paciente. (RNF de confidencialidade).
- O sistema deve emitir um recibo para o cliente, como o tempo máximo de 8 segundos após a transação. (RF “,” RNF de performace).

Portanto para nós, a questão de qualidade é tratada no processo de definição dos requisitos através da definição clara dos critérios de qualidade (não funcionais) que o software deverá atender. Essa abordagem [Cysneiros 99] [Chung 00] vem responder a um problema frequente no processo de produção de software, ou seja o não tratamento no ínicio do processo dos aspectos relacionados as características de qualidade do produto. Ou seja, aspectos de qualidade são tratados no próprio processo de definição do software. Essa estratégia integra de uma maneira coerente aspectos funcionais e aspectos não funcionais, podendo tratar no nível de abstração mais alto de opções de solução que ficavam escondidas muitas vezes até a etapa de programação.

Os requisitos não funcionais podem ser divididos [Macedo 99] em: requisitos de qualidade geral, de abrangência e de operação. Além disso pode-se classificar os requisitos não funcionais como requisitos não funcionais primários e requisitos não funcionais específicos. Por exemplo; precisão é um requisito não funcional de qualidade geral e portanto um requisito não funcional primário; precisão de valor já é um requisito não funcional específico, porque mais detalhado. Outras taxonomias de requisitos não funcionais existem [Chung 00], como por exemplo a utilizada por Sommerville [Sommerville 98], onde os requisitos não funcionais são divididos em: requisitos do processo, requisitos do produto e requisitos externos.

Abordar qualidade como requisitos não funcionais integra aspectos de qualidade na definição do próprio software. Com isso, a tarefa de gerenciar requisitos passa a cuidar não só de aspectos de funcionalidade, como também de aspectos de qualidade. Portanto, gerenciar requisitos torna-se um processo de gerência de qualidade, na medida em que procura tratar da alocação dos

requisitos (*requirements allocation*), de maneira a garantir , não só, que os requisitos estão sendo seguidos, como também são rastreáveis.

**Evolução**

O processo de construção de software será cada dia mais baseado no conceito de evolução [Leite 97b], ou seja estaremos sempre modificando algum software já existente. Na verdade, o próprio processo de definição de requisitos gera um feedback que acaba modificando os próprios requisitos. Portanto acreditamos ser falsa a idéia de congelar requisitos defendida por alguns autores. Concordamos com Manny Lehman [Lehman 94] que caracteriza o processo de software como: "*a complex, multi loop multilevel feedback system*".

No contexto evolutivo é importante saber da **impossibilidade da completeza** de um conjunto de requisitos. Definimos este conceito da seguinte maneira:

> *"O processo de definir requisitos é inerentemente incompleto, tendo em vista a grande complexidade do mundo real. É óbvio, no entanto, que sempre estaremos procurando ter requisitos os mais completos possíveis. A impossibilidade da completeza está relacionada a afirmação de Brooks [Brooks 87] sobre a bala de prata e a característica de homeostasia dos sistemas, ou seja a característica evolutiva do software, que procura se adaptar ao seu macrosistema*".

Acreditamos que para lidar com evolução é fundamental utilizar o conceito de *baseline* oriundo da área de configuração de sistemas. Um *baseline* é uma espécie de referencial que utilizamos num processo de mudança, uma âncora. Portanto, acreditamos que todo o processo de construção/evolução deve estar ancorado num *baseline*, para que se possa gerir as mudanças de uma maneira organizada. Qual o *baseline* mais indicado para tal? Cremos que esse *baseline* deve ser justamente os requisitos, já que são eles que definem o que deve ser feito e com que qualidade.

No entanto, como sabemos que os requisitos evoluem também, passamos a ter uma *baseline* que também evolue. A definição da *baseline* de requisitos [Leite 95] [Leite 97] explica que essa evolução se dá em dois eixos, um no que diz respeito a pontos de referência do modelo de processo de software e outro eixo no que diz respeito a progressão do processo de software no que se refere a mudança de nível de abstração. Ou seja o *baseline* de requisitos muda tanto num mesmo nível de abstração como entre níveis de abstração.

A característica fundamental da *baseline* de requisitos diz respeito a sua formação e a sua representação. A *baseline* se forma procurando moldar-se ao Universo de Informações e sua representação utiliza linguagem natural através de um léxico e de um conjunto de cenários. Essas representações facilitam a comunicação com os atores do Universo de Informações e portanto facilitam o processo de gerência da qualidade. Essas representações evoluem, como visto acima, e portanto teremos cenários que tanto descrevem situações no Universo de Informações, como cenários que descrevem situações que ocorrem no próprio código do sistema [Breitmam 98].

É claro que a complexidade de controlar essa *baseline* é não trivial, no entanto é uma maneira integrada de tentarmos garantir que a alocação dos requisitos, tanto funcionais como não funcionais seja registrada e rastreável. Assim, é possível efetivamente gerenciar os requisitos, principalmente num contexto volátil.

## Conclusão

Neste capítulo esboçamos uma série de conceitos relacionados a requisitos, qualidade e a engenharia de software e com base nesses conceitos propomos uma estrutura que possibilita uma gerência de qualidade tanto no nível funcional como no nível não funcional e que toma por princípio básico a constante evolução do software.

Resultados iniciais já demonstram a vantagem, de lidar-se com requisitos não funcionais de forma explícita [Cysneiros 99], como também demonstram a flexibilidade do esquema representacional [Leite]. Acreditamos que a visão integrada de qualidade, requisitos e evolução é extremamente benéfica para um correto entendimento do problema que estamos lidando, bem como de sua inerente complexidade, face ao volume de informações constante da *baseline* de requisitos.

## Bibliografia

[Chung 00] Chung, L, Nixon, B.A., Yu, E.,, Mylopoulos, J. Non-Functional Requirements in Software Engineering, Kluwer Academic Publishers, 2000.

[Breitman 98] Breitman, K; Leite, J.C.S.P. - Scenario Evolution: observations from a case study - Proceedings of the International Conference on Requirements Engineering, IEEE Computer Society Press, pp. 214-221, 1998.

[Breitman 99] Breitman, K; Leite, J.C.S.P; Finkelstein, A. – The World´s a Stage: A Survey on Requirements Engineering using a Real-Life Case Study, Journal of the Brazilian Computer Society: Vol.6, N.1, Pags. 13-37 (1999).

[Brooks 87] Brooks, F. – Essence and Accidents to Software Engineering – IEEE Computer, vol.4 no. 3, 1987, pp.10-19.

[Cysneiros 99] – Cysneiros, L.M.; Leite, J.C.S.P. – Integrating non-functional requirements into data modeling , in Proceedings of the Fourth IEEE International Symposium on Requirements Engineering, Limerick, Ireland, 1999 (to appear) (http://www.ul.ie/~isre99/).

[Fiorini 98] Fiorini, S.T; Staa, A.v., Martins, R.B. Engenharia de Software com CMM, Rio de Janeiro, Editora Brasport, 1998.

[Freeman 87] Freeman, P. A., Software Perspectives: The System is the Message, Addison-Wesley, Reading, Massachusetts, 1987.
[Harel 92] Harel, D. Bitting the Silver Bullet, IEEE Computer, vol. 25, n.1, Jan. 1992, pp. 8-19.

[Humphrey 95] Humphrey, W. S., A Discipline for Software Engineering, SEI Series on Software Engineering, Addison-Wesley Pub Co, 1995.

[Lehman 94] Lehman M M**,** *Evolution in the Context of Software Technology, Encyclopaedia of Software Engineering***,** Wiley and Co., 1994, vol. 2, pp. 1202 - 1208

[Leite 95] Leite J.C.S.P., Oliveira, A.P.A., A Client Oriented Requirements Baseline, in Proceedings of the Second IEEE International Symposium on Requirements Engineering, York, UK, IEEE Computer Society Press, 1995 pp. 108-115.

Qualidade de Software: Teoria e Prática, Orgs. Rocha, Maldonado, Weber, Prentice-Hall, São Paulo, 2001 Capítulo 17.

[Leite 97] Leite, J.C.S.P.; Rossi, G.; Maiorana, V.; Balaguer, F.; Kaplan, G.; Hadad, G.; Oliveros, A. - Enhancing a Requirements Baseline with Scenarios. Requirements Engineering Journal, Vol. 2, N. 4, Pags. 184 -- 198, 1997.

[Leite 97b] Leite, J.C.S.P.; Software Evolution, The Requirements Engineering View, keynote address 26 Jaio, Proceedings SoST´97, JAIO, SADIO, Buenos Aires, 1997, pp. 21-23.

[Leite] Leite, J.C.S.P.; Evolução de Software, (http://www.inf.puc-rio.br/~julio)

[Levenson 93] Levenson, N. and Tuner, C.S., An Investigation of the Therac-25 Accidents, IEEE Computer, Vol. 24, N.7, Jul.93, pp. 18-41.

[Macedo 99] Macedo, N.M.; Leite, J.C.S.P Integrando Requisitos Não Funcionais aos Requisitos Baseados em Ações Concretas. Anais do Workshop Iberoamericano de Engenharia de Requisitos e Ambientes de Software, CYTED & Instituto Tecnológico de Costa Rita (TEC) pp. 1-9,(1999).

[Ross 77] Ross, D. Structured Analysis (SA): A Language for Communicating Ideas, IEEE Transactions on Software Engineering, vol. 3, no.1, Jan. 1977, pp. 16-34.

[Sommerville 98] Sommerville, I, Software Engineering, Fifth edition, Addison-Wesley, 1998.

[Thomson] http://www.groups.dcs.stand.ac.uk/~history/Mathematicians/Thomson.html