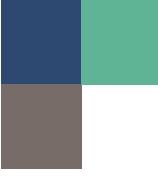




---

# GERENCIAMENTO ÁGIL DOS SISTEMAS



**Marco Ikuro Hisatomi**

# **GERENCIAMENTO ÁGIL DOS SISTEMAS**

1<sup>a</sup> edição

Londrina  
Editora e Distribuidora Educacional S.A.  
2020

**© 2020 por Editora e Distribuidora Educacional S.A.**

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Editora e Distribuidora Educacional S.A.

**Presidente**

Rodrigo Galindo

**Vice-Presidente de Pós-Graduação e Educação Continuada**

Paulo de Tarso Pires de Moraes

**Conselho Acadêmico**

Carlos Roberto Pagani Junior  
Camila Braga de Oliveira Higa  
Carolina Yaly  
Giani Vendramel de Oliveira  
Henrique Salustiano Silva  
Juliana Caramigo Gennarini  
Mariana Gerardi Mello  
Nirse Ruscheinsky Breternitz  
Priscila Pereira Silva  
Tayra Carolina Nascimento Aleixo

**Coordenador**

Henrique Salustiano Silva

**Revisor**

Valéria Cristina Gomes Leal

**Editorial**

Alessandra Cristina Fahl  
Beatriz Meloni Montefusco  
Gilvânia Honório dos Santos  
Mariana de Campos Barroso  
Paola Andressa Machado Leal

Dados Internacionais de Catalogação na Publicação (CIP)

---

Hisatomi, Marco Ikuro.

H673g      Gerenciamento ágil dos sistemas/ Marco Ikuro Hisatomi, -  
Londrina: Editora e Distribuidora Educacional S.A., 2020.  
44 p.

ISBN 978-65-87806-47-1

1. Gerenciamento 2. Ágil 3. Sistemas I. Título.

---

CDD 003

Raquel Torres – CRB 6/2786

2020

Editora e Distribuidora Educacional S.A.  
Avenida Paris, 675 – Parque Residencial João Piza  
CEP: 86041-100 — Londrina — PR  
e-mail: editora.educacional@kroton.com.br  
Homepage: <http://www.kroton.com.br/>

## SUMÁRIO

Controlando o escopo: as métricas e as estimativas	05
Gestão de riscos e de custos com COCOMO II	21
Gestão da Qualidade	37
Administração da manutenção do software	53

# **Controlando o escopo: as métricas e as estimativas**

Autoria: Marco Ikuro Hisatomi

Leitura crítica: Valéria Cristina Gomes Leal



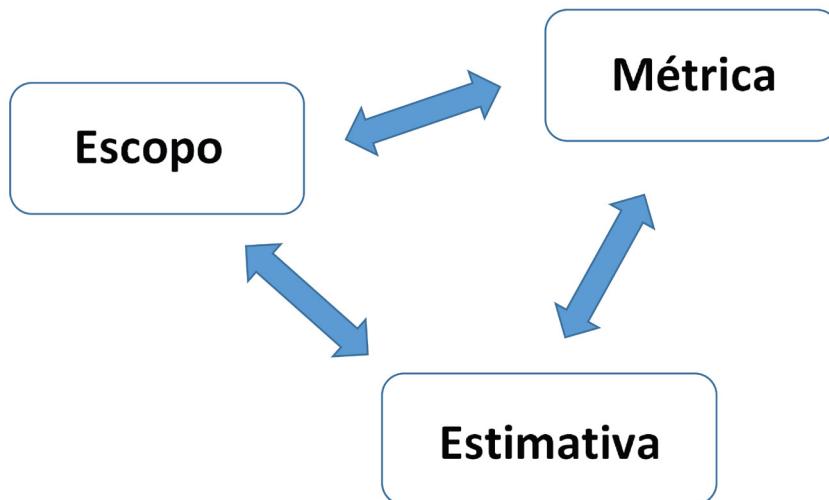
## **Objetivos**

- Saber como administrar o escopo do sistema.
- Saber como determinar as métricas para o desenvolvimento do sistema.
- Conhecer como determinar estimativas para o desenvolvimento.
- Perceber que as relações entre estimativa, métrica e escopo são interdependentes.

# 1. Introdução

Será que estamos preparados para gerenciar o escopo, visando entregar um produto conforme as necessidades reais do usuário dentro das condições ideais? Diante desse questionamento, vamos entender um dos princípios dos Métodos Ágeis, que é a interdependência que existe entre o escopo de um projeto de software a ser entregue e a estimativa de recursos, como mostra a Figura 1, considerando sempre métricas claras, pelas quais os membros da equipe percebem vantagens em medir e usar.

**Figura 1 – Interdependência entre escopo, métrica e estimativa**



Fonte: elaborada pelo autor.

Neste tema, apresentaremos como o gerenciamento ágil se diferencia de outras iniciativas, metodologias e técnicas de gerenciamento, em especial para o gerenciamento de escopo, baseado nas estimativas e métricas. A partir desse conhecimento, o estudante passará a ter habilidades de elaboração e entendimento das especificações de escopo, bem como do controle do escopo, das métricas e das estimativas ágeis. Assim, compreendendo as essências do gerenciamento de escopo embasado em Métodos Ágeis, será possível aumentar as chances de sucesso na construção de software, sobretudo quando se tratar de

funcionalidades e regras de negócio em constante mudanças ou em demanda permanente na era da informação.

## 2. Escopo em Métodos Ágeis

Vamos iniciar os estudos de escopo, estimativa e métrica resgatando dilemas que, habitualmente, vivenciamos no ambiente de desenvolvimento de software. Muitos argumentam: se tivesse mais tempo, poderíamos entregar mais funcionalidades. Outros ficam na dúvida: será que vamos conseguir entregar as funcionalidades do sistema no prazo previsto? E, ainda, há equipes que desconhecem seu próprio potencial de desenvolvimento de software.

Os Métodos Ágeis, como também é conhecido, antes mesmo da publicação do Manifesto Ágil, tinham como premissa um escopo não tão detalhado de todo o sistema a ser desenvolvido, nem mesmo uma documentação densa. Entendia Sommerville (2018, p. 58) que “o documento de requisitos do usuário é uma definição resumida contendo apenas as características mais importantes do sistema”.

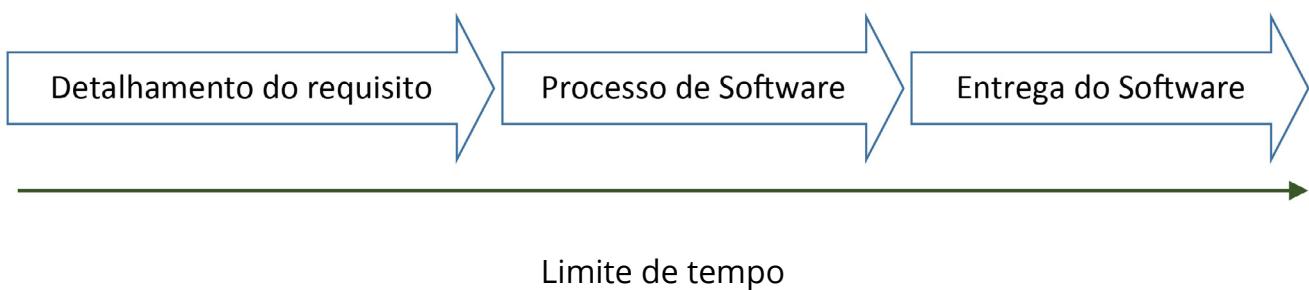
Dessa maneira, percebia-se que o escopo seria controlado, no detalhe, ao longo do desenvolvimento do projeto. Essa premissa é reforçada quando se pensa em quais métodos ágeis priorizam as funcionalidades atualizadas ao negócio do usuário. O detalhamento dos requisitos deve ser realizado o mais próximo possível da entrega do software ao usuário, em um prazo suficiente para implementar e disponibilizar.

Nesse quesito, entende-se que as entregas são previstas em períodos pré-estabelecidos e há um limite de tempo para a entrega do produto funcionando, o que agrega valor ao negócio do usuário. Esse limite de tempo é chamado de *Timebox* por alguns autores, e, para Massari (2018), corresponde a um intervalo de duração fixa, devendo qualquer tarefa

prevista nesse limite e não podendo essa duração ser estendida ou ultrapassada.

Portanto, para que o software (em uma determinada *release* ou versão) seja entregue com funcionalidades ou requisitos conforme as necessidades atualizadas do negócio do usuário, o detalhamento de tais requisitos deve ser ajustado com o usuário dentro desse limite de tempo. Aqui compreende-se que o processo de software ocorrerá naturalmente nas atividades de análise, construção e validação dentro do intervalo do *Timebox*.

**Figura 2 – Atividades do desenvolvimento dentro de um limite de tempo**



Fonte: elaborada pelo autor.

Em suma, para compreender bem como o Escopo será controlado, é preciso perceber as atividades demonstradas na Figura 2, sendo todas elas executadas dentro de um limite de tempo. Assim sendo, o controle do escopo fica sujeito a um prazo pré-estabelecido pelo método ágil. Em alguns exemplos praticados pelo SCRUM<sup>1</sup>, sugere-se um *Timebox*, chamado de *Sprint*, de duas a quatro semanas.

Reforçando a compreensão das entregas sucessivas, cada entrega do software será realizada no prazo estipulado de uma *Sprint*. É possível também ser assimilado por entregas incrementais, isto é, uma versão/*release* do software sofre uma ampliação com novas funcionalidades que são incrementadas ao final de cada *Sprint*. Consequentemente, abrange

<sup>1</sup> SCRUM é um *framework* para o desenvolvimento e manutenção de produtos complexos (SCRUM.ORG, [s.d.]).

a participação efetiva de usuários e demais *stakeholders*<sup>2</sup> e de toda a equipe do projeto, desde as especificações detalhadas dos requisitos até a validação de cada um deles.

Os métodos ágeis foram sendo aplicados pela engenharia de software ao longo dos anos, incorporando seus princípios de forma natural e complementar.

Os métodos ágeis se baseiam no desenvolvimento incremental; os incrementos são pequenos e, normalmente, novas versões do sistema são criadas e disponibilizadas para os clientes a cada duas ou três semanas, para que seja possível obter deles um feedback rápido nos requisitos que mudam. Além disso, esses métodos minimizam a documentação usando comunicação informal em vez de reuniões formais com documentos escritos. As abordagens ágeis de desenvolvimento de software consideram o projeto (design) e a implementação como as atividades centrais no processo de software. Elas incorporam outras tarefas a essas atividades, como a elicitação dos requisitos e os testes. ( SOMMERVILLE, 2018, p. 58)

## 2.1 Histórias do Usuário

Em Métodos Ágeis, cada funcionalidade a ser desenvolvida e incorporada no software passa a ser chamada de Histórias do Usuário. Ainda, em engenharia de software, essa história é tratada por requisito funcional de software. As histórias devem ser escritas pelo usuário e compreendidas por todos os envolvidos, de forma que ele próprio possa utilizá-las para validar a implementação, quando uma nova versão do software for entregue.

Se o usuário tem a responsabilidade de escrever as histórias, Pressman (2016, p. 73) trata como uma das atividades fundamentais nessa etapa do projeto o “ouvir” o usuário “para criar um conjunto de histórias que descreve o resultado, as características e as funcionalidades” que

<sup>2</sup> Stakeholders são partes interessadas, segundo PMBOK (PMI, [s.d.]): pessoas, grupos ou organizações que podem ser afetadas, direta ou indiretamente, pelo resultado de um projeto.

comporão o software futuro. Cada história poderá ser escrita em um cartão ou uma ficha em que o próprio usuário atribuirá um valor, denominado prioridade, ao que essa história representa para o seu negócio. Em muitos casos práticos, os cartões são escritos e colados em um quadro para a visualização de todos os envolvidos ou interessados, conforme ilustra a Figura 3.

**Figura 3 – Exemplo de mural com os cartões das histórias do usuário**



iStock – Akinbostanci/iStock.com.

O resultado da colaboração entre o cliente e a equipe ágil é uma descrição consistente e real das necessidades do usuário. Sommerville (2018) ressalta que o cliente do sistema discute cenários com os membros do time, para, assim, desenvolverem um “cartão de história” que reúna as necessidades do usuário.

Para ilustrar a importância de uma história de usuário, trazemos, no Quadro 1, o porquê de ela ser um insumo imprescindível no desenvolvimento baseado em Métodos Ágeis.

**Quadro 1 – Finalidade da história do usuário em cada fase**

<b>Fase do processo de software</b>	<b>Finalidade da História do Usuário</b>
Planejamento	Valor para o usuário final. Critério de aceitação. Tamanho da história para compor a lista da <i>Sprint</i> .
Construção e Design	Base para implementar em funcionalidades. Refatoração <sup>3</sup> no código-fonte.
Testes	Validação da funcionalidade pelos envolvidos no projeto.

Fonte: elaborado pelo autor.

Vamos ver agora como uma história pode ser escrita e percebida durante todas as fases do processo de software por meio de uma situação da realidade. Imaginemos que João é o usuário escritor da seguinte história: em um sistema de monitoramento de pacientes com problemas de pressão cardiovascular, é preciso permitir a manutenção de um paciente; adicionar um paciente, validando os dados pessoais; modificar os dados do paciente; e excluir o paciente do cadastro.

De posse dessa história, vejamos como fica a finalidade em cada uma das fases do processo de desenvolvimento de software:

<sup>3</sup> A refatoração é uma técnica disciplinada para reestruturar e otimizar um código-fonte existente, alterando sua estrutura interna sem alterar seu comportamento externo.

**Planejamento:** um cadastro fundamental para a finalidade do sistema.

Todos os dados do paciente devem ser validados na inclusão e na modificação.

**Construção e Design:** as funcionalidades de inclusão, modificação e exclusão devem ser implementadas com uma interface adequada e com mensagens objetivas, quando da validação dos dados do paciente. A refatoração do código-fonte deverá considerar todas as demais histórias relacionadas.

**Testes:** na entrega, o software deverá se comportar de acordo com a especificações escritas na história.

Agora que falamos sobre as premissas do gerenciamento de escopo, juntamente com o conceito de histórias do usuário, veremos como as estimativas serão definidas para cada uma dessas histórias.

### 3. Métrica em Métodos Ágeis

Inicialmente, Estimativa e Métrica são dois conceitos que se entrelaçam e se complementam, dependendo dos Processos e do Escopo. Enquanto a estimativa indica a previsão de esforços ou recursos a ser consumida na execução de uma determinada tarefa, atividade ou processo; a métrica define como devem ocorrer as medições: o que coletar, como tratar, o que avaliar e, sobretudo, como indicar a tomada de decisão.

A métrica sempre nos leva a um número (uma informação) que pode indicar se estamos no caminho correto ou não. Se conseguimos atingir uma meta em uma iteração, conquistando 100% das histórias de usuários entregues, é um excelente resultado. Contudo, em Métodos Ágeis, a medição deve continuar sendo um dos principais fatores para a melhoria contínua de processo e de habilidades da equipe.

Uma consideração a fazer sobre a métrica é que ela representa uma informação distinta para cada time de projeto de software. Se uma métrica faz sentido e auxilia na tomada de decisão para uma equipe, isso não significa que terá o mesmo valor para outra equipe. Ademais, pode ocorrer uma variação de eficácia de um projeto de software para outro. Essas considerações são relevantes principalmente por apresentarem características diferentes entre os projetos ou as equipes, como complexidade das histórias de usuários, participação do usuário final, tecnologia e ferramentas utilizadas no processo de software.

Portanto, para cada projeto, a métrica deve ter a validação de acordo com a sua realidade. Pressman (2016, p. 656) afirma que a “métrica deve ser validada empiricamente em uma ampla variedade de contextos antes de ser publicada ou usada para tomada de decisões”.

A utilização de métricas no processo de software será eficiente para o controle do escopo, em função da produtividade da equipe, por exemplo. Porém, as métricas serão muito eficientes em gestão da qualidade, quando será possível perceber o nível de satisfação do usuário final.

**Figura 4 - Ilustrando a melhoria contínua**



Fonte: jackaldu/iStock.com.

Em Métodos Ágeis, um dos princípios é a melhoria contínua nos processos e consequentemente do produto. A utilização de métricas é constante, conforme estimulado na Figura 4, porém não significa que se deva implementar uma métrica para todos os itens do processo. Elas são criadas cuidadosamente para agregar valor aos membros da equipe, e, como consequência, os processos e o produto sofrerão evoluções. Entre os princípios da agilidade, é preciso criar métricas com base em alguns que trarão benefícios diretos para o projeto, avaliando sempre uma tendência de velocidade e qualidade do processo de software.

Quando se opta por acompanhar a tendência da produtividade da equipe, com base em “Entrega de software em funcionamento, de algumas semanas a alguns meses, dando preferência a intervalos mais curtos”, é possível perceber se está ocorrendo um aumento gradativo em Histórias de Usuários entregues a cada período. Uma outra hipótese é escolher medir o tempo gasto na refatoração e na quantidade de bugs relacionados à arquitetura para seguir princípios como: “As melhores arquiteturas, requisitos e projetos são produzidos por equipe auto-organizadas” e “Atenção contínua à excelência técnica e um bom design aumentam a agilidade”. Essas e outras escolhas deverão partir da própria equipe de projeto, para que cada membro tenha a oportunidade de participar, adotando métricas que estimularão a constante e verdadeira melhoria.

Nem toda equipe ágil, segundo Pressman (2016), conseguirá aderir aos 12 princípios (previstos no Manifesto Ágil), atribuindo-lhes pesos iguais, simultaneamente. Assim, algumas preferem acatar a importância de certos princípios que são mais perceptíveis, mas que mantenham o **espírito ágil**.

## 4. Estimativa em Métodos Ágeis

Em Métodos Ágeis, a tarefa de estimar o esforço (ou prazo) é uma das ações mais importantes! Sabe-se que uma História do Usuário deve

estar completa e ser claramente compreendida por todos os membros do projeto, dado que a entrega dessa história no produto, ao final da Iteração, pode depender do designer, do desenvolvedor, do analista de banco de dados, do testador, ou seja, da maioria ou de todos os papéis do processo de software.

Antes de iniciar a estimativa das histórias do cliente do Backlog, vamos compreender mais um componente fundamental, a velocidade de desenvolvimento da equipe, que é o desempenho em relação à quantidade de histórias entregues pelo tempo (*Timebox*) da Iteração. Em outras palavras, segundo Sommerville (2016, p. 71), “para fazer as estimativas, eles usam a velocidade alcançada nas *sprints* anteriores, ou seja, quanto do *backlog* pôde ser concluído em uma única *sprint*”.

A partir de uma lista de requisitos de software (ou funcionalidades), chamado de *Backlog* do produto<sup>4</sup>, todos os membros do projeto escolhem um item (História do Usuário) segundo a sua prioridade. Em seguida, cada membro diz a sua estimativa em tamanho. Aqui é fundamental que cada indivíduo saiba o histórico de velocidade da equipe em cada Iteração. Assim, à medida que cada item é estimado, vai sendo formada uma lista do plano da Iteração, também chamada de *Backlog da Sprint*.

Vale salientar que a velocidade obtida em uma *Sprint* nem sempre pode ser considerada como sendo um desempenho constante nas próximas iterações. Várias situações podem ocorrer durante, o que que causa interferências em sua velocidade, como: modificação da história do usuário a pedido dele próprio; tecnologia ou habilidade de alguns dos membros em desacordo com as necessidades; e a estabilidade na produção de cada membro durante todo o tempo de *sprint*. Para Pressman (2016), os engenheiros de software apresentam grande variação nos estilos de trabalho e diferenças significativas no nível de

<sup>4</sup> Uma lista completa de todos os itens a serem desenvolvidos no projeto, devidamente escrita pelo usuário (mas que pode sofrer modificação a qualquer momento), o *Backlog* é compreendido como sendo os requisitos e/ou as funcionalidades do sistema.

habilidade, criatividade, organização, consistência e espontaneidade. Alguns se comunicam bem na forma escrita, por exemplo, enquanto outros não.

No ágil, o conceito de Velocidade é um paradigma diferente de homem/hora do esforço necessário para o desenvolvimento de um requisito ou caso de uso. Para reforçar o entendimento do que mais vale para um membro da equipe de desenvolvimento: a) o tempo a ser dispendido para entregar uma determinada funcionalidade ou b) o valor de uma funcionalidade (história do usuário) para o projeto? A resposta é sim para o valor da história do usuário e não para o tempo que se gasta para entregar essa mesma história.

Nesse ponto, é possível compreender melhor o critério de medir a velocidade da equipe: quanto mais histórias a equipe conseguir entregar em um determinado espaço de tempo, maior é a velocidade. Em outra percepção, pode-se dizer que em duas semanas a equipe conseguiu entregar mais valor para o cliente. Ou seja, a equipe é quem conseguiu cumprir com a meta de uma “versão” do software funcionando com um nível de qualidade que ela sabe fazer.

A seguir, traremos um exemplo próximo da realidade para termos o conhecimento correto da forma como a estimativa é realizada, antes do plano da iteração. Nele, será utilizada a técnica de Poker, usando a sequência de Fibonacci<sup>5</sup>, segundo a qual cada história do usuário recebe um valor, para cada membro da equipe. Segundo Layton (2019), devemos raciocinar assim: uma US de tamanho 5 indica que requer um pouco mais que a metade do esforço para uma US de 8; no entanto, requer duas vezes e meia de esforço para uma de tamanho 2, e esta, por sua vez, necessita de 66% do esforço comparado com a US de 3.

Pensem no cenário de um projeto denominado MCM para o desenvolvimento de um aplicativo *mobile* de monitoramento cardíaco

<sup>5</sup> A sequência de Fibonacci foi descoberta pelo matemático italiano Leonardo Fibonacci (1170-1250); também forma a chamada “proporção áurea”, um conceito visual utilizado em várias áreas.

de pacientes, em que existem nove histórias de usuário (*User Story*) e uma soma de 97 pontos de histórias (*Story Point*), como ilustrado na Figura 5. Os valores atribuídos para cada História do Usuário estão fundamentados na sequência de Fibonacci.

**Figura 5 – Painel de Estimativas do projeto**

SP	1	2	3	5	8	13	21	34
US	AA	AZ	BG			BX	AP	
		KM				BJ	RT	
							WA	

Fonte: elaborada pelo autor.

Embora o painel apresente a sequência de 1, 2, 3, 5, 8, 13, 21, 34..., poderão ser admitidos outros números (valores). Supondo que existam histórias com tamanho menor que 1, ou melhor, no decorrer da execução da *Sprint* estão conseguindo uma velocidade maior por terem habilidades maiores, a sua pontuação poderá ser redimensionada, criando a pontuação 0,5.

Por outro lado, caso a história seja um tanto longa, e, mesmo classificando em 34 pontos, a equipe constate dificuldades em completá-la em uma única iteração, deve-se dividi-la em duas ou mais *User Story*. É importante lembrar que tudo isso deve ter a avaliação do próprio usuário que irá usufruir dessa funcionalidade do sistema, mesmo que parcial.

É importante lembrar que, sempre que se for pontuar uma história de usuário, deve-se ter em mente o tamanho dela, e não o tempo que ela consumirá. Ou seja, é uma valorização (*Story Point*) em termos de esforço a ser dedicado pela equipe, em que todos são unanimes em dar a mesma pontuação. De acordo com Massari (2018), tal procedimento de atribuição de pontos é conhecido como estimativa Wideband Delphi, que é utilizada para a coleta de opiniões sobre um determinado assunto (no anonimato), sendo necessário repeti-la até que todos consigam emitir a sua opinião com a mesma pontuação, a partir de discussões sobre a complexidade ou extensão da história do usuário.

Nesse momento, todos da equipe devem ter o mesmo entendimento da história do usuário, em termos de composição, requisitos, regras de negócio e complexidade. Em outras palavras, é necessário saber todos os detalhes o suficiente para desenvolverem da maneira esperada pelo usuário final.

Um termo muito utilizado nas práticas ágeis é o Alcançável ou Realizável, do inglês *Achievable*. É claro que aqui está sendo considerada a funcionalidade ou a história do usuário totalmente pronta para ser disponibilizada em produção.

## 5. Gerenciamento do Escopo

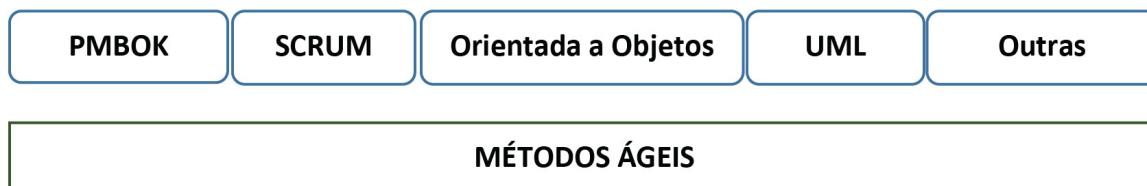
Após compreendermos como a Estimativa é construída, considerando as Métricas e o Escopo, precisamente sob a metodologia ágil, de entregas incrementais e gerenciadas por iterações, podemos dizer que o controle do projeto está no correto gerenciamento das Histórias do Usuário, em compreensão e priorização, com a colaboração obrigatória do Usuário Final.

Para alcançar o gerenciamento amplo e preciso do projeto de software, não basta ser ágil no sentido de rapidez apenas. Segundo Pressman (2016):

é essencial que o processo seja projetado de modo que a equipe possa adaptar e alinhar tarefas, possa conduzir o planejamento, compreendendo a fluidez de uma metodologia de desenvolvimento ágil; possa eliminar tudo, exceto os artefatos essenciais, conservando-os enxutos; e enfatize a estratégia de entrega incremental, conseguindo entregar ao cliente, o mais rapidamente possível, o software operacional para o tipo de produto e ambiente operacional. (PRESSMAN, 2016, p. 68)

Sendo assim, fica claro que, independentemente de técnicas de programação, metodologia ou processo de software, conforme ilustra a Figura 6, o ágil passará a fazer parte da cultura organizacional para transformar a maneira de agir, antes de se iniciar qualquer tarefa ou atividade relacionada ao software que será entregue ao usuário.

**Figura 6 – Relação das metodologias, ferramentas e técnicas com Ágil**



Fonte: elaborada pelo autor.

Para concluirmos o Tema 1, vale resgatar que a adoção da filosofia ágil é importante para obter a satisfação do cliente e ter uma constante e eficaz comunicação com o Usuário Final do sistema.

## ► Referências Bibliográficas

ABPMP. Association of Business Process Management Professionals. **Guide to the Business Process Management Body of Knowledge (BPM CBOK®)**. [s.d.]

Disponível em: <https://www.abpmp-br.org/educacao/bpm-cbok/>. Acesso em: 17 maio 2020.

ANACLETO, Thiago (org.). **Teste de Software**. São Paulo: Pearson Education do Brasil, 2016.

BECK, Kent *et al.* **Manifesto para Desenvolvimento Ágil de Software**. [s.d.]

Disponível em: <https://agilemanifesto.org/iso/ptbr/manifesto.html>. Acesso em: 16 maio 2020.

DIUANA, Gabriela (org.). **Qualidade de Software**. São Paulo: Pearson Education do Brasil, 2016.

FOWLER, Martin. **Refactoring.com**. [s.d.] Disponível em: <https://www.refactoring.com>. Acesso em: 16 maio 2020.

LAYTON, Mark C. **Gerenciamento Ágil de Projetos Para Leigos**. Rio de Janeiro: Alta Books, 2019.

MASSARI, Vitor L. **Gestão Ágil de Produtos com Agile Think Business Framework**: guia para certificação exin agile scrum product owner. Rio de Janeiro: Brasport, 2018.

PMI. **PMBOK Guia do CONHECIMENTO EM GERENCIAMENTO DE PROJETOS**.

[s.d.]. Disponível em: <https://www.pmi.org/pmbok-guide-standards/practice-guides>. Acesso em: 4 set. 2020.

PRESSMAN, Roger S. **Engenharia de Software**: uma abordagem profissional. Porto Alegre: Amgh, 2016.

SCRUM.ORG. **The home of scrum**. [s.d.]. Disponível em: <https://www.scrum.org>. Acesso em: 16 maio 2020.

SOMMERVILLE, Ian. **Engenharia de Software**. São Paulo: Pearson Education do Brasil, 2018.

# Gestão de riscos e de custos com COCOMO II

Autoria: Marco Ikuro Hisatomi

Leitura crítica: Valéria Cristina Gomes Leal

## ► Objetivos

- Conhecer a importância da gestão de riscos em projetos de software;
- Conhecer métodos de gestão de riscos.
- Conhecer quais os impactos dos custos em projetos de software.
- Conhecer modelagem de custos com COCOMO II.

## 1. Objetivo

Sobre o risco, podemos dizer que é uma ocorrência sobre a qual nem sempre temos a exata dimensão do impacto, principalmente quando não temos conhecimento de tais fatos. Apesar disso, é importante conhecermos os métodos de gestão de riscos no processo, para evitarmos prejuízos durante o desenvolvimento e após a entrega do software.

Gerenciar custos exige muita atenção de todos durante a realização de atividades do projeto, visto que uma atividade extra desnecessária causará gastos imprevisto, elevando, assim, o custo do projeto. Apresentamos a seguir a modelagem de custos com COCOMO II.

**Figura 1 – Diagrama do tema Gestão de Riscos e Modelagem de Custos**



Fonte: elaborada pelo autor.

Neste tema, iremos conhecer como as gestões de Riscos e de Custos auxiliam no gerenciamento do desenvolvimento de softwares para minimizar prejuízos, tanto pela contenção quanto pela prevenção de esforços ao longo do ciclo de vida dos sistemas.

A partir da Gestão de Riscos tradicional, será possível compreendermos quais são as características das atividades promovidas pelas organizações para evitar possíveis prejuízos decorrentes dos fatos não desejados ou planejados. Por meio do processo sistemático ISO

31000:2018 (ABNT, 2018)<sup>1</sup>, um *framework* que organiza logicamente as ações em gerenciamento de riscos, conhiceremos sobre identificação, análise, avaliação, tratamento e comunicação com as partes interessadas. Com isso, aspiramos certificar que nenhum esforço a mais seja promovido.

A área de gerenciamento de custos de um projeto de software é bem crítica, pois trata de atributos influenciado pelas áreas de processos, de pessoas e de requisitos, que sofrem mudanças constantemente. Porém, a partir da modelagem CoCoMo II, vamos conhecer os critérios e aspectos relevantes para gerenciar os custos do desenvolvimento de software, entendendo também os três submodelos para cada fase de projetos e aumentando a precisão nas estimativas propostas.

## 2. Riscos

Para um projeto de desenvolvimento de sistemas que envolve várias tarefas com características complexas e criativas, os engenheiros de software e gestores utilizam-se da fase de planejamento para elencar os possíveis riscos principais. Em muitas ocasiões, serão elencados os critérios de avaliação para continuar ou não com projeto em detrimento de fatos inesperados. Todavia, para Sommerville (2018, p. 635)

um plano de projeto é criado para registrar o trabalho a ser feito, quem irá fazê-lo, o cronograma de desenvolvimento e dos produtos de trabalho. Os gerentes usam o plano para apoiar a tomada de decisões de projeto e como uma maneira de medir o progresso.

Projetos de desenvolvimento de sistemas podem sofrer consequências indesejadas, como falta de precisão quanto à especificação da prestação do serviço de desenvolvimento; equívocos em delimitações do escopo e das funcionalidades do sistema a ser implementado; entre outros. É

<sup>1</sup> Brasileira, Norma: ABNT NBR ISO 31000:2018, 2018.

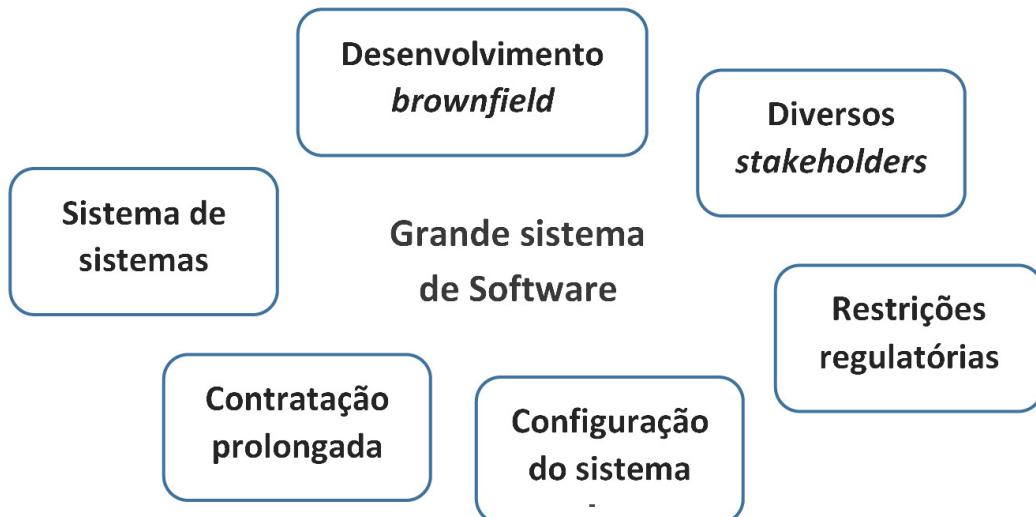
importante entender que as consequências podem ser pequenas, sendo necessário apenas ajustes, mas também podem chegar à suspensão do projeto por desentendimento entre as partes.

Vejamos um cenário hipotético de um acordo fechado entre o Cliente e o Desenvolvedor. Para esse projeto, o fornecedor oferece uma proposta que fique dentro do orçamento desejado pelo cliente, o qual concorda em delimitar as funcionalidades de maneira especificada para o software que será desenvolvido. É possível perceber que, nesse caso, poderá ocorrer divergência?

Nesse exemplo, segundo Sommerville (2018), convém se prevenir com a sinalização clara dos limites de escopo (funcionalidade especificada), o que serve de indicador. Com a definição desse indicador, ambos podem verificar variações antes que as divergências cheguem a um patamar que obrigue o cancelamento do contrato. Ainda nesse exemplo, é possível perceber que o gerenciamento de riscos inicia antes das atividades técnicas do desenvolvimento propriamente dito. Outra característica, é a experiência do gestor ou a cultura organizacional que, em muitas situações, pode antecipar certas discrepâncias para evitar ajustes de percurso devido a prejuízos.

Sommerville (2018) classifica sistemas grandes em potenciais sistemas, muito mais complexos e difíceis de compreensão em comparação aos sistemas pequenos ou com menor escopo, característica que requer maior esforços no gerenciamento de riscos. Embora o gerenciamento de grandes sistemas seja uma abordagem própria de planejamento, controle e execução do projeto, é de extrema importância conhecer as características dos grandes sistemas.

**Figura 2 – Características dos grandes sistemas**



Fonte: adaptada de Sommerville (2018).

Ilustradas na Figura 2, de acordo com os estudos de Sommerville (2018), as características de grandes sistemas de software são:

- Sistema de sistemas: são desenvolvidos por equipes diferentes, mas que se comunicam. Necessitam de maior dedicação das partes para a viabilização do crescimento e da continuidade da integração.
- Desenvolvimento *brownfield*<sup>2</sup>: é composto de vários sistemas que interagem com regras relacionadas, tornando-se resistentes ao desenvolvimento incremental. Requer negociações contínuas nos menores detalhes para manter a operação dos sistemas.
- Configuração do sistema: tem o mínimo de desenvolvimento de código para a criação de um novo sistema a partir da integração de vários. Requer o conhecimento aprofundado de todos os sistemas para garantir que a integração não seja afetada.
- Restrições regulatórias: os sistemas são integrados e possuem requisitos de conformidade específica. Exige uma documentação completa e atualizada.

<sup>2</sup> Na tradução literal “campos marrons”.

- Contratação prolongada: a movimentação de pessoal pode prejudicar o andamento do projeto. Requer a manutenção de um time coerente no maior tempo possível.
- Diversos *Stakeholders*: a diversidade em perspectivas e objetivos traz grande dificuldade para mantê-los próximos ao desenvolvimento. Deve envolver todos no processo de desenvolvimento para o alinhamento de objetivos e perspectivas.

É compreensível que a diversidade aumente o aparecimento de fatos que possam causar um impacto negativo ao projeto: pela integração de inúmeros sistemas; pela entrada e saída de membros do time; pela existência de muitos *stakeholders* com experiências e objetivos diversos; entre outras circunstâncias favoráveis ao risco.

## 2.1 Mitigação, monitoramento e gestão de riscos

O método *Risk Management, Mitigation and Monitoring* (RMMM), ou Mitigação, monitoramento e gestão de riscos, segundo Pressman (2016, p. 788), é uma maneira de planejar a contingência, visando à eficácia de como evitar, monitorar e gerenciar o risco. Esse método se torna eficaz, porque exige o envolvimento de todos para criar uma base histórica, que será utilizada para a projeção do impacto do risco identificado.

A mitigação é a escolha de um risco que possa ocorrer, sendo necessário, partir dele, determinar todos os prováveis fatores que causarão o risco. É importante para definir todas as ações sobre cada fator de risco para evitar que ele ocorra. Lembrando que se deve efetuar o mesmo procedimento para todos os riscos possíveis para o projeto de desenvolvimento de software.

O monitoramento, por sua vez, é responsabilidade do gestor, que deve supervisionar para que todos os riscos recebam a mitigação adequada.

Evidenciar um risco significa que as ações não foram suficientes para evitar o impacto, então a gestão do risco se torna ativa para conduzir ao patamar da normalidade o mais rápido possível, providenciando todas as atividades para colocar os fatores em funcionamento adequado ao processo e seguindo o plano de contingência.

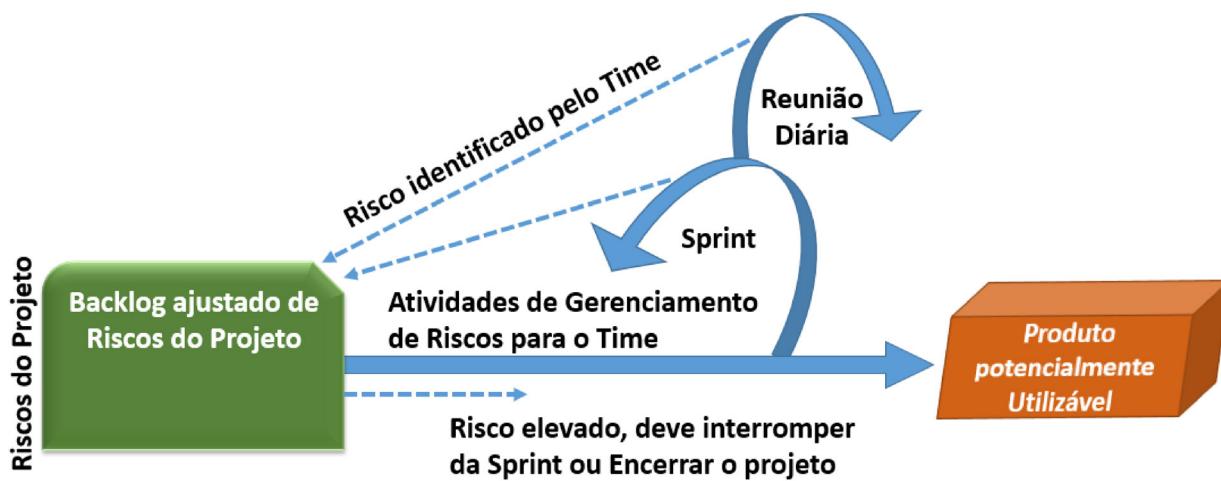
## 2.2 Riscos em Métodos Ágeis

Em Métodos Ágeis, também é considerado um desafio o desenvolvimento de sistemas com o gerenciamento de riscos em projetos de software devido a mudanças frequentes e por serem executadas atividades com foco em resultados. No *Scrum*, segundo Satpathy (2017), o propósito da comunicação diária pode revelar os riscos com maior agilidade,

sendo iterativo, o framework Scrum dá ampla oportunidade para a obtenção de feedback e definição de expectativas ao longo do ciclo de vida do projeto. Isso garante que os stakeholders do projeto, bem como o time, não sejam pegos de surpresa por má comunicação relacionada aos requisitos. (SATPATHY, 2017, p. 128)

Com base em característica ágeis que intensificam as ações de feedback diário e de revisão de produto e de processo em cada iteração, a Figura 3 destaca como o modelo propicia ao time ágil expor possíveis riscos e como estes são tratados durante a implementação do sistema, inclusive prevendo a interrupção das atividades da *Sprint* ou do projeto por completo.

**Figura 3 – Ciclo do gerenciamento de risco no Scrum**



Fonte: adaptada de Satpathy (2017).

Quando existe prejuízo em sistemas, um dos mais graves está na esfera dos negócios, ou seja, quando, na operacionalização do sistema, são descobertos impactos causados por falha na funcionalidade. Para esse caso, de acordo com Satpathy (2017), supõe-se que os Métodos Ágeis, por meio da flexibilidade, reduzem o risco de negócio, por permitirem adicionar ou modificar requisitos em curto prazo, podendo, assim, responder às ameaças e às oportunidades do ambiente de negócios, inclusive com custo mais baixo.

### 3. COCOMO II

A partir de um modelo de estimativa de tamanho de software (pontos por função, pontos por caso de uso, linhas de códigos, entre outros), o CoCoMo II (*Constructive Cost Model*), segundo Boehm (2000 *apud* LÓPEZ, 2005), é um método que estima esforço, quantidade de pessoas necessárias na equipe, prazo do serviço e custos de um projeto de desenvolvimento de *software*. O gerenciamento de projetos, contemplando todas as áreas de processo do Planejamento,

Monitoramento e Controle de projetos, pode usufruir dos propósitos desse para estimar esforço, prazo e custo.

O CoCoMo II tem na prática o uso de uma *baseline*<sup>3</sup>, segundo Boehm (2000 *apud* LÓPEZ, 2005), de exigências de um produto, formado pelo custo e pelo prazo (tempo), que, por sua vez, dependem de outros valores relativos ao projeto em desenvolvimento, como a equipe e o tamanho do sistema, conforme ilustrado na Figura 4, destacando a abrangência do CoCoMo II.

Figura 4 – Tamanho, Recurso e Custo



Fonte: elaborada pelo autor.

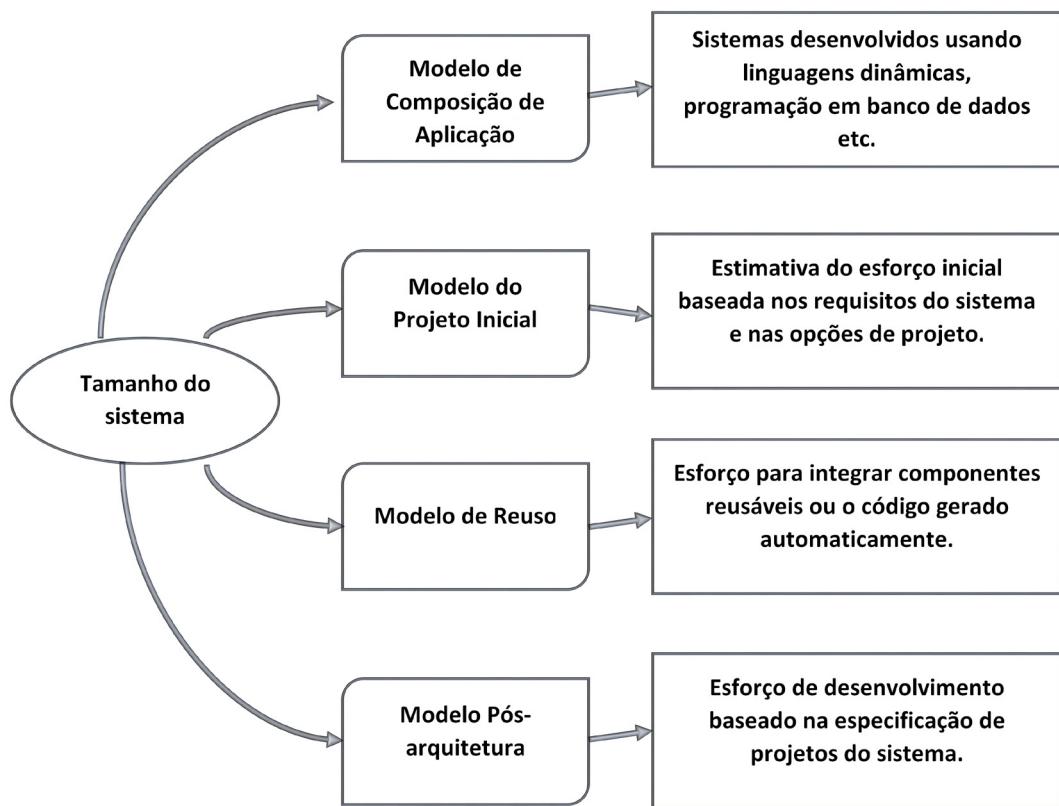
Embora o modelo CoCoMo II seja simples para entender, ele requer cuidados quanto aos detalhes e muito esforço anterior até alcançar a calibração para o ambiente de desenvolvimento que o adotará. Nesse caso, o dado mais importante é o histórico de projetos já desenvolvidos. Segundo Boehm (2000 *apud* LÓPEZ 2005), ele pode ser aplicado com os parâmetros nominais.

A adoção do modelo CoCoMo II é bem apropriada para esses tempos em que o mercado exige agilidade, uma vez que muitas metodologias são dinâmicas, com a reutilização de código e banco de dados programáveis (SOMMERVILLE, 2018), incorporando submodelos (Figura 5) para produzir estimativas mais detalhadas.

<sup>3</sup> Uma *baseline* é um conjunto de produtos aceitos e controlados que serão utilizados em atividades posteriores à sua aceitação (LEITE, 1997).

- Modelo de composição da aplicação: é apropriado para o estágio de prototipação de sistemas, linguagens de *script* ou programação em banco de dados, com o objetivo de estimar o esforço baseado em fórmula de tamanho/produtividade.
- Modelo de projeto inicial: é utilizado quando os requisitos são estabelecidos e as alternativas de arquitetura do software foram exploradas, com base em pontos de função que quantificam a funcionalidade independente da linguagem.
- Modelo de reuso: é usual quando existe a integração de componentes reutilizáveis e/ou o código de programa é gerado automaticamente.
- Modelo pós-arquitetura: é o modelo mais detalhado e envolve as etapas de construção real do software, após o projeto da arquitetura ser definido, refletindo a capacidade de pessoal e as características do produto e do projeto.

**Figura 5 – Modelos de estimativa CoCoMo II**



Fonte: elaborada pelo autor.

Com a intenção de oferecer sustentação adequada para as fases de planejamento de projetos, equipe de desenvolvimento, preparação inicial, replanejamento, negociação de contratos, avaliações de propostas e níveis de recursos (tecnológicos e humanos), o CoCoMo II estabelece os seguintes objetivos primários:

- Seguindo os processos determinados pelo ciclo de vida de software, segundo Jones (1986 *apud* LÓPEZ, 2005), é preciso elaborar a estimativa de custos, com base em, pelo menos, dez anos praticados no desenvolvimento.
- Deve ser adotada uma ferramenta específica pela equipe para manter uma base de dados, preservando um histórico das estimativas, que será utilizado futuramente para as melhorias no modelo de estimativa.
- É importante avaliar os efeitos de melhoria na tecnologia e nos custos pagos ao longo do ciclo de vida de desenvolvimento do software, por meio de ferramentas e técnicas implementadas em um *framework* analítico.

Para o uso adequado do modelo CoCoMo II, há um manual de definição<sup>4</sup> disponibilizado pela *University of Southern California* (USC, 2000).

Na sequência, acompanhemos os passos para demonstrar uma medição e um cálculo usando o modelo CoCoMo II. Partindo para uma visão mais prática do modelo, devemos verificar o processo considerando uma aplicação cujo tamanho do sistema, elaborado pela análise de ponto por função, é de 210 pontos (não ajustados). É importante lembrar que o CoCoMo II parte do critério de que o valor do ponto de função deve ser o não ajustado, pois o próprio modelo contempla o ambiente e suas características, o que impacta no resultado da estimativa de Esforço, Prazo e Custo.

<sup>4</sup> Documento com manual das definições do modelo CoCoMo II, elaborado pelo *Center for Software Engineering* da *University of Southern California* (USC) (USC, 2000).

O cálculo do ponto de função foi alcançado de acordo com o escopo de sistema e suas funcionalidades, conforme ilustrado na Figura 6.

**Figura 6 – Tamanho do sistema em ponto de função**

Aplicação : ENGENHARIA DE SOFTWARE			Projeto : GESTÃO DE CUSTOS												
Responsável : MESTRE			Revisor :												
Empresa : POS-GRADUAÇÃO			Tipo de Contagem : Projeto de Desenvolvimento												
Tipo de Função		Complexidade Funcional	Total PF IFPUG por Complexidade	%	Total PF Local FS por tipo de manutenção básica %										
SE	0	Baixa x 4	0		I= 0,00										
	0	Média x 5	0		A= 0,00										
	0	Alta x 7	0		E= 0,00										
<b>Qtd Total</b>	<b>0</b>	<b>Total</b>	<b>0</b>	<b>0,0%</b>	<b>0,00</b> 0,0%										
CE	0	Baixa x 3	0		I= 14,00										
	3	Média x 4	12		A= 2,00										
	1	Alta x 6	6		E= 0,00										
<b>Qtd Total</b>	<b>4</b>	<b>Total</b>	<b>18</b>	<b>28,6%</b>	<b>16,00</b> 31,4%										
ALI	6	Baixa x 7	42		I= 28,00										
	0	Média x 10	0		A= 7,00										
	0	Alta x 15	0		E= 0,00										
<b>Qtd Total</b>	<b>6</b>	<b>Total</b>	<b>42</b>	<b>66,7%</b>	<b>35,00</b> 68,6%										
AIE	0	Baixa x 5	0		I= 0,00										
	0	Média x 7	0		A= 0,00										
	0	Alta x 10	0		E= 0,00										
<b>Qtd Total</b>	<b>0</b>	<b>Total</b>	<b>0</b>	<b>0,0%</b>	<b>0,00</b> 0,0%										
Total PF não ajustados (contagem detalhada)			63	0,0% 4,8%	0,0%										
Total PF não ajustados (contagem estimativa)			62		28,6%										
Total PF não ajustados (contagem indicativa)			210	66,7%											
<p>% por Tipo de Função</p> <table border="1"> <tr> <td>EE</td> <td>0,0%</td> </tr> <tr> <td>SE</td> <td>0,0%</td> </tr> <tr> <td>CE</td> <td>28,6%</td> </tr> <tr> <td>ALI</td> <td>66,7%</td> </tr> <tr> <td>AIE</td> <td>0,0%</td> </tr> </table>						EE	0,0%	SE	0,0%	CE	28,6%	ALI	66,7%	AIE	0,0%
EE	0,0%														
SE	0,0%														
CE	28,6%														
ALI	66,7%														
AIE	0,0%														

Fonte: captura de tela da planilha de FATTO ([s.d.]).<sup>5</sup>

<sup>5</sup> Planilha para calcular o Ponto de Função como ferramenta para medição.

Para seguir o cálculo usando o modelo CoCoMo II, vamos utilizar o dado recém-mostrado com o valor de 210 pontos de função, que usaremos como insumo principal na ferramenta disponibilizada para o Modelo de Custo Construtivo, já preparado para considerar vários atributos que influenciam diretamente o prazo, o custo e os esforços necessários. Tal ferramenta, chamada Softwrecost.org COCOMO<sup>6</sup>, pode ser acessada para esse fim.

A fim de demonstrar a variação existente no resultado de esforço, prazo e custo, foram efetuadas duas simulações no submodelo Pós-arquitetura, sendo aqui denominadas de Simulação A e B. Ambas usam o mesmo tamanho inicial de 210 pontos de função, porém com as seguintes características, alterando apenas duas variáveis do custo de Software de Pessoal:

Simulação A (Figura 7):

- Capacidade do programador: nominal.
- Experiência de aplicação: nominal.
- Resultado em custo de \$ 12,742.
- Resultado de esforço de 15 pessoa/mês.
- Resultado de prazo de 9 meses.

<sup>6</sup> Software on-line para simular o cálculo de esforço, prazo e custo tem a parametrização para elaborar um resultado a partir do tamanho do sistema em Pontos de Função.

## Figura 7 – CoCoMo II – pós-arquitetura (Simulação A)

COCOMO II - Modelo de Custo Construtivo

Risco de Monte Carlo  
 Fora ▾  
 Cálculo automático  
 Fora ▾

Método de dimensionamento de tamanho de software [Pontos de Função ▾] Pontos de função <input type="text" value="210"/> Língua <input style="width: 100px;" type="text" value="Java"/> <small>Pontos de função não ajustados</small>																										
<b>Drivers de escala de software</b> Precedência <input type="text" value="Nominal"/> Arquitetura / Resolução de Riscos <input type="text" value="Nominal"/> Flexibilidade de Desenvolvimento <input type="text" value="Muito alto"/> Coesão da equipa <input type="text" value="Muito alto"/>  <b>Drivers de custo de software</b> Produtos																										
<table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">       Confiabilidade de software necessária  <input type="text" value="Baixo"/> </td> <td style="width: 30%; text-align: center;"> <b>Pessoal</b>        Capacidade do analista  <input type="text" value="Nominal"/>        Capacidade do Programador  <input type="text" value="Baixo"/> </td> <td style="width: 30%; text-align: center;"> <b>Plataforma</b>        Restrição de tempo  <input type="text" value="Muito alto"/>        Restrição de armazenamento  <input type="text" value="Nominal"/> </td> </tr> <tr> <td>       Tamanho da base de dados  <input type="text" value="Muito baixo"/> </td> <td>       Continuidade do pessoal  <input type="text" value="Nominal"/> </td> <td>       Volatilidade da plataforma  <input type="text" value="Nominal"/> </td> </tr> <tr> <td>       Complexidade do produto  <input type="text" value="Nominal"/> </td> <td>       Experiência de aplicação  <input type="text" value="Nominal"/> </td> <td>       Projeto  <input type="text" value="Nominal"/> </td> </tr> <tr> <td>       Desenvolvido para Reutilização  <input type="text" value="Nominal"/> </td> <td>       Experiência de plataforma  <input type="text" value="Nominal"/> </td> <td>       Uso de ferramentas de software  <input type="text" value="Nominal"/> </td> </tr> <tr> <td>       Correspondência da documentação às necessidades do ciclo de vida  <input type="text" value="Nominal"/> </td> <td>       Experiência de idioma e conjunto de ferramentas  <input type="text" value="Muito alto"/> </td> <td>       Desenvolvimento Multisite  <input type="text" value="Nominal"/> </td> </tr> <tr> <td></td> <td></td> <td>       Cronograma de desenvolvimento necessário  <input type="text" value="Nominal"/> </td> </tr> </table>			Confiabilidade de software necessária <input type="text" value="Baixo"/>	<b>Pessoal</b> Capacidade do analista <input type="text" value="Nominal"/> Capacidade do Programador <input type="text" value="Baixo"/>	<b>Plataforma</b> Restrição de tempo <input type="text" value="Muito alto"/> Restrição de armazenamento <input type="text" value="Nominal"/>	Tamanho da base de dados <input type="text" value="Muito baixo"/>	Continuidade do pessoal <input type="text" value="Nominal"/>	Volatilidade da plataforma <input type="text" value="Nominal"/>	Complexidade do produto <input type="text" value="Nominal"/>	Experiência de aplicação <input type="text" value="Nominal"/>	Projeto <input type="text" value="Nominal"/>	Desenvolvido para Reutilização <input type="text" value="Nominal"/>	Experiência de plataforma <input type="text" value="Nominal"/>	Uso de ferramentas de software <input type="text" value="Nominal"/>	Correspondência da documentação às necessidades do ciclo de vida <input type="text" value="Nominal"/>	Experiência de idioma e conjunto de ferramentas <input type="text" value="Muito alto"/>	Desenvolvimento Multisite <input type="text" value="Nominal"/>			Cronograma de desenvolvimento necessário <input type="text" value="Nominal"/>						
Confiabilidade de software necessária <input type="text" value="Baixo"/>	<b>Pessoal</b> Capacidade do analista <input type="text" value="Nominal"/> Capacidade do Programador <input type="text" value="Baixo"/>	<b>Plataforma</b> Restrição de tempo <input type="text" value="Muito alto"/> Restrição de armazenamento <input type="text" value="Nominal"/>																								
Tamanho da base de dados <input type="text" value="Muito baixo"/>	Continuidade do pessoal <input type="text" value="Nominal"/>	Volatilidade da plataforma <input type="text" value="Nominal"/>																								
Complexidade do produto <input type="text" value="Nominal"/>	Experiência de aplicação <input type="text" value="Nominal"/>	Projeto <input type="text" value="Nominal"/>																								
Desenvolvido para Reutilização <input type="text" value="Nominal"/>	Experiência de plataforma <input type="text" value="Nominal"/>	Uso de ferramentas de software <input type="text" value="Nominal"/>																								
Correspondência da documentação às necessidades do ciclo de vida <input type="text" value="Nominal"/>	Experiência de idioma e conjunto de ferramentas <input type="text" value="Muito alto"/>	Desenvolvimento Multisite <input type="text" value="Nominal"/>																								
		Cronograma de desenvolvimento necessário <input type="text" value="Nominal"/>																								
Manutenção <input type="text" value="Fora"/>																										
Custo das taxas de mão-de-obra de software por pessoa-mês (dólares) <input type="text" value="850"/> <input type="button" value="Calcular"/>																										
<b>Resultados</b>																										
Esforço de desenvolvimento de software (elaboração e construção) = 15,0 Horário pessoa-mês = 9,0 meses Custo = \$ 12742																										
Tamanho equivalente total = 11130 Distribuição da fase de aquisição do SLOC																										
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Estágio</th> <th>Esforço (pessoa-mês)</th> <th>Programação (meses)</th> <th>Pessoal Médio</th> <th>Custo (dólares)</th> </tr> </thead> <tbody> <tr> <td>Começo</td> <td>0,9</td> <td>1.1</td> <td>0,8</td> <td>\$ 765</td> </tr> <tr> <td>Elaboração</td> <td>3,6</td> <td>3,4</td> <td>1,1</td> <td>\$ 3058</td> </tr> <tr> <td>Construção</td> <td>11,4</td> <td>5,6</td> <td>2,0</td> <td>\$ 9684</td> </tr> <tr> <td>Transição</td> <td>1,8</td> <td>1,1</td> <td>1,6</td> <td>\$ 1529</td> </tr> </tbody> </table>	Estágio	Esforço (pessoa-mês)	Programação (meses)	Pessoal Médio	Custo (dólares)	Começo	0,9	1.1	0,8	\$ 765	Elaboração	3,6	3,4	1,1	\$ 3058	Construção	11,4	5,6	2,0	\$ 9684	Transição	1,8	1,1	1,6	\$ 1529	
Estágio	Esforço (pessoa-mês)	Programação (meses)	Pessoal Médio	Custo (dólares)																						
Começo	0,9	1.1	0,8	\$ 765																						
Elaboração	3,6	3,4	1,1	\$ 3058																						
Construção	11,4	5,6	2,0	\$ 9684																						
Transição	1,8	1,1	1,6	\$ 1529																						

Fonte: captura de tela do aplicativo Softwrecost.org ([s.d.]).

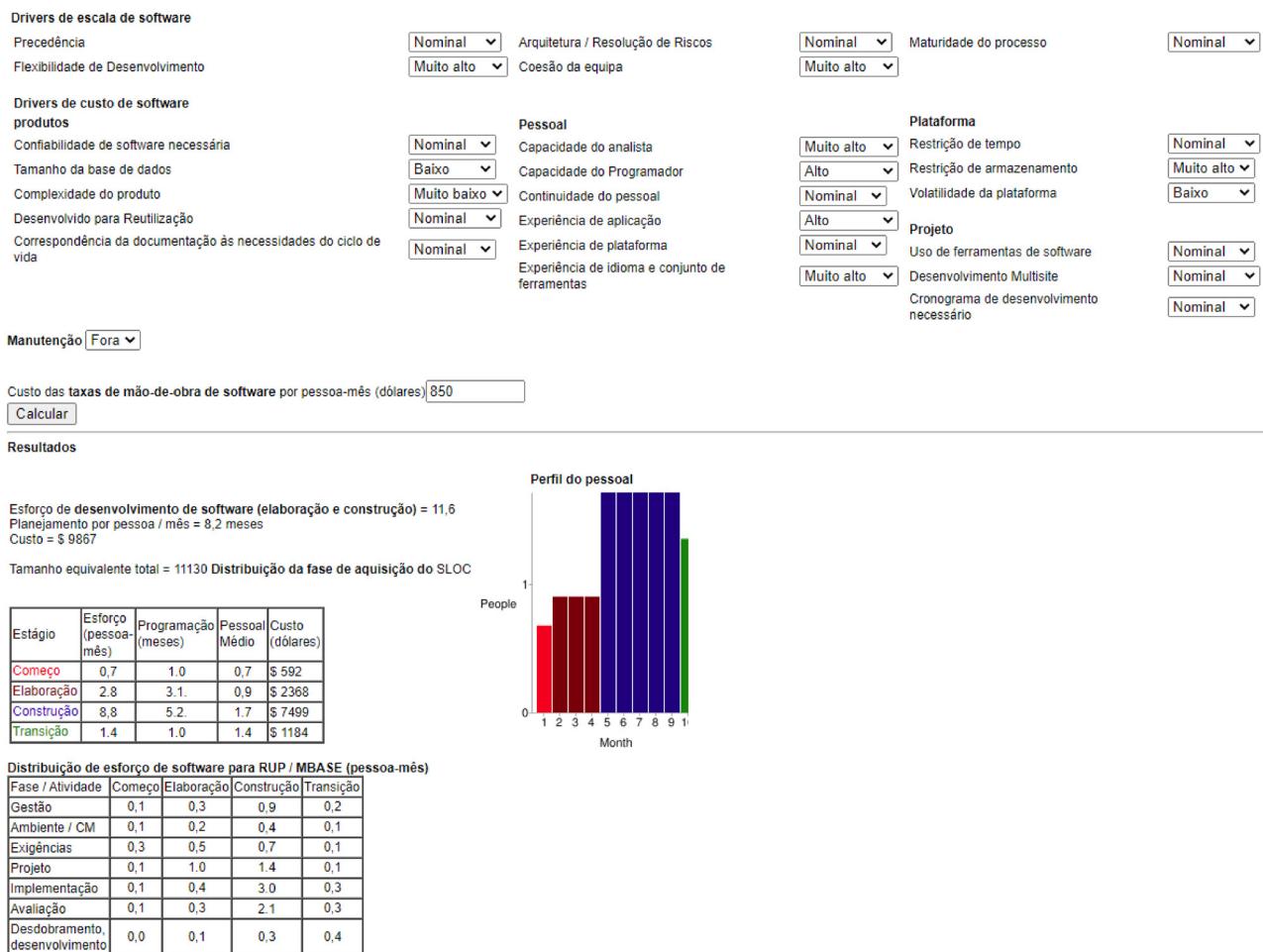
Na simulação B, é efetuada a alteração em dois parâmetros, pois considera que o desenvolvedor tem maior proficiência e experiência em relação à simulação A.

Simulação B (Figura 8):

- Capacidade do programador: **alta**.
- Experiência de aplicação: **alta**.
- Resultado em custo de \$ 9,867.

- Resultado de esforço de 11,6 pessoa/mês.
- Resultado de prazo de 8,2 meses.

**Figura 8 – CoCoMo II – pós-arquitetura (Simulação B)**



Fonte: captura de tela do aplicativo Softwarecost.org ([s.d.]).

Tanto a gestão de riscos quanto a de custos de um projeto ágil tendem à eficiência pelas diretrizes de constante participação do cliente, ou do dono do produto (pelo *Scrum*, denominado *Product Owner*<sup>7</sup>). Todos envolvidos devem estar em comunicação com a menor periodicidade possível para alterar histórias, o que é demandado pelos negócios, em reuniões diárias, testando e validando as funcionalidades no fechamento da iteração. É importante lembrar que tudo isso deve contar com a presença do time ágil do desenvolvimento do software.

<sup>7</sup> O *Product Owner* é o conhecedor das Histórias de Usuário e representante dos usuários finais.



## Referências Bibliográficas

- ABNT. Associação Brasileira de Normas Técnicas. **ISO 31000: Gestão de riscos-Diretrizes**. Rio de Janeiro: ABNT, 2018.
- FATTO. Identificação da contagem. [s.d.]. Disponível em: [encurtador.com.br/qyBT2](http://encurtador.com.br/qyBT2). Acesso em: 5 set. 2020.
- IT-SWARM.DEV. **O que são aplicações greenfield e brownfield?** Disponível em: <https://www.it-swarm.dev/pt/language-agnostic/o-que-sao-aplicacoes-greenfield-e-brownfield/967093897/>. Acesso em: 5 set. 2020.
- LEITE, Julio Cesar S. P. *et. al.* Enhancing a Requirements Baseline with Scenarios. **Requirements Engineering**, [s.l.], v. 2, n. 4, p. 184-198, 1997. Disponível em: <http://www-di.inf.puc-rio.br/~julio/Slct-pub/baseline.pdf>. Acesso em: 5 set. 2020.
- LÓPEZ, Pablo Ariel do Prado. COCOMO II-Um modelo para estimativa de custos de Gerência de Projetos. SIMPÓSIO DE INFORMÁTICA DA REGIÃO CENTRO DO RS, 4., 2005, Santa Maria. **Anais** [...]. Santa Maria: SIRC/RS, 2005. Disponível em <https://pdfs.semanticscholar.org/c90b/6117b6ae41b6c9b19bfff1af15a64b343d7b.pdf>. Acesso em 29 maio 2020.
- PRESSMAN, Roger S. **Engenharia de Software**: uma abordagem profissional. Porto Alegre: Amgh, 2016.
- SATPATHY, Tridibesh. **Um Guia para o Conhecimento em Scrum (Guia SBOK™)**. Arizona: SCRUMstudy, 2017.
- SOFTWARECOST.ORG. **COCOMO II-Constructive Cost Model**. [s.d.]. Disponível em: <http://softwarecost.org/tools/COCOMO/>. Acesso em: 5 set. 2020.
- SOMMERVILLE, Ian. **Engenharia de Software**. São Paulo: Pearson Education do Brasil, 2018.
- USC. University of Southern California. Center for Software Engineering. **COCOMO II Model Definition Manual**. 2000. Disponível em: <http://www.dmi.usherb.ca/~frappier/IFT721/COCOMOII.PDF>. Acesso em: 5 set. 2020.

# Gestão da Qualidade

Autoria: Marco Ikuro Hisatomi

Leitura crítica: Valéria Cristina Gomes Leal



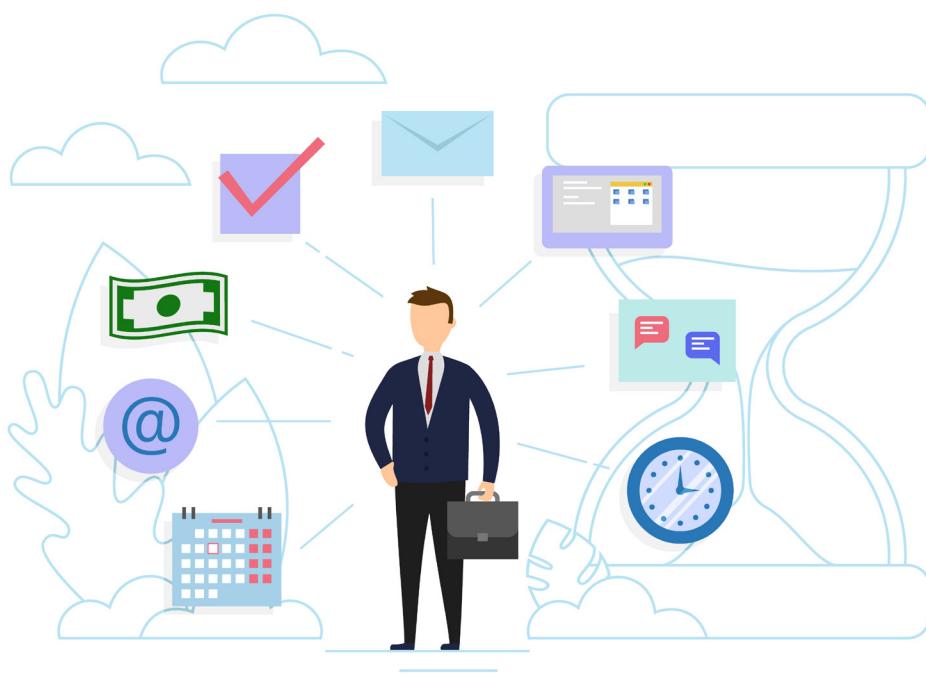
- Conhecer o que é gestão da qualidade.
- Conhecer métodos de controle da qualidade.
- Compreender que processos ágeis podem sofrer impactos com a gestão de qualidade.

## ► 1. Gestão da qualidade

O assunto deste tema é a Gestão da Qualidade! Sabemos que pode ser um grande desafio manter o comprometimento das entregas de software funcionando operacionalmente. O foco em qualidade está sempre presente no desenvolvedor de software, desde o início do processo, conhecendo o negócio, entendendo os requisitos e detalhando as histórias do usuário, a construção do código e a entrega do produto estão sendo guiadas pela luz da qualidade.

Vamos conhecer o que é a Gestão da Qualidade, entender quais metodologias podem ser contempladas nessa gestão para conseguir o efetivo controle dos processos do desenvolvimento de software e, por fim, conhecer as vantagens dos Métodos Ágeis no controle da qualidade de software.

**Figura 1 – Dinâmica da gestão da qualidade**



Fonte: iStock – Intpro/iStock.com.

Como podemos perceber na Figura 1, o gestor da qualidade precisa estar atento a todos os detalhes do processo de software, pois todos os recursos, artefatos, ações, comunicações, entre outros fatores afetam a qualidade do software em desenvolvimento. Para Bessin (2004, p. 2 *apud* PRESSMAN, 2016, p. 414), “a qualidade de software pode ser definida como uma gestão da qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam”.

Considerando que o ambiente global se apresenta em rápida mudança, as respostas em software devem acompanhar essa velocidade imposta, já que a maioria dos negócios está alicerçada em tecnologias da informação. Sendo assim, não basta cuidar dos fatores que afetam diretamente a qualidade, mas sim garantir que as mudanças serão atendidas com agilidade, aspirando à entrega de um software útil e satisfatório para o cliente.

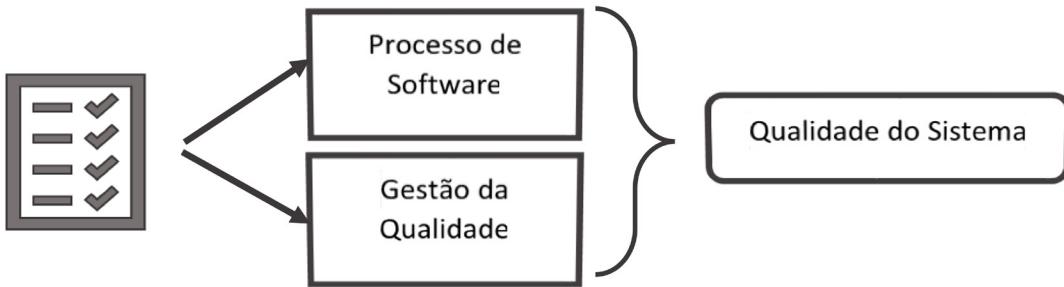
## 1.1 A respeito da qualidade de software

Definir o que é qualidade em software dependerá da tecnologia utilizada; do nível de conhecimento dos usuários; do quanto preciso deve ser o tratamento dos dados e das informações; da pressão da concorrência; e, ainda, da subjetividade de cada pessoa, com o seu grau de expectativa, da demanda de mercado e dos negócios no qual está sendo útil.

Ultimamente, o fator velocidade do ambiente mundial tem sido o norteador para os negócios; trata-se de uma mudança rápida, uma nova versão do software acrescida de modificações, com regras atualizadas. Sommerville (2018) reforça que implantar um novo software rapidamente poderá ajustar os requisitos, ademais é importante estar disposto para negociar a qualidade.

Para ilustrar esse cenário, a Figura 2 abre uma suposta lista de qualidades negociadas para serem contempladas ao longo do processo de desenvolvimento, a fim de resultar em um sistema de qualidade satisfatório para o cliente.

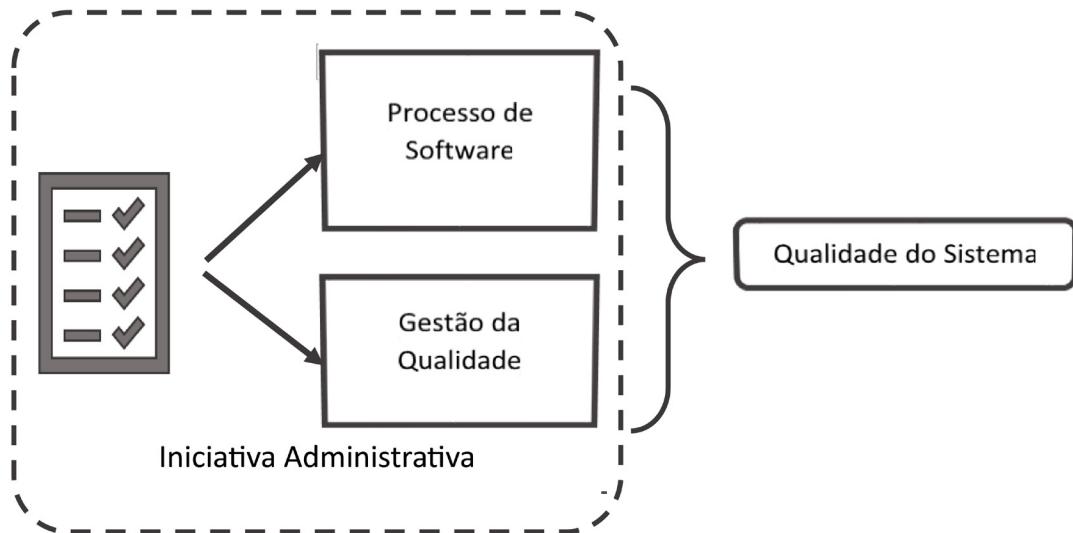
**Figura 2 – Qualidade negociada**



Fonte: elaborada pelo autor.

O processo de software prevê atividades diretamente relacionadas ao time ágil, com ações de cunho puramente técnico, como entendimento e detalhamento das histórias, construção, testes e entrega do produto parcial ou total a cada iteração do ciclo de vida. Contudo, sabemos que para a realização de todo esse processo o projeto depende de outras áreas organizacionais (Figura 3), como infraestrutura que suporte a construção de um software de alta qualidade e dispositivos de controle para evitar o caos no projeto por meio de iniciativas administrativas (PESSMAN, 2016).

**Figura 3 – Qualidade negociada com apoio administrativo**



Fonte: elaborada pelo autor.

Dando sequência, Pressman (2016) declara que a engenharia de software deve ter agilidade e ser adaptável ao problema, à equipe, à cultura organizacional e ao projeto.

Chegamos ao ponto base para sustentar a gestão da qualidade, por meio da melhoria do processo de software ou SPI (*Software Process Improvent*), que se comprehende da seguinte maneira: a) processo de software definido; b) abordagem organizacional; e c) estratégia para a melhoria. A organização que acredita na melhoria do processo de software percebe que a **cultura organizacional** é fundamental para suportar a transição tecnológica, determinar o nível em que se encontra, proporcionar a preparação para absorver as mudanças no processo e medir o nível após a adoção das mudanças (PRESSMAN, 2016).

Da mesma forma que a introdução de Métodos Ágeis nas organizações, quando a cultura anterior era diferente, a implantação do processo de melhoria contínua também pode causar uma resistência natural a ser considerada. Porém, fatalmente, ações em qualidade exigirão alguma adequação organizacional.



## 2. Métodos de controle da qualidade

Ao optar pelo desenvolvimento de software com qualidade, as organizações necessitam recorrer às técnicas e experiências catalogadas e praticadas. Porém, elas não bastarão por si, pois sempre que um método de sucesso for adotado, o ambiente organizacional, as habilidades pessoais, as tecnologias e as complexidades devem ser consideradas e adequadas a sua realidade.

Para tanto, vamos entender o SPI, que tem uma proposta ampla e completa, com atividades-base denominadas IDEAL<sup>1</sup>, uma sigla que vem das ações Iniciar, Diagnosticar, Estabelecer, Agir e Aprender. De acordo com SEI (2001, p. 1 *apud* PRESSMAN, 2016, p. 823), “modelo de melhoria organizacional que serve de roteiro para iniciar, planejar e implementar ações de melhoria”. Porém, antes vamos entender onde terá impacto.

Compreendendo as dimensões de qualidade, Garvin (1987 *apud* SOMMERVILLE, 2018) sugere que tenhamos uma visão transcendental para que essas dimensões sejam aplicadas em software: **desempenhar** as funcionalidades que geram valor ao usuário; surpreendem os **recursos** ao primeiro uso; **confiar** que o software está disponível quando precisar, sem falhas ou erros; estar em **conformidade** com os padrões locais relacionados; manter-se **durável** após modificações, sem efeitos colaterais; ter **facilidade de manutenção** e atualizações em tempo hábil; ser perceptível a **estética** do software, mesmo que subjetiva; ter a **percepção** de qualidade (ou não) a partir do conjunto de características do software, principalmente se comparado com outros.

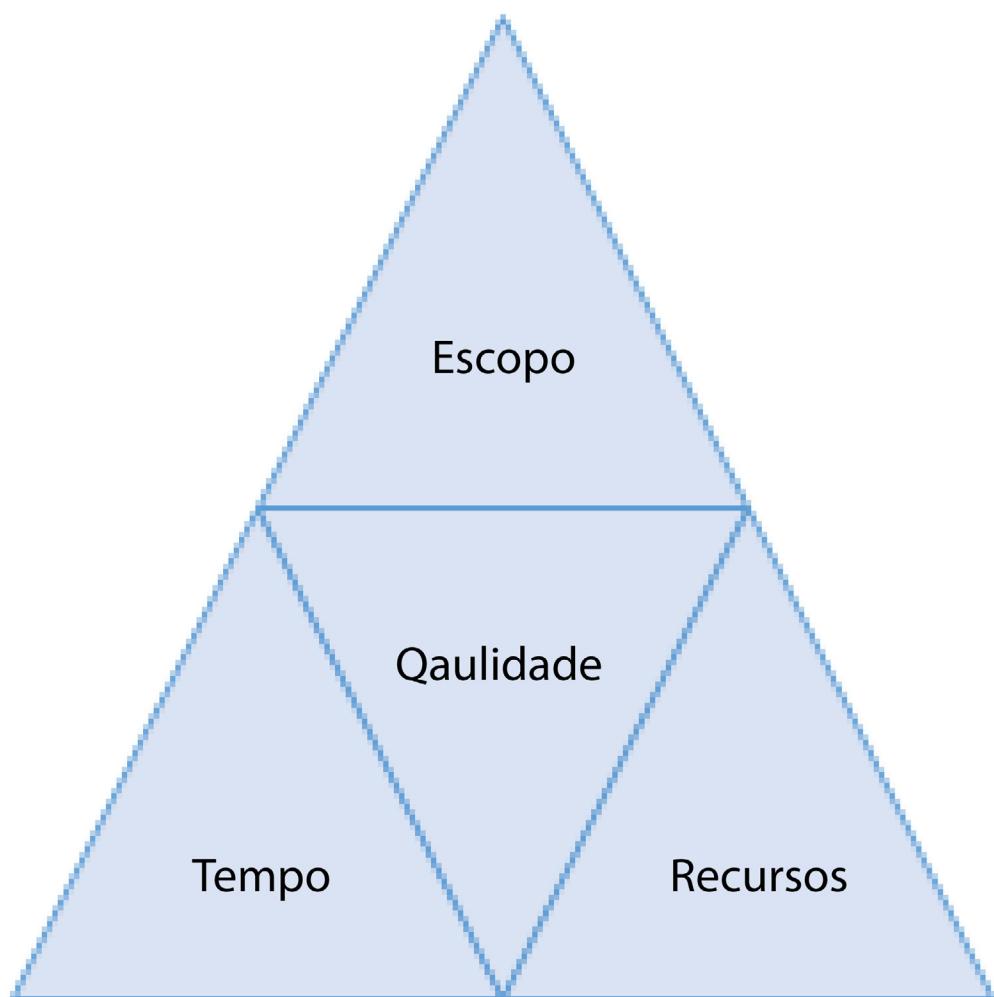
Talvez essas sejam as dimensões norteadores para alcançar, a partir do estabelecimento de objetivos para a equipe de desenvolvimento, a qualidade no software. Tendo isso em mente, como certificar-se de que essas dimensões foram contempladas pelo software? Além disso, como

<sup>1</sup> O IDEAL é um *roadmap* desenvolvido pela SEI (*Software Engineering Institute*) em 2001.

a equipe saberá que esses são os critérios esperados pelo cliente ou pelo projeto?

Nem tudo que se pretende de um nível ótimo de qualidade será possível, mas, por outro lado, desenvolver um software cheio de falhas poderá ser o último projeto da equipe. Venners (2003 *apud* PRESSMAN 2016) deixa uma reflexão para que qualquer equipe possa implementar uma gestão da qualidade de acordo com as necessidades atualizadas e conforme a cultura organizacional da atualidade, considerando o orçamento e o prazo disponíveis.

**Figura 4 – Equilíbrio entre a qualidade e as demais áreas do conhecimento**



Fonte: adaptada de Montes (2018).

O ideal é encontrar o equilíbrio (Figura 4) entre entregar os requisitos do usuário (escopo) e utilizar adequadamente os recursos destinados ao projeto dentro do prazo combinado com o cliente (tempo), alcançando, assim, um nível de qualidade satisfatório para a operacionalização dos negócios. Para certificar-se de que o equilíbrio irá acontecer, a equipe deve eleger um método para controlar a qualidade, desde que as definições desse método possuam os princípios de melhoria da filosofia de bom senso do SPI.

## 2.1 CMMI - maturidade do processo

Com foco em dar suporte e colaborar efetivamente com o processo de melhoria, por meio da gestão da qualidade, vamos entender o que é CMMI<sup>2</sup>, que sugere o gerenciamento da maturidade das organizações, sendo o principal indicador da qualidade em processo de software, e como os membros das equipes entendem, aplicam e praticam a engenharia de software (PRESSMAN, 2016). Sob a avaliação das atividades desempenhadas efetivamente em cada fase do processo, sempre referenciadas por metas específicas para cada prática ou atividade, faz-se uma classificação em níveis quanto à capacidade organizacional, indo desde o fato de a organização não alcançar suas metas e seus objetivos de acordo com as diretrizes definidas pelo CMMI até o nível em que a própria estrutura organizacional está preparada para continuar a melhoria do processo para atender às tendências do mercado e às necessidades dos clientes.

A Figura 5 ilustra cada um dos níveis, segundo o CMMI:

**Incompleto:** não atinge todas as metas e objetivos definidos pelo CMMI para a capacidade nível 1 da área de processo.

---

<sup>2</sup> O CMMI vem de *Capability Maturity Model Integration* e foi desenvolvido e atualizado pelo instituto SEI. É um metamodelo que sugere cinco níveis de maturidade no processo de software.

**Executado:** estão sendo executadas as tarefas necessárias para produzir os artefatos definidos.

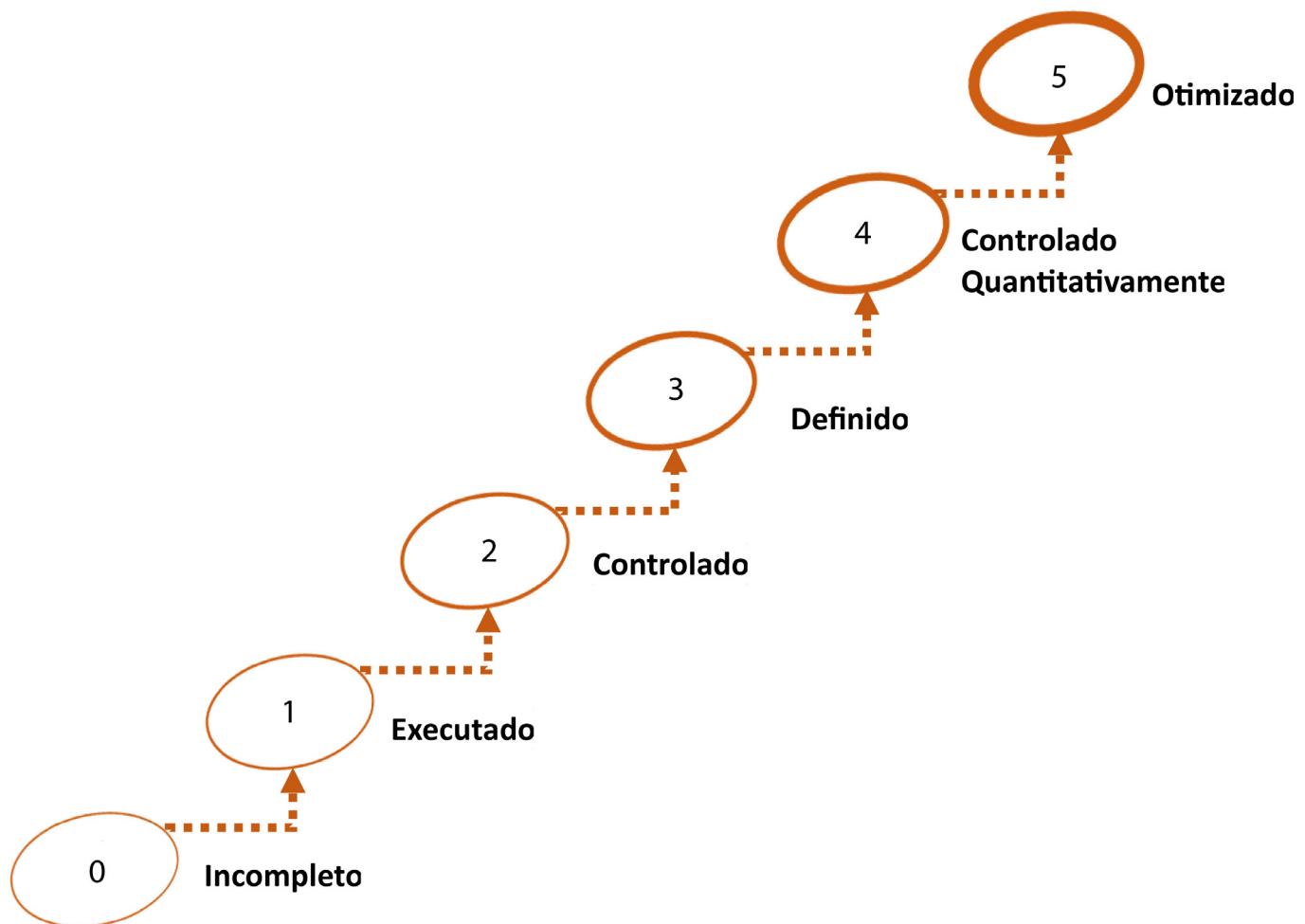
**Controlado:** todo o trabalho associado à área de processo está de acordo com uma política definida em termos de organização; todas as tarefas e produtos são “monitorados, controlados, revisados e avaliados quanto à conformidade com a descrição de processo” (CMMI, 2010, p. 42).

**Definido:** o processo é “adaptado com base no conjunto de processos padronizados da organização, de acordo com as regras de adaptação da organização e dos produtos acabados, medidas e outras informações de melhoria de processo para agregar valores ao processo organizacional” (CMMI, 2010, p. 43).

**Controlado quantitativamente:** a área de processo é “controlada e melhorada usando medição e avaliação quantitativa. São estabelecidos objetivos quantitativos para qualidade e desempenho de processo e utilizados como critérios no controle do processo” (CMMI, 2010, p. 43).

**Otimizado:** a área de processo é “adaptada e otimizada usando meios quantitativos (estatísticos) para atender à mudança de necessidades do cliente e melhorar continuamente a eficiência da área de processo em consideração” (CMMI, 2010, p. 44).

**Figura 5 – Níveis de maturidade do CMMI**



Fonte: adaptada de Chrissis (2011, p. 34).

Em cada um dos níveis, a qualidade é tratada para ser realizada formalmente, registrando os fatos e as medições conforme o plano estabelecido e com as diretrizes do CMMI, de acordo com cada nível de maturidade.

Vejamos um exemplo detalhado de um dos processos estruturados para a gestão da qualidade. Segundo o CMMI (2010)<sup>3</sup>, as áreas de validação e verificação descrevem conhecimentos relacionados diretamente com a qualidade no processo de desenvolvimento de software (CRHIISSIS, 2011), por meio da **Verificação** (para cada momento do processo de

<sup>3</sup> O modelo denominado CMMI for Development (CMMI-DEV) foi elaborado com um conjunto de diretrizes especificamente para o desenvolvimento de serviços e produtos.

software, as práticas de implementações devem ser verificadas). Com a verificação, entende-se que o procedimento deve estar previsto para cumprir o uso de uma determinada técnica ou configuração de uma determinada ferramenta, devendo esses procedimentos serem comparáveis com as práticas do CMMI.

Já pela **Validação**, de acordo com Chrassis (2011), todos os itens resultantes das atividades executadas devem ser conferidos de acordo com o modelo de referência ou com as especificações planejadas. Ela tem foco em validar os produtos ou os subprodutos relativos a uma atividade por membros da equipe, de acordo com as descrições definidas no plano da execução de tal atividade.

O CMMI é um *framework* largamente usado pelas grandes organizações e vem sendo adotado de forma customizada em empresas menores, porque tem uma proposta de melhoria contínua, baseada na maturidade de processos.

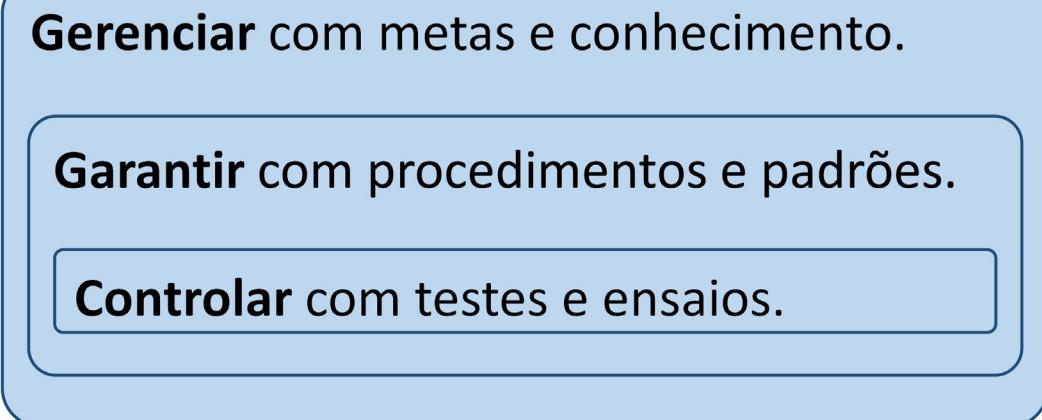
## 2.2 Da Engenharia de Software

É a partir das ações de um desenvolvedor, na prática das técnicas de engenharia de software, por meio da Gestão da Qualidade, tanto no entendimento de requisitos quanto na codificação dos programas, que um produto de software será entregue conforme a qualidade esperada pela equipe, pelo cliente e pelo usuário final. Sendo assim, além de conhecer profundamente as dimensões de qualidade, segundo Garvin (1987 *apud* SOMMERVILLE, 2018), a equipe deve estar preparada para seguir as diretrizes estratégicas da organização, com base na SPI, para estabelecer o detalhamento do propósito em qualidade no processo de desenvolvimento.

Para planejar e estruturar, a equipe deve conhecer os processos que conduzirão a um software de alta qualidade, por meio da engenharia de software, com base em processos de inspeção de código e de testes,

combinando com as medições e com a comunicação dos envolvidos em cada fase do projeto (PRESSMAN, 2016).

### **Figura 6 – Abrangência e escopo da gestão da qualidade**



Fonte: elaborada pelo autor.

Temos que dois processos são indispensáveis na gestão da qualidade, com base nos propósitos da engenharia de software, sendo possível perceber o foco e a abrangência para o efetivo planejamento e as ações. A Figura 6 mostra uma extensão maior, quando se trata de garantir que os procedimentos e os padrões estão adequados em relação ao controle de resultados por meio de ensaios e testes.

O **controle de qualidade** auxilia as pessoas para que sejam atingidas as metas da qualidade, a partir de tarefas completas e consistentes. Deve prever uma atividade antes de liberar o código para testes, por da inspeção ou dos ensaios para tentar revelar e corrigir erros, tendo como propósito descobrir o máximo de problemas a cada passo do processo de construção, como: falha na manipulação de dados e na comunicação entre interfaces ou erro na lógica de processamento. Se algum problema for encontrado, ele é computado no respectivo repositório das medições, devendo ser analisado pela equipe para que sejam tomadas as devidas providências. Os dados registrados nesse repositório devem estar elucidados a fim de que facilite para a equipe ajustar o processo.

Em suma, o objetivo é **evitar que defeitos em software sejam entregues ao cliente**.

A **garantia da qualidade** pode ser entendida como “provar os métodos”, sendo um gerenciamento racional de projeto e ações de controle de qualidade para o desenvolvimento de um software de alta qualidade (PRESSMAN, 2016). São ações de auditoria que levam a um relatório que confirma a eficiência do controle da qualidade. Por meio da garantia da qualidade, os envolvidos tomarão ciência sobre a qualidade do produto com o objetivo de demonstrar que o **processo está funcionando o suficiente para conquistar a confiança**. Além disso, caso existam problemas, todos os recursos estão previstos para as devidas melhorias no processo, nos padrões, nos procedimentos e, caso necessário, nos treinamentos.

As características aqui apresentadas pela Engenharia de Software poderão ser associadas a outras iniciativas ou modelos de gerenciamento da qualidade, conforme as percepções de ganho em cada perfil de organização ou equipe de desenvolvimento.

## 2.3 Qualidade em Métodos Ágeis

Algumas características praticadas pelos Métodos Ágeis favorecem a qualidade do software em desenvolvimento, apesar de não possuírem nenhuma documentação explicitando como deve ser realizada a verificação ou validação do produto ou do processo, como a inspeção ou a revisão formal. A melhoria do processo e do produto é obtida por meio dos processos previstos no *Scrum* (SOMMERVILLE, 2018), de Revisão da *Sprint* e Avaliação da *Sprint*, quando se discute todo o processo, desde o que ocorreu na iteração até os problemas de qualidade. Ou seja, o tratamento dado para a questão da qualidade está no conjunto das discussões.

Caso ocorram problemas (erros, faltas ou falhas) no produto, no resultado dessa avaliação/revisão, o time ágil pode decidir pela solução (como reparar) para a próxima *Sprint*, como uma atenção maior no processo da Refatoração (o que melhoraria a qualidade do produto). É importante lembrar que esse caso irá demandar mais esforço da equipe na tarefa da Refatoração, e, consequentemente, o plano da próxima iteração terá uma diminuição em quantidade de histórias de usuário. Em outras palavras, a atividade de Refatoração, individualmente ou em pares<sup>4</sup>, demandará maior esforço de verificação, sendo mais criteriosa, antes do check-in<sup>5</sup> dos programas.

Um ponto muito favorável dos Métodos Ágeis relativo à qualidade, por ser um processo minucioso pelo fundamento da programação em pares, é relatado por Sommerville (2018):

A programação em pares leva um conhecimento profundo de um programa já que ambos os programadores têm de compreender o programa em detalhes para continuar o desenvolvimento. Essa profundidade de conhecimento, às vezes, é difícil de alcançar em outros processos de inspeção e, assim, a programação em pares pode encontrar defeitos que talvez não fossem descobertos nas inspeções formais. No entanto, as duas pessoas envolvidas não conseguem ser tão objetivas quanto uma equipe de inspeção externa, pois estão examinando o seu próprio trabalho. (SOMMERVILLE, 2018, p. 679)

Porém, o próprio Sommerville (2018) ressalta alguns cuidados nessa questão para que o desenvolvedor (mesmo em pares) possa relatar os defeitos existentes:

- Ao imaginar situações em que possam ocorrer erros, a equipe deverá diminuir a quantidade de histórias na iteração para resolvê-los; nessa dinâmica, as pessoas podem atribuir valor maior nas

<sup>4</sup> Uma prática comum em Métodos Ágeis é a programação em pares (conhecida como *extreme Programming*), em que duas pessoas se responsabilizam pelo desenvolvimento de um código-fonte.

<sup>5</sup> O check-in (entrada) é um passo do processo realizado na ferramenta de gerenciamento de configuração, em que o check-in é a entrada para o Repositório definitivo e o check-out é a saída. (PRESSMAN, 2016, p. 649).

pontuações de histórias de usuários, pois assim seria possível continuar mantendo um indicador em quantidade desenvolvida por iteração, ou seja, manteria a velocidade de desenvolvimento, o que não irá refletir a realidade, se comparada com o histórico do projeto ou diante de outros projetos.

- Entre os pares pode ocorrer um mal-entendido, quando ambos entenderam com equívoco a funcionalidade. Como a documentação pode ser escassa, ficaria apenas na percepção do requisito ou da regra de negócio, dificultando encontrar a solução. Nesses casos, em uma auditoria externa, o apontamento do problema se torna mais eficiente.
- Um colega pode omitir o erro do outro, relutando em criticar um parceiro, mesmo que esteja prejudicando o projeto, em função da relação pessoal que existe entre os pares.

Finalizando o estudo do conteúdo da Gestão da Qualidade, temos o conhecimento para avaliar as necessidades de um time de desenvolvimento de sistemas para decidir por algum procedimento que aprimore o processo de software, a fim de melhorar a qualidade do software desenvolvido.

Vale ressaltar que no início da prática de procedimentos da qualidade, seja de planejamento, auditoria, verificação ou validação, é certo que será um processo contínuo para alcançar uma meta mais arrojada em qualidade, pois gradativamente a equipe vai apresentar evolução na confecção de aplicativos melhores – basta lembrarmos sobre os níveis de maturidade organizacional.

Também vimos que podemos optar por um ou outro modelo de qualidade, independentemente do ciclo de vida que a equipe esteja praticando, pois os procedimentos são adaptáveis às técnicas ou metodologias da engenharia de software.



## Referências Bibliográficas

- BECK, Kent et al. **Manifesto para Desenvolvimento Ágil de Software.** [s.d.]. Disponível em: <https://agilemanifesto.org/iso/ptbr/manifesto.html>. Acesso em: 16 maio 2020.
- CHRISSIS, Mary Beth. **CMMI® for Development-Guidelines for Process Integration and Product Improvement.** Boston: Pearson Education, 2011.
- CMMI. Product Team. **CMMI® for Development:** improving processes for developing better products and services. 2010. Disponível em: [https://resources.sei.cmu.edu/asset\\_files/TechnicalReport/2010\\_005\\_001\\_15287.pdf](https://resources.sei.cmu.edu/asset_files/TechnicalReport/2010_005_001_15287.pdf). Acesso em: 7 jun. 2020.
- MASSARI, Vitor L. **Gestão Ágil de Produtos com Agile Think Business Framework:** guia para certificação exin agile scrum product owner. Rio de Janeiro: Brasport, 2018.
- MONTES, Eduardo. **Site Escritório de Projetos.** 2018. Disponível em: <https://escritoriodeprojetos.com.br/eduardo-montes-pmp>. Acesso em: 5 set. 2020.
- PMI. **Project Management Institute.** [s.d.]. Disponível em: <https://www.pmi.org/brasil>. Acesso em: 16 maio 2020.
- PRESSMAN, Roger S. **Engenharia de Software:** uma abordagem profissional. Porto Alegre: Amgh, 2016.
- SCRUM.ORG. **The home of scrum.** [s.d.]. Disponível em: <https://www.scrum.org>. Acesso em: 16 maio 2020.
- SEI. Software Engineering Institute. **The IDEAL Model.** 2009. Disponível em: [https://resources.sei.cmu.edu/asset\\_files/Presentation/2001\\_017\\_001\\_23277.pdf](https://resources.sei.cmu.edu/asset_files/Presentation/2001_017_001_23277.pdf). Acesso em: 9 jul. 2020.
- SOMMERVILLE, Ian. **Engenharia de Software.** São Paulo: Pearson Education do Brasil, 2018.

# **Administração da manutenção do software**

Autoria: Marco Ikuro Hisatomi

Leitura crítica: Valéria Cristina Gomes Leal

## **Objetivos**

- Conhecer a manutenção no ciclo de vida dos sistemas de software.
- Conhecer os tipos de manutenção de software.
- Saber como controlar as alterações do software.

# 1. Introdução

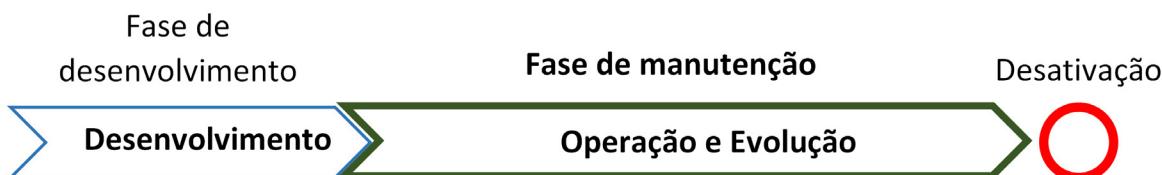
O objetivo deste tema é apresentar os principais aspectos da evolução do software, os fatores relevantes da administração de sua manutenção e as considerações sobre os tipos de manutenção.

Tradicionalmente, temos uma visão de que o software é desenvolvido por um determinado período e, depois que entra em operação, fica eternamente à disposição dos usuários, funcionando sem alterações e sem bugs até que um dia tenhamos que substitui-lo, porque houve mudança da plataforma operacional ou ele ficou totalmente desnecessário. No entanto, é muito maior a quantidade de intervenções nos sistemas durante a operação do que se imagina, devido ao volume crescente das exigências dos negócios. Sommerville (2018) descreve:

que os sistemas de software precisam se adaptar e evoluir para que continuem sendo úteis e que a modificação e a evolução de um software devem ser consideradas partes integrantes da engenharia de software.  
(SOMMERVILLE, 2018, p. 231)

Para Taentzer *et al.* (2019, p. 10), “a evolução é um fenômeno natural no ciclo de vida de sistemas de software de acordo com diversas razões para mudanças”.

**Figura 1 – A manutenção em evidência no ciclo de vida do software**



Fonte: adaptada de Tripathy (2015, p. 84).

A estratégia está muito mais no foco em mudança do que um plano que será mantido sem modificações por vários anos. A representação da abrangência da fase de manutenção em um determinado tempo do

ciclo de vida<sup>1</sup> do software pode ser compreendida pela Figura 1, que ilustra a fase do **desenvolvimento** do sistema e da **manutenção** (que representa a operação e a evolução) até a **desativação**.

Diante desse cenário, é necessário conhecer como fazer uma boa administração dos recursos para manter o sistema atendendo aos negócios, acompanhando as tecnologias e servindo às necessidades dos usuários finais, pois ele deve continuar sendo importante para a organização. Também é necessário conhecer os diferentes tipos de manutenção de software e decidir quanto aos investimentos para as efetivas modificações, a fim de avaliar se ele deve ser mantido, desativado, reprojeto ou substituído.

## ► 2. Operação e Evolução de Software

Para iniciar o entendimento da evolução, da forma de que os usuários realmente necessitam e desejam, vamos entender se todos os softwares podem sofrer modificações e quais os impactos dessas mudanças. Assim sendo, devemos entender como se dá a evolução do software e de seus aspectos fundamentais.

Na fase de concepção e construção de um sistema, até mesmo após a sua instalação, em plena fase de operação, é normal esperar que o sistema se estabilize e que perdure por muitos e muitos anos, oferecendo os benefícios esperados, conforme os requisitos “desenhados” nas primeiras atividades do projeto de desenvolvimento. A expectativa de uso por longas décadas é natural, porque, de acordo com Sommerville (2018), a organização espera que tenha um retorno sobre o investimento feito no seu desenvolvimento por meio da operação

---

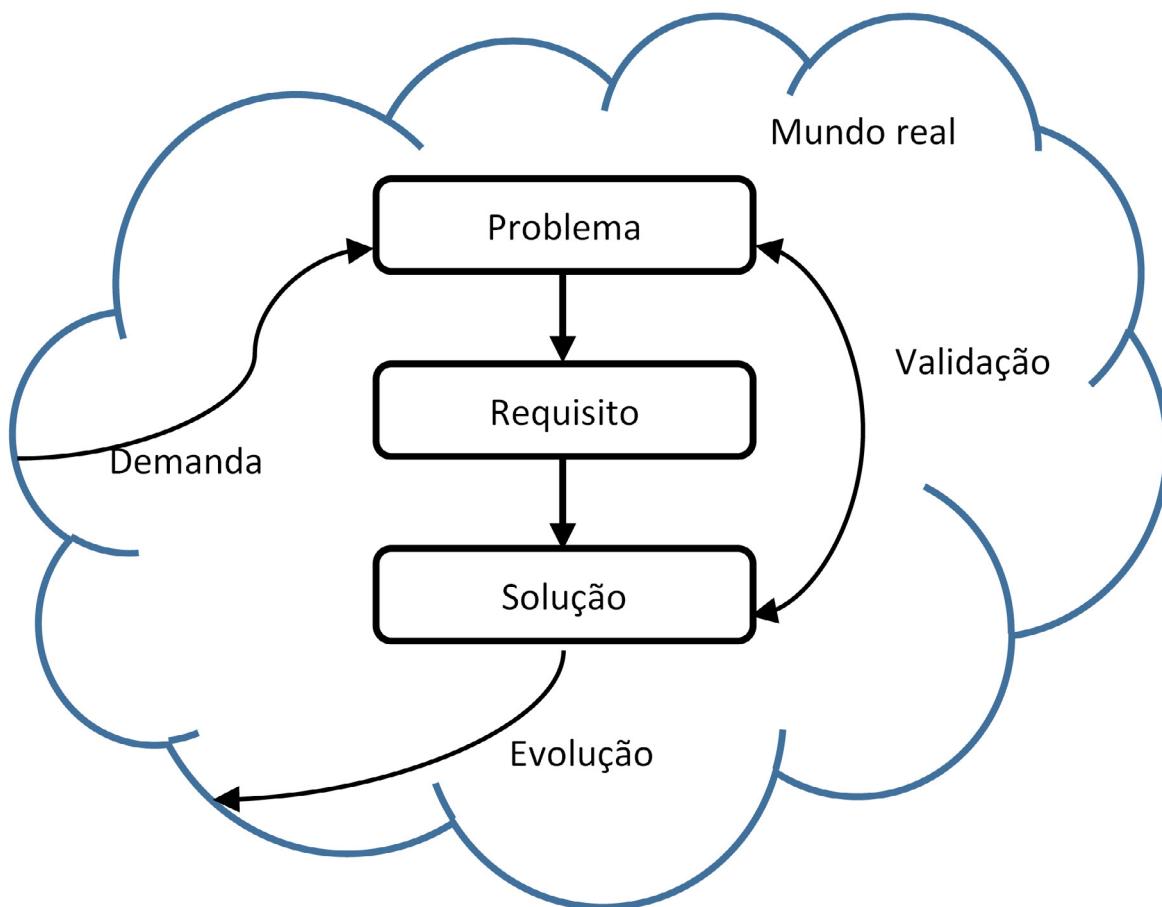
<sup>1</sup> De acordo com a NBR ISO/IEC 12207:1998 (ABNT, 1998), o ciclo de vida é a “Estrutura contendo processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software, abrangendo a vida do sistema, desde a definição de seus requisitos até o término de seu uso.”

do software, que viabiliza seus processos, seus procedimentos e suas regras de negócio essenciais para recuperar os altos valores investidos.

No mundo corporativo, a concorrência, em especial no ramo mercantil da atualidade, está tão acirrada que pequenos critérios de negociação ou detalhes na especificação de um produto são fundamentais. Esses requisitos podem estar no sistema, devendo a cada demanda (por solicitação do cliente ou por estratégia interna) um problema ser formalizado para que o software do sistema possa dar a solução.

Essas demandas são comuns devido às expectativas pessoais dos usuários ou a mudanças nas organizações. De acordo com Sommerville (2018), podem ocorrer ainda para realizar os ajustes das falhas encontradas em execuções operacionais, para adaptar o sistema às atualizações tecnológicas de software ou hardware e até mesmo para aumentar o desempenho ou melhorar características não funcionais. Como ilustrado na Figura 2, o sistema sofre interferências do mundo real e passa a sofrer adaptações até que decidam pela sua desativação.

**Figura 2 – Interação de demanda e evolução do software**



Fonte: adaptada de Tripathy (2015, p. 49).

Como as demandas acabam sendo representadas por Requisitos de software ou por Histórias de Usuário, o processo de implementação dessa modificação no software legado pode então ser definido para ser implementado utilizando as disciplinas da Engenharia de Software, de acordo com a metodologia adotada pela equipe ou pelo time de desenvolvimento de sistemas da organização. Contudo, a administração do sistema legado ou o gestor de projetos de evolução/manutenção de software, juntamente com o Time de Desenvolvimento/Manutenção (TDM<sup>2</sup>), deve dar uma tratativa particular. Sommerville (2018) comprehende que

A evolução de um software é particularmente cara nos sistemas corporativos em que cada um dos sistemas faz parte de um “sistema

<sup>2</sup> TDM é uma sigla criada pelo autor para facilitar a escrita neste material.

de sistemas” mais amplo. Nesses casos, não é possível considerar as mudanças em apenas um sistema; também é preciso examinar como elas afetam o sistema de sistemas mais amplo. Modificar o sistema pode significar que outros sistemas em seu ambiente também precisam evoluir para lidar com a mudança. (SOMMERVILLE, 2018, p. 232)

Para esse fenômeno, no qual uma modificação do software tem que ser gerenciada em um ambiente de múltiplos sistemas interdependentes, Hopkins e Jenkins (2008 *apud* Sommerville 2018) usam o termo **desenvolvimento de software brownfield**. Nele, compreender e analisar o impacto de uma mudança devem ser realizados no próprio sistema e nos demais sistemas que estão no mesmo ambiente operacional.

Aproveitando a preocupação quanto aos impactos, além de providenciar soluções das solicitações de evolução, o TDM deve assegurar a qualidade original da aplicação e, se possível, melhorar conforme definido no padrão ISO 25000 (ISO, 2014)<sup>3</sup>, independentemente do tipo da aplicação, seja pelo aspecto da manutenção, usabilidade, consistência ou correção, como argumentam Taentzer *et al.* (2019, p. 10). Aspectos da qualidade em evoluções serão descritos para que o administrador dessas evoluções nos sistemas da organização possa concebê-los em suas atividades de rotina, quando das solicitações de modificações no software.

Taentzer *et al.* (2019) elencam os seguintes fatores a serem considerados em qualidade no processo de evolução/manutenção do software:

- Qualidade funcional: corretude, consistência, confiabilidade e usabilidade.
- Qualidade não funcional: performance, manutenibilidade e segurança.

<sup>3</sup> Guia ISO da Engenharia de Sistemas e Software – Requisitos e Avaliação da Qualidade em Sistemas e Software. ISO/IEC 25000:2014 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) (ISO, 2014).

Vejamos uma situação que pode causar a perda de qualidade na fase de manutenção. Considerando que os reparos emergenciais são aqueles que nem sempre são analisados de forma completa, ou seja, fazem o exequível, Sommerville (2018) diz para atender a situação do momento, mesmo sabendo que essa implementação diminuirá a vida útil do sistema. Para recuperar a manutenibilidade do sistema, em condições administradas, o novo código deve passar pela Refatoração, diminuindo, assim, a degradação do software.

Para refletir sobre os requisitos emergentes, que complementam essa necessidade de estar preparado para atualizações constantes e em períodos cada vez menores, Pressman (2016) traz:

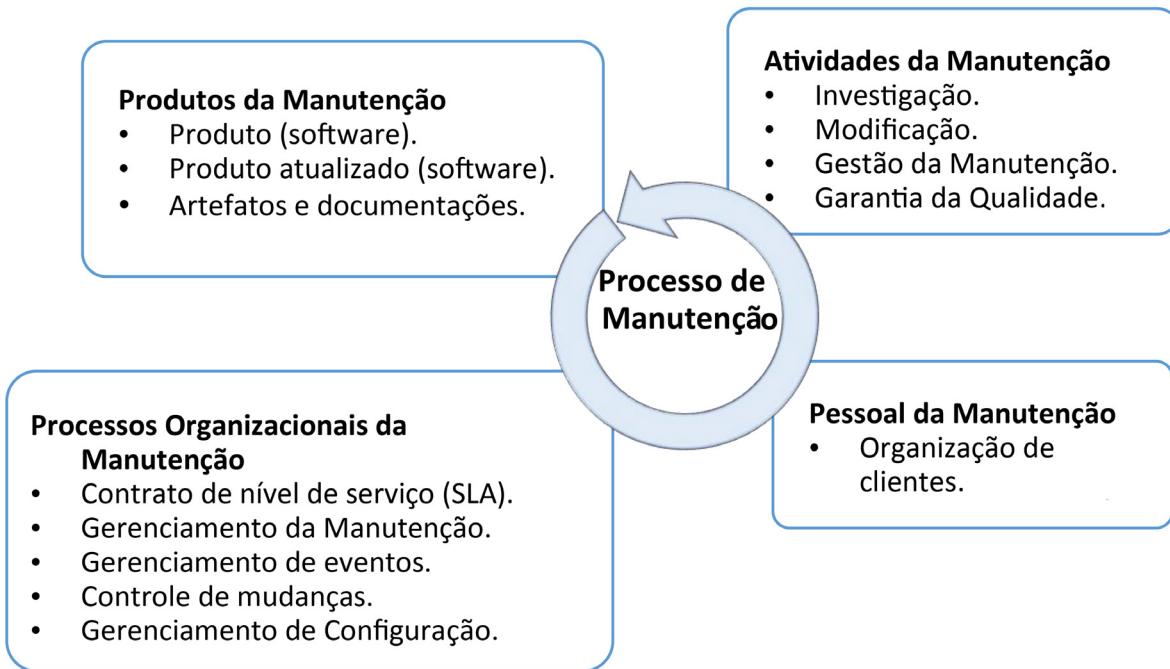
À medida que os sistemas se tornam mais complexos, até mesmo uma tentativa rudimentar de definir requisitos abrangentes está destinada ao fracasso. Uma definição de objetivos globais pode ser possível, um esboço dos objetivos intermediários pode ser alcançado, mas requisitos estáveis... sem chance! Os requisitos surgirão conforme todos os envolvidos na engenharia e construção de um sistema complexo aprenderem mais sobre esse sistema, sobre o ambiente no qual ele reside e sobre os usuários que vão interagir. (PRESSMAN, 2016, p. 846)

### ► 3. Processos da Manutenção do Software

As atividades envolvidas para a efetiva manutenção dos sistemas são semelhantes às utilizadas na fase de desenvolvimento de sistemas. Porém, o foco e as habilidades devem estar na adaptação e na correção dos produtos de software, transformando as solicitações de modificações de alto nível em código-fonte e mantendo a qualidade do software com as respectivas funcionalidades originais do ambiente operacional e as documentações atualizadas.

Vejamos as atividades envolvidas na manutenção, denominadas de Processos de Manutenção e divididas em quatro dimensões, na Figura 3. É importante destacar que existe uma correlação entre elas, e cada atividade requer habilidades, ferramentas e técnicas específicas.

**Figura 3 – Dos processos da manutenção**



Fonte: adaptada de Tripathy (2015, p. 38).

## Produtos da Manutenção

Para essa dimensão, devem ser mantidas, para cada elemento, as seguintes características: tamanho, idade, tipo de aplicação, composição e qualidade. Elas afetam diretamente no desempenho das atividades durante as modificações dos sistemas.

Entende-se por Produto (software) a aplicação que foi entregue juntamente com os demais artefatos correspondentes (arquivo de configuração de instalação, manuais de operação, documentações de design, entre outros). Já o Produto Atualizado (software) é a representação de uma versão do software composta por um arranjo, chamado de linha base (*baseline*), com todos os artefatos correspondentes. Por sua vez, os Artefatos e as

Documentações podem ser entendidos como todos os itens gerados ou utilizados na confecção da manutenção do software, como: projetos (designs), documento textual ou gráfico, componentes em código-fonte, evidências usadas em verificações ou validações, entre outros.

## Atividades da Manutenção

É a dimensão que corresponde ao desempenho de habilidades técnicas para investigar as necessidades solicitadas, sendo para correção, melhoria ou compulsória por lei/segurança. Ainda, segundo Tripathy (2015), verifica os impactos nos demais componentes, módulos ou sistemas dependentes da parte que será modificada.

A atividade de modificação está prevista nessa dimensão, na qual os especialistas utilizarão das técnicas adequadas (ou previstas no processo de software) para a construção/atualização de código-fonte, janelas de interação, interfaces com outros sistemas, verificação e testes unitários (de acordo com o previsto em processos da qualidade). Aqui vale lembrar que a manutenibilidade, um dos fatores da qualidade, segundo Pressman (2016, p. 797), “é um indicativo qualitativo da facilidade de corrigir, adaptar ou melhorar o software”.

Quanto ao gerenciamento previsto em Atividades da Manutenção, é esperado que o desenvolvedor responsável pela modificação efetue o devido controle de configuração de software (em conjunto com a dimensão dos Processos Organizacionais da Manutenção), criando uma linha base do software com todos os artefatos atualizados na atividade de modificação. Em garantia de qualidade, é desejado, segundo Tripathy (2015), que as modificações executadas em um sistema não danifiquem sua integridade, ficando a sugestão, nesse passo, do uso do teste de regressão, que é uma das técnicas apropriadas em garantia de qualidade para sistemas em modificação.

## **Processos Organizacionais da Manutenção**

É a dimensão na qual os gestores devem dispensar maior esforço, pois a complexidade dos sistemas em manutenção obrigará o uso de ferramentas e metodologias mais precisas no gerenciamento de cada área.

Antes de entendermos as atribuições para a área desse processo, vamos entender o que a organização, juntamente com a TDM, deve providenciar e estabelecer para que o processo seja administrado de forma clara e objetiva, a fim de garantir que custos, tempo e resultados sejam adequados às estratégias organizacionais. Devemos entender, entre as várias filosofias ou modelos de desenvolvimento/manutenção de software, qual foi utilizado para o sistema original, e conhecer em qual ou quais tecnologias foram construídos os programas do sistema original. Esse entendimento é necessário para adotarmos ou definirmos os procedimentos mais apropriados para a fase de manutenção do software, como: um guia de o que fazer em atividades de manutenção; as técnicas que serão úteis na implementação da mudança e os métodos que deverão seguir para cada situação da análise, implementação da mudança, garantia da qualidade e gestão de configuração; e os prazos para a realização das mudanças.

Para garantir o prazo de entrega das modificações solicitadas ao TDM, mesmo que seja uma equipe interna da organização, deve haver regras e contratos de nível de serviço (*Service-Level Agreement (SLA)*) com metas para solucionar a solicitação de mudança.

Tendo em vista que várias solicitações podem ser atendidas simultaneamente e que vários softwares estão passando por manutenções e evoluções, é certo que os recursos sofrem concorrências. Assim, é desejável um gerenciamento da manutenção para reger esse leque de atividades em paralelo, normalmente exercido por um gerente sênior. O gerenciamento da manutenção deve prever o controle de fluxo das solicitações de mudanças, classificando e determinando prioridades para cada uma, de acordo com as estratégias da organização ou do cliente. Uma

das fontes para determinar a prioridade das mudanças pode ser o relatório de defeitos gerado por *stakeholders*, que, ao final, serão consolidados no relatório de investigações. Oportunamente, Sommerville (2018, p. 709) cita que “o gerenciamento de mudanças é o processo de analisar os custos e os benefícios das mudanças propostas, aprovando as que têm bom custo-benefício e controlando quais componentes do sistema foram modificados.”

Por sua vez, o gerenciamento de configuração tem como propósito inter-relacionar todos os itens de software em uma determinada versão para compor uma linha base, considerada uma ferramenta fundamental para o controle de versão. A Figura 4 demonstra as várias versões de um produto da manutenção disponibilizadas para os usuários, o qual é controlado até a desativação de seu sistema.

**Figura 4 – Visão das versões do gerenciamento de configuração**



Fonte: elaborada pelo autor.

Se a cada versão uma *baseline* é guardada, isso implica em possibilidades de realizar os testes de regressão com os cenários e os casos de testes para garantir a confiabilidade do sistema após as modificações implementadas. Assim, é possível dar maior credibilidade ao processo da manutenção e ao gerenciamento de manutenção.

## Pessoal da Manutenção

Para concluir, vamos compreender a finalidade da dimensão **Pessoal da Manutenção** no processo de manutenção, conhecendo quais são as responsabilidades das pessoas envolvidas em uma modificação no sistema. Nesse sentido, há o grupo de pessoas da organização que administra a manutenção do software, no qual se encontra o TDM (engenheiros de software, programadores, testadores, entre outros), e o grupo da organização do cliente, que usufrui do software com os seus usuários e clientes.

Para o melhor direcionamento das solicitações de mudanças, apoiando-se no processo de gerenciamento de manutenção, podem ser mantidas informações de Pessoal da Manutenção. Isso permite uma análise de impacto mais adequada para a priorização das solicitações, tais como: tamanho da base de clientes, variabilidade (por segmento de negócio ou utilização efetiva de módulos do sistema), clientes que financiam o processo de manutenção, entre outras, que podem ser envolvidas no projeto para aumentar a precisão das decisões.

Aspectos importantes para tomar decisões sobre as solicitações de mudanças podem ser adotados pela TDM, o que colabora para diminuir prejuízos em esforços e custos, caso uma modificação seja aplicada em um momento inadequado para os clientes e o usuário e na visão estratégica da organização mantenedora do software. Na sugestão abordada por Sommerville (2018, p. 712), para a classificação das solicitações de mudanças:

poderia ter o nome de “grupo de desenvolvimento do produto”, responsável por tomar decisões a respeito de como um sistema de software deve evoluir. Esse grupo deve analisar e aprovar todas as solicitações de mudanças a menos que as alterações envolvam simples correções de erros menores em telas, páginas web ou documentos. Esses pequenos pedidos devem ser passados para implementação imediata.

Inversamente, a decisão pode ser tomada **analisando quais as consequências, caso a solicitação de mudança não seja efetuada**.

Algumas solicitações, na visão de Sommerville (2018), não têm impacto negativo para o negócio nem ganho substancial para o sistema, como a fonte da letra estar diferente do esperado no cabeçalho da consulta na lista de produtos do estoque, ou a somatória do relatório, que deveria estar alinhada à direita. Esses são casos que podem ser classificados com prioridade baixa.

Quando a característica é mais técnica e específica da equipe e do produto analisado, caso seja verificada uma forte tendência em aumentar as chances de adicionar novos defeitos, por se tratar de uma mudança que afeta muitos componentes, e o tempo de desenvolvimento for longo, para Sommerville (2018), é melhor colocar baixa prioridade e tentar subdividir em várias solicitações para diminuir os riscos de insucesso.

### 3.1 Classificação dos Tipos de Manutenções

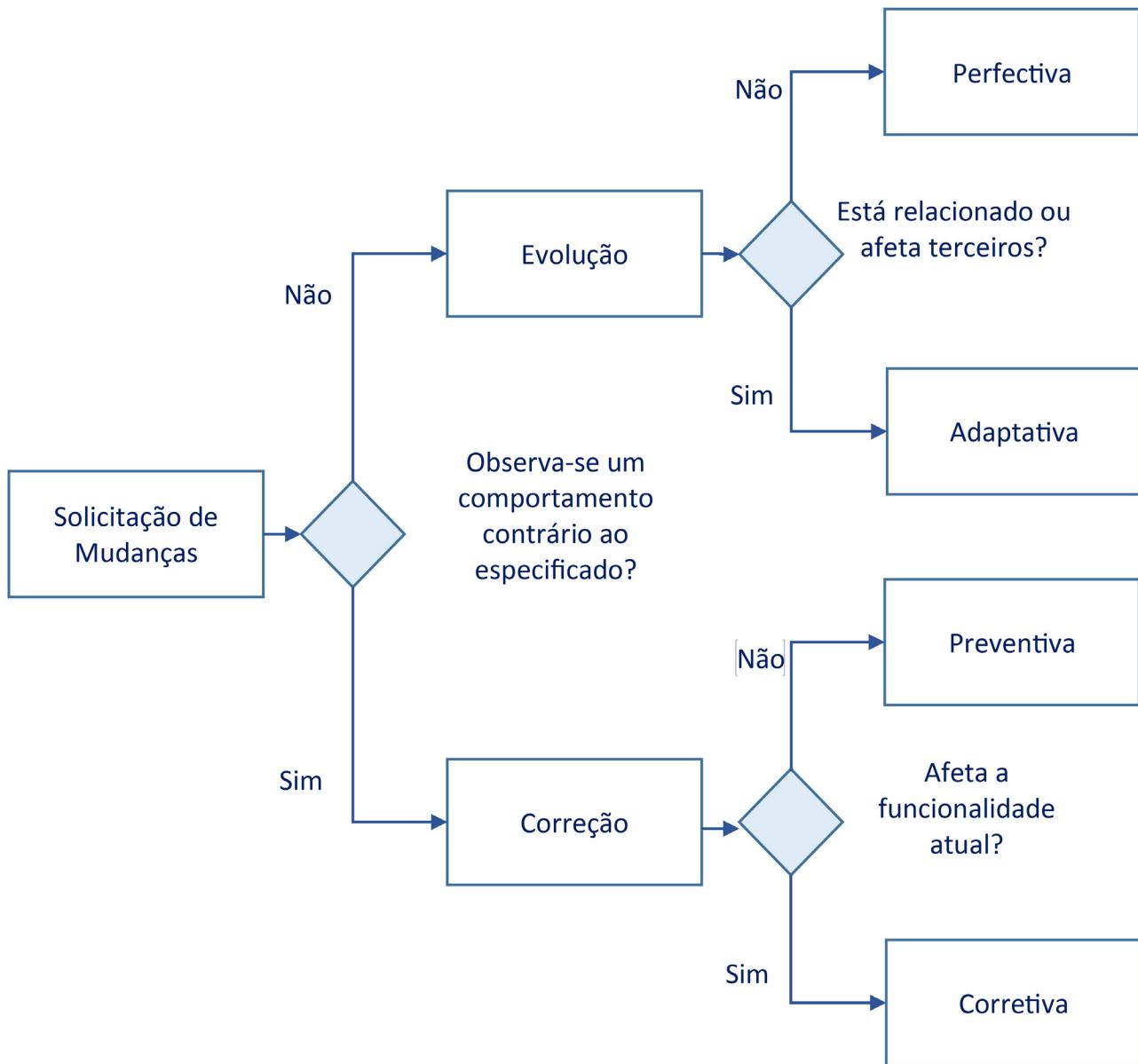
Dando continuidade ao procedimento de análise das manutenções, é adequado considerar o tipo de manutenção para priorizar de acordo com as características de resultado e de satisfação por parte do cliente. Destinada ao melhor planejamento das atividades de manutenção do software, a ISO/IEC 14764:2006 (ISO, 2006 p. 3) estabelece um padrão para o gerenciamento dessas atividades, não dando privilégio para uma determinada aplicação nem mesmo distinguindo por tamanho, complexidade ou criticidade.

O fluxo ilustrado na Figura 5 define os tipos de manutenções a partir da análise de uma formalização da solicitação da mudança. Inicialmente, essa solicitação se divide em dois escopos:

- Correção: quando se trata de uma descrição contrária ao que havia sido especificado e implementado.

- Evolução: quando se trata de uma novidade ainda não implementada no sistema.

**Figura 5 – Como classificar os tipos de manutenções**



Fonte: adaptada de ISO/IEC 14764 (ISO, 2006, p. 4).

Para o escopo da Evolução, temos:

- Adaptativa: quando a modificação afeta terceiros; nesse caso, deve evoluir por conta de uma atualização compulsória no ambiente operacional relacionada à tecnologia ou às interfaces com outros softwares.

- Perfectiva: melhoria em usabilidade, performance ou manutenibilidade, tanto pelo usuário, que necessita incluir/alterar/eliminar funcionalidades, quanto pelo time de desenvolvimento, que percebe a necessidade de refatoração ou recodificação para facilitar nas futuras modificações.

Agora, para o escopo da Correção, temos:

- Corretiva: quando a falha já ocorreu, sendo percebida na maioria das vezes pelo usuário. Portanto, é reativa a uma situação não desejada, podendo ser erro na codificação ou no entendimento dos requisitos do cliente/usuário.
- Preventiva: quando o time percebe, por isso de forma proativa, potenciais falhas por erro na construção ou por compreender melhor a operacionalização pelos usuários; essa manutenção visa prevenir possíveis bugs no futuro.

Embora o assunto seja extenso e complexo, com a abordagem elaborada neste tema, podemos perceber que o gerenciamento das manutenções são atividades correlatas à evolução do software ou à correção, sendo aplicáveis tanto para atender às novas necessidades quanto para prevenção de futuros problemas por situação não esperada, mas que demanda modificações imediatas.

Para concluir, vimos que a administração da manutenção do software pode ser controlada por um processo particular, em que várias técnicas e ferramentas se complementam para garantir qualidade e produtividade na execução das implementações solicitadas, seja pelo usuário/cliente ou pelo time interno.



## Referências Bibliográficas

- ABNT. Associação Brasileira de Normas Técnicas. **ISO/IEC 12207**: Tecnologia da Informação–Processos de Ciclo de Vida de Software. Rio de Janeiro: ABNT, 1998.
- ISO. International Standard. **ISO/IEC 14764**: Software Engineering — Software Life Cycle Processes — Maintenance. 2006. Disponível em: [http://mireilleblayfornarino.i3s.unice.fr/lib/exe/fetch.php?media=teaching:reverse:10.1109\\_ieeestd.2006.235774.pdf](http://mireilleblayfornarino.i3s.unice.fr/lib/exe/fetch.php?media=teaching:reverse:10.1109_ieeestd.2006.235774.pdf). Acesso em: 11 jun. 2020.
- ISO. International Standard. **ISO/IEC 25000**: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE. 2014. Disponível em: <https://www.iso.org/standard/64764.html>. Acesso em: 11 jun. 2020.
- PRESSMAN, R. S. **Engenharia de Software**: uma abordagem profissional. Porto Alegre: Amgh, 2016.
- SOMMERVILLE, I. **Engenharia de Software**. São Paulo: Pearson Education do Brasil, 2018.
- TAENTZER, G. *et al.* (ed.) **Managed Software Evolution–The Nature of Software Evolution**. 2019. Disponível em: <https://doi.org/10.1007/978-3-030-13499-0>. Acesso em: 11 jun. 2020.
- TRIPATHY, P.; NAIK, K. **Software Evolution and Maintenance**: a Practitioner's Approach. New Jersey: John Wiley & Sons, 2015.



---

**BONS ESTUDOS!**