WBA0450_v1.0

Gerenciamento Ágil dos Sistemas

Administração da Manutenção do *Software*

Bloco 1

Marco Ikuro Hisatomi

A qualidade na manutenção do software

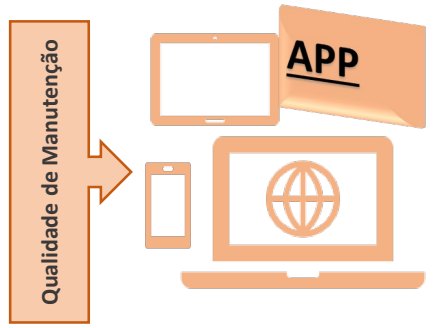
Ao implementar uma modificação no software, além de entregar de acordo com as especificações da solicitação de mudança (de correção ou de melhoria), deve garantir a qualidade conforme **aspectos** elencado por Taentzer *et al*. (2019) no processo de evolução do software:

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

Para você saber administrar a manutenção do software deve conhecer estes fatores!

*Sobre a **manutenibilidade** está disponível no Bloco 2.

Figura 1 – Qualidade de manutenção

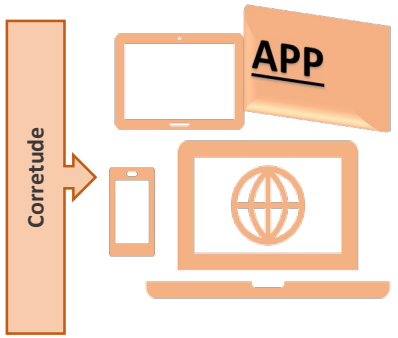


Fonte: elaborada pelo autor

Corretude em manutenção do software

- Relacionado ao requisito funcional, em que o software continua executando as funcionalidades e dando resultados de acordo com o que foi especificado e implementado inicialmente.
- Certificar-se da corretude comparando os resultados (em comportamentos semelhantes) da versão original e da **versão modificada**.
- Os testes automatizados podem ser úteis para manter e acrescentar os casos de testes em conjunto com os critérios de testes, visando a corretude.

Figura 2 – Corretude

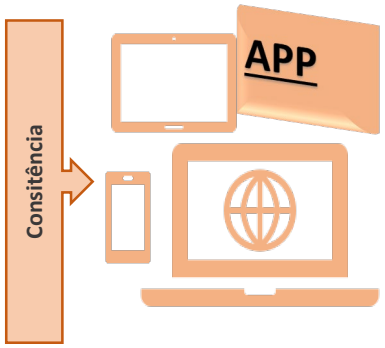


Fonte: elaborada pelo autor.

Consistência em manutenção do software

- Ter consistência de escopo: todos os elementos/componentes entre si. O software apresenta comportamento conforme esperado (externa).
- Estar consistente em requisitos especificados com o implementado (interna). Quando da criação do programa, podem estar consistentes, mas, ao **longo das modificações** podem ocorrer erros ou faltas de implementação (que provocam falhas de operação ou na integração com outros sistemas).
- Um teste unitário de integração continua (diariamente) pode verificar se está consistente.

Figura 3 – Consistência

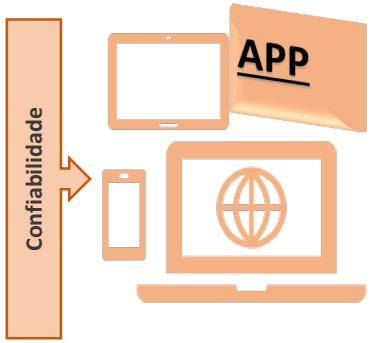


Fonte: elaborada pelo autor.

Confiabilidade em manutenção do software

- Neste fator, devemos conferir que, ao longo do tempo (no dia, na semana ou em período maior), o software apresenta o mesmo resultado ou comportamento.
- Independentemente do momento que usar o sistema, por ocasião de dados que foram **acrescentados ou modificados** em outros pontos do sistema, o usuário poderá confiar nos resultados apresentados. Inclusive, prevendo o adequado controle de acesso/alteração/eliminação de informações.
- Os testes automatizados podem conferir se o software apresenta bom nível de confiança.

Figura 4 – Confiabilidade

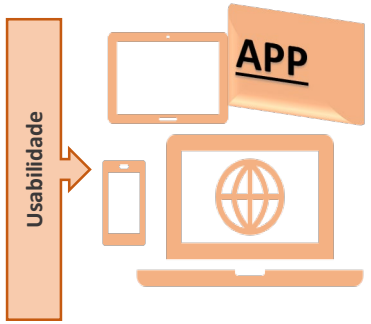


Fonte: elaborada pelo autor.

Usabilidade em manutenção do software

- **Após a manutenção efetuada**, o sistema continua com, ou possui, facilidade de uso, favorecendo o aprendizado rápido porque a disposição e comportamento do sistema está adaptado ao comportamento do usuário, de acordo com os negócios e/ou perfil de mercado.
- A percepção do usuário poderá conferir se software está com boa usabilidade porque é agradável, eficiente e, quando apresenta alguma falha, facilmente se recupera.
- O dono do sistema tem uma participação importante para validar se o sistema continua com a usabilidade aceitável.

Figura 5 – Usabilidade

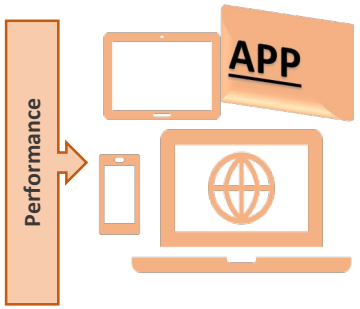


Fonte: elaborada pelo autor.

Performance em manutenção do software

- O quão rápido é para executar uma funcionalidade pelo sistema após a modificação. Ex.: por que ficou tão lento agora, se a mudança requerida nem foi para este relatório?
- Este fator é complexo, pois **nem sempre depende somente da aplicação**, o resultado da performance pode estar relacionado ao ambiente operacional
- Vale se certificar: a instalação/configuração (*setup*) está de acordo com os demais componentes do ambiente operacional e dos sistemas integrados.
- Um indicador deve ser usado para compreender as variantes e a possível necessidade de reprojeto (*redesign*).

Figura 6 – Performance

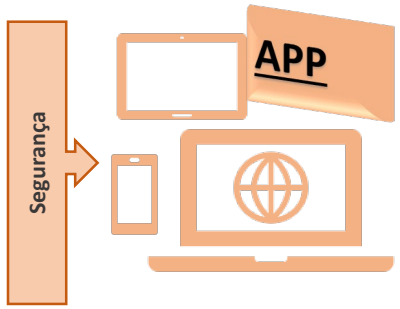


Fonte: elaborada pelo autor.

Segurança em manutenção do software

- Refere-se principalmente ao acesso não autorizado de usuários ou um ambiente que ameaça a operação (ou as informações) do sistema, tanto na fase de desenvolvimento como na fase de manutenção), pois, as tecnologias se apresentam em diferentes formas, compulsória e invasiva
- Segundo Taentzer *et al*. (2019, p. 208) “Preservar a segurança em sistemas de vida longa, requer um **suporte continuo e sistemático**, baseado em novos conhecimentos e evolução do software.”
- Portanto, convém estar atualizado sobre Modelo de Manutenção de Segurança (*Security Maintenance Model*) e demais conhecimentos relacionados.

Figura 7 – Segurança



Fonte: elaborada pelo autor.

Administração da Manutenção do *Software*

Bloco 2

Marco Ikuro Hisatomi

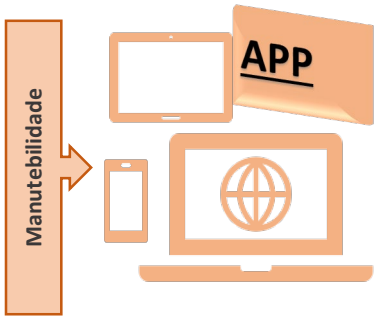
Manutebilidade em manutenção do software

- Parece obvio falar que a manutebilidade de um software seja um fator de qualidade no próprio processo de manutenção.
- Este é o fator importante para que possam aumentar a velocidade de resposta para correções ou para evoluções no sistema legado, o que reduz os custos em todo processo e, ainda, proporciona o adiamento do descarte

Fácil → **Analisar** **Modificar** **Testar**

- Seja para corrigir defeitos, entregar novos requisitos, adaptar ao ambiente operacional, entre outros.
- Ao adotar técnicas de entregas sucessivas por iteração aumentam as chances de manutenibilidade controlada.

Figura 8 – Manutebilidade

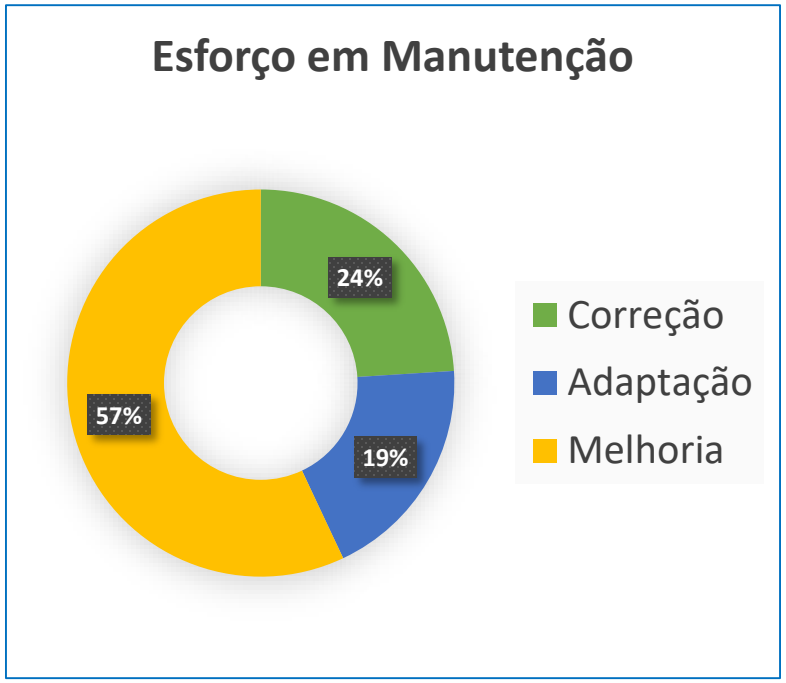


Fonte: elaborada pelo autor.

Manutebilidade em manutenção do software

- A **Refatoração** como recurso para mudar a realidade, apresentada por Sommerville (2018) para ser utilizado em qualquer modelo, preventivamente.
- Uma melhoria percebida pelo próprio Time para melhorar a estrutura, reduzindo a complexidade.
- Eliminação de “mau cheiros”: expressões condicionais extensas, métodos duplicados, métodos similares em subclasses e entre outros.

Figura 9 – Esforço em manutenção



Fonte: adaptada de Sommerville (2018, p. 247).

Administração da Manutenção do *Software*

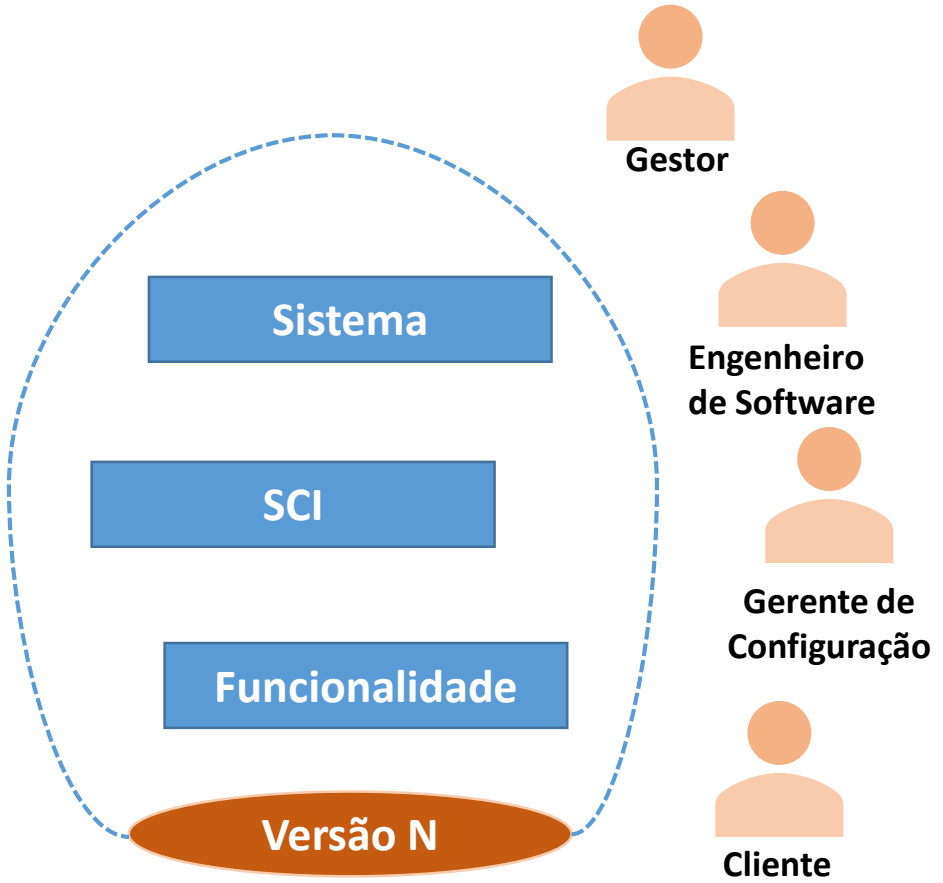
Bloco 3

Marco Ikuro Hisatomi

Processo de manutenção visando a qualidade

- Como garantir uma versão estável do Processo de Manutenção?
- Os responsáveis pela versão modificada são: gestor, engenheiros de software, gerente de configuração e cliente.
- Metodologia que facilita a criação, a implementação e a validação, a cada versão.
- Ferramenta com suporte ao Gerenciamento de Configuração dos SCI (*software configuration items*)

Figura 10 – Processo proposto

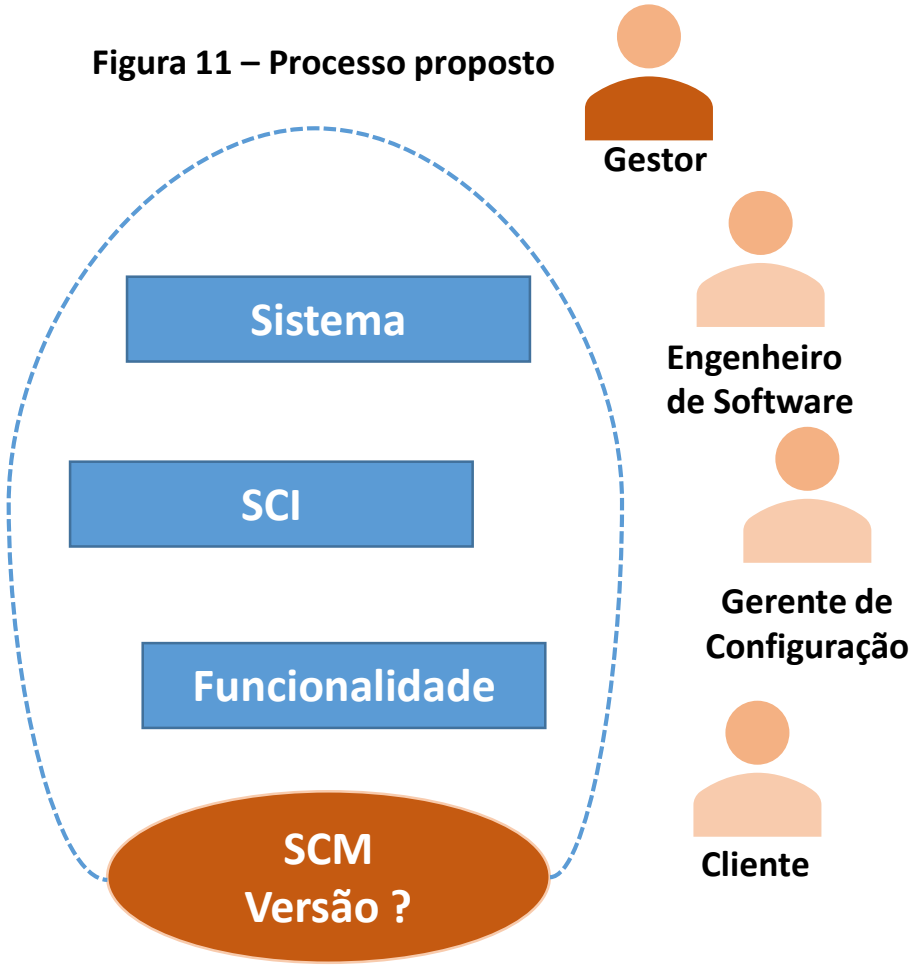


Fonte: elaborada pelo autor.

Processo de manutenção gerenciada

- O gestor, segundo Sommerville (2016), monitora o progresso do desenvolvimento.
- Monitora todos os SCI problemáticos e coordena as ações para a solução.
- Sistematiza a *build* de integração diária.
- Administra a comunicação entre os envolvidos e provê os recursos ao processo de manutenção.
- Audita por meio do SCM (*Software Configuration Management*).

Figura 11 – Processo proposto

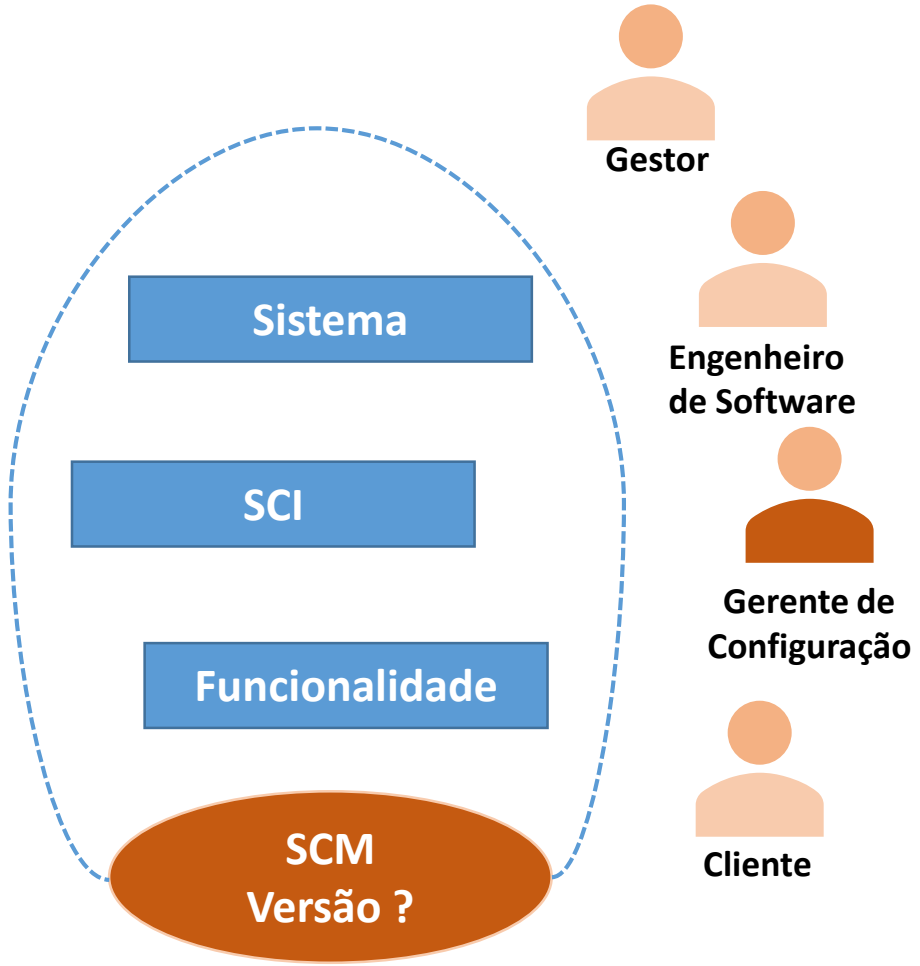


Fonte: elaborada pelo autor.

Processo de manutenção gerenciada

- O gerente de configuração, segundo Sommerville (2016), garante que procedimentos e políticas sejam seguidos para criar, alterar e testar o código.
- Monitora as avaliações e autorizações das solicitações do SCM.
- Coordena as movimentações no SCM e garante o rastreamento das versões dos SCI .

Figura 12 – Processo proposto

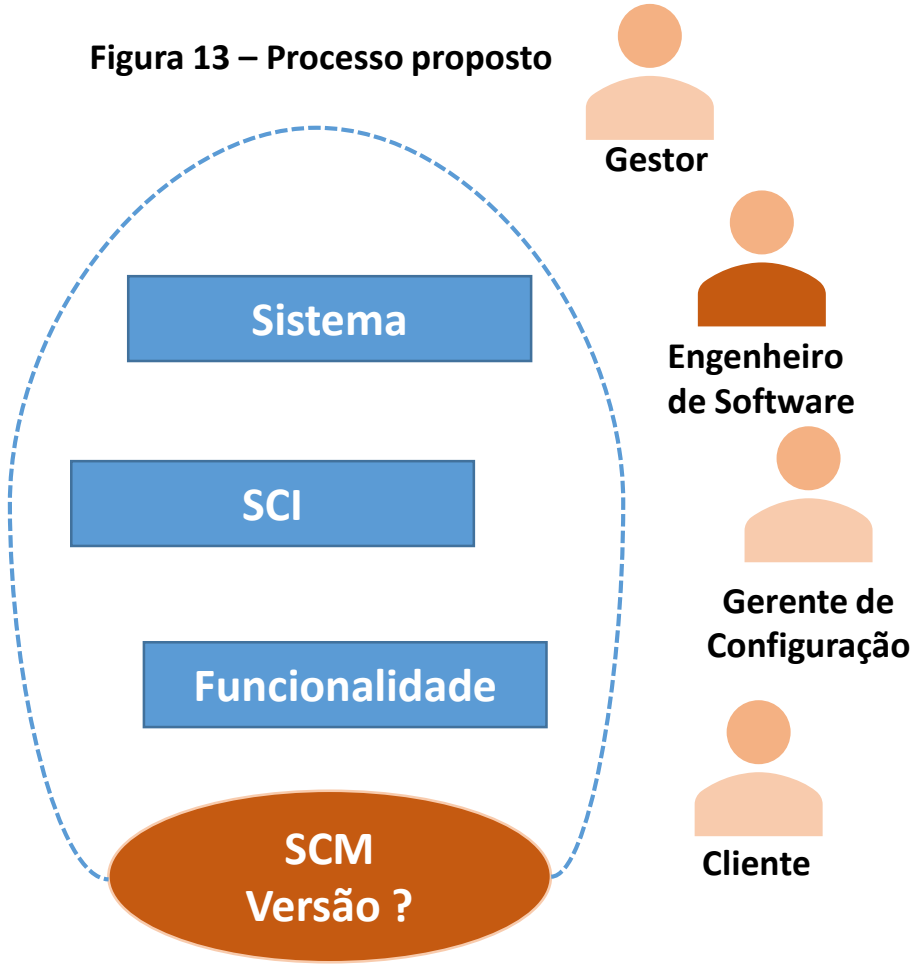


Fonte: elaborada pelo autor.

Processo de manutenção gerenciada

- O engenheiro de software criam artefatos consistentes (SOMMERVILLE, 2016).
- Mantém a comunicação constante quanto às necessidades e conclusão de tarefas.
- Garantem a propagação das modificações utilizando a ferramenta SCM.
- Monitoram componentes submetidos a alterações simultâneas, resolução de conflitos com mesclagem de alterações.

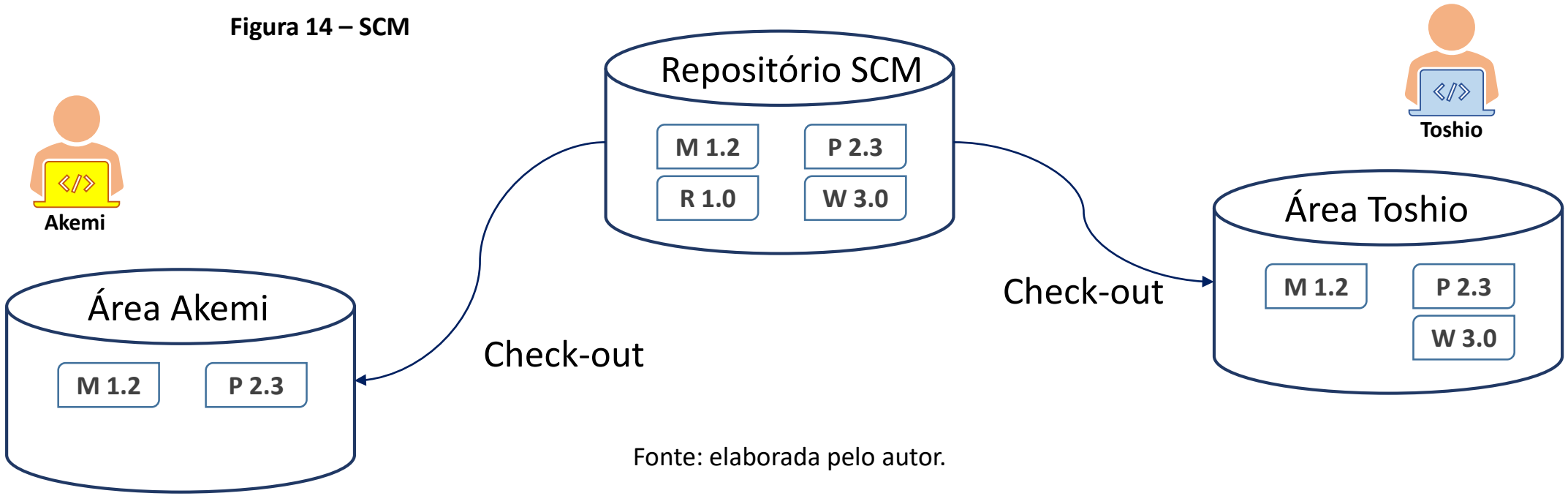
Figura 13 – Processo proposto



Fonte: elaborada pelo autor.

Processo de manutenção gerenciada (ilustrando SCM)

Figura 14 – SCM

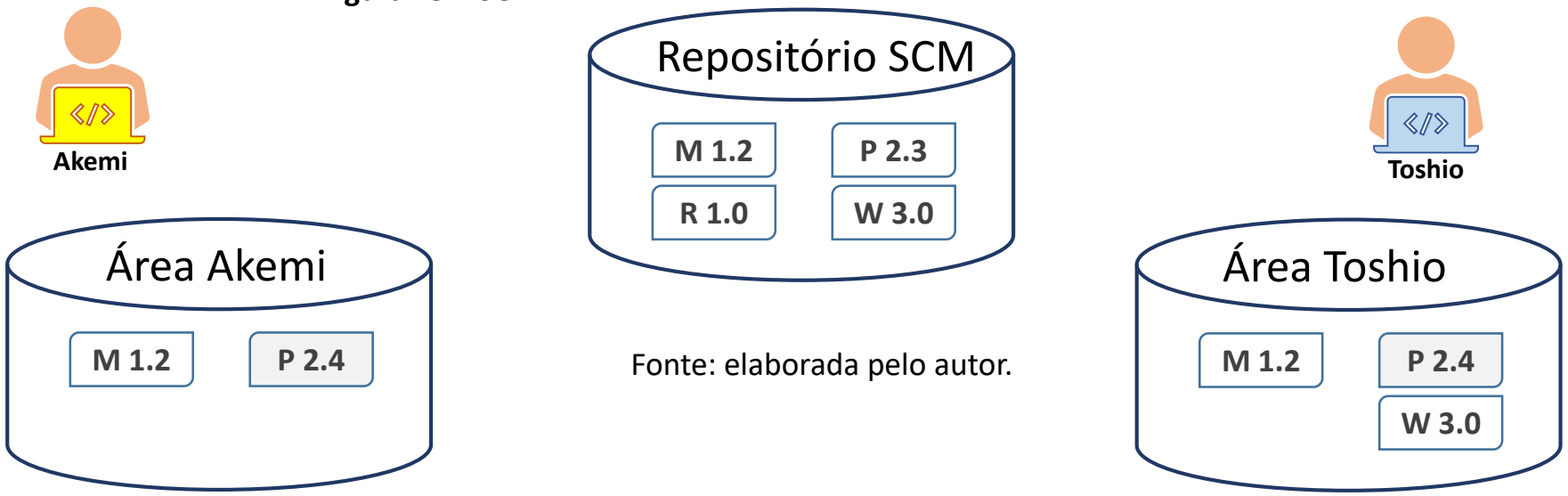


Fonte: elaborada pelo autor.

Para iniciar as atividades de modificação do código-fonte, o engenheiro de software deve efetuar o check-out.

Processo de manutenção gerenciada (ilustrando SCM)

Figura 15 – SCM



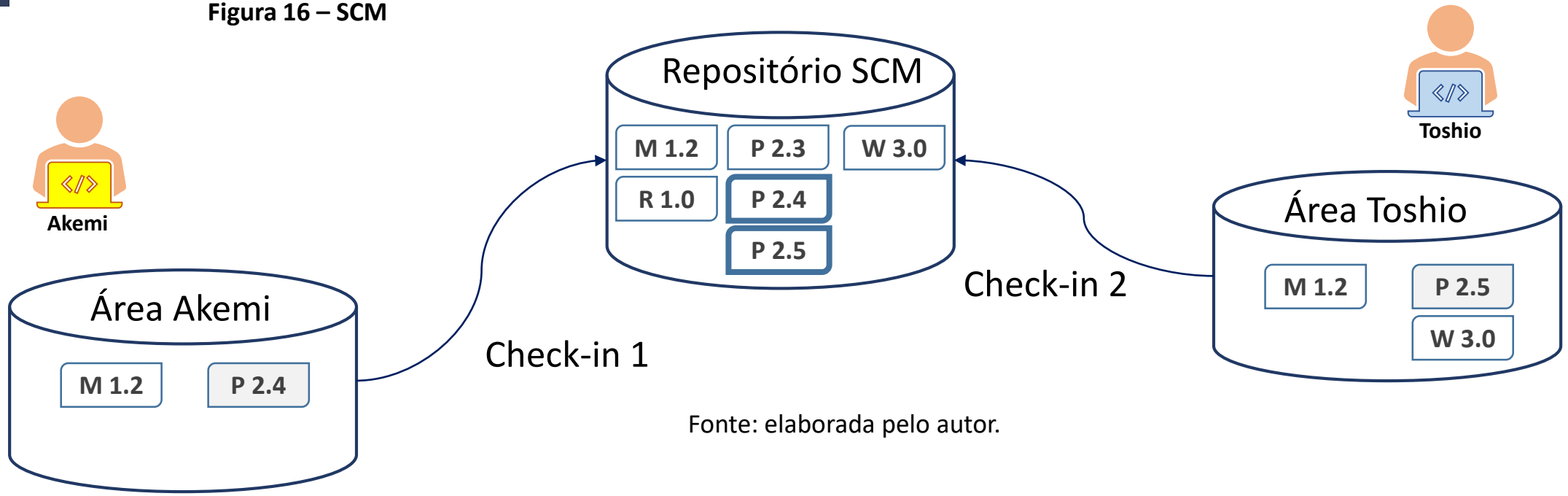
Fonte: elaborada pelo autor.

Em atividade

Enquanto estão em atividade, os engenheiros de software utilizam apenas as áreas de trabalho individual.

Processo de manutenção gerenciada (ilustrando SCM)

Figura 16 – SCM



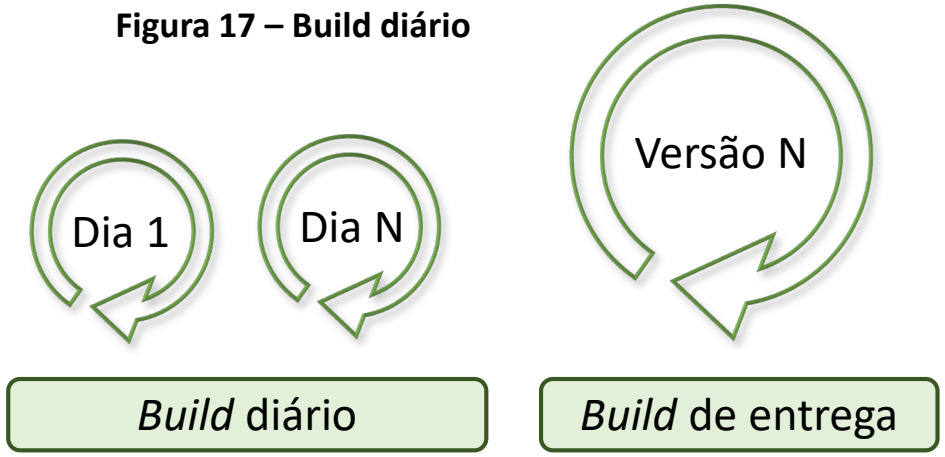
Fonte: elaborada pelo autor.

- Ao concluir as modificações, o engenheiro de software efetua o check-in.
- Quando o item sofre modificação simultaneamente, a ferramenta auxilia no merge do código-fonte (gerando a versão **2.5** do item **P**).

Build integrado – SCM com teste diário

- Visando a qualidade na entrega, Pressman (2016) sugere diariamente uma *Build* para revisão e integração de todos os SCI.
- Realização dos testes de integração diária.
- Possibilita a entrega sem *bugs* e no prazo.
- Uso continuo do SCM.

Figura 17 – Build diário



Fonte: elaborada pelo autor.

Teoria em Prática

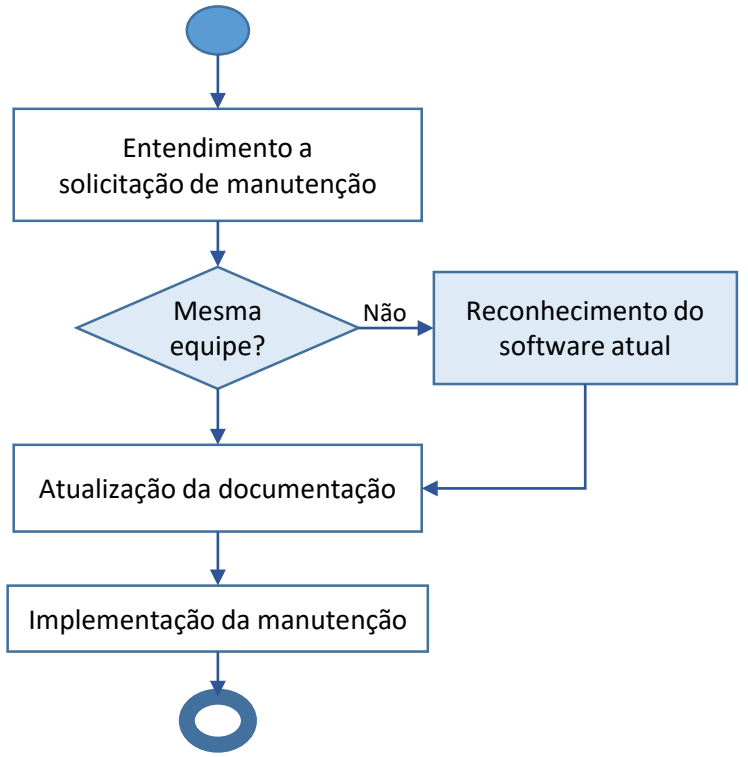
Bloco 4

Marco Ikuro Hisatomi

Reflita sobre a seguinte situação

- Segundo Sommerville (2018, p. 235): “quando há envolvimento de times diferentes, uma diferença fundamental entre o desenvolvimento e a evolução é que o primeiro estágio da implementação da mudança requer uma **compreensão do programa**”.
- Analisar o impacto da mudança para certificar-se que não afetará outra parte do sistema ou na integração com outros sistemas do mesmo ambiente operacional.

Figura 18 - Reconhecimento do software atual



Fonte: elaborada pelo autor.

Compreensão do programa (SOMMERVILLE, 2018, p. 234)

- Aplicativos de celulares podem ser menos complexos.
- Sistemas críticos embarcados podem necessitar da formalização com uma documentação estruturada para cada etapa do processo.
- Analisar componentes a serem alterados, os custos da mudança.
- Considerar que as origens das solicitações podem ter várias fontes:
  - Relatórios de defeitos emitidos pelos *stakeholders.*
  - Novas ideias do time de desenvolvimento para melhoria técnica.

Dica do Professor

Bloco 5

Marco Ikuro Hisatomi

Classificação os bugs

Segundo Pressman (2016, p. 432), quando a **falha** é percebida, significa que **já ocorreu** um desconforto para o usuário ou até prejuízo nos negócios do cliente.

O **defeito** aparece quando o sistema é submetido a situações não previstas ou funcionalidade diferente do especificado nos requisitos ou história de usuário (PRESSMAN, 2016 p. 432)

A causa fundamental de *bugs* é o **erro** acarretado por pessoas por **falta de habilidade**, **por desconhecimento** de técnicas da programação ou **por falta de entendimento** dos requisitos ou história de usuário (PRESSMAN, 2016 p. 432)

Figura 19 – Classificação de *bugs*

Falha

Defeito

Erro

Fonte: elaborada pelo autor.

Referências

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.

Referências

TAENTZER, G. *et al.* (Ed.) **Managed software evolution**: the nature of software evolution. Cham, Switzerland: Springer Open, 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. Hoboken: Wiley, 2014. Disponível em: http://93.174.95.29/main/61D09E290D2D96735502FA9944C49D0A. Acesso em: 11 jun. 2020.

Bons estudos!