GERÊNCIA DE CONFIGURAÇÃO E EVOLUÇÃO

AULA 5

Profª Adriana Bastos da Costa

CONVERSA INICIAL

Temos discutido ao longo da nossa caminhada que a gerência de configuração é uma disciplina fundamental quando falamos em projetos de desenvolvimento de software, tanto para organizar a construção de um novo software quanto para garantir uma manutenção segura e adequada de um software já em uso.

Todo software possui um ciclo de vida que envolve não só as etapas que ocorrem durante o desenvolvimento, mas também as etapas que ocorrem durante todo o período de manutenção e suporte, enquanto o software estiver ativo e em uso. A gerência de configuração se aplica ao ciclo de vida do software, e não apenas ao ciclo de vida de desenvolvimento do software.

Outro conceito importante para analisarmos é o fato de todo software passar por envelhecimento, que é algo natural, uma vez que o software sofre com as evoluções das tecnologias e das necessidades dos clientes, portanto, ele vai se desgastando ao longo do tempo. Por isso as manutenções são necessárias, para manter o software útil para as necessidades do cliente. Dessa forma, também temos o apoio da gerência de configuração, para garantir os componentes de software gerenciados e controlados ao longo do tempo.

Baseado nesse conceito, nesta etapa vamos explorar o que é a Lei de Lehman e como ela se aplica ao desenvolvimento de software.

Esta etapa está dividida em 5 tópicos principais, sendo eles:

- O que é a Lei de Lehman;
- Como a Lei de Lehman está estruturada;
- Ciclo de Vida do Software;
- Envelhecimento e Rejuvenescimento do Software;
- Evolução do hardware e sua contribuição para a evolução do software.

Vamos explorar os assuntos e entender como a Lei de Lehman envolve o desenvolvimento de software e se relaciona com o ciclo de vida de um software. Vamos também discutir sobre o envelhecimento e o rejuvenescimento do software e como a evolução do hardware serve de base para a evolução dos softwares.

TEMA 1 – O QUE É A LEI DE LEHMAN

Você já ouviu falar nas Leis de Lehman? Pois bem, é sobre isso que vamos discutir. Vamos discutir também sobre outros conceitos importantes que de alguma forma se relacionam com as Leis de Lehman.

Você já parou para pensar que várias empresas, que vivem do desenvolvimento de software, possuem duas atividades muito claras? Uma delas é desenvolver novos softwares, a partir dos requisitos e necessidades dos clientes. Mas também é muito comum equipes terem como responsabilidade suportar e manter softwares já em uso. Vimos até aqui os conceitos relacionados com evolução corretiva e evolutiva de software. Portanto, a evolução do software é comum e incentivada no mundo do desenvolvimento do software.

E isso ocorre, de acordo com Sommerville (2019), porque existe uma necessidade constante e inevitável de evolução por parte de todo software que espera manter-se ativo por um longo período de tempo. E isso ocorre de maneira natural com todo software, pois as necessidades do mercado e dos clientes mudam, além da evolução da tecnologia, que acaba exigindo novas funcionalidades do software.

Meir Manny Lehman foi um cientista e pesquisador na área da computação. Ele foi membro da Real Academia de Engenharia e professor na Escola de Ciências de Computação em Middlesex University. De 1972 até 2002, ele foi professor e Chefe do Departamento de Computação no Imperial College de Londres. E foi ele, junto com outros colegas de profissão, que organizou e difundiu os conceitos relacionados com a evolução do software.

A Lei de Lehman é uma fonte histórica de bom senso em software. Meir Lehman nasceu na Alemanha e transferiu-se para a Inglaterra na década de 30, trabalhando na IBM entre 1964 e 1972. Em 1974, ele publicou o que seria mundialmente conhecido como as *Leis de Lehman sobre evolução de software*. Os conceitos desenvolvidos e difundidos pelas Leis de Lehman eram relacionados, inicialmente, a sistemas do tipo E ou E-Programs.

Para entender melhor isso, vamos entender o que são sistemas do tipo E ou E-Programs. Inicialmente, precisamos entender que o contexto da IBM na década de 70 era de computadores que chamamos até hoje de mainframe. Ou seja, são computadores de grande porte, utilizados por grandes empresas para processar grandes quantidade de informações e transações simultâneas. Esses

computadores utilizam sistemas operacionais chamados de OS/360. O OS/360 foi um sistema operacional desenvolvido pela IBM para seus novos computadores mainframe System/360, anunciados em 1964. O OS/360 foi um dos primeiros sistemas operacionais a ter acesso direto a dispositivos de armazenamento como pré-requisito para sua operação.

Bom, sistemas do tipo E são os softwares desenvolvidos para resolver problemas do mundo real, operados por pessoas que se beneficiem deles em seu dia a dia, e foram criados para rodarem em mainframes com sistemas operacionais OS/360.

Os primeiros estudos, que se tem conhecimento, sobre os sistemas para OS/360 ficaram conhecidos como *leis de Lehman*, pois eram generalizáveis e aplicáveis a diferentes contextos na tomada de decisão, planejamento, desenvolvimento e manutenção de software. Ou seja, apesar de as leis terem sido pensadas para um tipo específico de sistemas e computadores, elas poderiam ser aplicadas em outros tipos, uma vez que eram genéricas e baseadas em bom senso.

Portanto, Lehman, com a ajuda de outras pessoas durante cerca de 30 anos, realizou estudos sobre o crescimento e a evolução de softwares. Desses estudos foram desenvolvidas leis, que na verdade são teorias gerais que buscam classificar as características da evolução de software.

No próximo tópico, vamos detalhar quais são as Leis de Lehman e como elas estão estruturadas e aplicadas nos softwares modernos.

TEMA 2 – COMO A LEI DE LEHMAN ESTÁ ESTRUTURADA

Meir Lehman, um grande cientista e pesquisador na área de TI, publicou o que seria mundialmente conhecido como *Leis de Lehman sobre evolução de software*. A partir de 1974 até o final do século XX, ele pesquisou e evoluiu com oito pontos sobre sistemas do tipo E ou E-Program, que são as chamadas, até hoje, de as 8 Leis de Lehman.

Os 8 pontos identificados e organizados por Lehman passaram a ser vistos como “lei” porque eram generalizáveis e aplicáveis a muitos contextos. Podiam ser usados em diferentes aplicações, como para tomada de decisão, planejamento, desenvolvimento e manutenção de software. As 8 leis de Lehman são as seguintes.

1. **Mudança contínua –** Um software deve ser continuamente adaptado, senão torna-se aos poucos cada vez menos satisfatório. A cada alteração no ambiente em que ele roda que exija melhorias, não as implementar no software o tornará progressivamente menos satisfatório naquilo para o que foi construído. O que se busca com essa lei é manter o software útil para o propósito para o qual ele foi construído.
2. **Complexidade crescente –** Se não forem tomadas medidas para reduzir ou manter a complexidade de um software, conforme ele é alterado sua complexidade irá aumentar progressivamente. Deve haver um esforço para reduzir a complexidade final de um sistema enquanto este recebe alterações. É o conceito ágil, muito difundido quando se utiliza essas metodologias para o desenvolvimento de software, relacionado com a refatoração. Ou seja, refatoração é a ação de melhorar o código buscando soluções mais simples. Dessa forma, a lei afirma que a solução seria aumentar os esforços em manutenções preventivas, o que resultaria no aprimoramento de sua estrutura, facilitando as manutenções posteriores.
3. **Auto regulação –** A curva pertinente ao processo de evolução de um software em relação a seus atributos e processos são autorreguláveis e próximos a uma curva normal, subindo até um teto, quando começa a diminuir. Ou, em outras palavras, essa lei sugere que o software possui uma dinâmica própria definida, o que decide as tendências da manutenção, e limita o número de possíveis mudanças, de forma que essa lei é uma consequência da organização responsável que determina a prioridade das mudanças baseada nos riscos, valores da mudança e custos envolvidos. Está relacionada também com a Lei da Conservação da Estabilidade Organizacional (1980). Essa lei complementar afirma que durante o ciclo de vida do software os resultados serão quase constantes, de modo que à substituição de recursos ou pessoas tenham efeitos imperceptíveis na evolução do software. Essa lei requer processo e organização.
4. **Conservação da estabilidade organizacional –** A velocidade de atividade global efetiva de um software em evolução deverá se manter invariável durante todo o ciclo de vida deste produto. As informações que são levadas em consideração para as tomadas de decisão que levam à evolução de um software tendem a ser constantes. A estabilidade

organizacional depende da efetiva manutenção e estabilidade do software, que precisa continuar atendendo o objetivo de negócio para o qual ele foi criado.

5. **Conservação de familiaridade –** Durante a vida útil de um software em evolução, a taxa de mudanças tende a ser proporcional ao domínio que a equipe detém. A taxa de evolução de um software está intimamente ligada ao grau de familiaridade dentre os profissionais que o mantém. Ou melhor, à medida que o software evolui, todos os envolvidos no seu uso ou manutenção devem manter o domínio de seu conteúdo e comportamento para conseguir uma evolução satisfatória. Precisam conhecer o escopo do software. Com as mudanças incrementais sendo quase constantes, a complexidade vai aumentando e com isso torna-se mais difícil obter esse domínio, podendo haver uma ou mais mudanças necessárias, que quando implementadas acarretam em mudanças comportamentais ou de limites de escopo do software. Neste momento, é preciso decidir se a mudança deve ser inserida no software em questão ou se um novo software deve ser projetado para atender à mudança necessária ao negócio.
6. **Crescimento Contínuo –** Todo software deve ter o conteúdo funcional continuamente ampliado durante seu ciclo de vida para manter a satisfação dos seus usuários e as necessidades do negócio. O projeto inicial não consegue incluir absolutamente tudo o que é necessário para atender o negócio e progressivamente precisará ser aumentado, até porque novas tendências surgem no mercado e os clientes passam a exigir mais funcionalidades. Ou seja, com a necessidade de manter o conteúdo do software funcional e útil, mudanças que não foram previstas na fase de desenvolvimento, mudanças para correção de erros, adições de novas funcionalidades ou melhorias em funções pré-existentes ocasionadas pelo domínio operacional em que o software está inserido deverão ser realizadas para manter a contínua satisfação dos usuários, ocasionando no aumento do conteúdo funcional do software. Essa lei envolve identificar onde o aumento das funcionalidades pode impactar, pois um requisito novo pode gerar alterações ou ajustes em requisitos já existentes, para manter a consistência e integridade do software.

7. **Qualidade diminuindo –** Os softwares desenvolvidos para resolver problemas do mundo real se depreciam progressivamente se eles não receberem as mudanças necessárias para adaptar-se ao que acontece em seu ambiente operacional durante todo o tempo de seu ciclo de vida útil. Ou seja, o software precisa se manter útil e atual, para que continue agregando valor para o negócio. Essa lei reforça a preocupação com a análise do todo, pois a qualidade do software tende a diminuir com a sua evolução, por possíveis problemas no desenvolvimento agravados pela possibilidade de mudanças externas, o que torna o ambiente imprevisível quanto ao surgimento de agravantes a sua qualidade, e mesmo que o software funcione satisfatoriamente em um presente momento não é possível garantir que este continue assim ao longo dos anos, o que torna necessárias medidas que evitem a detecção e correção contínua desses agravantes. Aqui podem ser utilizadas técnicas para garantir a rastreabilidade dos requisitos e sua constante consistências, além de técnicas de testes que busquem avaliar e executar testes de regressão. O foco deve ser garantir a qualidade do software como um todo.
8. **Sistema de feedback –** Os processos de manutenção e evolução de um software refletem sistemas de feedback em múltiplos níveis, loops e agentes e devem ser assim tratados para manter-se significativos. Os processos de evolução do software constituem um complexo sistema de realimentação que é constantemente realizado por seus usuários, podendo ser esses retornos positivos e negativos entre as versões, o que de certa forma estabelece o crescimento do software de acordo com a quantidade de retornos negativos e positivos, somado a outros fatores da organização. É preciso fazer uma análise de custo X benefício, buscando a melhoria contínua do software com foco em agregar cada vez mais valor ao negócio. Nessa lei é possível aplicar o conceito de PDCA (*plan, do, check e act*) que é usado na gestão dos negócios e se aplica perfeitamente bem à evolução de um software. Ou seja, é preciso continuamente buscar a melhoria contínua do software, com foco em atender aos objetivos estratégicos da empresa e os resultados de negócio esperados, mas sem esquecer de analisar as 8 Leis de Lehman, que ajudam a garantir uma análise adequada e uma manutenção mais organizada do software.

Até hoje as Leis de Lehman são válidas e úteis, pois elas são provocativas. Elas geram reflexão em pontos relevantes no que diz respeito a assuntos relacionados ao planejamento, construção e manutenção de software no seu aspecto tático-técnico.

Saiba mais

Para se aprofundar nas Leis de Lehman e nos conceitos relacionados com a evolução e manutenção de software, acessem os links indicados e ouçam os vídeos relacionados com os assuntos que discutimos até aqui: <https://www.youtube.com/watch?v=1kDSBc9ZouM>. e <https://technbiz.eximia.co/>. Acessados em: 26 set. 2022.

Essa tendência de pensar em formas melhores de se fazer e manter software não é exclusividade das Leis de Lehman. Na década de 90, um grupo de estudiosos da área de TI iniciaram experiências para trazer ao desenvolvimento de software princípios preconizados pela indústria automobilística no modelo Lean da Toyota. Dessa forma, surgiram os conceitos ágeis que são aplicados no desenvolvimento de software através de várias metodologias ágeis.

Dessa forma, podemos dizer que no início do século XXI foi promulgada a proposta de uma nova "Lei" sobre evolução de software, conhecido mundialmente como *Manifesto Ágil*. Sim, o manifesto ágil é genérico o suficiente para ser entendido como uma filosofia ou uma lei de desenvolvimento de software, não é mesmo?

Como o manifesto ágil foi criado? Você já ouviu falar, pois bem, vamos relembrar.

Um grupo de especialistas em TI, desenvolvedores e consultores, que já vinham praticando novas formas de organização de equipes, técnicas de identificação de requisitos, interação, análise, desenvolvimento, testes e evolução reuniram-se em uma estação de esqui em Utah (USA) e começaram a organizar pensamentos em torno de uma forma diferente de fazer software. Organizaram então um manifesto.

O Manifesto dizia: "Estamos descobrindo maneiras melhores de desenvolver softwares, fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar":

- Indivíduos e interação entre eles mais que processos e ferramentas;

- Software em funcionamento mais que documentação abrangente;
- Colaboração com o cliente mais que negociação de contratos; e
- Responder a mudanças mais que seguir um plano.

Ou seja, mesmo havendo valor nos itens à direita, deveríamos valorizar mais os itens à esquerda. Além dos valores, o manifesto ágil contém 12 princípios ágeis, que devem ser entendidos e trabalhados para o seu sucesso, sendo eles:

- Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor;
- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento – processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas;
- Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos;
- Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto;
- Construir projetos ao redor de indivíduos motivados – dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho;
- O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara;
- Software funcional é a medida primária de progresso;
- Processos ágeis promovem um ambiente sustentável – os patrocinadores, desenvolvedores e usuários devem ser capazes de manter indefinidamente passos constantes;
- Contínua atenção à excelência técnica e bom design aumenta a agilidade;
- Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito;
- As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis;
- Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

As Leis do século XX, conhecida como *Leis de Lehman*, são mais genéricas que o Manifesto do século XXI, conhecido como Manifesto Ágil. Pois

enquanto o primeiro apontava para pontos que são consenso sob o aspecto técnico no desenvolvimento de software, o segundo propõe uma abordagem menos industrial ou menos comando-controle, uma abordagem que envolve questões técnicas, mas também questões sociais, em que um bom artefato técnico é consequência da interação humana que o cria e mantém.

Enquanto as Leis de Lehman relatavam uma percepção objetiva e prática, racional e técnica, o Manifesto Ágil sugere uma grande quebra de paradigma. Ele envolve implementar um processo de mudança cultural onde o ponto principal não é o artefato e as tarefas, mas a mobilização das pessoas para em conjunto e com técnicas apropriadas desenvolverem a melhor solução possível, o foco está na qualidade e na satisfação do cliente.

TEMA 3 – CICLO DE VIDA DE SOFTWARE

As Leis de Lehman que estudamos nos temas anteriores desta etapa nos levam a entender que um software possui um ciclo de vida. Você já parou para pensar que várias coisas possuem um ciclo de vida? As pessoas, os produtos e também os softwares.

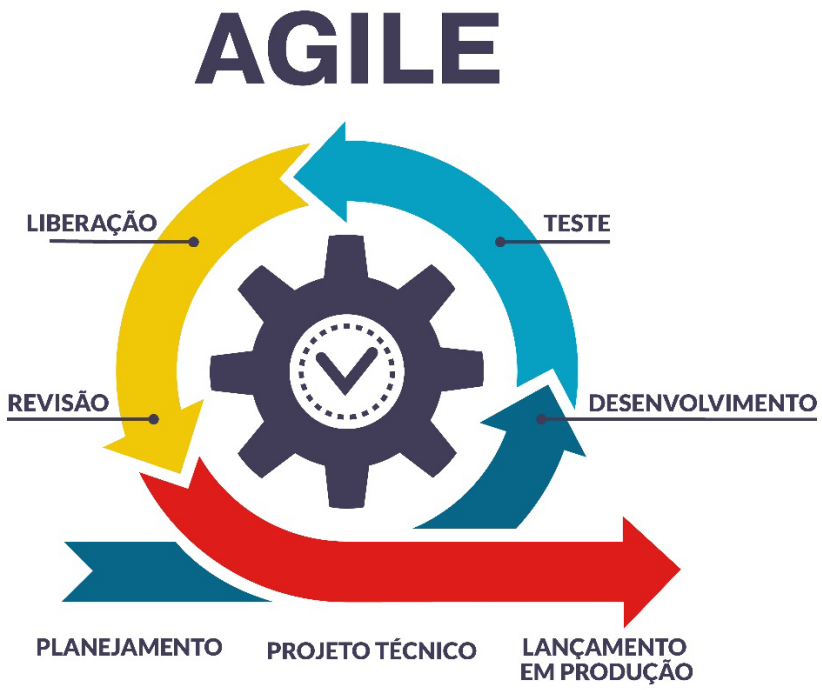
O ciclo de vida é a estrutura contendo processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software, abrangendo a vida do sistema, desde a definição de seus requisitos até o término de seu uso. Sim, o ciclo de vida passa pelo desenvolvimento do software e possui etapas também durante seu suporte e manutenção, quando ele já está em produção e sendo usado de forma comercial.

Quando falamos em desenvolvimento de software, o modelo de ciclo de vida é a primeira escolha a ser feita, pois determina como a construção do software será organizada. Mas quando falamos em manutenção e suporte do software, o papel do ciclo de vida é mostrar como está a evolução de maturidade do software, até que o mesmo caia em declínio e seja substituído.

Os principais ciclos de vida de desenvolvimento de software conhecidos no mercado e mais utilizados são: Cascata, Scrum, RUP, XP, entre outros.

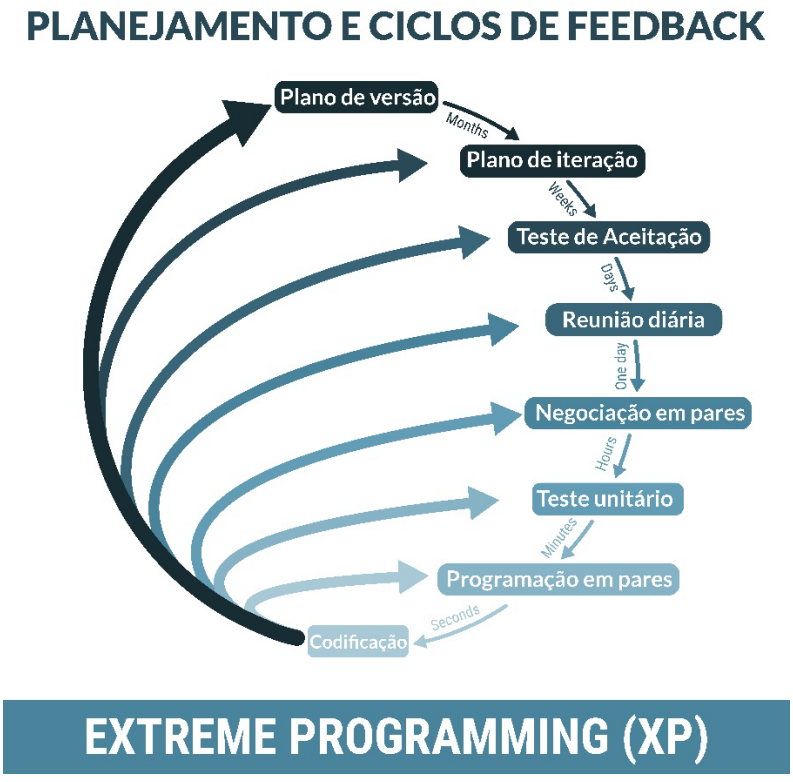
Cada ciclo de vida possui uma organização própria, com fases bem definidas que envolvem as atividades, as funções e responsabilidades de cada membro da equipe. O que diferencia um processo de software do outro é a ordem em que as fases vão ocorrer, o tempo e a ênfase dados a cada fase, as atividades presentes, e os produtos entregues.

Figura 1 – Ciclo de Vida Scrum



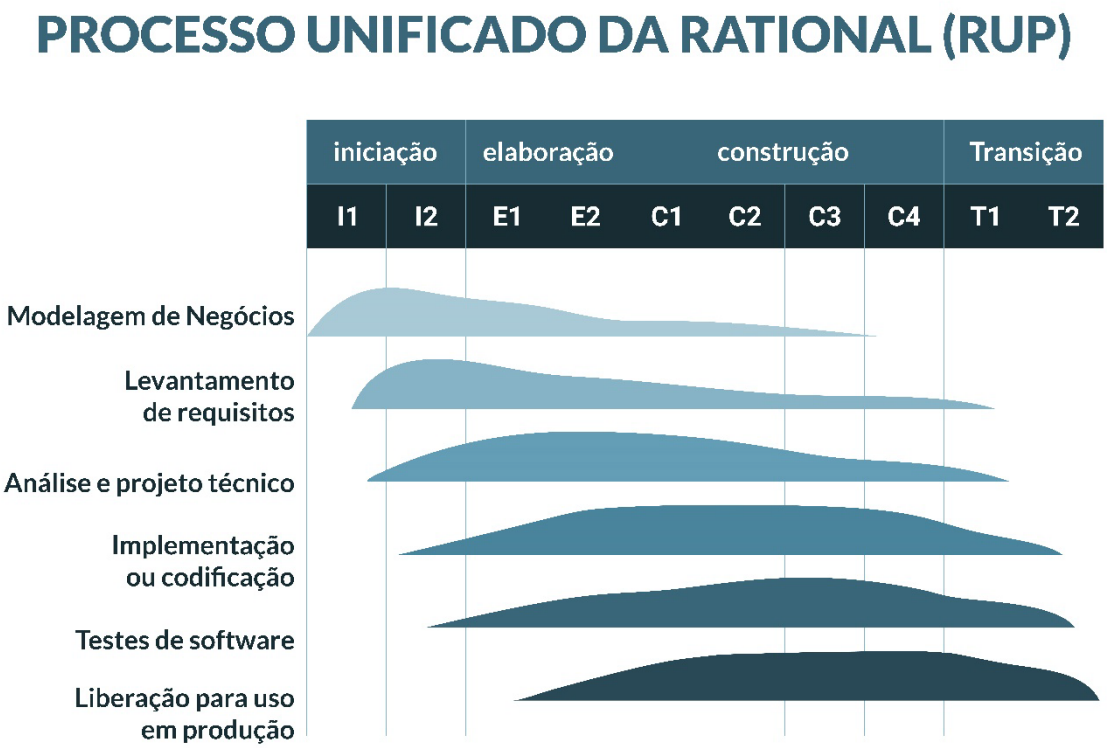
Créditos: WinWin artlab/Shutterstock.

Figura 2 – Ciclo de vida XP



Créditos: KostiantynL /Shutterstock.

Figura 3 – Ciclo de vida RUP



Créditos: KostiantynL /Shutterstock.

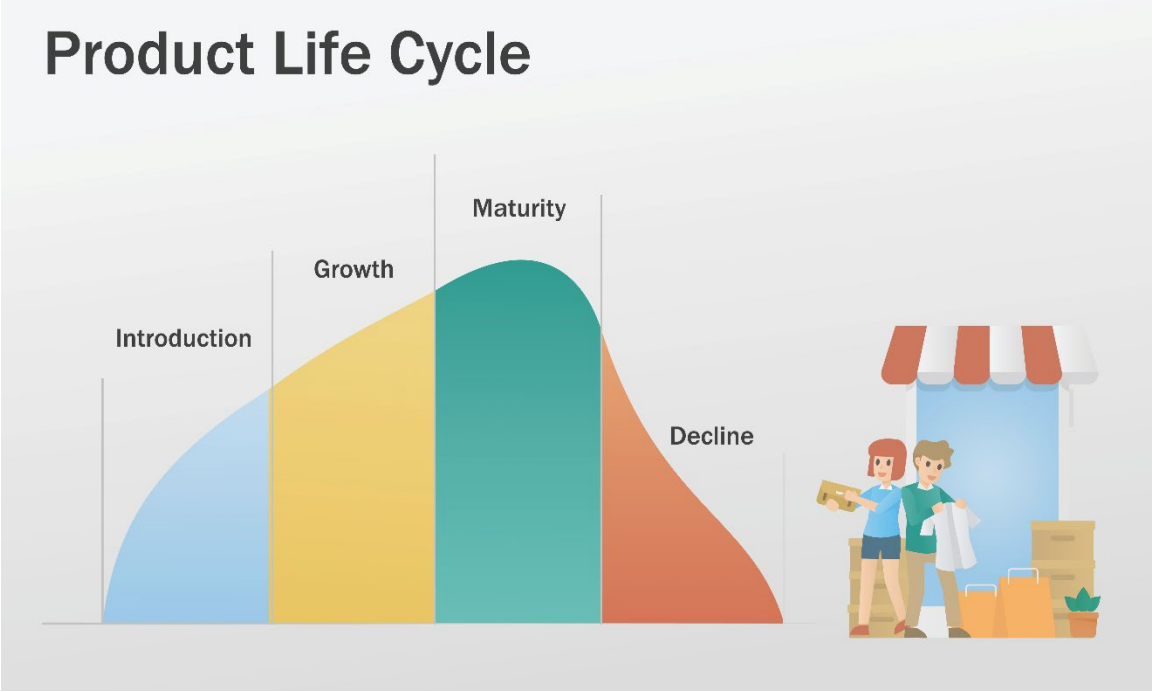
Quando o software está pronto e em uso no ambiente de produção pelos usuários, ele passa a ser visto como um produto. Dessa forma, como produto de software, ele tem um ciclo de vida diferente, que envolve fases relacionadas ao seu amadurecimento, envelhecimento e declínio ou desuso.

Bom se pensarmos na definição de vida como "propriedade que caracteriza os organismos cuja existência evolui do nascimento até a morte", faz todo sentido pensar que os produtos também passem pelo mesmo ciclo. Tal qual organismos vivos, produtos também nascem, evoluem e morrem. Mas para conseguir entender o ciclo de vida de um produto, precisamos entender a diferença entre projeto e produto.

Quando falamos de projetos de software, lidamos com coisas que possuem início, meio e fim muito bem delimitados pelo escopo da solução. Os ciclos de vida, tanto tradicionais quanto ágeis, com levantamentos de requisitos, passagem de bastão entre etapas e o lançamento de parte do produto no final do processo, caracterizam o desenvolvimento de um software e suas etapas, de maneira geral. Esses ciclos de vida, quando aplicados ao projeto de software, possuem um escopo a ser atendido.

Produtos, por outro lado, não possuem escopo fechado, as melhorias vão surgindo conforme a necessidade do mercado, do cliente e da tecnologia. Essas melhorias precisam ser gerenciadas, priorizadas e implementadas de maneira organizada e segura.

Figura 4 – Ciclo de vida do produto de software



Créditos: Appleing/Shutterstock.

Todo produto, até mesmo o produto de software, percorre uma jornada que vai desde o seu desenvolvimento até uma eventual descontinuação.

Podemos definir o que é o ciclo de vida do produto como um conjunto de etapas que todo produto percorre do seu projeto e concepção até o momento em que ele é descontinuado e retirado do mercado. O conceito de ciclo de vida do produto foi desenvolvido pelo economista alemão Theodore Levitt. Segundo Levitt, nenhum produto “vive” para sempre; mais cedo ou mais tarde o seu ciclo de vida chegará ao fim. Isso também ocorre com o software, que pode ser substituído por um novo software, mais modernos e mais adequado às novas necessidades do negócio. Mas até que isso ocorra, o software passa por muita manutenção, e precisa ser gerenciado constantemente.

Mas voltando ao ciclo de vida do produto, Levitt propôs um modelo que divide o ciclo de vida dos produtos nas seguintes fases:

- Desenvolvimento;
- Introdução;
- Crescimento;
- Maturidade;
- Declínio.

Não vamos aqui falar da fase de Desenvolvimento, pois esta é a fase quando o software está sendo construído. Nesta fase se aplica outro ciclo de vida, como os que vimos anteriormente.

Focando nas demais fases do ciclo de vida do produto proposto por Levitt, temos o seguinte detalhamento do objetivo de cada fase.

- **Introdução** – na fase de introdução, o software já está construído. Ele, então, é colocado em produção para ser utilizado pelos usuários e clientes. Essa é a fase onde os usuários começam a conhecer o potencial do software, identificando pontos fortes e pontos fracos, que precisam ser melhorados. Aqui já podem ser identificadas melhorias evolutivas e corretivas.
- **Crescimento** – na fase de crescimento, é muito comum que o software continue passando por evoluções e melhorias. As melhorias precisam ser gerenciadas e priorizadas, para que sejam implementadas de maneira organizada, para que defeitos não sejam introduzidos no software, junto com novos requisitos. A gerência de configuração é fundamental para garantir a qualidade dos componentes do software, mantendo o software seguro e íntegro.
- **Maturidade** – é a fase quando o software está mais estável, quando quase não são mais identificadas evoluções corretivas. As melhorias evolutivas ainda podem surgir, mas cada vez em menor número, pois o software já está maduro o suficiente para atender ao negócio. É o momento em que o cliente passa a pensar se novas melhorias evolutivas deverão ser ainda implementadas no software ou se vale mais a pena partir para projetar um novo software. Não tem como prever quanto tempo dura cada fase do ciclo de vida, mas sabemos que não é algo curto e nem rápido. Um software tem uma expectativa de vida realmente bastante grande. Um bom exemplo disso são os sistemas em mainframe com mais

de 20 ou 30 anos que ainda são usados por grandes instituições financeiras.

- **Declínio** - cedo ou tarde todos os produtos chegarão à fase de declínio do ciclo de vida. Após atingir a maturidade e render bons resultados para a empresa, os softwares deixam de representar vantagens e precisam ser descontinuados. Pode ser que fique muito caro manter novas melhorias por conta do custo da manutenção. Nesse momento é muito comum que a empresa comece a pensar em outras soluções para substituir o software, seja desenvolvendo um novo software mais moderno seja pensando em soluções prontas disponíveis no mercado e que atendam às necessidades da empresa.

Entendidos os conceitos relacionados com o ciclo de vida do desenvolvimento do software e do produto do software? Vejam que são complementares, mas possuem conceitos diferentes e que envolvem gestões diferentes. Mas um ponto é certo, em ambos os ciclos de vida, a gerência de configuração é fundamental para manter o controle e a segurança do que está sendo inserido no software.

Pensando um pouco mais nos conceitos que envolvem os ciclos de vida, é possível perceber que eles refletem o envelhecimento e rejuvenescimento de um software. E é exatamente sobre isso que vamos discutir no próximo tema.

TEMA 4 – ENVELHECIMENTO E REJUVENECIMENTO DO SOFTWARE

Já vimos que o software possui uma vida longa, que envolve desde sua construção até seu envelhecimento e declínio.

O envelhecimento do software é um processo pelo qual a qualidade do código do software diminui ou se torna desatualizada, levando a vários problemas técnicos. Isso pode ocorrer por vários e diferentes motivos. Qualquer software ou programa de computador está sujeito a um ciclo de envelhecimento que muda gradualmente suas características e desempenho para pior. Eventualmente, o software perde a utilidade e os usuários precisam atualizar para uma versão mais nova e atual.

Essa é a razão pela qual grandes empresas como a Google, Microsoft, Apple e outros gigantes da tecnologia regularmente lançam atualizações de software importantes. Conseguem se recordar de atualizações de recursos como

novas e melhores versões dos sistemas operacionais existentes? A Apple faz isso constantemente com o IOS dos celulares, não é mesmo?

Mas uma pergunta bastante intrigante e que nos leva a pensar é: o software se degrada com o tempo?

E a resposta é sim! Todo software se degrada com o tempo devido ao avanço do hardware, das tecnologias em geral ou às mudanças que ocorrem nas necessidades dos usuários.

Conforme o software envelhece, ele gradualmente deixa de atender aos seus propósitos e os usuários começam a ter problemas de lentidão ou até mesmo de instabilidade. É possível que o software dispare travamentos e travamentos frequentes do sistema ou até mesmo pare de funcionar completamente. O Windows XP é talvez o exemplo perfeito de envelhecimento de software, concorda? Pesquise sobre ele para compreender tudo o que ocorreu com esta versão do Windows e como a Microsoft tratou o assunto. É uma história muito interessante.

Mas voltando para o nosso assunto principal, embora não possamos evitar o envelhecimento do software, é possível ter uma imagem bastante clara do que pode estar causando o envelhecimento. Se temos a visão do problema, podemos tomar ações específicas para estender a vida útil do software. Construir um software envolve um grande investimento, portanto, quanto mais o software for utilizado e agregar valor para os negócios, melhor será o retorno do investimento feito.

Existem alguns pontos básicos que podem ser entendidos como problemas comuns que levam o software a se degradar com o tempo.

- **Atualizações de hardware e da tecnologia –** os avanços da tecnologia e, principalmente, do hardware estão ocorrendo em um ritmo muito rápido. Se os desenvolvedores de software não conseguirem adaptar o código ao progresso do hardware, o software rapidamente se tornará obsoleto.
- **Acúmulo de erros ao longo do tempo –** já é conhecido e aceito que não existe software livre de bugs ou de erros, todo software terá erro, por conta da complexidade ou dos diferentes cenários de funcionalidades existentes. É muito comum, conforme você continua executando seus programas de computador, mais e mais erros podem se acumular. Isso coloca pressão adicional no código do software.

- **Dados e falta de integridade de arquivo** – o impacto na integridade dos dados é algo que você não pode evitar. Os dados originais ou o código do software mudam involuntariamente com o tempo. Pode ser que cada vez mais erros ocorram durante as fases de gravação, leitura, processamento ou armazenamento de dados, afetando a integridade dos dados.
- **Memória "inchada e vazando" –** se os programas de computador não conseguem liberar os recursos de memória de que não precisam mais, eles basicamente esgotam a memória disponível. Como resultado, o computador não executará corretamente o código do software e vários erros podem se acumular. Isso ocorre até memo com nossos computadores. Sabe quando precisamos reiniciar o computador para melhorar um problema de lentidão, por exemplo? Pois é, pode ser que a memória tenha ficado ocupada demais, não conseguindo processar de maneira satisfatória os comandos realizados pelo usuário.

Todos esses pontos acima aceleram o processo de envelhecimento do software. Mas por outro lado, é possível tomar ações que ajudam a evitar o envelhecimento do software.

- **Obtenha as últimas atualizações –** uma maneira de evitar o envelhecimento do software é instalar regularmente as atualizações mais recentes disponibilizadas. Os desenvolvedores de software lançam atualizações constantemente para melhorar seus programas, adicionar novos recursos e corrigir erros conhecidos. Desta forma, o software existente pode se adaptar às necessidades dos usuários ou às novas plataformas de tecnologia existentes. As novas versões podem também melhorar a segurança do computador.
- **Limpe seu sistema regularmente –** evite instalar software desnecessário. Crie o hábito de limpar regularmente o sistema de arquivos inúteis e arquivos temporários. Não se esqueça de verificar o sistema em busca de infecções por vírus e malware. O malware é conhecido por alterar arquivos de programas, o que acelera o processo de envelhecimento do software. Instale e utilize um antivírus no computador.
- **Libere os recursos do computador de esforços desnecessários –** desinstale, periodicamente, os programas de que não precisa mais. Tenho o controle do que está instalado no computador. Limite o número

de programas que são iniciados automaticamente na inicialização. Mantenha as coisas simples sempre que possível.

- **Reinstale o software –** desinstalar e reinstalar o software de vez em quando ajuda a estender sua vida útil ou duração de uso. Ao instalar uma nova cópia desse software ou programa, você pode reparar arquivos corrompidos ou retornar para configurações mais otimizadas, por exemplo.

Falamos até agora sobre o envelhecimento do software, mas e sobre rejuvenescimento do software, isso pode ocorrer?

Na verdade, o termo rejuvenescimento do software está relacionado com as ações ou métodos usados para prevenir ou retardar o envelhecimento do software. Em outras palavras, essas ações têm como objetivo remover os erros de software acumulados ao longo do tempo, liberar recursos do sistema e corrigir problemas de corrupção de dados. Ou seja, o rejuvenescimento do software está relacionado com minimizar o impacto do tempo, retardando o seu envelhecimento.

Podemos utilizar a manutenção de um carro como uma analogia para entender este conceito. Sabemos que um carro tem um tempo de vida, como todo produto, não é mesmo? Mas, se um carro é bem cuidado, passa por revisões constantes e é cuidado de maneira adequada, ele dura mais funcionando bem. Não é verdade? Pois bem, o mesmo se aplica ao software.

Algumas técnicas de rejuvenescimento de software envolvem ações que agem contra seu envelhecimento.

- Reinicializar o sistema é uma das técnicas de rejuvenescimento de software mais populares e úteis. Não é esse o primeiro método de solução de problemas que vem à mente quando o computador ou telefone para de funcionar?
- A instalação limpa do sistema operacional é outro método de rejuvenescimento de software acessível ao usuário. Em essência, isso significa instalar uma cópia com configurações originais da versão mais recente do sistema operacional. É preciso, portanto, desinstalar completamente a versão anterior do sistema operacional, formatar o dico rígido para que todos os dados sejam excluídos, e então, instalar uma nova cópia do sistema operacional.

Dessa forma, podemos entender que como qualquer produto, o software também possui um ciclo de vida. É normal que os programas de computador sofram a um declínio gradual no desempenho ao longo do tempo, o que é chamado de envelhecimento do software. Os usuários podem executar uma série de ações para evitar o envelhecimento do software, mas não podem interromper totalmente o processo. É por isso que novos softwares precisam ser projetados e implementados, para continuar atendendo às necessidades dos clientes e dos negócios.

TEMA 5 – EVOLUÇÃO DO HARDWARE E SUA CONTRIBUIÇÃO PARA A EVOLUÇÃO DO SOFTWARE

É de conhecimento de todos que o hardware e o software são elementos que fazem parte de um computador, onde cada um deles tem sua função para o desempenho e bom funcionamento.

O **hardware** corresponde aos componentes físicos do computador, ou seja, são as peças e aparatos eletrônicos que, ao se conectarem, fazem o equipamento funcionar.

O **software** é a parte lógica, referente aos sistemas que executam as atividades, ou seja, são os programas e aplicativos que fazem com a máquina funcione, deixando de ser apenas um conjunto de componentes eletrônicos e passando a processar transações matemáticas e lógicas.

Quando estudamos a história de TI, percebemos que a evolução do hardware e do software ocorrem de maneira bastante homogênea, com um impulsionando o outro o tempo todo. Algumas vezes o hardware e a tecnologia impulsionam a construção de softwares mais complexos e mais potentes, às vezes, as necessidades de softwares mais especializados pressionam a indústria do hardware a evoluir.

Tanto o hardware quanto o software estão presentes em celulares, TVs, computadores, tablets, impressoras e até mesmo nas máquinas de lavar e micro-ondas mais modernos.

O que temos visto é uma explosão de softwares cada vez mais complexos e dependentes de tecnologia. Isso nos leva a compreender como a evolução do hardware apoia e sustenta a evolução do software.

Atualmente muito se tem falado de *Internet of Things (IoT)*, ou melhor, em português, Internet das Coisas. Mas afinal, o que envolve a Internet das Coisas?

Segundo o site Oracle (2014):

> A Internet das Coisas (IoT) descreve a rede de objetos físicos incorporados a sensores, software e outras tecnologias com o objetivo de conectar e trocar dados com outros dispositivos e sistemas pela internet. Esses dispositivos variam de objetos domésticos comuns a ferramentas industriais sofisticadas.

Créditos: Gorodenkoff /Shutterstock.

Temos percebido e vivido essa evolução, onde a internet das coisas se tornou uma das tecnologias mais importantes do século XXI. Não é mais coisa de ficção científica ou de desenho animado, virou realidade relógios com função de monitoramento cardíaco, carros autônomos, iluminação residencial controlada por um aparelho de celular, entre outras aplicações disponíveis ao alcance de um número cada vez maior de consumidores.

Atualmente, é possível conectar objetos do cotidiano – eletrodomésticos, carros, termostatos, babás eletrônicas – à Internet por meio de dispositivos incorporados, é possível uma comunicação perfeita entre pessoas, processos e outras coisas. Os softwares estão embarcados no hardware, permitindo essa conexão com a internet e o funcionamento da internet das coisas.

A velocidade que temos presenciado a internet das coisas chegar cada vez em mais lares e indústrias está sendo impulsionada pela computação de baixo custo, pela nuvem, por conceitos como big data, análise avançada e tecnologias móveis. Dessa forma, coisas físicas ou objetos podem compartilhar e coletar dados com o mínimo de intervenção humana. Nesse mundo conectado em que vivemos, os sistemas digitais podem gravar, monitorar e ajustar cada

interação entre os objetos. E para isso só basta que eles estejam conectados. O mundo físico encontra o mundo digital, e eles trabalham em conjunto.

Toda essa evolução ocorreu por conta do aumento da velocidade e da capacidade da internet, e por conta de desenvolvimento de softwares que permitem e apoiam a conexão dos diferentes objetos, trazendo uma infinidade de aplicações para o mercado e para a sociedade como um todo.

Embora a ideia de internet das coisas já exista há algum tempo, vários avanços em diversas tecnologias permitiu a aplicação do IoT de forma mais efetiva. Vamos analisar alguns desses avanços.

- **Acesso à tecnologia de sensores de baixo custo e baixa potência** – sensores acessíveis e confiáveis estão possibilitando a tecnologia IoT para mais fabricantes, difundindo assim novas possibilidades de conexão.
- **Conectividade** – uma série de protocolos de rede para a Internet facilitou a conexão de sensores à nuvem e a outras coisas para transferência eficiente de dados, melhorando a comunicação entre os diferentes dispositivos.
- **Plataformas de computação na nuvem** – o aumento da disponibilidade de plataformas na nuvem permite que empresas e consumidores acessem a infraestrutura de que precisam para aumentar a escala sem precisar gerenciar tudo, confiando mais na prestação dos serviços pelos provedores de serviços em nuvem.
- **Machine learning (ou em português, aprendizado de máquina) e análise avançada** – com os avanços em *machine learning* e análise avançada de dados, além do acesso a quantidades grandes e variadas de dados armazenados na nuvem, as empresas podem obter insights de maneira mais rápida e fácil. É possível prever ações e movimentos necessários para os negócios de maneira mais assertiva. O surgimento dessas tecnologias continua a ultrapassar os limites da IoT e os dados produzidos pela IoT também alimentam essas tecnologias. Vira um ciclo de impulsionamento contínuo de desenvolvimento.
- **Inteligência artificial (IA)** – os avanços nas redes neurais trouxeram o NLP (natural-language processing, processamento de linguagem natural) aos dispositivos de IoT (como assistentes pessoais digitais Alexa, Cortana e Siri) e os tornaram atraentes, acessíveis e viáveis para uso doméstico.

Com toda essa evolução que estamos presenciando, podemos concordar que a IoT está reinventando o automóvel, permitindo carros conectados. Acessar carros remotamente, passa a ser realidade. Por exemplo, pré-aquecendo o banco do carro antes que o motorista entre nele ou chamando um carro remotamente por telefone. Com a capacidade da IoT de habilitar a comunicação dispositivo a dispositivo, os carros poderão reservar seus próprios compromissos de serviço, quando necessário.

Uma outra aplicação bastante difundida é a relacionada com questões de saúde. Vimos surgir a telemedicina e a teleconsulta, permitindo os pacientes a terem um contato com médicos sem que ambos saíssem de suas casas. Os relógios com monitoramento cardíaco e de pressão sanguínea passaram a conectar pacientes e hospitais, para o atendimento mais assertivo e rápido em caso de emergências.

Comandar as luzes e monitorar as câmeras de segurança de uma casa passou a ser algo feito de maneira remota, através de um aplicativo instalado no celular, por exemplo.

Vejam quanta evolução está ocorrendo na sociedade como um todo, estamos vivendo a história! E isso só é possível pela evolução da tecnologia e pelo desenvolvimento de softwares capazes que aproveitar esta tecnologia atendendo às demandas dos consumidores.

FINALIZANDO

Nesta etapa, discutimos sobre a importância de manter de forma organizada e útil a evolução do software. Discutimos também o caráter irrevogável das mudanças em software, ou seja, mudanças são inevitáveis para manter o software útil e atual. Se um software não está sofrendo mudanças e evoluções, é preciso analisar se esse software é útil e se está agregando valor para o negócio. Esse questionamento é válido quando paramos para pensar que se um software foi criado para agregar valor ao negócio, contribuir para gerar resultados melhores e que, como a evolução da tecnologia, as necessidades dos clientes e as tendências do mercado é algo irreversível, logo, é muito sensato que o software precise ser evoluído para acompanhar as necessidades do negócio. Concordam?

Foi exatamente esse contexto discutido ao longo desta etapa e espero que tenha feito sentido para você e te ajude a compreender o caráter de evolução contínua de todo software, buscando manter-se útil e atual para seus clientes, para o mercado e para agregar valor para o negócio.

Vamos continuar evoluindo e aprofundando nos conceitos que envolvem a Gerência de Configuração em outro momento. Nos vemos em breve.

REFERÊNCIAS

O que é IOT? **Oracle**, 19 fev. 2014. Disponível em: <https://www.oracle.com/br/internet-of-things/what-is-iot/>. Acesso em: 24 set. 2022.

PAULA FILHO, W. de P. **Engenharia de Softwares**: produtos. 4. ed. Rio de Janeiro: LTC, 2019.

PFLEEGER, S. L. **Engenharia de Software:** teoria e prática. 2. ed. São Paulo: Prentice Hall, 2004.

SBROCCO, J. H. T. de C. **Metodologias ágeis:** Engenharia de Software sob medida. 1. ed. São Paulo: Érica, 2012.

SOMMERVILLE, I. **Engenharia de software.** 10. Ed. São Paulo: Pearson, 2019.

VETORAZZO, A. de S. **Engenharia de Software**. Porto Alegre: SAGAH, 2018.