Tópico 04

Fundamentos de Sistemas de Informação

Desenvolvimento de Sistemas de Informação

1. Introdução

Em uma determinada organização, é possível a existência de vários recursos tecnológicos dando suporte a sistemas de informação ou mesmo atuando de forma isolada (por exemplo, uma rede de comunicação de dados). Em especial, os softwares. Sabemos que eles podem atuar como aplicativos ou suporte a outros sistemas (p.ex. Sistema Operacional). Ocorre que, de uma forma ou de outra, quando se fala em software, temos de pensar que sua construção passa por algumas atividades, organizadas de alguma forma, formando um processo de desenvolvimento de software. Caso contrário, o que teremos são soluções muito ruins que geram muitos erros ou mesmo não atendem aos requisitos para os quais foram demandadas. Desde já, entenda que você, isso mesmo, você, caro aluno, será um profissional que utilizará a Engenharia de Software em vários momentos de sua vida acadêmica e profissional.

Assim, precisamos utilizar de um arsenal de ferramentas, processos, técnicas e metodologias para se obter bons produtos de software e, assim, atender aos requisitos de negócio de nossa organização. Estamos atuando, dessa forma, dentro de uma das áreas da Ciência da Computação, chamada de **Engenharia de Software**. Primeiramente, vejamos os conceitos de software e Engenharia de Software. Pressman (2011, p 29) diz que

o Software de computador é o produto que profissionais de software desenvolvem e ao qual dão suporte no longo prazo. Abrangem programas executáveis em um computador de

qualquer porte ou arquitetura, conteúdos (apresentados à medida que os programas são executados), informações descritivas tanto na forma impressa (hard copy) como na virtual, abrangendo praticamente qualquer mídia eletrônica.
A engenharia de software abrange um processo, um conjunto de métodos (práticas) e um leque de ferramentas que possibilitam aos profissionais desenvolverem software de altíssima qualidade. Chamamos de Engenheiros de Software os profissionais atuantes nessa área da Computação.

Cria-se software para computadores da mesma forma que qualquer produto bem-sucedido: aplicando-se um processo adaptável e ágil que conduza a um resultado de alta qualidade, atendendo às necessidades daqueles que usarão o produto. Aplica-se uma abordagem de engenharia de software Pressman (2011, p. 29).

Então, o que temos aqui é o emprego dos recursos da Engenharia de Software para a obtenção (desejo) de produtos de software de qualidade (o conceito de qualidade será definido mais à frente). Veremos como surge a necessidade de desenvolvimento de um software e, partir dela, quais recursos (e como) estão à nossa disposição e deverão ser utilizados.
Importante ressaltar que há um conjunto grande de recursos da engenharia de software, de forma que, para cada produto a ser produzido, há sempre uma ou mais formas (processos) de fazê-lo.

No texto aqui referenciado, está um excelente material adicional para seus estudos que irá possibilitar um maior entendimento, do ponto de vista prático, de desenvolvimento de software. O foco de interesse do documento é o elemento software de sistemas de

software. O texto especifica melhor algumas normas e padrões adotados na literatura para desenvolvimento de software.

Clique aqui.

Boa leitura!

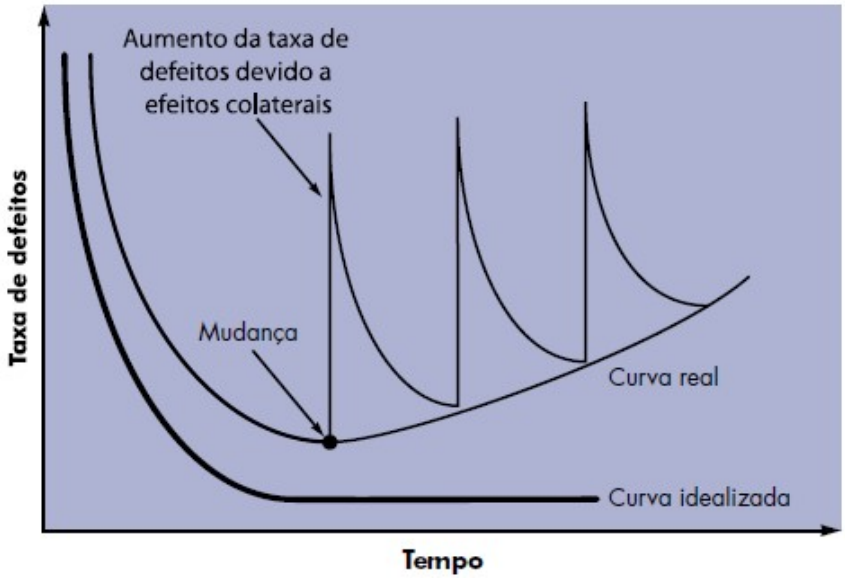
2. Engenharia De Software, Ciclo De Vida E Construção De Software

A grande questão inicial é: por que a Engenharia de Software? Vejamos um breve histórico para entendermos os problemas graves existentes antigamente quando se falava em desenvolvimento de software. De acordo como Prado (2014, p. 122):

> *"Na década de 1970, o desenvolvimento do software realizado pelas organizações de TI era desorganizado, desestruturado e sem planejamento, o que gerava um produto de má qualidade, prazos estourados, inexistência de documentação, que, muitas vezes, não atendia as necessidades do cliente. Essa era ficou conhecida como a* ***crise do software****. Nesse cenário, surgiu a necessidade de tornar o desenvolvimento do software um processo estruturado, planejado, entregue no prazo e no orcamento, levando ao nascimento da* ***engenharia de software****".*

O que se observa é que, nos anos anteriores à adoção da Engenharia de Software, havia muitos (e graves) problemas com os produtos de software existentes. De fato, conforme Pressman (2011, p. 30), naquela época (40 ou 50 anos atrás), ninguém poderia imaginar que milhões de programas de computador

teriam de ser corrigidos, adaptados e ampliados à medida que o tempo passasse. A realização dessas atividades de "**manutenção**" absorve mais pessoas e recursos do que todo o esforço aplicado na criação de um novo software. O que se desenha é a seguinte situação: softwares mal desenvolvidos geram muitos problemas e necessidades de manutenções futuras, envolvendo grandes custos para isso ou simplesmente deixando de ser utilizados (perda do dinheiro investido). Para sanar isso, conforme aumenta a importância do software, a comunidade da área tenta desenvolver tecnologias que tornem mais fácil, mais rápido e mais barato desenvolver e manter programas de computador de alta qualidade. Assim, foi preciso compreender as particularidades envolvidas com os softwares e sua forma de desenvolvimento.



Relação tempo x quantidade de defeitos em software

Na figura anterior, podemos perceber o comportamento dos defeitos existentes em um software. A curva mais realçada e inferior (Curva idealizada) representa uma queda na quantidade de defeitos na medida em que o tempo passa, o que seria ideal. Ocorre que, por suas particularidades, o software segue a "curva real". Nessa curva, podemos visualizar que, após certo tempo, ocorre uma mudança (p.ex., mudança de um requisito de negócio em virtude de uma nova lei), forçando uma manutenção

nesse produto. Era de se esperar que essa curva continuasse “seu caminho”, mas a intervenção realizada pode gerar outros defeitos (efeito colateral) e novos defeitos deverão ser corrigidos. Esse processo se repete enquanto o software existir.

Conforme Pressman (2011, p. 32), um software possui as seguintes características:

- Software é desenvolvido ou passa por um processo de engenharia; ele não é fabricado no sentido clássico;
- Software não “se desgasta”, isto é, diferentemente de um hardware (que desgasta seus equipamentos com o tempo), Software não é suscetível aos males ambientais. Quando um hardware tem defeito, é substituído por outro igual. Isso não ocorre com o software;
- Embora a indústria caminhe para a construção com base em componentes, a maioria dos softwares continua a ser construída de forma personalizada (sob encomenda), pois sempre haverá necessidades específicas de cada organização e que devem ser incorporadas ao software com customizações e algumas melhorias.

Para a produção de software (produto) com qualidade desejada, precisamos utilizar uma metodologia (método, estratégia) de construção organizada, que pode ser ajustada e melhorada com o tempo. Em Engenharia de Software, chamamos isso de **Processo de Desenvolvimento de Software** (ou, simplesmente, processo de software). Para Pressman (2011, p. 40), **Processo** é um conjunto de atividades, ações e tarefas realizadas na criação de algum produto de trabalho (*work product*). Uma atividade esforça-se para atingir um objetivo amplo (por exemplo, comunicar-se com os interessados) e é utilizada independentemente do campo de aplicação, do tamanho do projeto, da complexidade de esforços ou do grau de rigor com que a engenharia de software será aplicada.
Uma ação (por exemplo, projeto de arquitetura) envolve um

conjunto de tarefas que resultam num artefato de software fundamental (por exemplo, um modelo de projeto de arquitetura). Uma tarefa se concentra em um objetivo pequeno, porém, bem definido (por exemplo, realizar um teste de unidades) e produz um resultado tangível.

Pressman (2011, p.40) afirma que o processo é uma abordagem adaptável que possibilita às pessoas (a equipe de software) realizar o trabalho de selecionar e escolher o conjunto apropriado de ações e tarefas. A intenção é a de sempre entregar **software dentro do prazo e com qualidade suficiente** para satisfazer àqueles que patrocinaram sua criação e àqueles que irão utilizá-lo. Sendo assim, é fundamental o alinhamento (aderência) do processo de desenvolvimento de software aos processos de negócio, isto é, não se pode construir um software sem o conhecimento das particularidades do contexto de negócio onde ele irá operar. Nesse sentido, Laudon (2000, p.35) define um **processo de negócio** (ou processo organizacional) como sendo um conjunto de atividades logicamente relacionadas que definem como tarefas organizacionais específicas serão executadas. Refere-se, ainda, às maneiras únicas através das quais o trabalho, as informações e o conhecimento são coordenados em determinada empresa.

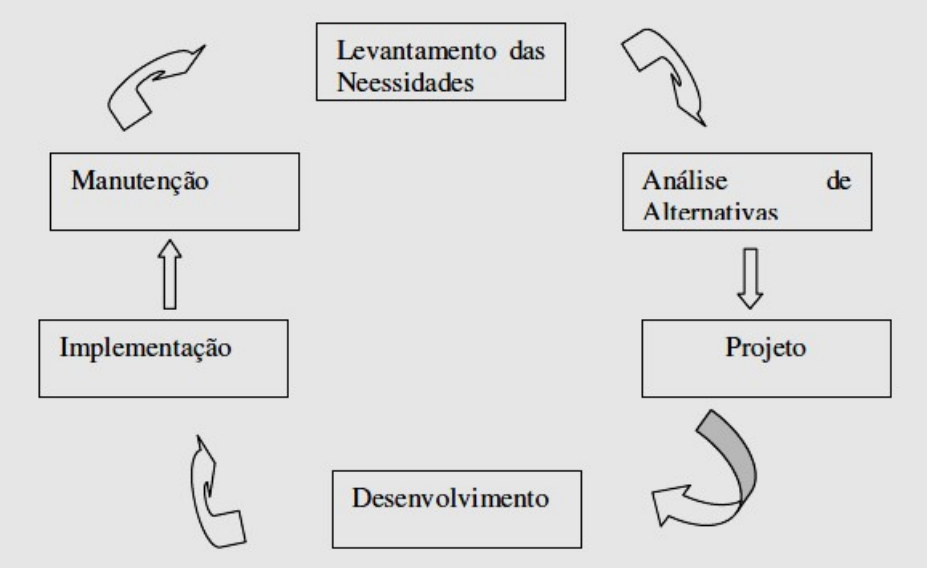
Os **mitos** criados em relação ao software — crenças infundadas sobre o software e sobre o processo usado para criá-lo — remontam aos primórdios da computação. Os mitos possuem uma série de atributos que os tornam insidiosos. Por exemplo: eles parecem ser, de fato, afirmações razoáveis (algumas vezes, contendo elementos de verdade), têm uma sensação intuitiva e frequentemente são promulgados por praticantes experientes “que entendem do riscado”.

> Mito 1: já temos um livro que está cheio de padrões e procedimentos para desenvolver software. Ele não supre meu pessoal com tudo que eles precisam saber? Mito 2: se o cronograma atrasar, poderemos acrescentar mais programadores e ficar em dia (algumas vezes, denominado conceito da “horda mongol”). Mito 3: se eu decidir terceirizar o projeto de software, posso simplesmente relaxar e deixar essa empresa realizá-lo
>
> PRESSMAN (2011).

Ciclo De Vida De Software

Em geral, podemos entender que um produto de software irá passar por algumas fases em sua existência, desde sua concepção até sua extinção. Chamamos isso de **Ciclo de Vida de um Software**, o qual está associado a um processo de desenvolvimento de software. Em outras palavras, podemos entender que, ao se desenvolver um software, ele irá passar por um ciclo, organizado em etapas/fases, as quais estarão organizadas de alguma forma, a depender de como o modelo adotado precisa evoluir. Por exemplo: um processo de desenvolvimento pode gastar pouca energia na fase/etapa de projeto, focando nos requisitos. Já outro modelo pode focar na implementação de grande parte de seus esforços. A figura a seguir ilustra esse conceito.

Ciclo de vida do software



Modelo geral de um ciclo de vida de um produto de software.

Este ciclo compreende algumas fases ou etapas (PUCRS, 2020, p. 01):

1. O levantamento das necessidades também chamado de análise de requisitos, que identifica as necessidades de informações da organização;
2. A análise de alternativas consiste na identificação e avaliação de sistemas alternativos;
3. Projeto trata da construção das especificações detalhadas para o projeto selecionado. Essas especificações incluem o projeto das interfaces, banco de dados, características físicas do sistema, tais como número, tipos e localizações das estações de trabalho, hardware de processamento, o cabeamento e os dispositivos de rede;
4. Desenvolvimento, que inclui o desenvolvimento ou aquisição do software, a provável aquisição do hardware e o teste do novo sistema;
5. Implementação, que ocorre após o sistema ter passado satisfatoriamente por testes de aceitação. O sistema é transferido do ambiente de desenvolvimento para o ambiente de produção. O sistema antigo (se existir) deve migrar para o novo;
6. Manutenção refere-se a todas as atividades relacionadas a um sistema depois que ele é implementado. Deve incluir atividades tais como a correção de software que não funcione corretamente e a adição de novos recursos aos sistemas em resposta às novas demandas dos usuários.

Uma variação das etapas/fases descritas acima encontra-se em Pressman (2011, p. 40), descritas através de **5 atividades**

principais:

- **Comunicação:** a intenção é compreender os objetivos das partes interessadas (gestores, usuário comuns, etc.) para com o projeto e fazer o levantamento das necessidades que ajudarão a definir as funções e características do software;
- **Planejamento:** a atividade de planejamento cria um "mapa" que ajuda a guiar a equipe na sua jornada. O mapa — denominado plano de projeto de software — define o trabalho de engenharia de software, descrevendo as tarefas técnicas a serem conduzidas, os riscos prováveis, os recursos que serão necessários, os produtos resultantes a serem produzidos e um cronograma de trabalho;
- **Modelagem**: cria-se um "esboço" da coisa, de modo que se possa ter uma ideia do todo — qual será o seu aspecto em termos de arquitetura, como as partes constituintes se encaixarão e várias outras características;
- **Construção**: esta atividade combina geração de código (manual ou automatizada) e testes necessários para revelar erros na codificação;
- **Emprego**: o software (como uma entidade completa ou como um incremento parcialmente efetivado) é entregue ao cliente, que avalia o produto entregue e fornece feedback, baseado na avaliação.

Outro conceito muito importante aplicado ao desenvolvimento de software são as **Linguagens de Programação**. Conforme descrito por Gudwin (2020, p.3):

> *"Para se implementar um algoritmo em um computador, é necessário descrevê-lo de uma forma que o computador esteja apto a executá-lo. Essa descrição é feita por intermédio de uma "linguagem de programação". O próprio conjunto de instruções de um processador pode ser entendido como uma "linguagem de programação".*

> *Entretanto, essa linguagem normalmente não é a mais adequada para a descrição de um programa, uma vez que os algoritmos necessários podem ser sofisticados, e essa linguagem primitiva, também chamada de "linguagem de máquina", não é nem um pouco amigável ao programador, demandando um esforço muito grande na elaboração de programas mais complexos. Sendo assim, foram desenvolvidas, ao longo da história da computação, diversas "linguagens de programação", cada qual, a seu tempo, introduzindo facilidades e recursos que foram tornando a tarefa de programar mais fácil e menos suceptível a erros.* ***Atualmente, com as linguagens visuais (também chamadas por alguns de linguagens de quarta geração), programar deixou de ser uma arte restrita a um grupo de indivíduos, para tornar-se uma ferramenta a mais dentro do escopo do usuário comum****".*

Todo projeto de desenvolvimento de software precisa pensar em uma linguagem de programação mais apropriada para os tipos de entrega que irão ocorrer. Por exemplo: um software que irá atuar na Internet, em seu comércio eletrônico, não pode utilizar uma mesma linguagem que irá programar circuitos digitais de um computador. Elas têm propósitos diferentes. De acordo com Gudwin (2020, p.3), as linguagens de programação podem ser divididas dentro uma taxonomia, sendo: de Baixo Nível, Não-Estruturadas, Procedurais, Funcionais, Orientadas a Objeto, Específicas a Aplicações e Visuais. Assim, para os algoritmos que iremos construir, se faz necessária a escolha da linguagem de programação mais apropriada. Há um vasto conjunto de linguagens de programação disponíveis, como C, C#, C++, Fortran, Prolog, SQL, Java, R, Python, etc. Um profissional de tecnologia da informação deve conhecer sempre, pelo menos, 2 ou 3 dessas linguagens para poder encarar problemas com maior resiliência.

Feita essa importante consideração quanto à ferramenta de construção de algoritmos (as Linguagens de Programação), é importante ressaltar que a decisão de qual modelo de desenvolvimento seguir dependerá da análise de alguns fatores, como as características técnicas e funcionais do produto desejado, dos recursos disponíveis para desenvolvimento, da cultura organizacional para projetos, da capacidade técnica das pessoas envolvidas no desenvolvimento, dentre outros. A figura a seguir, definida em Pressman (2011, p. 54), apresenta variações no desenvolvimento dessas atividades:

Variações em atividades de desenvolvimento de software.

No item a) da figura, temos um processo linear, mais rígido e que pode ser utilizado em ambientes cujos requisitos de negócio são mais estáveis, sujeitos a poucas mudanças e onde o produto final somente será entregue ao término de todas as atividades. Um dos modelos mais conhecidos com essas características é o Modelo Cascata (*Waterfall*). No item b), há algumas iterações (refluxos) devido a uma maior possibilidade, em algum momento, dos requisitos de negócio mudarem ou, mesmo ainda, não se encontrarem em um ponto ideal de entendimento. À medida que vão sendo identificados, podem evoluir e, junto com eles, a complexidade do produto final, que será entregue também ao final do desenvolvimento. No item c), temos um modelo de desenvolvimento em um fluxo evolutivo, que, diferentemente dos modelos anteriores, já permite uma entrega parcial de um produto ao final de um primeiro ciclo do fluxo. Essa abordagem é muito interessante, pois já dá a possibilidade de os usuários conviverem com uma solução parcial, a fim de que possam validar seus requisitos e entenderem melhor seus objetos e alinhamento com o ambiente de negócios. É essa metodologia que vem sendo amplamente utilizada no cenário atual e que envolve as abordagens chamadas de Ágeis, como **SCRUM** e **TDD** – *Test Driven Development* / Desenvolvimento orientado a teste, os quais serão abordados com maiores detalhes em outras disciplinas de seu curso.

Uma questão importante a respeito de um processo de desenvolvimento de software é **como selecionar um processo adequado de desenvolvimento de software.** A escolha de um modelo de desenvolvimento de software deverá levar em conta as características da aplicação, isso é, domínio (abrangência) do problema, tamanho, complexidade, etc. Assim, se nosso produto estará voltado para um pequeno supermercado, com volume de compras muito pequeno, poucos usuários e sem grandes infraestruturas de computação (servidores modernos, redes de computadores complexas, etc.), podemos pensar num modelo mais simples,

que considera o ambiente (negócio) mais estático, sem grandes mudanças com o passar do tempo. Agora, se estamos pensando em uma solução para uma grande prefeitura gerenciar seu sistema de saúde (atendimentos e internações), nossa complexidade aumenta, tanto do ponto de vista do volume e diversidade de dados, quanto de usuários e regras de negócio (muitas das vezes, impostas por Leis, que mudam constantemente).

Nesse cenário, devemos pensar num modelo de desenvolvimento de software mais robusto, que incorpore mudanças ambientais grandes sem grandes impactos na solução em operação. Como vimos nos exemplos acima, também é importante considerar a Tecnologia a ser adotada na sua construção (paradigma de desenvolvimento, linguagem de programação, mecanismo de bancos de dados, etc.) e sua forma de organização (se é terceirizada ou própria, por exemplo), em que o produto será desenvolvido (dentro da organização ou fora) e o perfil da equipe de desenvolvimento (nível técnico, experiência, etc.).

Um problema fundamental da qualidade de software é definir claramente os objetivos que se pretende atingir com um projeto. Para fazê-lo, é preciso enumerar características desejáveis do software, que idealmente devem incluir valores quantitativos e qualitativos a serem respeitados. O material aqui referenciado apresenta diversas formas de se melhorar a qualidade do software, através de normas de qualidade, de técnicas de inspeção, testes de software e de modelos de melhoria do processo de testes.

Clique aqui

Fonte: ARAÚJO (2020)

3. Conceitos De Qualidade E Manutenção De Software

Como todo produto que é produzido, espera-se que um software atenda a um mínimo de critérios de qualidade. O usuário comum espera, no mínimo, a não ocorrência do que eles chamam de "bugs", ou erros de execução do software. Mas, será que qualidade de software está limitada somente a "bugs"? A resposta é não. Primeiramente, não podemos nos esquecer de que o software é um PRODUTO, gerado por um processo de desenvolvimento, conforme vimos nas seções anteriores. Mas, o que seria então **qualidade do software**?

De acordo com Sommerville (2011, p. 457), a avaliação da qualidade de software é um processo subjetivo, em que a equipe de gerenciamento de qualidade precisa usar seu julgamento para decidir se foi alcançado um nível aceitável de qualidade. **Ela precisa considerar se o software é adequado para sua finalidade (funcionalidades) ou não**. Em Pressman (2011, p.359), a qualidade de um projeto engloba o grau de atendimento às funções e características especificadas no modelo de requisitos. A qualidade de conformidade focaliza o grau em que a implementação segue o projeto e o sistema resultante atende suas necessidades e as metas de desempenho.

Para Sommerville (2011, p. 457), a **qualidade subjetiva de um sistema de software** baseia-se, também, em grande parte, em suas características não funcionais (tempo de processamento, segurança, facilidade de uso, etc.). Isso reflete a experiência prática do usuário — se a funcionalidade do software não é a esperada, os usuários frequentemente apenas contornam esse problema e encontram outras maneiras de fazer o que

querem. No entanto, se o software for muito lento ou não confiável, será praticamente impossível aos usuários atingir seus objetivos.

Se produzimos um sistema de software de péssima qualidade, perdemos porque ninguém irá querer comprá-lo. Se, por outro lado, gastamos um tempo infinito, um esforço extremamente grande e grandes somas de dinheiro para construir um software absolutamente perfeito, então, isso levará muito tempo para ser completado, e o custo de produção será tão alto que iremos à falência. Ou perdemos a oportunidade de mercado ou então simplesmente esgotamos todos os nossos recursos. Dessa maneira, os profissionais desta área tentam encontrar aquele meio-termo mágico onde o produto é suficientemente bom para não ser rejeitado logo de cara, como, por exemplo, durante uma avaliação, mas também não é o objeto de tamanho perfeccionismo e trabalho que levaria muito tempo ou que custaria demasiadamente para ser finalizado.

PRESSMAN (2011)

Importante entender que **o processo de desenvolvimento usado tem uma influência significativa sobre a qualidade de software** e que bons processos são mais suscetíveis de conduzir o software de boa qualidade. **O gerenciamento e a melhoria de qualidade de processo podem gerar softwares com menos defeitos.** Assim, o estudo da qualidade de software pode se voltar ao produto ou ao processo de desenvolvimento.

Um dos pontos positivos na busca da qualidade do software é a utilização de modelos, normas e padrões de qualidade de software (produto e processo). Conforme Araújo (2020, p.2), os **modelos de qualidade** objetivam avaliar o produto de software, segundo diferentes aspectos baseados na visão do usuário. Para padronizar internacionalmente as características de implementação do software, foram criadas algumas normas, como a **ISO 9126**, que possui um **conjunto de atributos** que têm impacto na capacidade do software de manter o seu nível de desempenho dentro de condições estabelecidas por um dado período de tempo, como Funcionalidade, Confiabilidade e Usabilidade, e a **ISO 12119** (norma foi publicada em 1994) que trata da **avaliação de pacotes de software**, também conhecidos como "software de prateleira". Ambas as normas focam na qualidade do produto de software.

Quanto aos processos de desenvolvimento de software, citamos modelos de referência importantes, como o CMMi e o MPS.Br. De acodo com Sommerville (2011, p. 504), o **CMMi** (*Capability Maturity Model – Modelo de Maturidade de Capacidade*) é uma tentativa de integrar a multiplicidade de modelos de capacidade com base na noção de maturidade de processos (incluindo seus próprios modelos). O framework do CMMI tem duas instâncias (formas de implementação), por **estágio e continua**, e destina-se a ser um framework de melhoria de processos com ampla aplicabilidade em uma gama de empresas. O CMMI é estruturado em cinco níveis de maturidade, de 1 a 5, em que o nível 1 é o menos maduro e o nível 5 é o mais maduro. Cada nível de maturidade, com exceção do nível 1, é composto de várias áreas de processo (*process areas* – PAs).

Já o modelo **MPS.Br** (Melhoria de Processo de Software – Brasil) é uma iniciativa brasileira para definição de um modelo de avaliação de qualidade de processo de software, desenvolvimento por liderança da Associação para Promoção da Excelência do Software Brasileiro (SOFTEX). Conforme Softex

(2020), seu objetivo é Melhoria de Processo de Software e Serviços; além disso, que o modelo MPS seja **adequado ao perfil de empresas com diferentes tamanhos e características, públicas e privadas, embora com especial atenção às micro, pequenas e médias empresas**. Também se espera que o modelo MPS seja compatível com os padrões de qualidade aceitos internacionalmente e que tenha como pressuposto o aproveitamento de toda a competência existente nos padrões e modelos de melhoria de processo já disponíveis. O MPS define sete níveis de maturidade: A (Em Otimização), B (Gerenciado Quantitativamente), C (Definido), D (Largamente Definido), E (Parcialmente Definido), F (Gerenciado) e G (Parcialmente Gerenciado).

Outro item importante na busca pela qualidade de software é a realização de **Testes**. De acordo com Pressman (2011, p. 402), Teste é um conjunto de atividades que podem ser planejadas com antecedência e executadas sistematicamente. Por essa razão, deverá ser definido para o processo de software um modelo (template – modelo) para o teste — um conjunto de etapas no qual pode-se colocar técnicas específicas de projeto de caso de teste e métodos de teste. De forma mais objetiva, pensemos nos Testes contendo duas atividades macro, chamadas de **verificação** e **validação (V&V).** Verificação refere-se ao conjunto de tarefas que garantem que o software implementa corretamente uma função específica. Validação refere-se a um conjunto de tarefas que asseguram que o software foi criado e pode ser rastreado segundo os requisitos do cliente. Importante ressaltar que, conforme Pressman (2011, p. 402), **o teste proporciona o último elemento a partir do qual a qualidade pode ser estimada e, mais pragmaticamente, os erros podem ser descobertos**.

Após as devidas ações voltadas para a qualidade do software (produto), ele passará a operar no ambiente real da empresa, no

ambiente de “produção”. Dizemos que o software estará em **Manutenção**. De acordo com Sommerville (2011, p. 29):

> *“Poucos sistemas de software são completamente novos, e faz muito mais sentido ver o desenvolvimento e a manutenção como processos contínuos. Em vez de dois processos separados, é mais realista pensar na engenharia de software como um processo evolutivo, no qual o software é constantemente alterado durante seu período de vida em resposta as mudanças de requisitos e as necessidades do cliente. A mudança aumenta os custos de desenvolvimento de software, porque, geralmente, significa que o trabalho deve ser refeito. Isso é chamado retrabalho”.*

Para Sommerville (2011, p. 30), existem duas abordagens que podem ser adotadas para a redução de custos de retrabalho:

1. **Prevenção de mudanças**, em que o processo de software inclui atividades capazes de antecipar as mudanças possíveis antes que seja necessário qualquer retrabalho. Eles podem experimentar o protótipo e refinar seus requisitos antes de se comprometer com elevados custos de produção de software;
2. **Tolerância a mudanças**, em que o processo foi projetado para que as mudanças possam ser acomodadas a um custo relativamente baixo. Isso normalmente envolve alguma forma de desenvolvimento incremental.

O vídeo a seguir apresenta, de forma simples, alguns conceitos de qualidade de produto e processo de software. O vídeo faz boas relações de produtos de software e produtos cotidianos. Preste bastante atenção nos termos utilizados relacionados à Engenharia de Software.

4. Conclusão

Ao final desta unidade, fizemos uma boa caminhada para entendimento de como produtos de software são produzidos, avaliados por sua qualidade e entregues (e evoluídos). Vimos que as atividades de desenvolvimento de softwares estão abarcadas na área da Ciência da Computação, chamada de Engenharia de Software. É muito importante que nós, profissionais da área de desenvolvimento de sistemas de informações, apliquemos suas técnicas, ferramentas e processos. Aliás, processo de desenvolvimento de software foi visto como um conjunto de atividades técnicas, geralmente, agrupadas em fases ou etapas. Não perca de vista os conceitos de Qualidade estudados, pois eles irão acompanhá-lo em toda sua vida profissional da área de Sistemas de Informação.

5. Referências

ARAÚJO, R.B.S., CHALEGRE, V.C. **Qualidade de Produtos de Software. Notas de aula**. Disponível em <https://www.cin.ufpe.br/~processos/TAES3/Livro/00-LIVRO/10-Qualidade%20de%20Produtos%20de%20Software-v6_CORRIDIDO.pdf>

GUDWIN, R,R. Linguagens de Programação: Mini e Microcomputadores: Software – Notas de Aula. UNICAMP. Campinas – SP. Disponível em <ftp://vm1-dca.fee.unicamp.br/pub/docs/vonzuben/ea877_1S99/ling_prog.pdf>.

LUCENA, F.N.. **Vamos entender construção de software?**. Disponível em <https://docs.google.com/document/d/1WCAc8bKjoVDyXXav7zU9x44TPUwsALdKtA1qzb_nVYw/edit>.

LAUDON, K.C., L., J.P.. **Sistemas de Informação Gerenciais**. 9 ed. São Paulo – SP: Editora Pearson, 2011.

PRADO, E., ARAÚJO, L., ORNELAS, R.. **Fundamentos de Sistemas de Informação**. Ed. 1. Editora Elsevier. São Paulo – SP, 2014.

PRESSMAN, R.S. **Engenharia de Software: uma abordagem profissional**. Ed.7. Editora McGrawHill. Porto Alegre – ES, 2011.

PUCRS. **O Ciclo de Vida do Desenvolvimento de Sistemas – Notas de Aula**. Disponível em <http://www.pucrs.br/edipucrs/online/projetoSI/6-Engenharia/O_Ciclo_de_Vida_do_Desenvolvimento_de_Sistemas.pdf>.

SOFTEX. **MPS.BR – Melhoria de Processo do Software Brasileiro**. Disponível em <https://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Software_2012-c-ISBN-1.pdf>.

SOMMERVILLE, I. **Engenharia de Software**. Ed. 9. Editora Pearson. São Paulo – SP. 2011.

YouTube. (2014). Wellington Duraes. **Qualidade de Software — Webcast de Wellington Durães.** 11min49. Disponível em:

< https://www.youtube.com/watch?v=n8sAGdxmsaQ>.

Parabéns, esta aula foi concluída!

O que achou do conteúdo estudado?

Péssimo Ruim Normal Bom Excelente

Deixe aqui seu comentário

Mínimo de caracteres: 0/150

Enviar