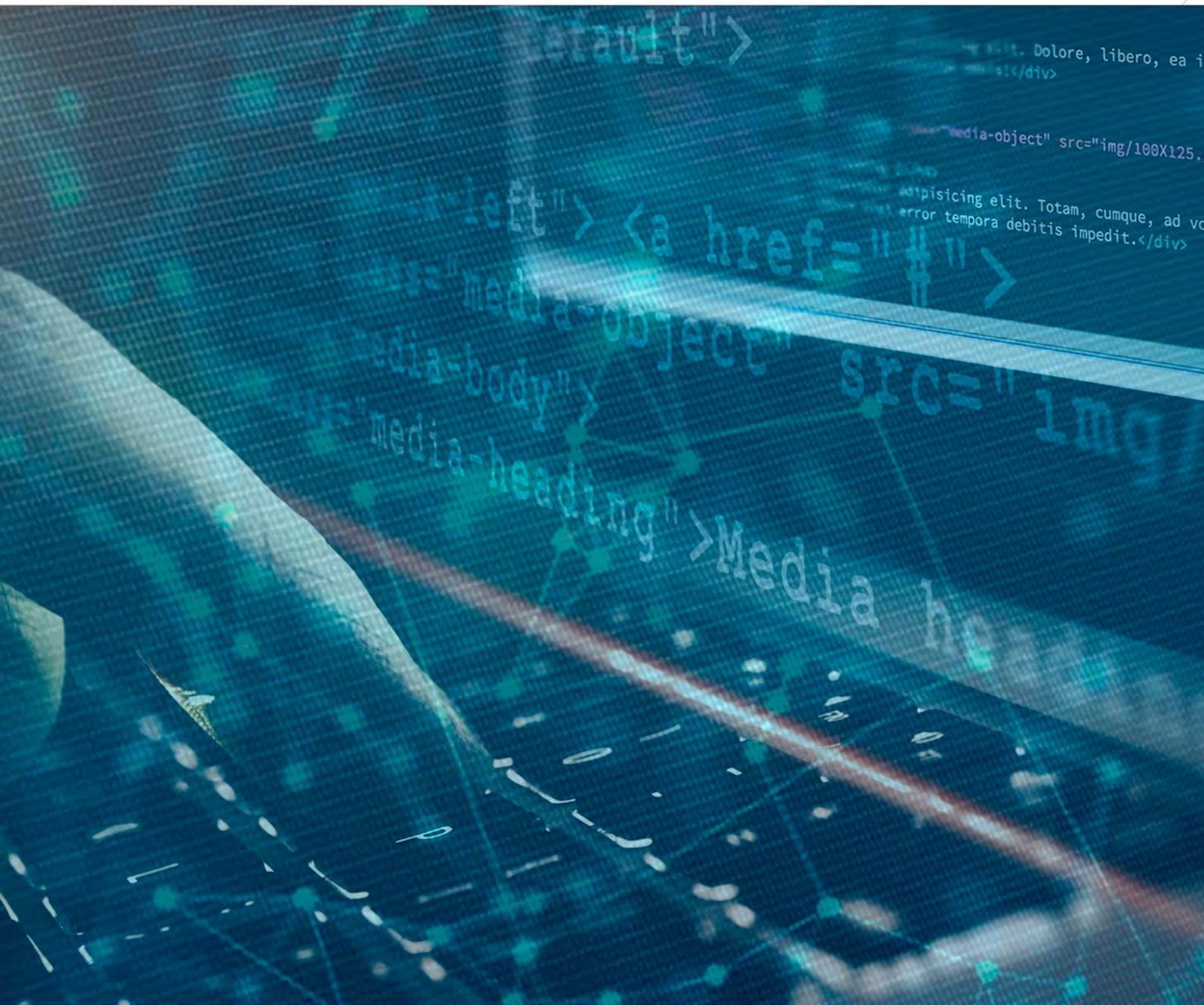


1

TEXTO BASE

ENGENHARIA DE SOFTWARE



Texto base

1

Elementos e Características de um software

Prof. João de Deus Freire Junior

Resumo

Antes de estudar a disciplina e o processo de Engenharia de Software, é muito importante compreender os principais elementos e características de um software. Cada elemento de um software é essencial para a operação, evolução e manutenção do software. Entender as características do software possibilita a compreensão de sua natureza peculiar, seu processo de fabricação e tempo de vida. Com esse conhecimento será possível ter um melhor aprendizado da Engenharia de Software.

1.1. Introdução

Como podemos definir um software? Quais as diferenças entre software e hardware? Como um software é formado e quais são suas principais características? Todas essas perguntas serão respondidas nesta aula. Essas informações fornecem um conhecimento abrangente do que é um software e a base para entendimentos de conceitos relacionados a engenharia de software.

1.2. Diferenças entre hardware e software

O hardware é toda a parte física de um computador ou outro dispositivo computacional. Ele permite que as instruções enviadas pelos programas sejam executadas, que os usuários interajam com as aplicações e, entre outras coisas, que dados e informações sejam armazenados. Exemplos de hardware são: mouse, teclado, fones de ouvido, monitores e etc. Devido a sua natureza física o hardware se desgasta ao longo do tempo o que o torna passível de reparação ou inutilização.



Figura 1.1. Notebook, Celular e Mouse - Exemplos de Hardware. Disponível em:
<https://unsplash.com/>



Figura 1.2. Processador em placa-mãe - Exemplos de Hardware. Disponível em:
<https://unsplash.com/>

Em contrapartida, o software é a parte lógica que faz com que os computadores ou outros dispositivos computacionais funcionem. O software fornece instruções para o hardware de como ele deve funcionar e quais funções executar. Exemplos de software são: aplicativos para dispositivos móveis, sistemas operacionais, processadores de textos, jogos e diversos outros programas de computador. Pela sua natureza lógica, o software não se desgasta com o tempo, mas pode se deteriorar.



Figura 1.3. Tela de Celular exibindo aplicativos - Exemplos de Software.
Disponível em: <https://unsplash.com/>



Figura 1.4. Área de Trabalho de um notebook exibindo programas - Exemplos de Software. Disponível em: <<https://unsplash.com/>>.

1.3. O que é um software?

O software pode ser melhor definido pelas partes que o formam. Segue uma definição de software que segue essa lógica:

Software consiste em: (1) instruções (programas de computador) que, quando executadas, fornecem características, funções e desempenho desejados; (2) estruturas de dados que possibilitam aos programas manipular informações adequadamente; e (3) informação descritiva, tanto na forma impressa como na virtual, descrevendo a operação e o uso dos programas. (PRESSMAN, 2011, p. 32)

Esta definição deixa claro os elementos que formam um software, que são: instruções, estrutura de dados e informação descritiva (documentação). Qualquer um desses elementos são essenciais para o funcionamento, evolução e manutenção do software.

O software é um produto produzido por profissionais especializados, ele pode dar suporte a operações computacionais, de negócios, médicas e etc. Ele pode ser disponibilizado gratuitamente para uso em massa, pode servir para divulgação de outros produtos, pode ser o diferencial competitivo de grandes corporações ou pode ser a única interação entre clientes e a empresa.

O software está cada vez mais essencial em nosso dia a dia. Não vivemos um dia sequer sem operar ou consumir informação de algum software.

1.4. Elementos de um software

Os principais elementos de um software são: instruções, estrutura de dados e documentos.

As instruções, quando executadas, produzem a função e desempenho desejados pelo software. As instruções dizem o que os dispositivos e máquinas devem fazer. O conjunto de instruções de um software é seu código-fonte. Este código pode ser escrito com uso de linguagem de programação como Java, Python e etc.

As estruturas de dados possibilitam que os programas manipulem adequadamente os dados e produzam informações. As estruturas de dados armazenam dados de forma eficiente. Elas podem ser: filas, pilhas, árvores binárias, objetos relacionais e etc.

Os documentos descrevem a operação e uso do programa. A documentação do software detalha a estrutura do software, sua arquitetura, componentes, funcionamento e regras. A documentação bem elaborada é essencial para evolução e manutenção do software.

1.5. Características de um software

1.5.1. Software é desenvolvido ou passa por um processo de engenharia; ele não é fabricado no sentido clássico.

Tanto software como hardware requerem a construção de um produto, porém, seus métodos de fabricação são totalmente diferentes. Ambas as atividades são dependentes de pessoas, mas a relação entre pessoas envolvidas e trabalho realizado é completamente diferente nesses produtos. Os custos de software concentram-se no processo de engenharia. Isso significa que projetos de software não podem ser geridos como se fossem projetos de fabricação. (PRESSMAN, 2011)

1.5.2. O Software não “se desgasta”, mas, se deteriora.

À medida que o tempo passa, os componentes de um hardware sofrem os efeitos cumulativos de poeira, vibração, impactos, temperaturas extremas e vários outros males ambientais ele começa a desgastar-se.

O software não é suscetível aos males ambientais que fazem com que o hardware se desgaste. Portanto, ele não se desgasta. Mas, o ambiente de negócios, a tecnologia, os processos, as organizações, sociedades, leis, regras mudam e todos esses aspectos podem fazer com que o software fique obsoleto e se deteriore.

De acordo com Pressman (2011), outro aspecto de desgaste ilustra a diferença entre hardware e software. Quando um componente de hardware se desgasta, ele é substituído por uma peça de reposição. Não existem peças de reposição de software. Cada defeito de software indica um erro no projeto ou no processo pelo qual o projeto foi traduzido em código-fonte. Portanto, as tarefas de manutenção de software, que envolvem solicitações de mudanças, implicam em complexidade maior do que a de manutenção de hardware.

1.5.3. Embora a indústria caminhe para a construção com base em componentes, a maioria dos softwares continua a ser construída de forma personalizada (sob encomenda)

À medida que a disciplina de Engenharia de Software evolui, cada vez mais componentes reutilizáveis são criados. Muitas empresas inclusive comercializam componentes ou softwares prontos chamados “pacotes” para outras empresas. Desta

forma, as organizações não precisam construir aplicações ou componentes de software já existentes no mercado, elas só precisam pagar pela aquisição ou uso dos serviços.

Porém, a maioria das aplicações ainda são construídas de forma personalizada pelas organizações por possuírem elevado valor agregado, sendo inclusive o principal diferencial dessas empresas. Isto é muito real nas empresas/plataformas digitais como: Uber, Nubank e Airbnb.

1.5. Referências

PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.

Figuras 1, 2, 3 e 4, Site Unsplash. Disponível em <<https://unsplash.com/>>. Acesso em: 07 jan. 2021.

2

TEXTO BASE

ENGENHARIA DE SOFTWARE

Texto base

2

Curva de Falhas de Software e Hardware e Tipos de Software

Prof. João de Deus Freire Junior

Resumo

Para melhor entendimento da importância do software, sua abrangência e como somos dependentes dele é necessário que conheçamos os diversos tipos de software, suas implicações e usos. O estudo da natureza e ciclo de vida de software e hardware exibe as diferenças entre os dois e como as falhas se apresentam para cada um. Esse entendimento guia o comportamento e práticas dos desenvolvedores de software.

1.1. Introdução

Como me relaciono com o software no meu dia-a-dia? Os meus aplicativos de celular também são softwares? Quais são os tipos de software? Terei tantos problemas com o software quanto tenho com o hardware? Um software é alterado com a mesma frequência que um hardware? Todas essas perguntas serão respondidas nesta aula. Essas informações ajudam a entender como o software está inserido em nosso dia-a-dia, no mundo de negócios e organizações em geral, além de entender o ciclo de vida peculiar do software.

1.2. Curva de defeitos de hardware

A curva de defeitos de um hardware tem um comportamento bem diferente da curva de defeitos de um software. Ela está relacionada à natureza física do hardware e de muitos outros produtos físicos. O hardware apresenta uma alta taxa de defeitos no início de seu ciclo de vida, isso acontece até que o equipamento e seus componentes sejam ajustados corretamente a sua função e ambiente. Após a correção desses defeitos e estabilização do produto, a taxa de defeitos do hardware se mantém estável e baixa até o fim de seu ciclo de vida quando é verificada uma alta taxa de defeitos novamente devido ao desgaste dos produtos por efeitos cumulativos de poeira, vibração, impactos e temperaturas extremas e vários outros males ambientais. Essa curva de defeitos do

hardware é chamada de “curva da banheira”. Vejam na figura abaixo a curva de defeitos do hardware ao longo de seu ciclo de vida.

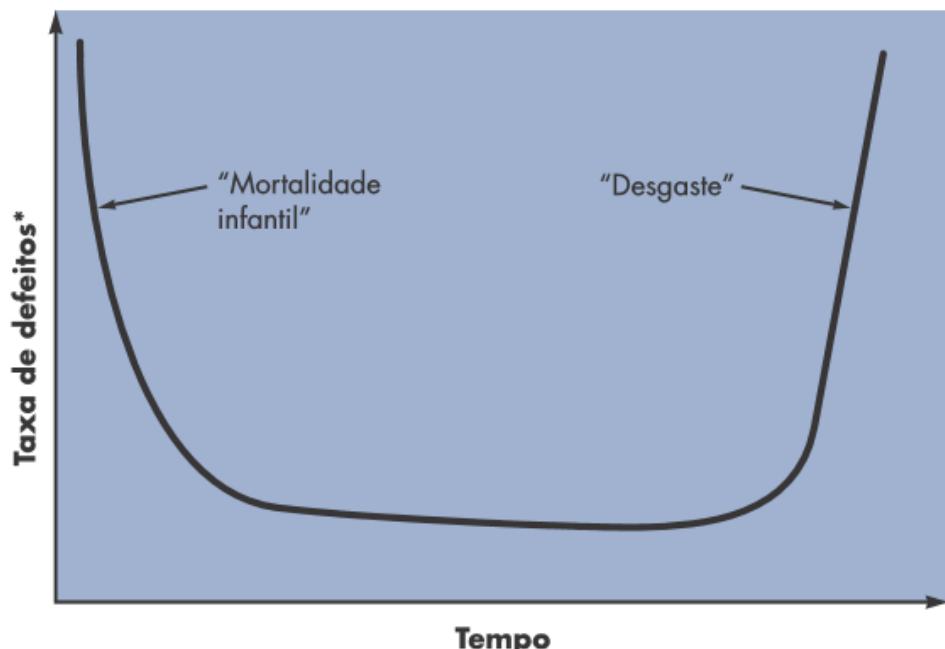


Figura 1.1. Curva de defeitos de hardware. PRESSMAN, R. S.(2011)

1.3. Curva de defeitos do software

A curva de defeitos de um software é bastante peculiar e tem um comportamento bem diferente da curva de defeitos de um hardware. Ela também tem uma alta taxa de defeitos no início do ciclo de vida do software assim como o hardware, porém, ela não se estabiliza ao longo do tempo como a do hardware. Diferente do hardware, o software sofre diversas mudanças durante seu ciclo de vida. Essas mudanças acontecem para corrigir defeitos identificados tarde, para desenvolver melhorias no software, adaptações e evoluções. A cada mudança, como efeito colateral, há um aumento na taxa de defeitos daquele software. O software é estabilizado, mas o processo se reinicia com uma nova mudança. Vejam na figura abaixo a curva de defeitos do software ao longo de seu ciclo de vida.

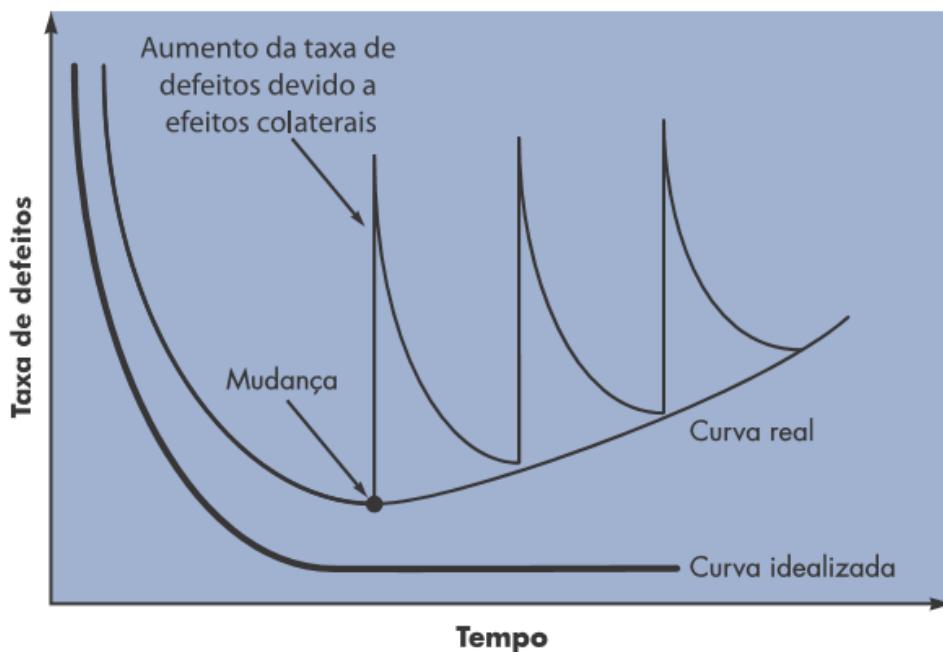


Figura 1.2. Curva de defeitos de software. PRESSMAN, R. S.(2011).

1.4. Tipo de software

O software é a tecnologia única mais importante no cenário mundial. O software se tornou uma tecnologia indispensável para negócios, ciência e engenharia; ele viabilizou a criação de novas tecnologias (por exemplo, engenharia genética e nanotecnologia), a extensão de tecnologias existentes (por exemplo, telecomunicações) e a mudança radical nas tecnologias mais antigas (por exemplo, indústria gráfica); se tornou a força motriz por trás da revolução do computador pessoal; já vemos pacotes de software sendo comprados pelos consumidores em lojas de bairro; o software evoluiu lentamente de produto para serviço, na medida que empresas de software ofereceram funcionalidade imediata (just-in-time), via um navegador Web ou aplicativos móveis; companhias de software se tornaram as maiores e mais influente companhias da era industrial criando a era digital; a Internet, iria evoluir e modificou tudo: de pesquisa em bibliotecas a compras feitas pelos consumidores, incluindo discurso político, hábitos de namoros de jovens e de adultos não tão jovens. (PRESSMAN, 2011)

O software foi incorporado em sistemas de todas as áreas: transportes, medicina, telecomunicações, militar, industrial, entretenimento, máquinas de escritório e etc. Para atender toda essa demanda foram criados vários tipos de software. Segue na tabela abaixo os tipos mais importantes:

Tabela 1.1. Tipos de Software

Tipos de Software	Descrição e Exemplos
-------------------	----------------------

Básico	Programas de apoio a outros programas. Ex.: Sistema Operacional, Drivers e Compiladores.
De Tempo Real	Monitora, analisa e controla eventos do mundo real. Ex.: Controle de Tráfego Aéreo, Aviões, Trânsito e Usinas.
Comercial	Operações Comerciais e Tomada de Decisões Administrativas. Ex.: Sistemas de Administração Empresarial, Vendas e etc.
Científico e Engenharia	Algoritmos de Processamento de Números. Exemplos: Cálculos e etc.
De Computador Pessoal	Processamento de textos, planilhas eletrônicas, aplicativos de diversões e etc. Ex.: Pacote Office, Jogos e etc.
De Inteligência Artificial	Algoritmos não numéricos para resolver problemas que não sejam favoráveis à computação ou à análise direta. Ex.: Robôs, Chatbots e etc.
Embutido	Controla produtos e sistemas de mercados industriais e de consumo. Ex.: Lavadora, Iluminação e etc.
Mobile	Aplicativo para dispositivos móveis.

Fonte: autoral

1.5. Tipo de software - Categorias modernas

Com a evolução da tecnologia é possível observar novos tipos de software que, mesmo que possam ser classificados nas categorias apresentadas no tópico anterior, devem ser destacados separadamente também devido ao grande número de corporações e usuários que os utilizam. Segue na tabela abaixo esses tipos de software mais modernos:

Tabela 1.2. Tipos de Software - Categorias modernas

Tipos de Software	Descrição e Exemplos
SAAS	Software como um Serviço. Ex.: Hospedagem de Sites e etc.
Redes Sociais	Comunicação, Conteúdo colaborativo e propositivo. Ex.: Facebook, Instagram e etc.
Plataformas	Fornecem a base para oferecimento e aquisição de serviços. Ex.: Uber e Airbnb.
E-Commerce	Fornecem a base para oferecimento e aquisição de produtos. Ex.: Magalu, Extra e etc.

1.6. Você quer assistir?

Segue uma indicação de vídeo para estudo complementar. Trata-se de um vídeo com

explicações simples sobre as diferenças entre hardware e software:
<<https://youtu.be/RM8bBzHggu8>>. Acesso em 20/01/2021.

1.5. Referências

Fonte Figuras 1.1, 1. 2: PRESSMAN, R. S.(2011).

Fonte Tabelas 1.1, 1. 2: Criação autoral.

PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.

3

TEXTO BASE

ENGENHARIA DE SOFTWARE

Texto base

3

Evolução de Software

Prof. João de Deus Freire Junior

Resumo

Como base para o entendimento dos grandes desafios que encontramos na produção de software é necessário que conheçamos como o software evolui até a complexidade e variedade que conhecemos hoje. Esta aula explica como e quando começamos a produzir software, como ele evoluiu, quais influências externas o impactou e como chegamos ao que conhecemos nos dias atuais.

1.1. Introdução

Quando se iniciou o desenvolvimento de software? Como era o software na década de 60, 70 ou 80? É possível dividir a evolução do software em eras? O crescimento exponencial da necessidade de software foi acompanhado pela criação de boas práticas de desenvolvimento de software? Todas essas perguntas serão respondidas nesta aula. Como base para o entendimento dos grandes desafios que encontramos na produção de software é necessário que conheçamos como o software evoluiu até a complexidade e variedade que conhecemos hoje em dia.

1.2. Evolução de Software

A década de 1950 é um marco para o desenvolvimento de software. É nela que se inicia a primeira era do desenvolvimento de software. Da década de 1950 até os nossos dias tivemos quatro eras de desenvolvimento de software. (PFLEEGER, 2003)

Segue nos tópicos abaixo as principais características de cada era do software:

1.2.1. A primeira era do desenvolvimento de software - 1950 a 1965

A lista abaixo apresenta as principais características desta era de desenvolvimento do software:

- O desenvolvimento de software era considerado uma arte;

- Havia poucos métodos sistemáticos para o desenvolvimento;
- O desenvolvimento de software não era gerenciado;
- O hardware sofria contínuas mudanças e era o centro das atenções;
- O software era customizado, ou seja, adequado às necessidades do usuário final, e a sua distribuição era limitada;
- O software era desenvolvido e utilizado pela mesma pessoa ou organização;
- Não havia documentação, todas as informações necessárias sobre o software estavam na cabeça das pessoas que o desenvolveram (one's head);
- O processamento de dados era em lote (batch).



Figura 1.1. Mainframe IBM da década de 60

1.2.2. A segunda era do desenvolvimento de software - 1963 a 1974

A lista abaixo apresenta as principais características desta era de desenvolvimento do software:

- Surgimento da multiprogramação e dos sistemas multiusuários;
- Desenvolvimento de técnicas interativas homem -máquina;
- Utilização de sistemas de tempo real;
- Surgimento da 1ª geração de Sistema Gerenciadores de Banco de Dados;
- Nascem as software houses e os produtos de software;
- O software era produzido para ampla distribuição em um mercado multidisciplinar, em várias áreas de conhecimentos;
- Surge o conceito de biblioteca de software;
- Devido à falta de metodologias de desenvolvimento e de documentação, a manutenção era praticamente impossível.



Figura 1.2. Primeiros dispositivos computacionais com possibilidade de interação com usuários. Disponível em: <<https://unsplash.com/>>.

1.2.3. A terceira era do desenvolvimento de software - 1973 a 1978

A lista abaixo apresenta as principais características desta era de desenvolvimento do software:

- Surgimento dos sistemas distribuídos e paralelos;
- Desenvolvimento das redes locais e globais de computadores;
- Necessidade de elevada demanda por acesso imediato a dados por parte dos usuários;
- Criação dos computadores de uso pessoal (PC - personal computers) e estações de trabalho (workstations);
- Uso generalizado de microprocessadores;
- Grande consumos de computadores;
- Os computadores se tornam acessíveis.



Figura 1.3. e 1.4 Primeiros computadores pessoais. Disponível em: <<https://unsplash.com/>>.

1.2.4. A quarta era do desenvolvimento de software - 1985 aos dias atuais

A lista abaixo apresenta as principais características desta era de desenvolvimento do software:

- Tecnologias orientadas a objetos
- Sistemas especialistas e software de inteligência artificial usados na prática
- Software de rede neural artificial
- Computação Paralela
- Internet
- Dispositivos móveis
- Redes sociais



**Figura 1.5. e 1.6 Smartphone e notebooks modernos. Disponível em:
<<https://unsplash.com/>>.**

1.3. Você quer assistir?

Segue uma indicação de vídeo para estudo complementar. Trata-se de um vídeo com explicações simples sobre a evolução de software com várias imagens que ajudam a conhecer as especificidades da primeira era de software: LIPPEL, Vinicius. Evolução dos Softwares: 1950 a 1965. Disponível em: <<https://youtu.be/lNq-g6QKlzA>>. Acesso em: 07 jan. 2021

1.5. Referências

Fonte Figuras 1.1, 1. 2,1.3,1.4,1.5,1.6: Site Unsplash. Disponível em: <<https://unsplash.com/>>. Acesso em: 07 jan. 2021.

PFLEEGER, Shari Lawrence. Engenharia de software - teoria e prática . 2 .ed. São Paulo: Pearson (livros universitarios), 2003

4

TEXTO BASE

ENGENHARIA DE SOFTWARE



Texto base

4

Crise de Software

Prof. João de Deus Freire Junior

Resumo

O processo de desenvolvimento de software é complexo e a demanda por software cresceu e ainda cresce exponencialmente. O software traz diferenciais competitivos para as organizações, serviços base e suportam as operações. Ele é essencial. Porém, há muitos problemas encontrados nos projetos de construção, melhorias e manutenção de software. De fato, vivemos uma crise do software.

1.1. Introdução

Por que tantos problemas são encontrados ao utilizar software? Porque os projetos de desenvolvimento de software falham tanto em estimativas de prazo e custo? Estamos em uma crise do software? Todas essas perguntas serão respondidas nesta aula. Será apresentada a crise de software, os principais problemas associados a ela e suas causas.

1.2. O que é a crise do software?

A Crise do Software foi um termo que surgiu nos anos 70 que expressava as dificuldades do desenvolvimento de software frente ao rápido crescimento da demanda por software, da complexidade dos problemas a serem resolvidos e da inexistência de técnicas de desenvolvimento de sistemas.

1.3. Problemas relacionados a crise de software

Há diversos problemas associados à Crise do Software que são percebidos pelos usuários, desenvolvedores e outros envolvidos com a atividade de criação, evolução e manutenção de software. Esses problemas são as “dores” sentidas por todos envolvidos com software. Eles têm prejudicado muito as corporações, as empresas de tecnologia, entre outros. Segue abaixo quatro exemplos desses problemas:

1.3.1. As estimativas de prazo e de custo frequentemente são imprecisas

As estimativas de prazo e custo necessários para desenvolvimento de software geralmente falham e são muito maiores do que o previsto. Os prazos maiores do que os previstos afetam muito as corporações, pois, frequentemente o prazo pode estar associado ao início de uma nova operação de negócios, lançamento de novo produto e etc.; as corporações são afetadas também pelos custos maiores que os orçados causando cancelamento de projetos, prejuízos, perdas financeiras e etc. As falhas nas estimativas estão associadas às causas abaixo:

- As equipes de desenvolvimento de software não dedicam tempo para coletar dados sobre o processo de desenvolvimento de software.
- As equipes de desenvolvimento de software não tem nenhuma indicação sólida de produtividade, não podendo avaliar com precisão a eficácia de novas ferramentas, métodos ou padrões.

1.3.2. A produtividade das pessoas da área de software não tem acompanhado a demanda por seus serviços

O software é a tecnologia única mais importante no cenário mundial. O software se tornou uma tecnologia indispensável para negócios, ciência e engenharia; ele viabilizou a criação de novas tecnologias (por exemplo, engenharia genética e nanotecnologia), a extensão de tecnologias existentes (por exemplo, telecomunicações) e a mudança radical nas tecnologias mais antigas (por exemplo, indústria gráfica); se tornou a força motriz por trás da revolução do computador pessoal; já vemos pacotes de software sendo comprados pelos consumidores em lojas de bairro; o software evoluiu lentamente de produto para serviço, na medida que empresas de software ofereceram funcionalidade imediata (just-in-time), via um navegador Web ou aplicativos móveis; companhias de software se tornaram as maiores e mais influente companhias da era industrial criando a era digital; a Internet, iria evoluir e modificou tudo: de pesquisa em bibliotecas a compras feitas pelos consumidores, incluindo discurso político, hábitos de namoros de jovens e de adultos não tão jovens. (PRESSMAN, 2011)

Os sistemas têm de ser construídos e entregues mais rapidamente devido a todo esse aumento exponencial da demanda; sistemas maiores e até mais complexos são requeridos; sistemas devem ter novas capacidades que antes eram consideradas impossíveis. Como os métodos de engenharia de software existentes não conseguem lidar com isso, novas técnicas de engenharia de software precisam ser desenvolvidas para atender a essas novas demandas. A produtividade das pessoas da área de software não tem acompanhado a demanda por seus serviços. (SOMMERVILLE, 2011)

1.3.3. A qualidade de software às vezes é menos que adequada

Com o aumento da demanda de desenvolvimento de software exposto no tópico anterior, as empresas do setor de Tecnologia da Informação têm sido pressionadas a oferecer soluções de maior qualidade em prazos cada vez menores.

Em contrapartida, uma pesquisa recente publicada em 2012 e feita com organizações nacionais revelou que 43% dos projetos de TI, em média, foram cancelados ou entregues com falhas comprometedoras no processo e/ou produto. (RIBEIRO et al., 2011).

A qualidade de software é comumente menor que adequada. Os problemas de qualidade entre outros fatores estão associados a falta de conceitos qualitativos sólidos de garantia de qualidade e utilização de práticas e ferramentas para assegurar a qualidade do software entregue.

1.3.4. O software existente é difícil de manter

De acordo com a IEEE a definição de engenharia de software é:

“A aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação de engenharia ao software.” (IEEE, 1993)

Essa definição deixa claro a importância da manutenção de software no processo de engenharia de software. A manutenção começa logo no começo do uso do software. Logo que o software é liberado para os usuários finais, e em alguns dias, erros começam a ser relatados à equipe de desenvolvimento do software. Em adição, mudanças são solicitadas por grupos de usuários para adaptar o software às suas necessidades e para melhor atender suas operações de negócios e clientes. Os usuários sempre precisarão de algumas melhorias para fazer o software funcionar em seu mundo.

Com isso, o desafio da manutenção do software começou. As equipes de sustentação e desenvolvimento do software terão que trabalhar paralelamente na correção de bugs, solicitações de adaptação e melhorias que devem ser planejadas, programadas e, por fim, executadas. Logo, a fila já cresceu muito e o trabalho ameaça devorar os recursos disponíveis. Com o passar do tempo, sua organização descobre que está gastando mais tempo e dinheiro com a manutenção dos programas do que criando novas aplicações. De fato, não é raro uma organização de software despender de 60% a 70% de todos os recursos com manutenção de software. (PRESSMAN, 2011)

Os motivos de muito trabalho na manutenção de software são muitos. Osborne e Chikofsky fornecem uma resposta parcial:

“Muitos softwares dos quais dependemos hoje têm em média de 10 a 15 anos. Mesmo quando esses programas foram criados, usando as melhores técnicas de projeto e codificação conhecidas na época [e muitos não foram], o tamanho do programa e o espaço de armazenamento eram as preocupações principais. Eles então migraram para novas plataformas, foram ajustados para mudanças nas máquinas e na tecnologia dos sistemas operacionais e aperfeiçoados para atender a novas necessidades dos usuários – tudo isso sem grande atenção na arquitetura geral. O resultado são estruturas mal projetadas, mal codificadas, de lógica pobre e mal documentadas em relação aos sistemas de software, para os quais somos chamados a fim de mantê-los rodando.”(OSBORNE, 1990, p.10-11)

As más práticas de desenvolvimento e má elaboração ou inexistência de documentação de software levam a grande dificuldade na manutenção do software.

1.3.5. Resumo dos problemas associados à crise de software

A tabela abaixo apresenta um resumo dos problemas associados à crise de software.

Tabela 1.1. Variáveis a serem consideradas na avaliação de técnicas de interação.

Problema	Status na Crise
Prazos e Custos em relação às estimativas	Mais altos
Produtividade das Pessoas	Baixa
Qualidade de Software	Baixa
Dificuldade de manter software	Alta

1.4. Causas da Crise de Crise de Software

As causas associadas à crise de software são diversas. Vamos ressaltar neste tópico as três principais.

1.4.1. Natureza do software

A própria natureza do software é uma das causas da crise do software devido às suas próprias características. Segue essas características:

- O software é um elemento de sistema lógico e não físico (produto intangível). Consequentemente, o sucesso é medido pela qualidade de uma única entidade e não pela qualidade de muitas entidades manufaturadas.
- O software não se desgasta, mas se deteriora!

À medida que o tempo passa, os componentes de um hardware sofrem os efeitos cumulativos de poeira, vibração, impactos, temperaturas extremas e vários outros males ambientais ele começa a desgastar-se.

O software não é suscetível aos males ambientais que fazem com que o hardware se desgaste. Portanto, ele não se desgasta. Mas, o ambiente de negócios, a tecnologia, os processos, as organizações, sociedades, leis, regras mudam e todos esses aspectos podem fazer com que o software fique obsoleto e se deteriore.

1.4.2. Falhas das pessoas responsáveis pelo desenvolvimento de um software

As falhas das pessoas responsáveis pelo desenvolvimento de um software é uma das causas da crise do software. Isso acontece porque:

- Alguns gerentes de TI não tem nenhum background em software;
- Os profissionais da área de software têm recebido pouco treinamento formal em

- novas técnicas para o desenvolvimento de software;
- Alguns profissionais e gestores de TI são resistentes a mudanças.

1.4.3. Mitos de Software

Os mitos de software são inverdades sobre o processo de engenharia de software que propagam desinformação e confusão e eles são também uma das causas da crise do software. Os mitos podem ser:

- Administrativos;
- De clientes;
- Profissionais.

1.5. Você quer ler?

Segue uma indicação de leitura para estudo complementar. Trata-se de um artigo que apresenta e discute pontos da crise de software e de uma crise mais contemporânea a de significado: [Da crise do software à crise de significado: a importância de aprender a compreender](https://medium.com/software-zen/da-crise-do-software-%C3%A0-crise-de-significado-a-import%C3%A1ncia-de-aprender-a-compreender-5ff8fb920e37). Disponível em: <<https://medium.com/software-zen/da-crise-do-software-%C3%A0-crise-de-significado-a-import%C3%A1ncia-de-aprender-a-compreender-5ff8fb920e37>>. Acesso em: 07 jan. 2021

1.6. Referências

PFLEEGER, Shari Lawrence. Engenharia de software - teoria e prática . 2 .ed. São Paulo: Pearson (livros universitarios), 2003

RIBEIRO, A.; PRADO, D.; ARCHIBALD, R. Pesquisa sobre Maturidade e Sucesso em Gerenciamento de Projetos de Sistemas de Informação (software). Relatório TI 2010. [S.l]: Maturity by Project Category Model (MPCM), 2011. Disponível em: <http://www.maturityresearch.com/novosite/2010/downloads/PesquisaMaturidade2010_Relatorio-TI_Completo_V3.pdf>. Acesso em: 21 mar. 2012.

OSBORNE, W. M., and E. J. Chikofsky, “Fitting Pieces to the Maintenance Puzzle,” IEEE Software, January 1990, pp. 10–11.

PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.

5

TEXTO BASE

ENGENHARIA DE SOFTWARE



Texto base

5

Mitos de Software

Prof. João de Deus Freire Junior

Resumo

Há muitas inverdades, más tradições e práticas ditas como verdades e bons procedimentos no processo de engenharia de software. Elas são chamadas de mitos de software. Os mitos de software propagam desinformação e confusão ocasionando muitos problemas no desenvolvimento de software.

1.1. Introdução

O que é verdade e o que é mentira do processo de desenvolvimento de software? Quais são os mitos de software? Qual é a verdade relacionada ao processo de engenharia de software relacionada a cada mito? Essas perguntas serão respondidas nesta aula. Serão apresentados os mitos de software. O conhecimento deles é muito importante para reconhecê-los, evitá-los e combatê-los com boas práticas.

1.2. O que são os mitos de software?

Os mitos criados em relação ao software são crenças infundadas sobre o software e sobre o processo usado para criá-lo. Eles parecem ser, de fato, afirmações razoáveis (algumas vezes contendo elementos de verdade) e frequentemente são promulgados por praticantes experientes ou gestores. (PRESSMAN, 2011)

1.3. Mitos Administrativos

De acordo com Pressman (2011, p.46) os mitos administrativos acontecem porque:

“Gerentes com responsabilidade sobre software, assim como gerentes da maioria das áreas, frequentemente estão sob pressão para manter os orçamentos, evitar deslizes nos cronogramas e elevar a qualidade. Como

uma pessoa que está se afogando e se agarra a uma tábua, um gerente de software muitas vezes se agarra à crença num mito do software, para aliviar a pressão (mesmo que temporariamente).”

Segue nos tópicos abaixo alguns exemplos de mitos administrativos de acordo com Pressman (2011, p.46).

1.3.1. Só precisamos de um manual repleto de padrões e procedimentos para a construção de software.

- **Mito:**

Já temos um manual repleto de padrões e procedimentos para a construção de software. Isso com certeza por si só oferecerá ao meu pessoal tudo o que eles precisam saber

- **Realidade:**

Será que o manual é usado? Os profissionais sabem que ele existe? Ele reflete a prática moderna de desenvolvimento de software? Ele é completo? Na realidade, um manual por si só nunca oferecerá à equipe tudo que eles precisam saber.

1.3.2. Podemos já desenvolver software de alta qualidade já que temos computadores de última geração.

- **Mito:**

Meu pessoal tem ferramentas de desenvolvimento de software de última geração; afinal, compramos os mais novos computadores.

- **Realidade:**

É preciso muito mais do que os mais recentes computadores para se fazer um desenvolvimento de software de alta qualidade.

1.3.3. Sempre podemos adicionar mais programadores para resolver nosso problema e tirar o atraso.

- **Mito:**

Sempre que estivermos atrasados, podemos adicionar mais programadores e tirar o atraso.

- **Realidade:**

O desenvolvimento de software não é um processo mecânico igual à manufatura. Acrescentar pessoas em um projeto torna-o ainda mais atrasado. Pessoas podem ser acrescentadas, mas somente de uma forma planejada.

1.4. Mitos de Clientes

De acordo com Pressman (2011, p.46) os mitos de clientes acontecem porque:

“O cliente solicitante do software computacional pode ser uma pessoa na mesa ao lado, um grupo técnico do andar de baixo, de um departamento de marketing/vendas, ou uma empresa externa que encomendou o projeto por contrato. Em muitos casos, o cliente acredita em mitos sobre software porque gerentes e profissionais da área pouco fazem para corrigir falsas informações. Mitos conduzem a falsas expectativas (do cliente) e, em última instância, à insatisfação com o desenvolvedor.”

Segue nos tópicos abaixo alguns exemplos de mitos de clientes de acordo com Pressman (2011, p.46).

1.4.1. Uma declaração geral dos objetivos é suficiente para se começar a escrever programas - podemos preencher os detalhes mais tarde.

- **Mito:**

Somente uma declaração geral dos objetivos é suficiente para se começar a escrever programas - podemos preencher os detalhes mais tarde e já iniciar o desenvolvimento.

- **Realidade:**

O Uma definição inicial ruim é a PRINCIPAL causa de fracassos dos esforços de desenvolvimento de software.

É fundamental uma descrição formal e detalhada do domínio da informação, função, desempenho, interfaces, restrições de projeto e critérios de validação.

1.4.2. Os requisitos de projeto modificam-se continuamente, mas as mudanças podem ser facilmente acomodadas, porque o software é flexível.

- **Mito:**

Os requisitos de projeto modificam-se continuamente, mas as mudanças podem ser facilmente acomodadas, porque o software é flexível. E mudanças tardias não causarão grandes impactos no desenvolvimento.

- **Realidade:**

Uma mudança, quando solicitada tarde num projeto, pode ser maior do que mais do que uma ordem de magnitude mais dispendiosa do que a mesma mudança solicitada nas fases iniciais.

1.5. Mitos profissionais

De acordo com Pressman (2011, p.47) os mitos profissionais são:

“Os mitos que ainda sobrevivem nos profissionais da área têm resistido por mais de 50 anos de cultura de programação. Durante seus primórdios, a programação era vista como uma forma de arte. Modos e atitudes antigas dificilmente morrem.”

Segue nos tópicos abaixo alguns exemplos de mitos profissionais de acordo com Pressman (2011, p.47).

1.5.1. Assim que escrevermos o programa e o colocarmos em funcionamento nosso trabalho estará completo

- **Mito:**

Assim que escrevermos o programa e o colocarmos em funcionamento, nosso trabalho estará completo. Podemos nos dar por satisfeitos com a entrega e partir para uma nova demanda.

- **Realidade:**

Os dados da indústria indicam que entre 50% e 70% de todo esforço gasto num programa serão despendidos depois que ele for entregue pela primeira vez ao cliente.

1.5.2. Enquanto não tiver o programa "funcionando", eu não terei realmente nenhuma maneira de avaliar sua qualidade

- **Mito:**

Enquanto não tiver o programa "funcionando" completamente em ambiente de produção ou validação, eu não terei realmente nenhuma maneira de avaliar sua qualidade

- **Realidade:**

Um programa funcionando é somente uma parte de uma configuração de software que inclui todos os itens de informação produzidos durante a construção e manutenção do software.

1.6. Você quer ler?

Segue duas indicações de estudo complementar. Uma para leitura e outra para assistir.

- Assista o vídeo:

SOUZA, Adler Diniz de. Mitos de Software e Engenharia de Software. 2016.
Disponível em: <<https://youtu.be/CKSpC6VP6TU>>. Acesso em: 19 dez. 2019.

- Leia o artigo:

FONSECA, Gabi. Os principais mitos do desenvolvimento de Software. 2011. Disponível em: <<https://www.profissionaisti.com.br/os-principais-mitos-do-desenvolvimento-de-software/>>. Acesso em: 19 dez. 2019.

1.6. Referências

PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.

6

TEXTO BASE

ENGENHARIA DE SOFTWARE



Texto base

6

Impacto das Mudanças

Prof. João de Deus Freire Junior

Resumo

Uma mudança, quando solicitada tardiamente num projeto, pode ser maior do que mais do que uma ordem de magnitude mais dispendiosa do que a mesma mudança solicitada nas fases iniciais. As mudanças impactam muito o andamento dos projetos podem inclusive inviabilizar a continuidade de um projeto devido ao grande impacto no todo. Por isso, as fases iniciais de levantamento de requisitos de um projeto são essenciais para seu sucesso.

1.1. Introdução

Posso solicitar mudanças no software que está sendo desenvolvido quando eu quiser? As mudanças são facilmente recebidas e absorvidas sem impacto pela equipe de desenvolvimento? O impacto das mudanças diferem de acordo com o momento que são solicitadas? Essas perguntas serão respondidas nesta aula. É vital entendermos qual o impacto das solicitações de mudanças no ciclo de desenvolvimento de software.

1.2. Curva de defeitos do software

A curva de defeitos de um software é bastante peculiar e tem um comportamento bem diferente da curva de defeitos de um hardware. Ela apresenta uma alta taxa de defeitos no início do ciclo de vida do software, assim como ocorre com o hardware. No entanto, ao contrário do hardware, a curva de defeitos do software não se estabiliza ao longo do tempo. Isso se deve ao fato de que o software está sujeito a múltiplas mudanças durante o seu ciclo de vida. Essas mudanças ocorrem para corrigir defeitos identificados tarde, implementar melhorias no software, realizar adaptações e promover evoluções. A cada mudança realizada, como efeito colateral, há um aumento na taxa de

defeitos do software. Embora o software possa ser estabilizado, o processo se reinicia com uma nova mudança. A figura 1.1 ilustra a curva de defeitos do software ao longo de seu ciclo de vida.

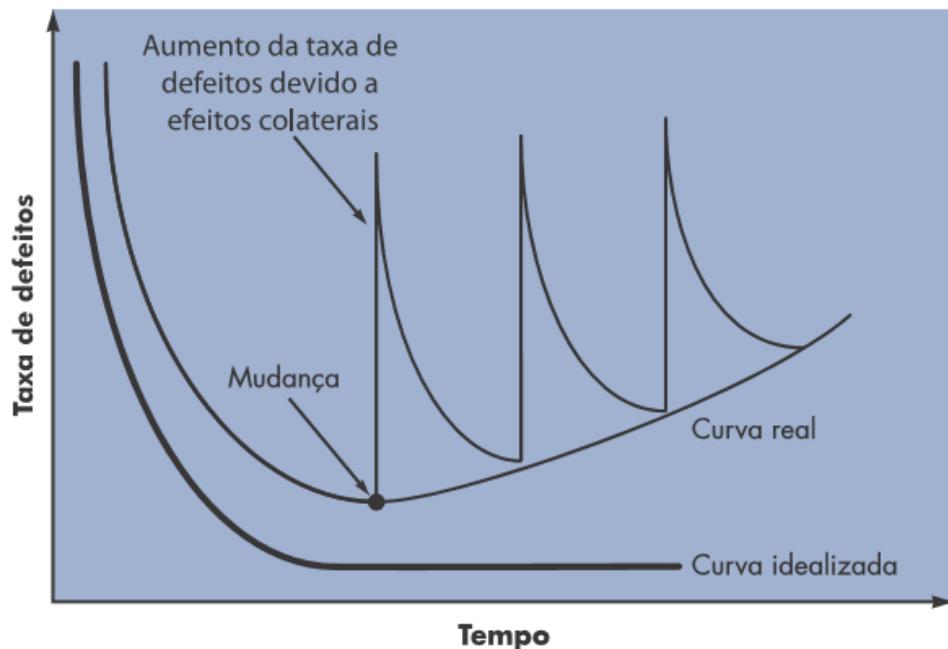


Figura 1.1. Curva de defeitos de software

1.3. Impacto das mudanças

O conhecimento acumulado de desenvolvimento de software (baseada em décadas de experiência) afirma que o impacto de mudanças aumentam de forma não linear conforme o projeto avança (Figura 1.2, curva em preto contínuo). É de certa forma fácil absorver uma mudança quando uma equipe de software está levantando requisitos (no início de um projeto). Pode-se ter de alterar um detalhamento do uso, de uma funcionalidade, ampliar uma lista de funções, alterar algo relacionado ao desenho das interfaces com o usuário ou editar uma especificação por escrito. Os custos de tal trabalho são mínimos e o tempo demandado não afetará negativamente o resultado do projeto. Mas, se adiantarmos alguns meses, o que aconteceria? O time de desenvolvimento estará em meio aos testes de validação (que ocorrem relativamente no final do projeto) e um importante interessado está requisitando uma mudança funcional grande, que muitas vezes provoca mudanças estruturais no software. A mudança requer uma alteração no projeto da arquitetura do software, o projeto e desenvolvimento de vários novos componentes, modificações em vários outros componentes, especificação de novos testes, e assim por diante. Os impactos crescem rapidamente e não serão triviais o tempo e custos necessários para assegurar que a mudança seja feita sem efeitos colaterais inesperados. (PRESSMAN, 2011)

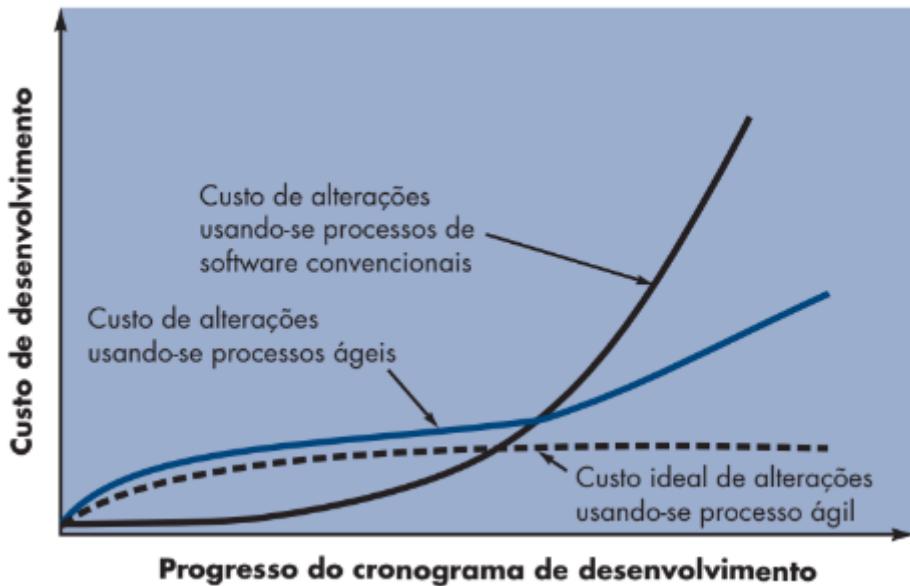


Figura 1.2. Custos de alterações como uma função do tempo em desenvolvimento

1.4. Você sabia?

Há vários exemplos conhecidos de grandes projetos impactados por mudanças ou defeitos identificados na fase de manutenção do software em produção. Segue nos próximos tópicos alguns exemplos.

1.4.1. Espaçonave da NASA

Em sua missão a Marte em 1998, a espaçonave Climate Orbiter acabou perdida no espaço. Embora a falha tenha confundido os engenheiros por algum tempo, foi revelado que um subcontratado da equipe de engenharia não conseguiu fazer uma conversão simples de unidades inglesas para métricas. Um lapso embaraçoso que enviou a nave de US \$125 milhões fatalmente perto da superfície de Marte, após tentar estabilizar sua órbita muito baixa. Os controladores de voo acreditam que a espaçonave invadiu a atmosfera de Marte, onde as tensões associadas prejudicaram suas comunicações, deixando-a voando pelo espaço em uma órbita ao redor do sol.

1.4.2. Ariane 5

O mais novo foguete de lançamento de satélite não tripulado da Europa utilizou o software funcional de seu antecessor, o Ariane 4. Infelizmente, os motores mais rápidos do Ariane 5 exploraram um bug que não foi encontrado nos modelos anteriores. Trinta e seis segundos após o lançamento inicial, os engenheiros do foguete apertaram o botão de

autodestruição após várias falhas do computador. Em essência, o software tentou amontoar um número de 64 bits em um espaço de 16 bits. As condições de estouro resultantes travaram os computadores principal e de backup (que estavam executando exatamente o mesmo software).

O Ariane 5 havia custado quase US \$8 bilhões para ser desenvolvido e carregava uma carga útil de satélite de US \$500 milhões quando explodiu.

1.4.3. Explosão de gasoduto soviético

O oleoduto soviético tinha um nível de complexidade que exigiria um software de controle automatizado avançado. A CIA (Central de Inteligência dos Estados Unidos) foi informada das intenções soviéticas de roubar os planos do sistema de controle. Trabalhando com a empresa canadense que projetou o software de controle de dutos, a CIA fez com que os projetistas criassem deliberadamente falhas na programação para que os soviéticos recebessem um programa comprometido. Alega-se que, em junho de 1982, falhas no software roubado levaram a uma explosão massiva ao longo de parte do oleoduto, causando a maior explosão não nuclear da história do planeta.

1.5. Você quer ler?

Segue duas indicações de estudo complementar. Trata-se de um artigo que explica quanto um erro de software pode custar para as empresas e apresenta vários links de outros artigos que detalham o tema.

- Leia o artigo:

ROBERTO, Jones. Quanto os erros em software podem custar para a sua empresa? 2019.

Disponível em:

<<https://medium.com/xp-inc/quanto-os-erros-em-software-podem-custar-para-a-sua-empresa-409b77d08bd6>>. Acesso em: 07 jan. 2021.

1.6. Referências

PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.

ROBERTO, Jones. Quanto os erros em software podem custar para a sua empresa? 2019. Disponível em: <<https://medium.com/xp-inc/quanto-os-erros-em-software-podem-custar-para-a-sua-empresa-409b77d08bd6>>. Acesso em: 07 jan. 2021.

7

TEXTO BASE

ENGENHARIA DE SOFTWARE



Texto base

7

Fundamentos da Engenharia de Software

Prof. João de Deus Freire Junior

Resumo

A Engenharia de Software foi criada com o intuito de resolver ou minimizar os impactos da crise de software. Ela é o emprego de boas práticas no desenvolvimento de software, a aplicação de processos, métodos e ferramentas adequadas para o desenvolvimento econômico e de qualidade do software. As preocupações e metas de engenharia de software miram a evolução e sistematização do processo de desenvolvimento de software.

1.1. Introdução

O que é engenharia de software? Quais são os principais fundamentos e elementos da engenharia de software? Quais são as preocupações e metas da engenharia de software? Essas perguntas serão respondidas nesta aula. O entendimento do que é a engenharia de software, suas preocupações e metas é a base para a aplicação de boas práticas no desenvolvimento e manutenção de software

1.2. O que é engenharia de software?

Uma boa definição de Engenharia de software é que ela é o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter software de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais. (Bauer apud Pressman, 2011)

Outra boa definição também citada pelo Pressman e oriunda do dicionário internacional de engenharia é que Engenharia de software é a aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e

na manutenção de software; isto é, a aplicação de engenharia ao software. (IEEE apud Pressman, 2011)

Um resumo e simplificação dessas duas definições poderia ser que a engenharia de software prescreve a aplicação de abordagens e processos sistemáticos para possibilitar o desenvolvimento de software de maneira econômica e com qualidade.

1.3. Preocupação da engenharia de software

A engenharia de software se preocupa com a sistematização do processo de criação e manutenção de software. Ela tem por objetivo apoiar o desenvolvimento profissional de software. Ela inclui técnicas que apoiam especificação, projeto e evolução de programas. (Sommerville, 2011).

1.4. Principais metas da engenharia de software

As principais metas da engenharia de software são:

1. Melhorar a qualidade de produtos de software;
2. Aumentar a produtividade do pessoal técnico e;
3. Aumentar a satisfação dos clientes.

Por isso, a disciplina de engenharia de software se preocupa com todos os aspectos da produção de software desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo usado. (Sommerville, 2011)

Segue abaixo uma afirmação de Sommerville (2011, p.5) sobre as preocupações da engenharia de software:

“A engenharia de software não se preocupa apenas com os processos técnicos do desenvolvimento de software. Ela também inclui atividades como gerenciamento de projeto de software e desenvolvimento de ferramentas, métodos e teorias para apoiar a produção de software. Engenharia tem a ver com obter resultados de qualidade requeridos dentro do cronograma e do orçamento.”

1.5. Camadas da Engenharia de Software

Podemos dividir a atuação da engenharia de software em camadas. Contudo, todas essas camadas devem ter como foco o comprometimento organizacional com a qualidade, a promoção de uma cultura de aperfeiçoamento contínuo de processos, e é esta cultura que, no final das contas, leva ao desenvolvimento de abordagens cada vez mais efetivas

na engenharia de software. A pedra fundamental que sustenta a engenharia de software é o foco na qualidade. (PRESSMAN, 2011)

As camadas da engenharia de software são: Processo, Métodos e Ferramentas. Todas essas camadas como afirmado acima tem o foco de proporcionar mais qualidade ao software. Elas servem como base e suportam umas às outras.

Segue abaixo uma figura que exibe as camadas da engenharia de software:



Figura 1.1. Camadas da Engenharia de Software

1.5.1. Camada de Processo

A camada base da engenharia de software é a camada de processo. Ela constitui o elo de ligação entre os métodos e ferramentas e define a sequência em que os métodos serão aplicados.

Segue uma explicação do Pressman (2011, p. 40) sobre essa camada:

“A base para a engenharia de software é a camada de processos. O processo de engenharia de software é a liga que mantém as camadas de tecnologia coesas e possibilita o desenvolvimento de software de forma racional e dentro do prazo. O processo define uma metodologia que deve ser estabelecida para a entrega efetiva de tecnologia de engenharia de software. O processo de software constitui a base para o controle do gerenciamento de projetos de software e estabelece o contexto no qual são aplicados métodos técnicos, são produzidos produtos derivados (modelos, documentos, dados, relatórios, formulários etc.), são estabelecidos marcos, a qualidade é garantida e mudanças são geridas de forma apropriada.”

1.5.2. Camada de Métodos

A segunda camada da engenharia de software é a camada de métodos. Ela fornece os detalhes de como fazer para construir o software.

Segue uma explicação do Pressman (2011, p. 40) sobre essa camada:

“Os métodos da engenharia de software fornecem as informações técnicas para desenvolver software. Os métodos envolvem uma ampla gama de tarefas, que incluem: comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte. Os métodos da engenharia de software baseiam-se em um conjunto de princípios básicos que governam cada área da tecnologia e inclui atividades de modelagem e outras técnicas descritivas.”

1.5.3. Camada de Ferramentas

A terceira camada da engenharia de software é a camada de ferramentas. As ferramentas são suporte automatizado aos métodos.

Segue uma explicação do Pressman (2011, p. 40) sobre essa camada:

“As ferramentas da engenharia de software fornecem suporte automatizado ou semi automatizado para o processo e para os métodos. Quando as ferramentas são integradas, de modo que as informações criadas por uma ferramenta possam ser usadas por outra, é estabelecido um sistema para o suporte ao desenvolvimento de software, denominado engenharia de software com o auxílio do computador.”

1.6. Você quer ler?

Segue duas indicações de estudo complementar. Trata-se do livro do Pressman que é base da disciplina de engenharia de software.

- Leia o capítulo 1 do livro abaixo, página 40:
PRESSMAN, Roger S.; MAXIM, Bruce R.. Engenharia de Software: UMA ABORDAGEM PROFISSIONAL. Disponível em:
[<https://integrada.minhabiblioteca.com.br/#/books/9788580555349/pageid/40>](https://integrada.minhabiblioteca.com.br/#/books/9788580555349/pageid/40)

1.6. Referências

- PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.
- SOMMERVILLE, Ian.(2011) Engenharia de Software. 9.ed. São Paulo: Pearson Prentice Hall, 2011.
- Figuras 1.1: PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.

- Naur, P., and B. Randall (1969), Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee, NATO, 1969.

8

TEXTO BASE

ENGENHARIA DE SOFTWARE

Texto base

8

Modelos de Processo de Engenharia de Software

Prof. João de Deus Freire Junior

Resumo

A Engenharia de Software foi criada com o intuito de resolver ou minimizar os impactos da crise de software. Ela é o emprego de boas práticas no desenvolvimento de software e também é necessário que tenhamos um roteiro contendo uma série de passos que podem nos ajudar a desenvolver produtos de software com qualidade, dentro de prazos e custos previstos que satisfaçam as expectativas dos clientes. Chamamos esses roteiros de modelos de processo de engenharia de software. Nesta aula, vamos discutir os diferentes tipos de modelos, suas aplicações e diferenças.

1.1. Introdução

O que são os modelos de processo de engenharia de software? Qual modelo devo utilizar? Quais são os principais modelos de processo de engenharia de software? Qual o modelo é o mais recomendado para cada situação? Todas essas perguntas serão respondidas nesta aula. O entendimento do que são os modelos de processo de engenharia de software, suas principais características e suas aplicações é extremamente importante para que os profissionais de tecnologia da informação tenham competência de saber qual aplicar para suas próprias necessidades.

1.2. O que são os modelos de processo de engenharia de software?

Uma boa definição do que são os modelos de processo de engenharia de software é a contextualização feita pelo Pressman (2011) que quando se trabalha na elaboração de um produto ou sistema, é importante seguir uma série de passos previsíveis — um roteiro que ajude a criar um resultado de alta qualidade e dentro do prazo estabelecido. O roteiro

é denominado “processo de software”. Os modelos são diferentes roteiros para desenvolvimento de softwares.

Além de fornecer um roteiro para desenvolvimento de software definindo as atividades necessárias a serem realizadas e a sequência delas, o modelo de processo também estabelece o fluxo de processo, neste aspecto, o modelo descreve como são organizadas as atividades metodológicas, bem como as ações e tarefas que ocorrem dentro de cada atividade em relação à sequência e ao tempo. (Pressman, 2011)

Os fluxos de processo podem ser lineares, iterativo, evolucionário e paralelo. Segue uma representação gráfica e breve explicação de cada um deles:



Figura 1.1. Fluxo de processo linear

Um fluxo de processo linear executa cada uma das cinco atividades metodológicas em sequência, começando com a de comunicação e culminando com a do emprego. (Pressman, 2011)

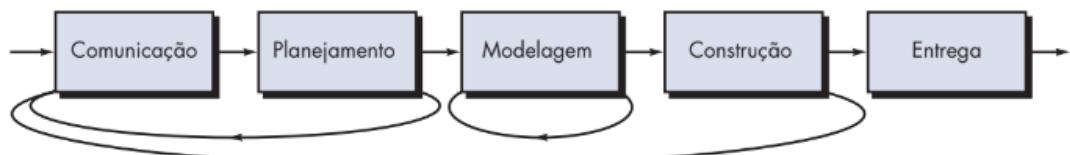


Figura 1.2. Fluxo de processo iterativo

Um fluxo de processo iterativo repete uma ou mais das atividades antes de prosseguir para a seguinte. (Pressman, 2011)

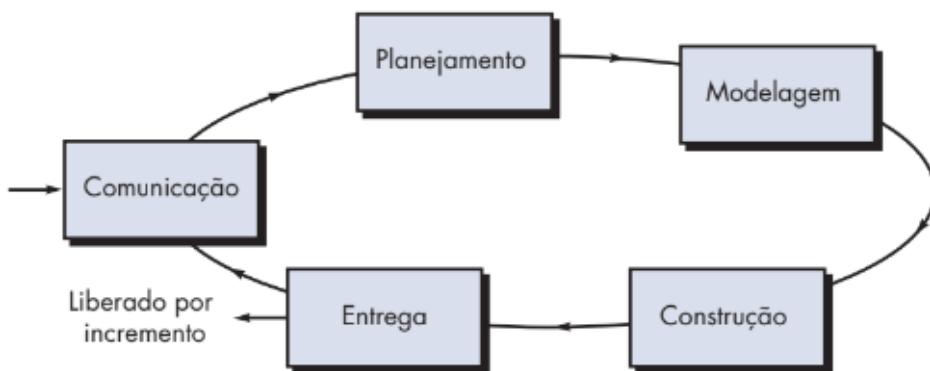


Figura 1.3. Fluxo de processo evolucionário

Um fluxo de processo evolucionário executa as atividades de uma forma “circular”. Cada volta pelas cinco atividades conduz a uma versão mais completa do software. (Pressman, 2011)

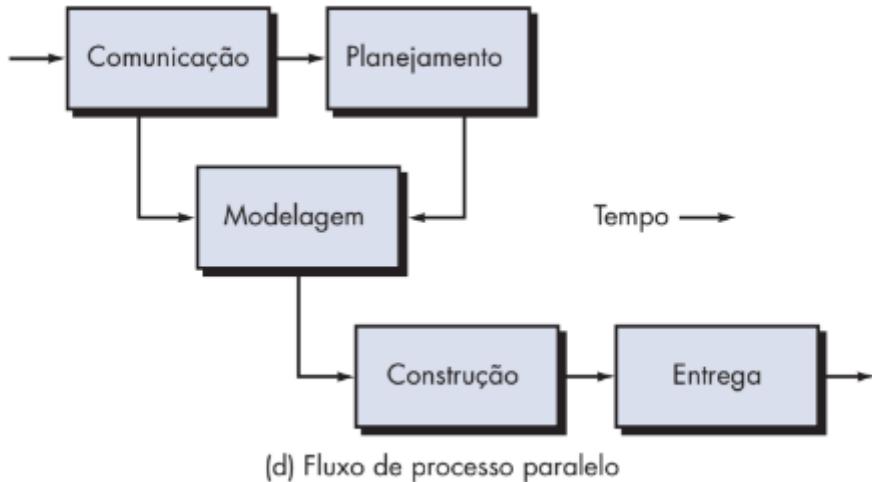


Figura 1.4. Fluxo de Processo Paralelo

Um fluxo de processo paralelo executa uma ou mais atividades em paralelo com outras atividades (por exemplo, a modelagem para um aspecto do software poderia ser executada em paralelo com a construção de um outro aspecto do software). (Pressman, 2011)

1.3. Qual modelo de processo de engenharia de software devo utilizar?

O modelo deve ser sempre escolhido com base em três pontos que variam de acordo com a solução tecnológica que estivermos desenvolvendo. Esses pontos são:

- Natureza do projeto e do produto
- Métodos e ferramentas utilizados
- Controles e Produtos intermediários desejados

1.4. Modelo Cascata ou Ciclo de Vida Clássico

O modelo cascata ou ciclo de vida clássico é o modelo mais antigo e o mais amplamente usado da engenharia de software, ele é modelado em função do ciclo da engenharia convencional e requer uma abordagem sistemática, sequencial ao desenvolvimento de software.

Esse modelo é recomendado quando os requisitos de um problema são bem compreendidos, quando adaptações bem definidas são feitas em sistemas já existentes ou quando novos desenvolvimentos e melhorias são desenvolvidos para sistemas ou ambientes de negócios razoavelmente estáveis. (Pressman, 2011).

Esse modelo é subdividido em fases. Segue abaixo uma breve explicação de cada uma das suas fases.

1.5.1. Comunicação

Essa fase envolve a coleta de requisitos em nível do sistema, pouca quantidade de projeto e a análise é de alto nível. Ela fornece a visão essencial quando o software deve fazer interface com outros elementos (hardware, pessoas e banco de dados).

1.5.2. Planejamento

Essa fase é responsável pela tradução dos requisitos de alto nível do software para um conjunto de representações que podem ser utilizados para formação de estimativas de custo, cronograma e definição de formato de acompanhamento de projeto.

1.5.3. Modelagem

Nesta fase a coleta de requisitos é intensificada e focada no software, concentra-se em entender o domínio da informação, a função, desempenho e interfaces exigidos. Os requisitos de sistema e de software são documentados e revistos com o cliente.

1.5.4. Construção

Nesta fase ocorre a tradução das representações do projeto para uma linguagem “artificial” resultando em instruções executáveis pelo computador. Nesta fase as funcionalidades do sistemas são testadas e validadas se estão de acordo com as especificações do projeto.

1.5.5. Emprego ou Manutenção

Nesta fase o software é entregue em ambiente produtivo para ser utilizado pelos usuários finais. Inicia-se o período de suporte e sustentação do software em operação e melhorias podem ser efetuadas de acordo com o feedback dos clientes finais.

1.5.6. Representação gráfica do Modelo Cascata

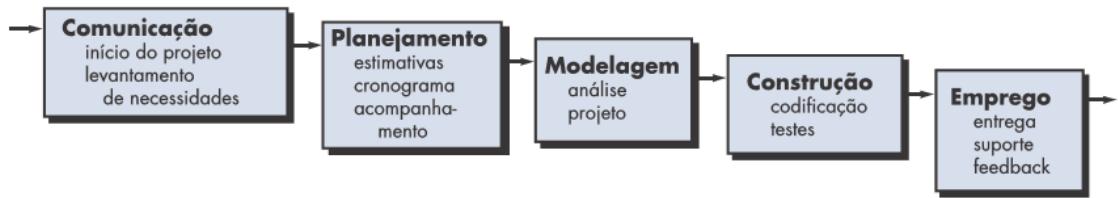


Figura 1.5. Modelo Cascata

1.5.7. Observações sobre o Modelo Cascata

Alguns problemas desse modelo são que projetos reais raramente seguem o fluxo sequencial. No início do processo é difícil estabelecer explicitamente todos os requisitos. No começo dos projetos sempre existe uma incerteza natural. Outro ponto é que o cliente deve ter paciência, uma versão executável do software só fica disponível numa etapa avançada do desenvolvimento. Porém, embora o Ciclo de Vida Clássico tenha fragilidades, ele é significativamente melhor do que uma abordagem casual de desenvolvimento de software. (Pressman, 2011)

1.6. Paradigma da Prototipação

Esse paradigma possibilita ao desenvolvedor criar um modelo do software a ser construído. Ele serve como um mecanismo para identificar os requisitos de software e é apropriado para quando o cliente define um conjunto de objetivos gerais para o software,

mas não identificou em detalhes os requisitos de entrada, processamento e saída. (Pressman, 2011)

Segue uma representação gráfica desse modelo:

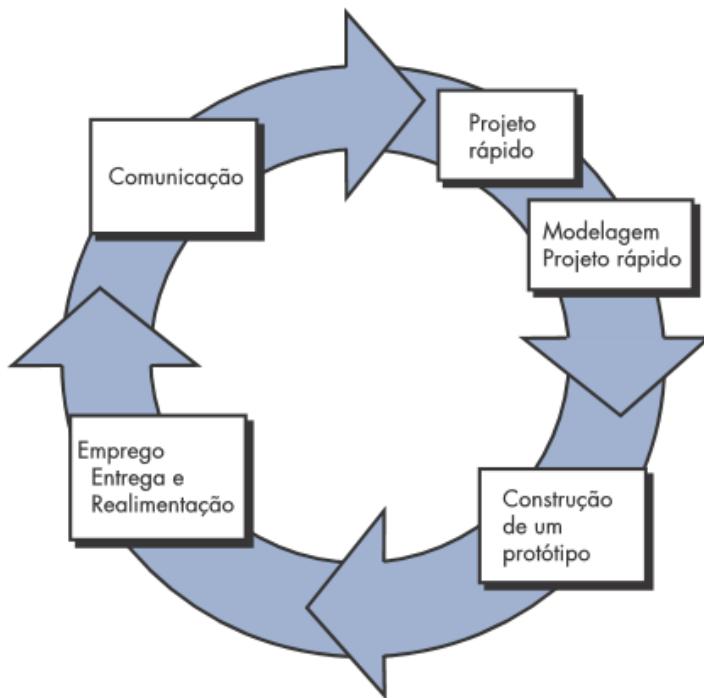


Figura 1.6. Paradigma da Prototipação

Esse modelo também é composto por etapas, segue uma breve explicação sobre cada uma delas:

1.6.1. Comunicação

Nesta etapa, os objetivos gerais do projeto são definidos, os feedback de clientes são discutidos e se necessário, incorporados a uma nova versão do protótipo.

1.6.2. Projeto Rápido

Nesta etapa ocorre a representação dos aspectos do software que são visíveis ao usuário (abordagens de entrada e formatos de saída).

1.6.3. Modelagem Projeto Rápido

Nesta etapa, desenvolvedor e cliente definem os objetivos gerais do software,

identificam quais requisitos são conhecidos e as áreas que necessitam de definições adicionais.

1.6.4. Construção de um protótipo e Emprego, Entrega e Retroalimentação

O projeto rápido é implementado, entregue, avaliado e essa avaliação serve como insumo para um novo ciclo de prototipação.

1.6.5. Observações sobre o Paradigma de Prototipação

Alguns problemas desse modelo são que o cliente pode não entender que o software que ele vê não considerou, durante o desenvolvimento, a qualidade global e a manutenção a longo prazo. Ele pode não aceitar bem a ideia que a versão final do software vai ser construída e "força" a utilização do protótipo como produto final. Em adição, o desenvolvedor normalmente utiliza o que está disponível com o objetivo de produzir rapidamente um protótipo, com o tempo ele pode acostumar-se com isso e esquece que essa abordagem não é apropriada para o desenvolvimento do produto final. (Pressman, 2011)

Ainda que possam ocorrer problemas, a prototipação é um ciclo de vida eficiente. A chave é definir as regras do jogo logo no começo. O cliente e o desenvolvedor devem concordar que o protótipo seja construído para auxiliar na descoberta de requisitos.

1.7. Modelo em Espiral

Esse modelo engloba as melhores características do ciclo de vida Clássico e da Prototipação, adicionando um novo elemento, a **Análise de Riscos**. Ele segue a abordagem de passos sistemáticos do Ciclo de Vida Clássico incorporando-os numa estrutura iterativa que reflete mais realisticamente o mundo real. Usa a Prototipação, em

qualquer etapa da evolução do produto, como mecanismo de redução de riscos.
(Pressman, 2011)

Segue uma representação gráfica desse modelo:

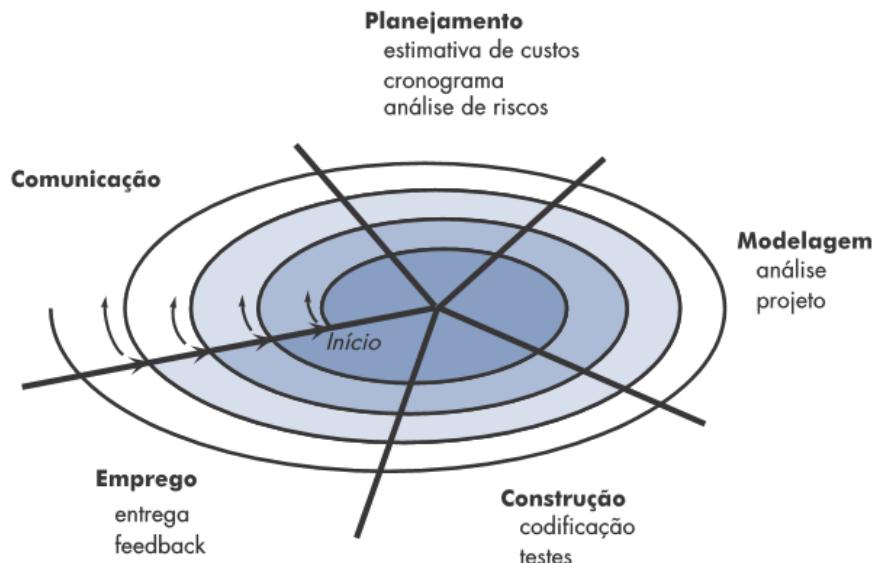


Figura 1.7. Modelo em Espiral

Esse modelo também é composto por etapas, segue uma breve explicação sobre cada uma delas:

1.7.1. Comunicação

Nesta etapa, os objetivos gerais do projeto são definidos, os feedback de clientes são discutidos e se necessário, incorporados a uma nova versão do produto.

1.7.2. Planejamento

Nesta etapa, os objetivos gerais do projeto são detalhados, alternativas são consideradas e restrições definidas. Estimativas de custo e prazo são definidas e uma análise de risco é efetuada.

1.7.3. Modelagem

Nesta etapa ocorre a análise de alternativas, requisitos e identificação / resolução de riscos. Os requisitos são detalhados para que o produto seja construído.

1.7.4. Construção e Emprego

Nestas etapas ocorre o desenvolvimento do produto no nível seguinte a validação do produto e os feedbacks dos clientes são recebidos e utilizados como insumo para planejamento das novas fases do ciclo em espiral.

1.8. Desenvolvimento baseado em Técnicas de 4^a Geração

De acordo com Pressman (2011) o desenvolvimento baseado em técnicas de 4^a geração se concentra na capacidade de se especificar o software para uma máquina em um nível que esteja próximo à linguagem natural. Engloba um conjunto de ferramentas de software que possibilitam que:

- o sistema seja especificado em uma linguagem de alto nível e;
- o código fonte seja gerado automaticamente a partir dessas especificações.

1.9. Desenvolvimento baseado em componentes

O modelo de desenvolvimento baseado em componentes desenvolve aplicações a partir de componentes de software pré-empacotados. O modelo de desenvolvimento baseado em componentes conduz ao:

- reúso do software e a reusabilidade proporciona uma série de benefícios mensuráveis aos engenheiros de software;
- A equipe de engenharia de software pode conseguir uma redução no tempo do ciclo de desenvolvimento e custo do projeto.

1.6. Você quer ler?

Segue duas indicações de estudo complementar. Trata-se do livro do Pressman que é base da disciplina de engenharia de software.

- Leia o capítulo 2 do livro abaixo, páginas 51 a 69:
PRESSMAN, Roger S.; MAXIM, Bruce R.. Engenharia de Software: UMA ABORDAGEM PROFISSIONAL. Disponível em:
<https://integrada.minhabiblioteca.com.br/#/books/9788580555349/pageid/40>

1.7. Referências

- PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.
- Figuras 1.1, 1.2, 1.3, 1.4, 1.5, 1.6 e 1.7: PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.
-

9

TEXTO BASE

ENGENHARIA DE SOFTWARE



Texto base

9

Abordagens de Desenvolvimento de Software

Prof. João de Deus Freire Junior

Resumo

A Engenharia de Software foi criada com o intuito de resolver ou minimizar os impactos da crise de software. Ela é o emprego de boas práticas no desenvolvimento de software A fim de nos ajudar a aplicar estas melhores práticas podemos utilizar uma das abordagens de desenvolvimento de software que podem nos ajudar a desenvolver produtos de software com qualidade, dentro de prazos e custos previstos que satisfaçam as expectativas dos clientes. Nesta aula, vamos discutir as diferentes abordagens, suas aplicações e diferenças.

1.1. Introdução

O que podemos desenvolver software de forma mais dinâmica? O que é modelo incremental? Quais são as principais abordagens de desenvolvimento de software? Qual a abordagem é a mais recomendada para cada situação? Todas essas perguntas serão respondidas nesta aula. O entendimento das abordagens de desenvolvimento de software, suas principais características e suas aplicações é extremamente importante para que os profissionais de tecnologia da informação tenham competência de saber qual aplicar para suas próprias necessidades.

1.2. Modelo de Processo Incremental

No modelo incremental, o sistema é entregue ao cliente em incrementos e cada incremento fornece parte da funcionalidade. Os requisitos são priorizados e os requisitos de prioridade mais alta são incluídos nos incrementos iniciais. Uma vez que o desenvolvimento de um incremento é iniciado, os requisitos são congelados e os requisitos para os incrementos posteriores podem continuar evoluindo e incluir requisitos

já implementados. Esse modelo combina elementos dos fluxos de processos lineares e paralelos. Conforme exibido na figura 1.1, o modelo incremental aplica sequências lineares, de forma escalonada, à medida que o tempo vai avançando. Cada sequência linear gera “incrementais” (entregáveis/aprovados/liberados) do software de maneira similar aos incrementais gerados por um fluxo de processos ágeis. (Pressman, 2011).

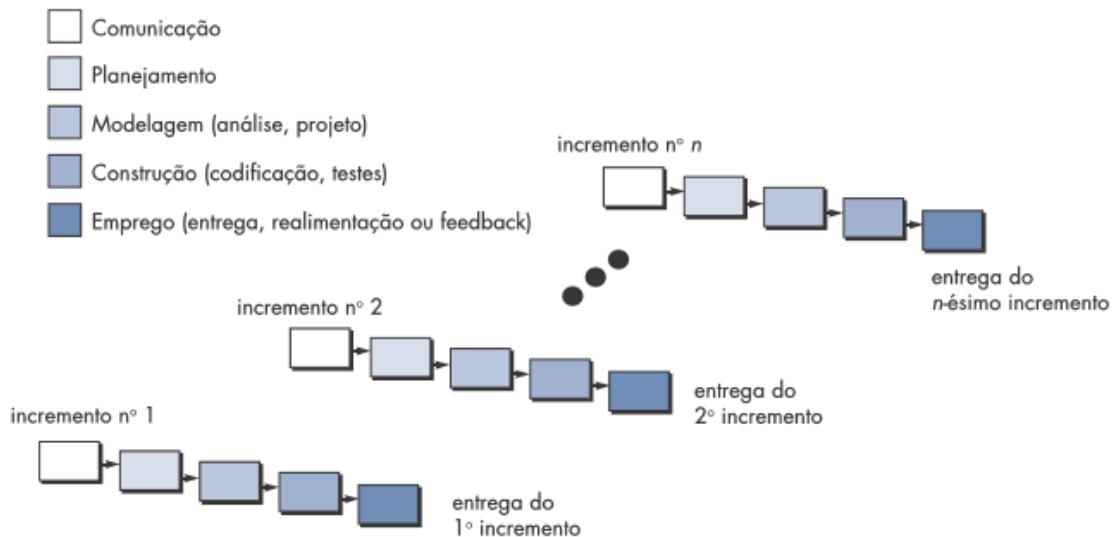


Figura 1.1. Modelo de Processo Incremental

As vantagens desse modelos são :

1. Incrementos podem ser entregues regularmente ao cliente e, desse modo, a funcionalidade do sistema é disponibilizada mais cedo.
2. Os incrementos iniciais agem como protótipos para elucidar os requisitos para incrementos posteriores do sistema.
3. Menor risco de falha geral do projeto
4. Os serviços de sistema de mais alta prioridade tendem a receber mais testes

1.3. A abordagem RUP

É uma abordagem baseada na UML(Linguagem de Modelagem Unificada). Ela busca cobrir todos os aspectos do desenvolvimento de software. Ela é fortemente focada na documentação do sistema e normalmente é descrita a partir de três perspectivas:

1. Dinâmica: mostra as fases ao longo do tempo.
2. Estática: mostra atividades de processo.
3. Prática: sugere bons princípios e práticas de desenvolvimento.

A abordagem RUP é uma tentativa de aproveitar os melhores recursos e características dos modelos tradicionais de processo de software, mas caracterizando-os de modo a implementar muitas das melhores práticas do desenvolvimento ágil de software. O RUP reconhece a importância da comunicação com o cliente e de métodos sequenciais para descrever a visão do cliente sobre um sistema. Ele enfatiza o importante

papel da arquitetura de software e ajuda o arquiteto a manter o foco nas metas corretas, tais como confiabilidade e reutilização. Ele propõe um fluxo de processo iterativo e incremental, proporcionando a sensação evolucionária que é essencial no desenvolvimento de software moderno. (Pressman, 2011)

Segue uma representação gráfica dessa abordagem:

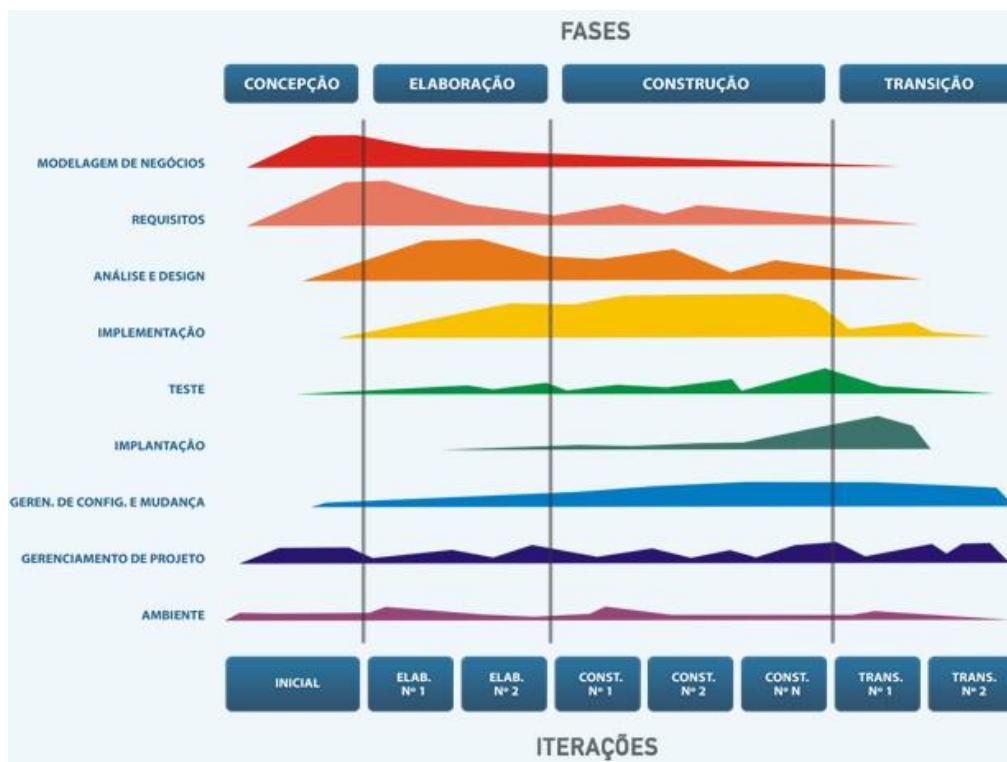


Figura 1.2. A abordagem RUP - Fases e Atividades

1.3.1. Fases do RUP

- Concepção
 - Estabelecer o domínio de negócio para o sistema
- Elaboração
 - Desenvolver um entendimento do domínio do problema e a arquitetura do sistema
- Construção
 - Projeto, programação e teste de sistema
- Transição
 - Implantar o sistema no seu ambiente operacional

1.3.2. Boas Práticas do RUP

- Desenvolver o software iterativamente
- Gerenciar requisitos
- Usar arquiteturas baseadas em componentes
- Modelar o software visualmente

- Verificar a qualidade de software
- Controlar as mudanças do software

1.4. Manifesto Ágil

O Manifesto Ágil se trata de um documento que foi assinado e divulgado em 2001, por Kent Beck e outros dezesseis renomados desenvolvedores, autores e consultores da área de software (batizados de “Agile Alliance”- “Aliança dos Ágeis”) assinaram o “Manifesto para o Desenvolvimento Ágil de Software” (“Manifesto for Agile Software Development”). (Pressman, 2011)

O documento se inicia da seguinte maneira:

Desenvolvendo e ajudando outros a desenvolver software, estamos desvendando formas melhores de desenvolvimento. Por meio deste trabalho passamos a valorizar:

- Indivíduos e interações acima de processos e ferramentas
- Software operacional acima de documentação completa
- Colaboração dos clientes acima de negociação contratual
- Respostas a mudanças acima de seguir um plano

Ou seja, embora haja valor nos itens à direita, valorizamos os da esquerda mais ainda.

Normalmente, um manifesto é associado a um movimento político emergente: atacando a velha guarda e sugerindo uma mudança revolucionária (espera-se que para melhor).

De certa forma, é exatamente do que trata o desenvolvimento ágil. Embora as ideias básicas que norteiam o desenvolvimento ágil tenham estado conosco por muitos anos, só há menos de duas décadas que se consolidaram como um “movimento”.

A figura abaixo é uma representação gráfica das principais ideias desse movimento:



Figura 1.3. O Manifesto Ágil

1.4. A abordagem Scrum

A palavra Scrum é de origem inglesa e é o nome de uma atividade que ocorre durante a partida de Rugby. Quando utilizada para definir a abordagem de desenvolvimento de software, ela se refere a um método de desenvolvimento ágil de software concebido por Jeff Sutherland e sua equipe de desenvolvimento no início dos anos 1990. Mais recentemente, foram realizados desenvolvimentos adicionais nos métodos gráficos Scrum por Schwaber e Beedle. (Pressman, 2011)

Os princípios da abordagem Scrum são consistentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades estruturais: requisitos, análise, projeto, evolução e entrega. Em cada atividade metodológica, ocorrem tarefas a realizar dentro de um padrão de processo chamado sprint. O trabalho realizado dentro de um sprint (o número de sprints necessários para cada atividade metodológica varia dependendo do tamanho e da complexidade do produto) é adaptado ao problema em questão e definido, e muitas vezes modificado em tempo real, pela equipe Scrum. O fluxo geral do processo Scrum é ilustrado na Figura 1.4. (Pressman, 2011)

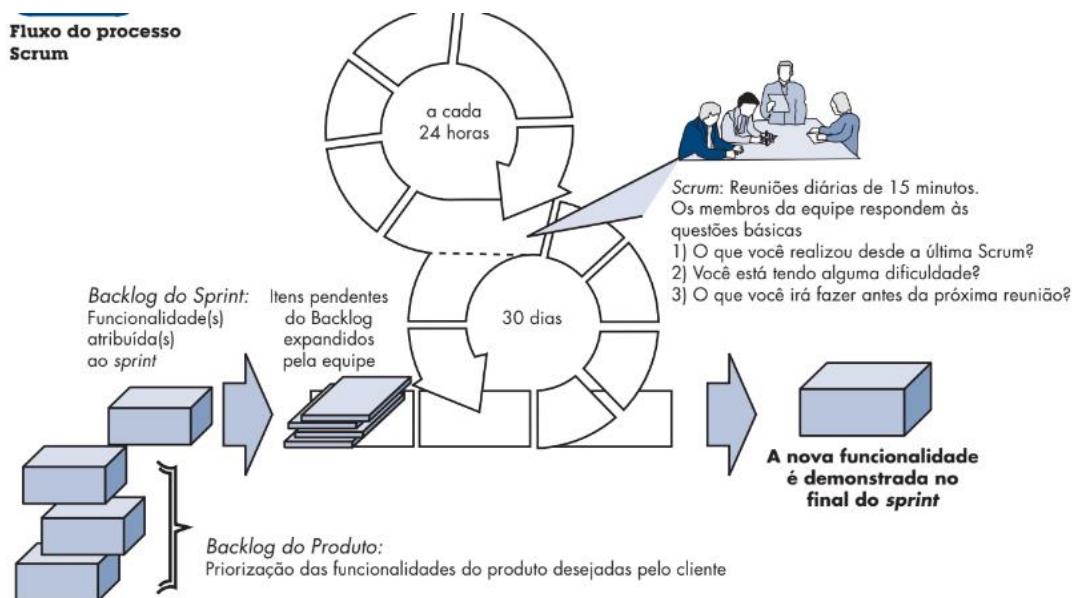


Figura 1.4. Fluxo do processo Scrum

Em cada ciclo de entrega do Scrum, itens do backlog do produto são retirados e avaliados pela equipe de desenvolvimento junto com os representantes de negócios do produto. Estes itens são entendidos e estimados. Os itens que estiverem dentro da

capacidade da equipe para implementação são incluídos na sprint atual e desenvolvidos durante o período da sprint que pode ser de 15, 20 ou até 30 dias. Durante o período da sprint são realizadas reuniões diárias para acompanhamento e alinhamento do trabalho e no fim dela, é realizada uma de entrega dos itens desenvolvidos, chamada de reunião de revisão e outra para avaliar a sprint, chamada de retrospectiva.

As vantagens do Scrum são a adaptabilidade a mudanças devido aos pequenos ciclos de desenvolvimento (sprints), a proximidade do cliente / negócios do desenvolvimento que possibilita maior assertividade na entrega de valor e por fim, desenvolvimento da equipe através das avaliações constantes.

1.6. Você quer ler?

Segue uma indicação de estudo complementar. Trata-se de um artigo do site scrum.org que explica o que é o Scrum.

- O que é Scrum? Disponível em: <<https://www.scrum.org/resources/what-is-scrum>>

1.6. Referências

- PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.
- Figuras 1.1, 1.2, 1.3 e 1.4: PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.

10

TEXTO BASE

ENGENHARIA DE SOFTWARE



Texto base

10

Introdução a UML

Prof. João de Deus Freire Junior

Resumo

A UML é a linguagem de modelagem unificada. Trata-se de uma linguagem de modelagem de soluções e sistemas. Ela auxilia na compreensão da solução sistêmica na perspectiva interna estática e iterativa do sistema e na perspectiva do cliente. Ela é formada por diagramas que possibilitam essa visão completa de solução. Nesta aula, será exibida as principais definições da UML e seus diagramas.

1.1. Introdução

Há alguma linguagem de modelagem de sistemas? É possível projetarmos uma solução antes de a construirmos? O que significa UML? Quais são os principais diagramas da UML? Todas essas perguntas serão respondidas nesta aula. O entendimento da linguagem UML é essencial para modelagem adequada e abrangente de sistemas, esse conhecimento possibilita conhecer várias perspectivas que um sistema pode ser modelado e projetado antes do seu desenvolvimento. Desta forma, podemos mitigar e diminuir significativamente as falhas e fracassos no desenvolvimentos de projetos de soluções tecnológicas.

1.2. O que é UML

UML (Unified Modeling Language – linguagem de modelagem unificada) é uma linguagem-padrão para descrever/documentar projeto de software. A UML pode ser usada para visualizar, especificar, construir e documentar os artefatos de um sistema de software. (Pressman, 2011)

Grady Booch, Jim Rumbaugh e Ivar Jacobson desenvolveram a UML na década de 1990 com muito feedback da comunidade de desenvolvimento de software. A UML combinou um grupo de notações de modelagem concorrentes usadas pela indústria do software na época. (Pressman, 2011)

Em 1997, a UML 1.0 foi apresentada ao OMG (Object Management Group), uma associação sem fins lucrativos dedicada a manter especificações para ser usadas pela indústria de computadores. A UML 1.0 foi revisada tornando-se a UML 1.1 e adotada mais tarde naquele ano. O padrão atual é a UML 2.0 e agora é um padrão ISO. (Pressman, 2011)

1.3. Principais Diagramas da UML

A UML 2.0 fornece 13 diferentes diagramas para uso na modelagem de software. Os principais diagramas são:

- De classe;
- Distribuição;
- Caso de uso;
- Sequência;
- Comunicação;
- Atividade;
- Estado.

De acordo com Sommerville (2011), através desses diagramas podemos modelar os sistemas em perspectivas diferentes. Por exemplo:

1. Uma perspectiva externa, em que você modela o contexto ou o ambiente do sistema.
2. Uma perspectiva de interação, em que você modela as interações entre um sistema e seu ambiente, ou entre os componentes de um sistema.
3. Uma perspectiva estrutural, em que você modela a organização de um sistema ou a estrutura dos dados processados pelo sistema.
4. Uma perspectiva comportamental, em que você modela o comportamento dinâmico do sistema e como ele reage aos eventos.

Sommerville (2011) afirma também que os cinco principais diagramas da UML podem representar a essência de um sistema. Eles os fazem da seguinte forma:

1. Diagramas de atividades, que mostram as atividades envolvidas em um processo ou no processamento de dados.

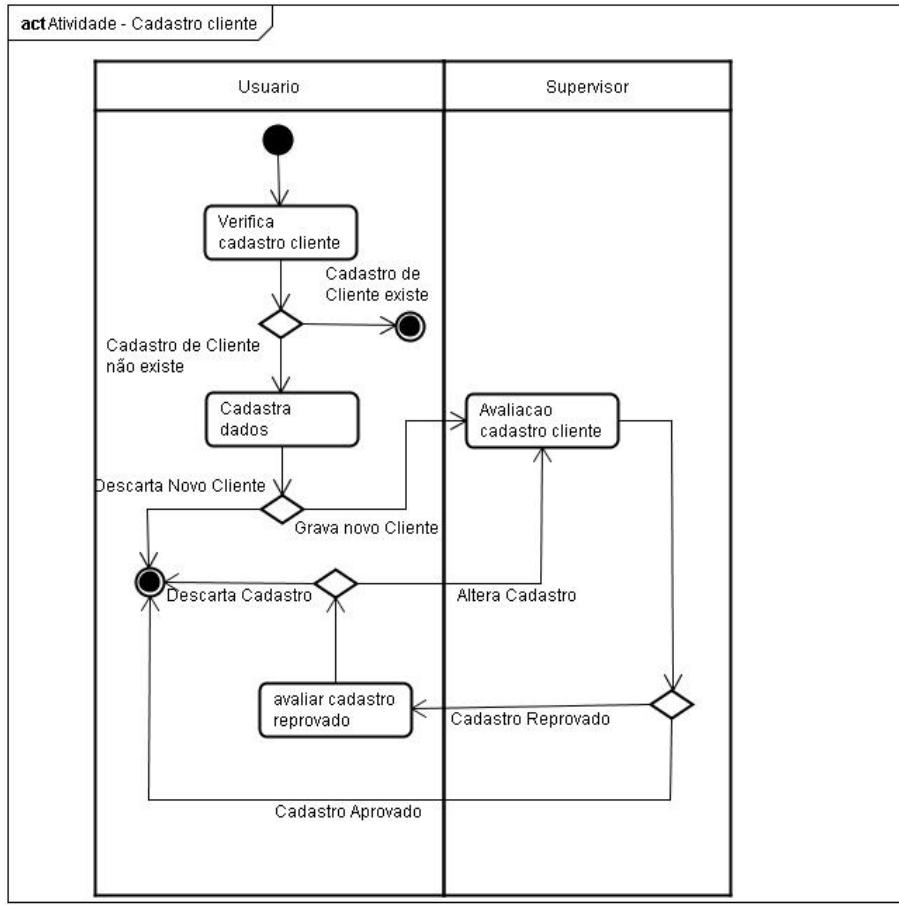


Figura 1.1. Diagrama de Atividades. Fonte: SOMMERVILLE, Ian.(2011)

2. Diagramas de casos de uso, que mostram as interações entre um sistema e seu ambiente.

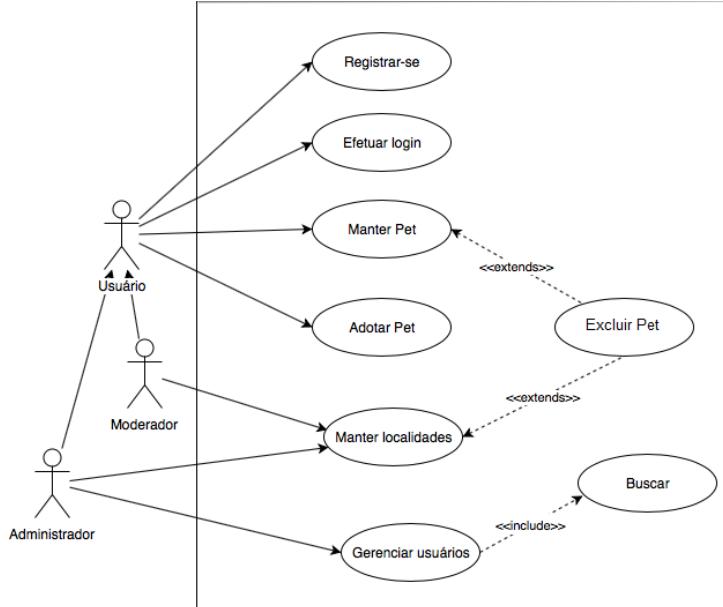


Figura 1.2. Diagrama de Caso de Uso. Fonte: SOMMERVILLE, Ian.(2011)

3. Diagramas de sequência, que mostram as interações entre os atores e o sistema, e entre os componentes do sistema.

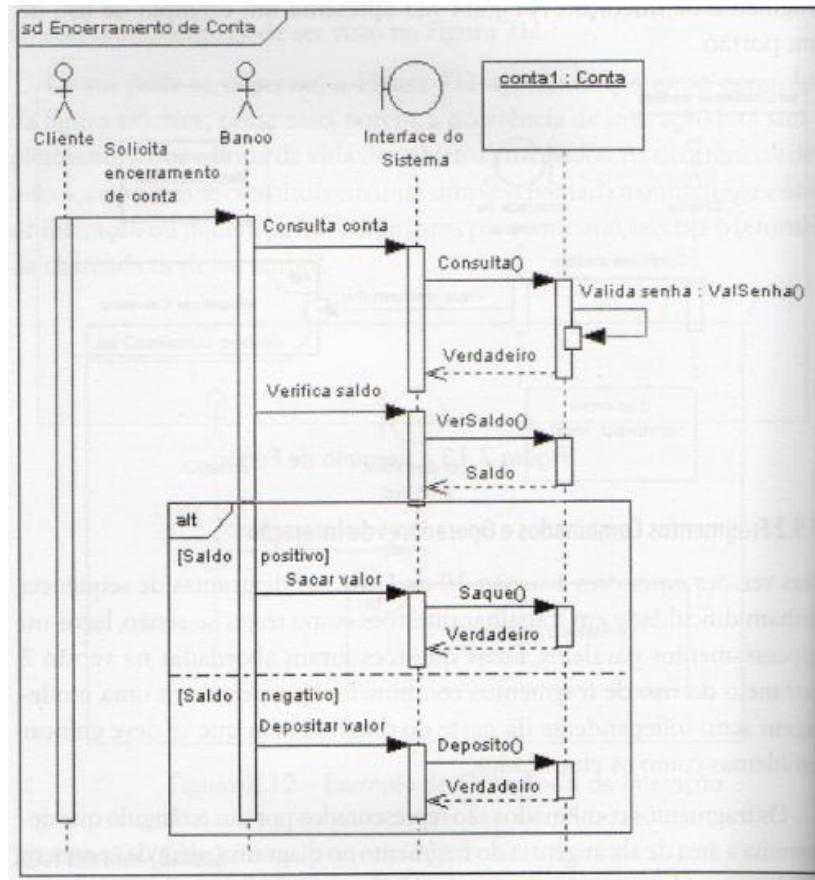


Figura 1.3. Diagrama de Sequência. Fonte: SOMMERVILLE, Ian.(2011)

4. Diagramas de classe, que mostram as classes de objeto no sistema e as associações entre elas.

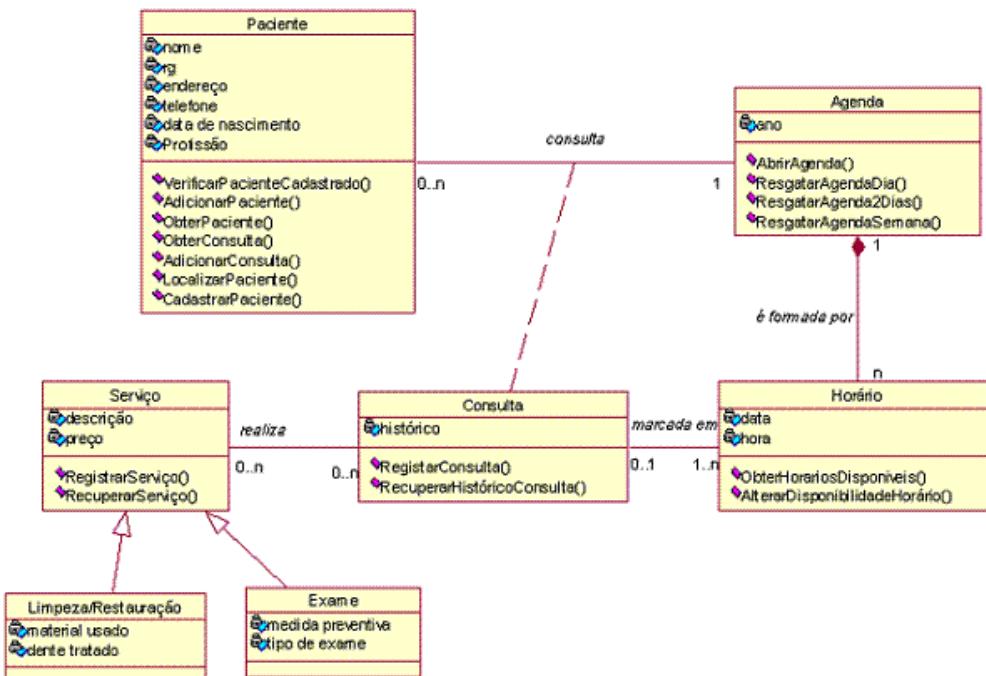


Figura 1.4. Diagrama de Classe. Fonte: SOMMERVILLE, Ian.(2011)

5. Diagramas de estado, que mostram como o sistema reage aos eventos internos e externos.



Figura 1.5. Diagrama de Estados. Fonte: SOMMERVILLE, Ian.(2011)

1.4. Você quer ler?

Segue uma indicação de estudo complementar. Trata-se de um breve guia da linguagem UML.

- FOWLER, Martin. UML Essencial: Um breve guia para a linguagem-padrão de modelagem de objetos.. 2005. Disponível em: <[https://integrada.minhabiblioteca.com.br/#/books/9788560031382/epubcfi/6/8\[idloc_003.xhtml-itemref\]/!4\[eid604\]/20\[eid709\]/6\[eid716\]@0:49.4](https://integrada.minhabiblioteca.com.br/#/books/9788560031382/epubcfi/6/8[idloc_003.xhtml-itemref]/!4[eid604]/20[eid709]/6[eid716]@0:49.4)>. Acesso em: 20 dez. 2019.

1.5. Referências

- PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.
- SOMMERVILLE, Ian.(2011) Engenharia de Software. 9.ed. São Paulo: Pearson Prentice Hall, 2011.

- Figuras 1.1, 1.2, 1.3, 1.4 e 1.5: SOMMERVILLE, Ian.(2011)

11

TEXTO BASE

ENGENHARIA DE SOFTWARE



Texto base

11

Diagrama de Caso de Uso

Prof. João de Deus Freire Junior

Resumo

A UML é a linguagem de modelagem unificada. Trata-se de uma linguagem de modelagem de soluções e sistemas. Ela auxilia na compreensão da solução sistêmica na perspectiva interna estática e iterativa do sistema e na perspectiva do cliente. Ela é formada por diagramas que possibilitam essa visão completa de solução. Nesta aula, será exibido o Diagrama de Caso que é um dos principais diagramas da UML ao fornecer uma visão externa da solução.

1.1. Introdução

Como posso projetar uma solução sistêmica na perspectiva do cliente? Como posso construir uma representação gráfica das metas dos clientes ao utilizar o sistema? O que é o diagrama de caso de uso? Quais são os principais elementos do diagrama de caso de uso? Todas essas perguntas serão respondidas nesta aula. O entendimento da dos elementos, usos e formato do diagrama de caso de uso da UML possibilitará aos analistas uma poderosa ferramenta de modelagem para expressar a visão do cliente ao utilizar o sistema. Desta forma, podemos mitigar e diminuir significativamente as falhas e fracassos no desenvolvimentos de projetos de soluções tecnológicas.

1.2. Principais Diagramas da UML

A UML 2.0 fornece 13 diferentes diagramas para uso na modelagem de software. Os principais diagramas são:

- De classe;
- Distribuição;

- Caso de uso;
- Sequência;
- Comunicação;
- Atividade;
- Estado.

De acordo com Sommerville (2011), através desses diagramas podemos modelar os sistemas em perspectivas diferentes. Por exemplo:

1. Uma perspectiva externa, em que você modela o contexto ou o ambiente do sistema.
2. Uma perspectiva de interação, em que você modela as interações entre um sistema e seu ambiente, ou entre os componentes de um sistema.
3. Uma perspectiva estrutural, em que você modela a organização de um sistema ou a estrutura dos dados processados pelo sistema.
4. Uma perspectiva comportamental, em que você modela o comportamento dinâmico do sistema e como ele reage aos eventos.

Sommerville (2011) afirma também que os cinco principais diagramas da UML podem representar a essência de um sistema. Eles os fazem da seguinte forma:

1. Diagramas de atividades, que mostram as atividades envolvidas em um processo ou no processamento de dados.
2. Diagramas de casos de uso, que mostram as interações entre um sistema e seu ambiente.
3. Diagramas de sequência, que mostram as interações entre os atores e o sistema, e entre os componentes do sistema.
4. Diagramas de classe, que mostram as classes de objeto no sistema e as associações entre elas.
5. Diagramas de estado, que mostram como o sistema reage aos eventos internos e externos.

1.3. Diagrama de Caso de Uso

1.3.1. Definições Gerais

O diagrama de caso de uso por meio de uma linguagem simples, demonstra o comportamento externo do sistema, mostra o sistema pela perspectiva do usuário,

indicando as funções e serviços oferecidos e quais usuários utilizarão cada um deles. (Pressman, 2011)

Esse diagrama é o diagrama mais abstrato, informal e flexível, sendo usado no início da modelagem. Ele pode ser modificado ao longo do desenvolvimento do software, serve de base para os demais diagramas da UML. (Pressman, 2011)

Os elementos principais do Diagrama UC são: Atores, Caso de Uso e Associações.

1.3.1. Os atores

Os atores representam os papéis dos diversos usuários do sistema. Cada ator tem uma ou mais metas específicas que quer atingir/obter ao usar o sistema. Os atores podem representar um ser humano, um periférico de hardware ou outro sistema.

Um ator é representado, na UML, pelo “stickman”.



Figura 1.1. Diagrama de Caso de Uso - Ator e Metas. Fonte: PRESSMAN, R. S. (2011)

1.3.2. O caso de uso

O caso de uso é a meta do usuário ao utilizar o sistema e ele é representado, na UML, por uma elipse com o Nome do UC em seu interior.

Há algumas regras para escrita dos nomes dos casos de uso, segue abaixo as principais:

- Deve ser único, intuitivo e autoexplicativo;
- Deve definir o resultado observável que o Caso de Uso fornece ao Ator;
- Deve estar na perspectiva do ator que “dispara” o UC;
- Deve descrever o comportamento sustentado pelo UC;
- Deve iniciar com verbo no infinitivo.



Figura 1.2. Caso de Uso. Fonte: PRESSMAN, R. S. (2011)

1.3.3. Associações

As associações representam um diálogo completo entre um Ator e um UC. Segue um exemplo de um estudante interagindo com um sistema de catálogo de disciplinas para fazer matrícula.

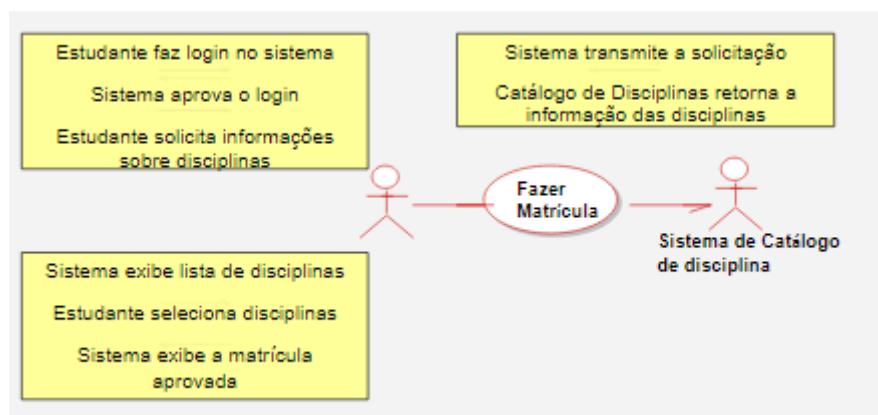


Figura 1.3. Associações. Fonte: PRESSMAN, R. S. (2011)

1.3.3. Tipos de Associações

- **Especialização/Generalização:** ocorre quando há características semelhantes (e com poucas diferenças entre si) entre Casos de Uso ou entre Atores (nunca entre UC e Ator!);
- **Inclusão:** usada quando existe uma situação ou rotina comum a mais de um caso de uso (evitar repetições); estereótipo <<INCLUDE>>; seta aponta para o UC incluído;
- **Extensão:** usada para descrever cenários opcionais de um UC, só quando certa condição for satisfeita; estereótipo <<EXTEND>>; seta aponta para o UC que estende.

1.3.4. Exemplo de Diagrama de Caso de Uso

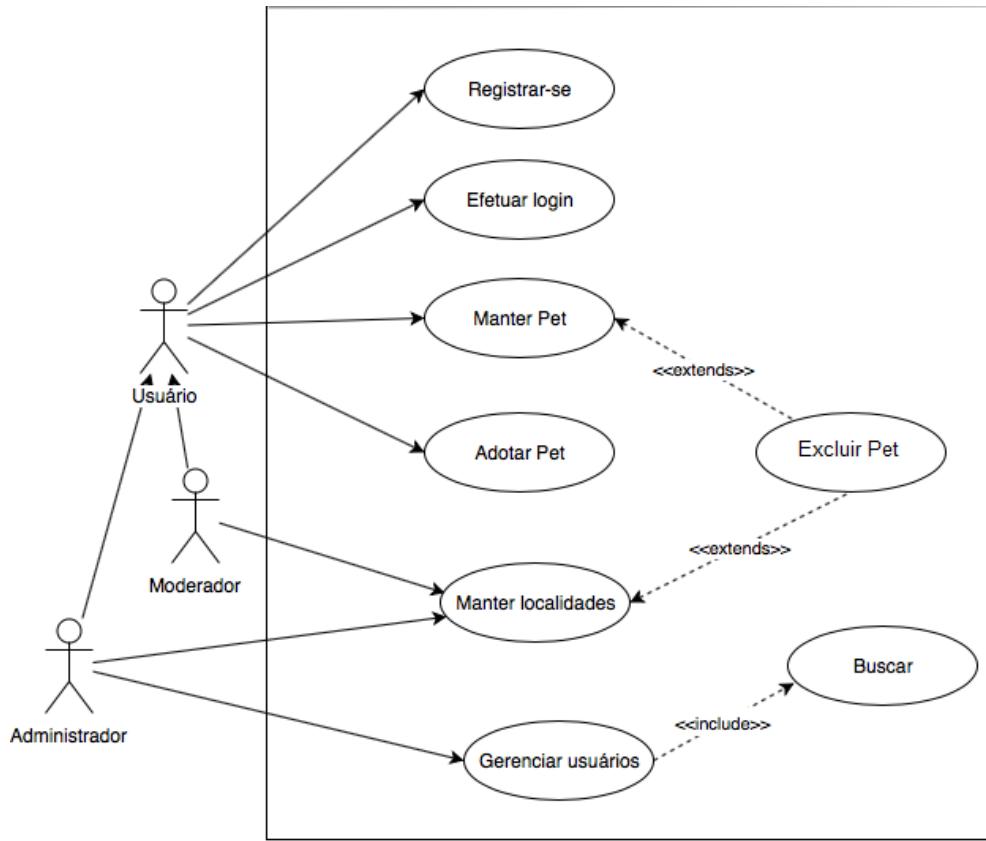


Figura 1.4. Exemplo de Caso de Uso. Fonte: PRESSMAN, R. S. (2011)

1.4. Você quer ler?

Segue uma indicação de estudo complementar. Trata-se de um breve guia da linguagem UML.

- LARMAN, Craig. Utilizando UML e Padrões: Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo.. 2007. Disponível em:
<https://integrada.minhabiblioteca.com.br/#/books/9788577800476/pageid/33>. Acesso em: 20 dez. 2019..

1.5. Referências

- PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.

- Figuras 1.1, 1.2, 1.3, 1.4 e 1.5: PRESSMAN, R. S.(2011)

12

TEXTO BASE

ENGENHARIA DE SOFTWARE



Texto base

12

Diagrama de Classe, Diagrama de Sequência e Diagrama de Atividades

Prof. João de Deus Freire Junior

Resumo

A UML é a linguagem de modelagem unificada. Trata-se de uma linguagem de modelagem de soluções e sistemas. Ela auxilia na compreensão da solução sistêmica na perspectiva interna estática e iterativa do sistema e na perspectiva do cliente. Ela é formada por diagramas que possibilitam essa visão completa de solução. Nesta aula, será exibido os Diagrama de Classe, Diagrama de Sequência e Diagrama de Atividades que são três dos principais diagramas da UML.

1.1. Introdução

Como posso projetar uma solução sistêmica na perspectiva interna do sistema? Como posso construir uma representação gráfica das estruturas do sistema? O que são os diagramas de classe, atividades e sequência? Quais são os principais elementos desses diagramas? Todas essas perguntas serão respondidas nesta aula. O entendimento dos elementos, usos e formato desses diagramas da UML possibilitará aos analistas uma poderosa ferramenta de modelagem para expressar a visão interna do sistema. Desta forma, podemos mitigar e diminuir significativamente as falhas e fracassos no desenvolvimentos de projetos de soluções tecnológicas.

1.2. Principais Diagramas da UML

A UML 2.0 fornece 13 diferentes diagramas para uso na modelagem de software. Os principais diagramas são:

- De classe;

- Distribuição;
- Caso de uso;
- Sequência;
- Comunicação;
- Atividade;
- Estado.

De acordo com Sommerville (2011), através desses diagramas podemos modelar os sistemas em perspectivas diferentes. Por exemplo:

1. Uma perspectiva externa, em que você modela o contexto ou o ambiente do sistema.
2. Uma perspectiva de interação, em que você modela as interações entre um sistema e seu ambiente, ou entre os componentes de um sistema.
3. Uma perspectiva estrutural, em que você modela a organização de um sistema ou a estrutura dos dados processados pelo sistema.
4. Uma perspectiva comportamental, em que você modela o comportamento dinâmico do sistema e como ele reage aos eventos.

Sommerville (2011) afirma também que os cinco principais diagramas da UML podem representar a essência de um sistema. Eles os fazem da seguinte forma:

1. Diagramas de atividades, que mostram as atividades envolvidas em um processo ou no processamento de dados.
2. Diagramas de casos de uso, que mostram as interações entre um sistema e seu ambiente.
3. Diagramas de sequência, que mostram as interações entre os atores e o sistema, e entre os componentes do sistema.
4. Diagramas de classe, que mostram as classes de objeto no sistema e as associações entre elas.
5. Diagramas de estado, que mostram como o sistema reage aos eventos internos e externos.

1.3. Diagrama de Classe

1.3.1. Definições Gerais

O diagrama de classe é o mais utilizado da UML. Ele permite a visualização das classes utilizadas pelo sistema e como elas se relacionam. Ele apresenta uma visão estática da organização das classes, definindo sua estrutura lógica. Ele pode ser usado

para definir um modelo lógico de um banco de dados (mas, uma classe não corresponde a uma tabela no BD). (Pressman, 2011)

Ele é formado por classes que são compostas por: nome da classe, lista de atributos e lista de operações (métodos).

1.3.1. As classes

As classes no diagrama de classes são compostas por: nome da classe, lista de atributos e lista de operações (métodos).

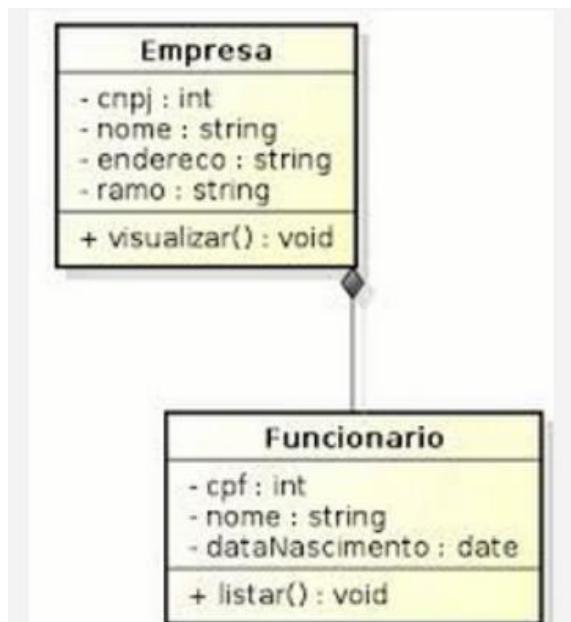


Figura 1.1. Classes. Fonte: PRESSMAN, R. S. (2011)

1.3.2. Associações

As associações são compartilhamento de informações e colaboração para execução dos processos do sistema; descrevem o vínculo entre os objetos de uma ou mais classes. (Pressman, 2011)

As associações exibem a navegabilidade (caminho da informação) e multiplicidade (números mínimo e máximo de objetos envolvidos);

1.3.3. Tipos de Associações

- **Unária:** relacionamento de objeto de uma classe com objetos da mesma classe.

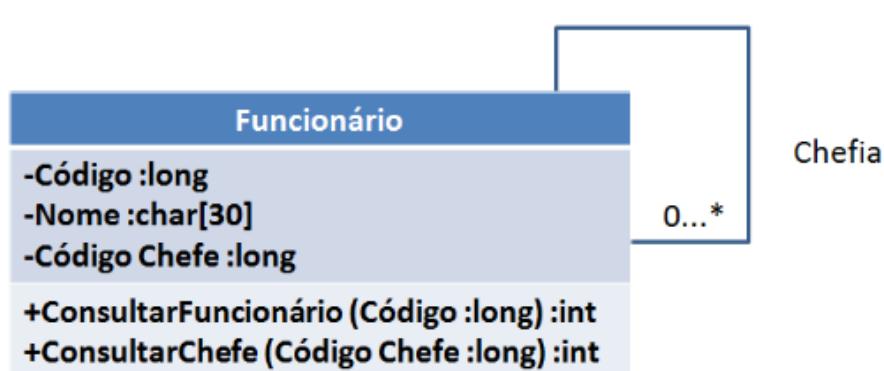


Figura 1.2. Associação Unária. Fonte: PRESSMAN, R. S. (2011)

- **Binária:** quando há relacionamento de objetos de duas classes.

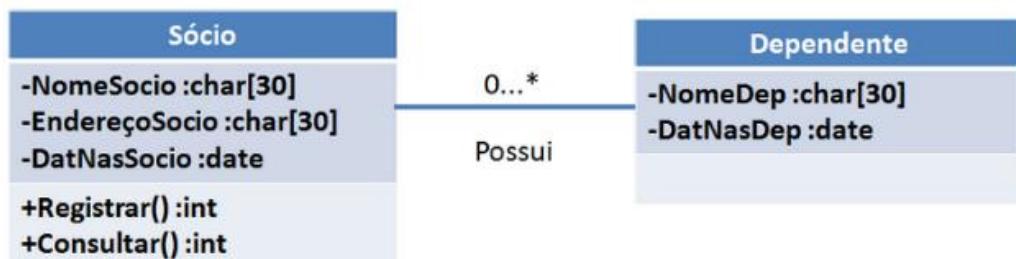


Figura 1.3. Associação Binária. Fonte: PRESSMAN, R. S. (2011)

- **Ternária:** conectam objetos de mais de duas classes.

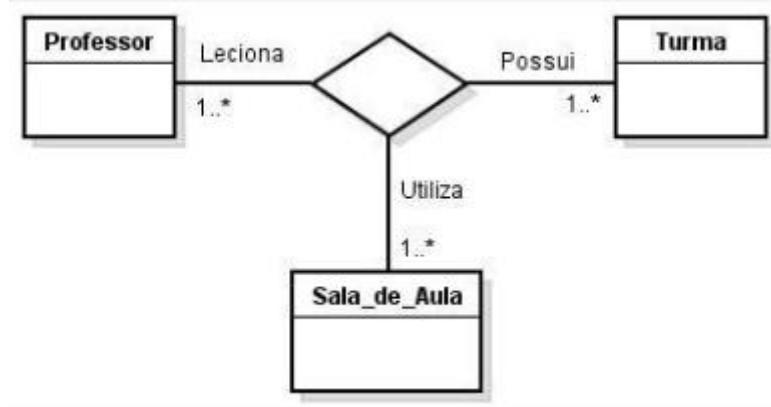


Figura 1.4. Associação Ternária. Fonte: PRESSMAN, R. S. (2011)

- **Agregação:** demonstra relação todo/parte entre objetos associados.

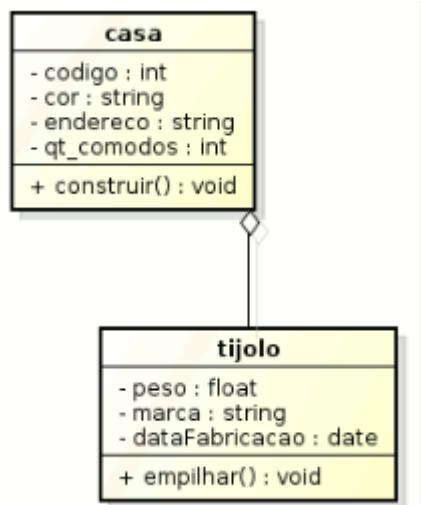


Figura 1.5. Associação Agregação. Fonte: PRESSMAN, R. S. (2011)

- **Composição:** variação da agregação, com vínculo mais forte.

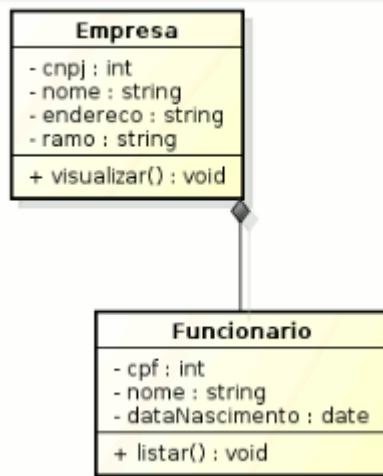


Figura 1.6. Associação Composição. Fonte: PRESSMAN, R. S. (2011)

- **Especialização / Generalização:** identifica classes-mãe e classes-filhas, com ocorrência de herança.

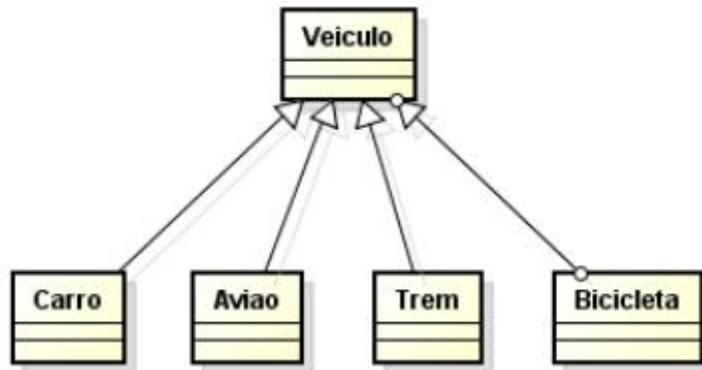


Figura 1.7. Associação Especialização / Generalização. Fonte: PRESSMAN, R. S. (2011)

1.3.4. Exemplo de Diagrama de Classe

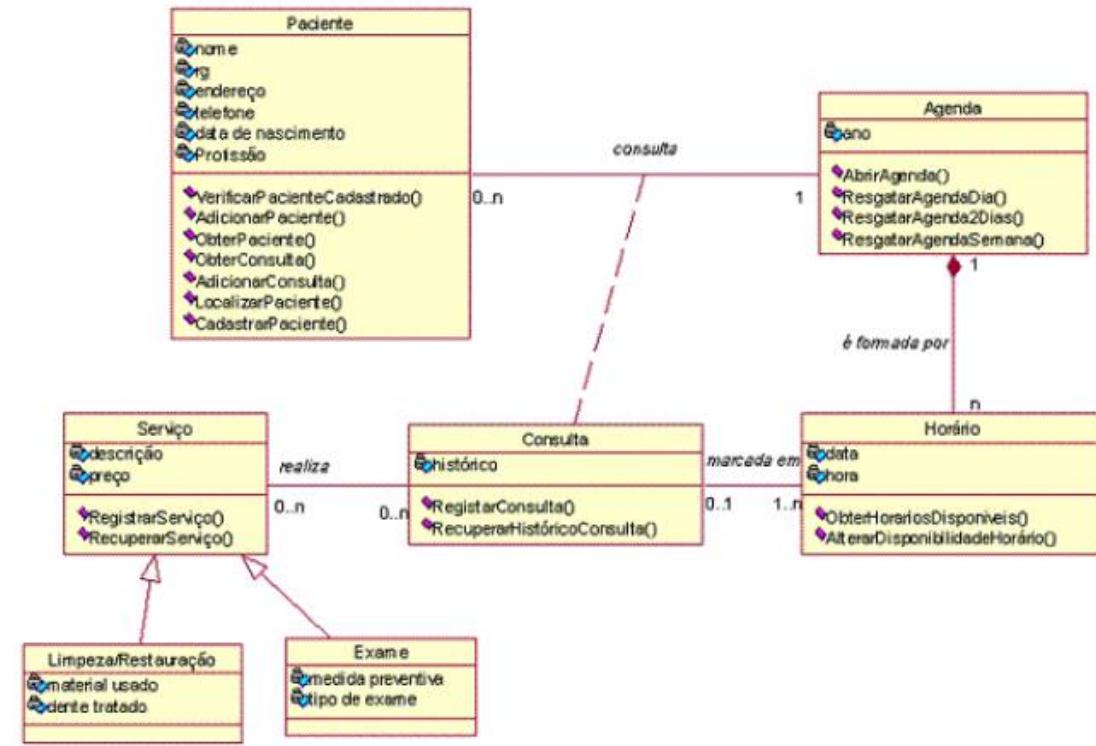


Figura 1.8. Exemplo de Caso de Uso. Fonte: PRESSMAN, R. S. (2011)

1.4. Diagrama de Sequência

1.4.1. Definições Gerais

O diagrama de sequência procura determinar a sequência de eventos que ocorrem em um processo. Ele identifica quais métodos devem ser disparados entre os atores e objetos envolvidos e em qual ordem. (Pressman, 2011)

O diagrama de sequência baseia-se no Diagrama de UC: normalmente há um diagrama de sequência para cada UC (cada UC é um processo disparado pelo Ator). Ele também depende do Diagrama de Classes: classes dos objetos declarados, além dos métodos. (Pressman, 2011)

1.4.2. Elementos

De acordo com Pressman (2011), o diagrama de sequência é composto pelos seguintes elementos:

- **Atores:** são os mesmos dos Diagramas UC; aqui, possuem “linha de vida”;
- **Objetos:** representam as instâncias das classes do processo ilustrado no Diagrama de Sequência; um objeto pode existir desde o início do processo ou ser criado durante a execução do mesmo;
- **Linha de Vida:** representa o tempo que um objeto existe durante o processo. Representada por uma linha fina e tracejada, é interrompida com um “X” quando o objeto é destruído;
- **Foco de Controle (Ativação):** indica os períodos que um objeto participa ativamente do processo. Na linha da vida, é representado por uma linha grossa.
- **Mensagens (Estímulos):** demonstram a ocorrência de eventos, que forçam a chamada de um método em um dos objetos envolvidos no processo.

Representadas por uma seta entre dois componentes (quem envia para quem recebe). O texto contido nelas indicam o evento ocorrido e o método chamado.

- **Mensagens de Retorno:** identifica a resposta a uma mensagem para o objeto que a chamou; pode retornar informações específicas ou flag de sucesso ou não.
- **Autochamadas:** mensagens que partem da linha de vida do objeto e atingem a linha de vida do mesmo objeto.
- **Condições de Guarda:** estabelecem uma regra ou condição para que uma mensagem possa ser disparada.

1.4.3. Exemplo de Diagrama de Sequência

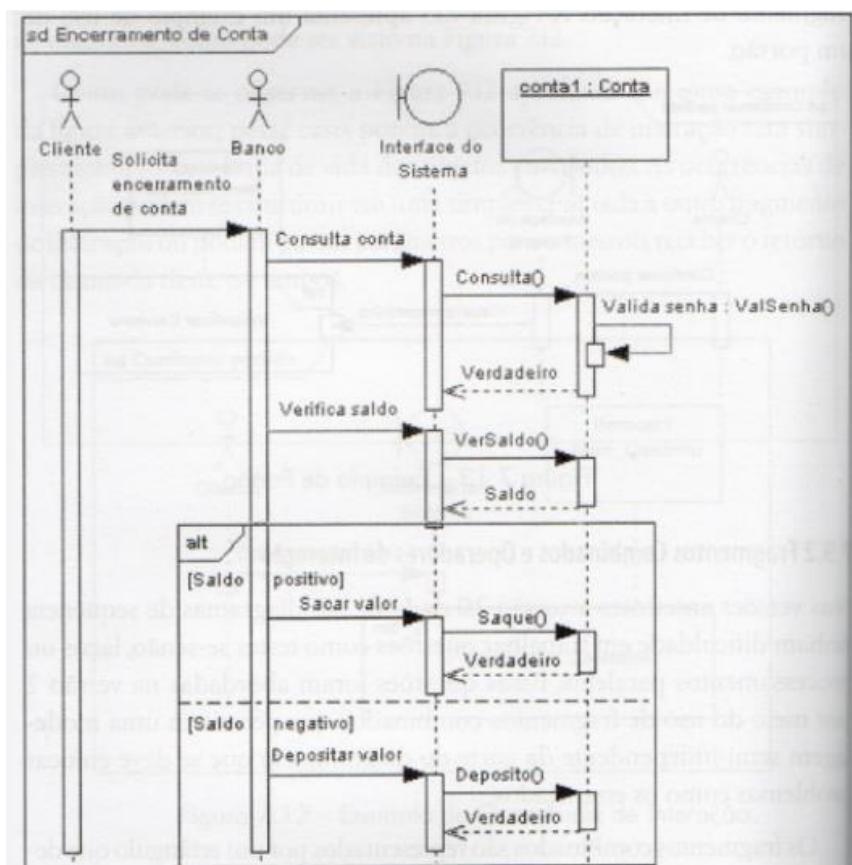


Figura 1.9. Exemplo de Diagrama de Sequência. Fonte: PRESSMAN, R. S. (2011)

1.5. Diagrama de Atividades

1.5.1. Definições Gerais

O diagrama de atividades dá maior ênfase ao nível de algoritmo (um dos mais

detalhistas). Ele é utilizado para modelar atividades, que podem ser um método, algoritmo ou um processo completo. Ele é semelhante aos antigos “fluxogramas”. Ele possui partição de atividades para representar o fluxo de um processo que envolve diversos atores (ou departamentos, setores, etc...). (Pressman, 2011)

1.5.2. Exemplo de Diagrama de Atividades

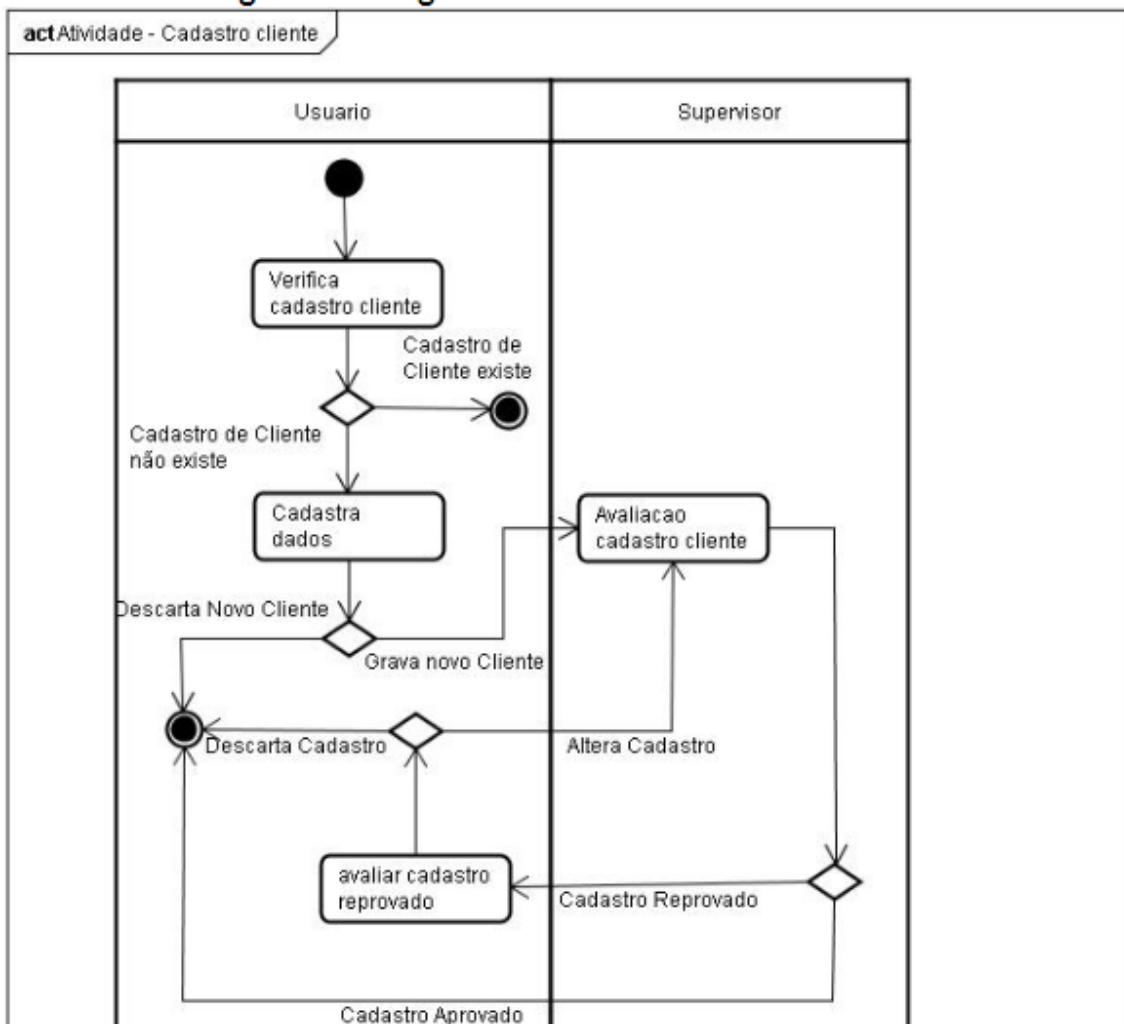


Figura 1.10. Exemplo de Diagrama de Atividades. Fonte: PRESSMAN, R. S. (2011)

1.6. Você quer ler?

Segue uma indicação de estudo complementar. Trata-se de um breve guia da linguagem UML.

- DEVMEDIA. Conhecendo os diagramas da UML através de um estudo de caso. 2011. Disponível em: <<https://www.devmedia.com.br/conhecendo-os-diagramas-da-uml-um-introductorio/1>>

<diagramas-da-uml-atraves-de-um-estudo-de-caso/22550>. Acesso em: 20 dez. 2019.

1.7. Referências

- PRESSMAN, R. S.(2011) Engenharia de Software: uma abordagem profissional. 7.ed. Porto Alegre: Bookman, 2016.
- Figuras 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10: PRESSMAN, R. S.(2011)