AULA 1 – INTRODUÇÃO À ENGENHARIA DE SOFTWARE

OBJETIVO DA AULA

Compreender os conceitos iniciais sobre software e a aplicação do contexto de engenharia ao cenário de produção de software.

APRESENTAÇÃO

O profissional de tecnologia da informação é responsável por uma atividade essencial no dia a dia de qualquer negócio: a resolução de problemas.

Os softwares são programas de computadores desenvolvidos para um cliente específico ou para o mercado em geral, e visam possibilitar com que os seus usuários tenham o seu trabalho afetado positivamente por meio da tecnologia.

O contexto de engenharia atribuído à elaboração de software tem o objetivo de aplicar métodos, práticas e ferramentas na resolução de problemas, visando garantir eficiência, praticidade e segurança aos processos apoiados por ele.

Nesta aula vamos aprender um pouco sobre o papel da engenharia de software na elaboração de softwares.

1. A IMPORTÂNCIA DO SOFTWARE NA ATUALIDADE

Hoje é praticamente impossível pensar em qualquer atividade pessoal ou profissional sem o apoio de um software. Os softwares podem estar presentes em pequenas soluções através de sistemas embutidos, em geladeiras e tvs, por exemplo, ou grandes sistemas de informação, como os de bancos e companhias aéreas.

A nossa vida pessoal também é muito afetada diariamente por conta do uso dos softwares, e neste caso podemos citar o controle dos nossos compromissos diários em agendas eletrônicas, marcações de consultas médicas, até a compra de ingressos para shows e cinema, isso sem contar com todas as facilidades que os softwares que estão nos smartphones nos proporcionam.

Vale a pena dizer que o profissional de tecnologia da informação, por ter como especialidade a resolução de problemas, acaba transitando pelos diversos níveis organizacionais, a saber, operacional, tático e estratégico.

FIGURA 1 | **Pirâmide Organizacional**

Fonte: Laudon e Laudon.

Para cada nível, necessidades diferentes são observadas, pois o gerente do nível estratégico tem a necessidade de observar o dado de uma maneira diferente da visão que necessita o gerente operacional.

Assim, toda vez que o profissional de software estiver envolvido na produção de algum software, é importante conhecer bem os *stakeholders*. Os *stakeholders* são os profissionais interessados e envolvidos no projeto, aqueles que podem tomar decisão durante o andamento do projeto.

Alguns exemplos de *stakeholders* são os gerentes, o patrocinador do projeto e os *key-users,* que são usuários importantes da parte do cliente. É importante ressaltar que alguns dos próprios membros da equipe de desenvolvimento também são considerados *stakeholders*.

2. SOFTWARE

Mas o que é o software afinal? Em um contexto simples, os softwares são programas de computadores criados para a resolução de um ou mais problemas.

Esses programas podem ser desenvolvidos como produtos que vão apoiar o trabalho de um ou mais profissionais. Esses produtos podem ser divididos em duas categorias: os softwares genéricos e os softwares por encomenda.

Os softwares genéricos são desenvolvidos para que sirvam para várias empresas de um mesmo ramo. Por exemplo, um software para controlar o estoque de uma empresa pode ser facilmente empregado em outra empresa que também deseja controlar o seu estoque.

Já os softwares por encomenda tratam especificamente o problema do cliente solicitante, sendo desenvolvido para uma realidade particular e, portanto, pertencente exclusivamente a quem o encomendou. Um exemplo desse tipo de software seria um sistema construído para controlar as atividades acadêmicas de uma determinada escola.

3. O PROFISSIONAL DE SOFTWARE

A partir daí temos o profissional de software atuando como facilitador técnico para que outros profissionais possam ter as suas atividades diárias influenciadas pelo apoio dos softwares.

A tarefa desse profissional é observar e entender as necessidades do cliente, para poder analisar e propor uma solução computacional que tornará viável, ágil e simplificada a execução das tarefas do cliente.

Este é um ponto interessante para o profissional de tecnologia da informação, de uma forma geral, pois ele passa a conhecer o domínio de negócio ao qual ele está prestes a construir uma solução, entendendo o funcionamento, os processos e as regras de negócio. Uma responsabilidade e tanto, não acha?

A área de tecnologia proporciona diversos desafios que devem ser encarados pelo profissional de TI. Cada um desses desafios está relacionado com o papel e cargo desses profissionais. Por ser uma tarefa muito difícil elencar todos os papéis que um profissional de TI pode possuir, alguns dos mais comuns conseguimos observar na lista a seguir:

a) Desenvolvimento: apoiam atividades organizacionais a partir de sistemas que facilitam o controle, coordenação e tomada de decisão nos negócios.

- Analista de Sistemas;
- Analista de Requisitos;
- Programador;
- Analista de Testes;
- Administrador de Dados.

b) Suporte: tem a responsabilidade de manter os sistemas computacionais e a infraestrutura em funcionamento e com a garantia da segurança.

- Analista de Suporte;
- Analista de Redes;
- Analista de Infraestrutura;
- Administrador de Banco de Dados.

Além dessas duas grandes áreas, ainda temos cargos relacionados à Análise dos Dados, Governança, dentre outras áreas relacionados à tecnologia da informação.

4. ENGENHARIA DE SOFTWARE

Para que tudo isso seja possível, nasceu a Engenharia de Software, uma disciplina que prevê o desenvolvimento profissional de software, propondo métodos, técnicas e ferramentas que permita o desenvolvimento de um produto com qualidade.

Muito antes de tratar o desenvolvimento de software como engenharia, ocorreu a chamada de "Crise do Software". Esse termo se origina dos anos 1970 e relata o momento em que a Engenharia de Software praticamente não existia, algo que se tornou crítico, uma vez que esse período apresentou um crescimento substancial na atividade de desenvolvimento de software em todo o planeta.

Especificamente este termo foi cunhado na apresentação de uma palestra feita por Edsger Dijkstra na *Association for Computing Machinery The Humble Programmer* (EWD340), que gerou artigo publicado no periódico *Communications of the ACM*.

O artigo *The Humble Programmer* (1972) pode ser encontrado no link www.cs.utexas.edu/users/EWD/ewd03xx/EWD340.PDF.

Alguns aspectos conseguem direcionar quais eram os sintomas que apontavam a Crise do Software:

- Projetos que não cumpriam o orçamento previsto;
- Projetos que ultrapassavam o prazo determinado no início do desenvolvimento;
- Software com defeito;
- Software com qualidade pobre e muito difíceis de serem entendidos;
- Projetos difíceis de manutenir;
- Softwares que não entregavam aquilo que era pedido pelo cliente.

Nesse aspecto, a Engenharia proporcionou aos profissionais de desenvolvimento uma espécie de ponto zero, que ajudava os desenvolvedores a utilizarem e compartilharem as melhores práticas para o desenvolvimento, sempre a partir da definição de técnicas, ferramentas e processos que foram bem-sucedidos em projetos diversos.

5. CAUSAS DA FALHA DE PROJETOS DE SOFTWARES

Ainda com todas as definições da Engenharia de Software, os projetos podem falhar caso não sejam considerados alguns aspectos importantes. A lista a seguir apresenta alguns dos motivos clássicos que fazem com que os projetos acabem falhando:

- Escolha errada da tecnologia a ser empregado no projeto;
- Escolha errada do processo de desenvolvimento do software;

- Mudanças rápidas nas regras de negócio;
- Inflexibilidade do sistema ou do processo;
- Pouco tempo de investimento no planejamento do projeto;
- Prazos irreais (subestimação ou superestimação do esforço);
- Falta de recursos adequados (pessoas, máquinas, ferramentas, ambiente etc.);
- Falhas no processo de captura, especificação e gerência de requisitos;
- Baixa participação dos *stakeholders* no projeto;
- Baixo índice de *feedback* pelo cliente;
- Falhas no gerenciamento de riscos;
- Pouco suporte da alta direção.

O profissional de software deve estar atento para cada uma dessas causas e deve considerar os possíveis planos de contingência para o caso de falhas nos quesitos mais importantes, principalmente aqueles que se tem menor controle, como a baixa participação dos *stakeholders*.

6. OS DESDOBRAMENTOS DA ENGENHARIA DE SOFTWARE

O desenvolvimento de software passou por uma evolução natural agregando valores entre pessoas, hardware e softwares disponíveis.

Antes, os softwares eram produzidos para atender a problemas departamentais. Por exemplo, softwares de contabilidade, departamento de pessoal e estoque. Mais precisamente, o objetivo era a automação de processos.

Mas as empresas entenderam que era necessário que os softwares de cada parte de uma organização se comunicassem, evitando assim o registro de informações em duplicidade. Agora o foco estava no aumento da produtividade.

A integração dos sistemas foi essencial para que as informações pudessem ser interpretadas de maneira corporativa, gerando eficiência e criando um diferencial competitivo em relação aos concorrentes diretos.

A aplicação de metodologias de desenvolvimento e o uso de ferramentas que apoiam esses processos tornou-se uma prática, aumentando a qualidade e permitindo que todo um processo de continuidade e aperfeiçoamento do software se tornasse uma realidade.

Atualmente outras iniciativas para o aumento na produtividade do desenvolvimento do software são aplicadas nas fábricas de software, como o compartilhamento de projetos construídos em equipe, o uso de ferramentas de versionamento, e a aplicação dos modelos ágeis de desenvolvimento, fazendo com que a elaboração de software seja facilitada, e que as melhores práticas sejam reconhecidas e aplicadas, tudo isso graças à Engenharia de Software!

CONSIDERAÇÕES FINAIS

Nesta aula inicial abordamos não só a conceituação de software, mas também dedicamos um tempo a entender o importante papel do analista de sistemas nos processos que apoiam a construção de soluções para os mais diversos problemas dos mais distintos domínios.

Traçamos um panorama sobre a forma como o software é importante em nosso dia a dia atualmente, e como era o cenário descrito como “crise do software” em tempos em que não havia a preocupação com a metodologia e os processos para desenvolvimento de software, assim como havia poucas ferramentas de apoio para a atividade de desenvolvimento de software.

Descrevemos também a evolução no conceito de produção de software, apontando quais foram as principais evoluções e como estas ajudaram na melhoria desta atividade essencial para os dias atuais.

MATERIAIS COMPLEMENTARES

Artigo que deu origem ao termo “crise do software” de Edsger Dijkstra: *The Humble Programmer*. Disponível em: https://www.cs.utexas.edu/users/EWD/ewd03xx/EWD340.PDF. Acesso em: 28 out. 2022.

Artigo: Por que os Softwares falham? Disponível em: http://www.linhadecodigo.com.br/artigo/2054/por-que-projetos-de-software-falham.aspx. Acesso em: 28 out. 2022.

REFERÊNCIAS

KENNETH C. LAUDON; JANE P. LAUDON. *Sistemas de informação gerenciais*. 11ª edição. Editora Pearson, 2014.

PRESSMAN, R. G. *Engenharia de Software.* 9ª ed. Rio de Janeiro: McGraw-Hill, 2021.

SOMMERVILLE, I. *Engenharia de Software.* 10ª ed. São Paulo: Pearson Addison. Wesley, 2019.

AULA 2 – ATIVIDADES DO PROCESSO DE SOFTWARE

OBJETIVO DA AULA

Compreender quais são as atividades utilizadas no processo de software, apresentando quais são os objetivos de cada uma delas, bem como estabelecer quais são os artefatos que devem ser elaborados em cada processo.

APRESENTAÇÃO

A construção de software deve considerar o estabelecimento de um processo de elaboração organizado, que deve ser avaliado conforme o objetivo e as características de cada projeto de software.

Esta aula apresenta as atividades essenciais para que o software seja desenvolvido sem que nenhuma fase, seja "esquecida", com o objetivo de aplicar as etapas fundamentais que devem ser aplicadas em qualquer processo.

As atividades básicas do processo de software são: especificação, desenvolvimento, validação e evolução. É importante ressaltar que elas são conduzidas e organizadas de maneira diferente de acordo com o processo de desenvolvimento escolhido. A correta aplicação das mesmas certamente facilitará o processo de desenvolvimento, tanto no quesito de completude, quanto o de busca pela qualidade.

1. PROCESSO DE SOFTWARE

Em Engenharia de Software devemos escolher o processo de software que deverá ser implementado, ou seja, o método que utilizaremos no desenvolvimento do produto como, por exemplo, o processo em cascata ou incremental.

Antes disso, precisamos conhecer o que chamamos de atividades ou disciplinas do processo de software. Essas disciplinas são normalmente organizadas em 5 atividades básicas: especificação, desenvolvimento, validação e evolução.

Embora a literatura apresente variações quanto a essa organização de atividades, é importante destacar que elas serão facilmente definidas como etapas da maioria dos processos de software que estudaremos. O que muda é a forma como essas disciplinas são organizadas e implementadas.

Os Processos de Software são as fases indispensáveis para produzir e manter um software. Cada atividade possui uma organização de suas atividades técnicas e gerências, e envolvem métodos, ferramentas, artefatos e regras que possibilitam sistematizar e organizar o desenvolvimento e manutenção de produtos de software.

Essas atividades visam definir:

- Os artefatos (qualquer produto gerado no desenvolvimento de um software, como Códigos-fonte, diagramas, documentos que serão desenvolvidos);
- Quando e como eles serão desenvolvidos;
- O perfil de quem irá desenvolvê-los;
- A razão do desenvolvimento de cada artefato;
- Onde os artefatos serão desenvolvidos.

Além de ajudar na organização e sistematização do desenvolvimento do software, ao final de cada atividade os artefatos também podem ser utilizados como documentação final do software.

É certo que a correta aplicação das atividades do processo de software agregarão incontáveis benefícios ao projeto como um todo. É notório que os principais benefícios percebidos dizem respeito à clareza da equipe desenvolvedora sobre o software porque todos os membros da equipe são capazes de compreender o que deve ser feito ao longo do desenvolvimento do produto, além de também saberem o papel de cada desenvolvedor no projeto.

Vale também ressaltar o crescimento da equipe naquele processo de desenvolvimento, pois cada vez que o processo é repetido, mais assertivo a equipe ficará em termos de custo, organização do cronograma e definição dos prazos de entrega.

O uso das atividades de um processo de software torna-se fundamental sempre que for necessária a construção de um software, buscando a sua qualidade e a sua continuidade para que seja prolongada a vida útil do produto. Sendo assim, vamos conhecer as tarefas e ferramentas destinadas a cada atividade.

2. ESPECIFICAÇÃO

É a atividade aplicada à identificação das necessidades do software. Conhecer o que o sistema faz e o que como o usuário terá o seu trabalho afetado por ele.

Também conhecida como engenharia de requisitos, a atividade de especificação visa compreender e identificar os requisitos do sistema. Um requisito é uma característica que o sistema deve contemplar, uma funcionalidade que o sistema deve possuir para satisfazer os contratos e especificações feitas pelo cliente ou usuário.

Nesta disciplina detalhamos cada requisito para conhecer a totalidade de funcionalidades e restrições que um software deve possuir. Esta é uma adse crítica, e necessita de que o analista de sistemas se aproprie de cada detalhe, cada processo que deverá ser implementado para que a elaboração do software seja bem-sucedida. É importante destacar que um erro nessa fase é crítico, pois impactam em toda a cadeia de construção do software.

Artefatos são produtos gerados a partir de um trabalho realizado em uma fase da atividade de processo de software. Nesta primeira atividade, um artefato importante é a geração de um documento chamado de documento de requisitos, que reúne todos os requisitos identificados para satisfazer a solicitação do cliente.

Existem quatro atividades no processo de engenharia de requisitos:

- Estudo da viabilidade;
- Elicitação e análise de requisitos;
- Especificação de requisitos;
- Validação de requisitos.

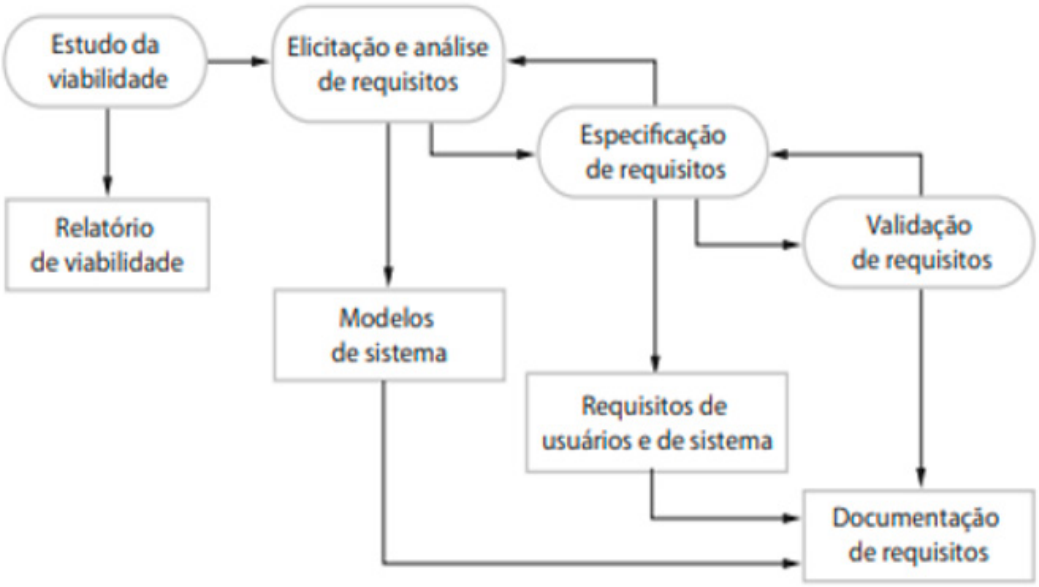
No estudo da viabilidade é realizado um levantamento sobre questões como restrições orçamentais e entendimentos sobre se a construção do sistema deve prosseguir, ou não, considerando o interesse da parte do cliente e da parte da empresa ou equipe de desenvolvimento.

Na elicitação e análise, todos os requisitos são identificados, resultando no entendimento do que o sistema deverá ser capaz de executar. É possível usar protótipos do sistema para ajudar no entendimento da totalidade dos requisitos identificados.

Na especificação de requisitos são detalhados os requisitos identificados, produzindo uma lista com a especificação de cada requisito, tanto no contexto de sua funcionalidade quanto nas restrições de hardware e software pertinentes e necessários para a sua execução.

Por fim, na validação, os requisitos são verificados quanto a sua consistência e completude, nesse momento deve-se atualizar o documento de requisitos para que os erros encontrados sejam resolvidos e integrados na versão final do documento.

FIGURA 1 | **Os Requisitos da Engenharia de Processos**



Fonte: SOMMERVILLE, 2019

Ainda sobre os artefatos gerados nessa fase, podem ser criados Diagramas Entidade Relacionamento (DER), para especificação do banco de dados que irá compor o sistema, e também o Diagrama de Caso de Uso da UML (linguagem unificada de modelagem), que possibilitará exibir o relacionamento do usuário com cada funcionalidade do sistema. Outra ferramenta da UML que pode ser útil é o diagrama de atividades que permite o detalhamento do passo a passo na execução de cada requisito.

3. DESENVOLVIMENTO

A atividade de desenvolvimento envolve o projeto e a implementação do software. Esta fase requer a tradução de tudo o que foi feito na fase anterior em sistemas executáveis.

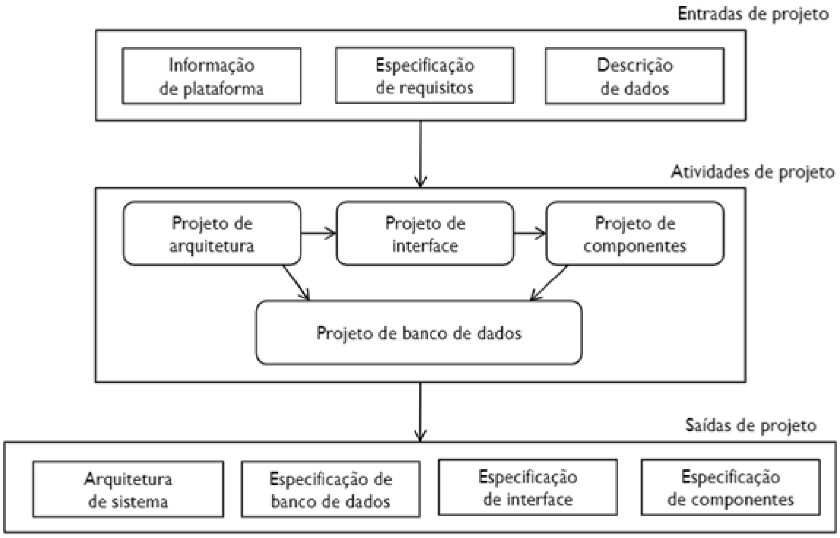
Em termos de projeto, deve-se descrever a estrutura geral do software, incluindo arquitetura (linguagem de programação a ser utilizada, por exemplo), interfaces, algoritmos e banco de dados necessários para implementar o sistema.

Nesse ponto é importante observar os padrões de implementação que são normalmente definidos pela empresa. Alguns exemplos dessas definições são modelos de identação, padronização na nomenclatura de variáveis e objetos de banco de dados, definição de comentários no código, organização na criação das funções, procedimentos, *triggers* e tudo mais.

Na implementação, independente da linguagem adotada, é importante considerar que todo sistema precisa ter continuidade e para isso é preciso que sejam considerados aspectos de manutenibilidade e compreensão.

Nesta fase, os artefatos gerados normalmente são o Diagrama de Estrutura Modular(DEM) para visualização dos módulos, submódulos e empacotamento, apresentando visualmente a definição dos programas a serem desenvolvidos. No caso do banco de dados, a modelagem dos dados é convertida para instruções de banco de dados, normalmente em linguagem SQL. Pode-se também refinar os diagramas entidade relacionamento (DER) e/ou os diagramas de classes e de máquina de estados da UML.

FIGURA 2 | **Um Modelo Geral do Processo do Projeto**



Fonte: SOMMERVILLE, 2019

A Figura 2 mostra os artefatos produzidos nesta atividade, além de especificar as saídas após a finalização desta fase de construção do software.

É importante conhecer quais são as tarefas de cada parte do projeto:

- No projeto de arquitetura definimos a arquitetura geral do sistema, como os seus módulos ou subsistemas;
- No projeto de interface estabelecemos aspectos de como os componentes do sistema se comunicarão, ou seja, um componente se comunica com o outro sem precisar saber como ele foi implementado;
- No projeto de componentes deve-se projetar o funcionamento de cada parte do sistema;
- No projeto de banco de dados é especificada a camada de dados do sistema e as suas estruturas.

Ao fim desta atividade temos um produto que ainda carece de validação e testes para que seja colocado em produção.

4. VALIDAÇÃO

Também chamada de atividade de verificação e validação, esta fase tem o objetivo de apurar se o sistema construído atende às especificações do cliente, avaliando os resultados da execução de cada operação previamente definida.

Nesta atividade são aplicados testes de software, que apresentam cenários capazes de simular a operação do sistema, avaliando se o mesmo atinge os resultados esperados.

Aqui são verificados e validados os procedimentos desenvolvidos para garantir a qualidade do software. É preciso verificar a presença de erros e se os requisitos estão sendo atendidos de acordo com a necessidade dos usuários.

Além dos testes, existem outros tipos de verificação do software, como inspeções e revisões, que podem ocorrer não apenas nesta atividade, mas também ao longo de todas as atividades anteriores.

Na validação passamos por testes de componente, de sistema e de aceitação. Estes testes visam garantir o comportamento do sistema em três estágios, para que o sistema não seja testado como uma unidade única, mas como um conjunto integrado.

Aqui o desenvolvimento é finalizado e disponibilizado para uso no usuário. Os procedimentos são aprovados pelos *stakeholders*, e na maioria das vezes um treinamento é realizado com os usuários do negócio, pois o usuário precisa saber manusear o sistema computacional e conhecer como o computador irá atender suas necessidades.

Ao final desta atividade, o sistema está pronto para o chamado teste beta, que significa que o sistema está pronto para ser utilizado comercialmente. Neste teste alguns usuários fazem uso do produto no dia a dia, revelando aos desenvolvedores possíveis erros ou problemas encontrados. Após os devidos ajustes, o sistema enfim está liberado para ser comercializado.

5. EVOLUÇÃO

Esta última atividade tem o objetivo de garantir que o sistema possa ser flexível ao ponto de se ajustar às mudanças de requisitos, legislação e demais alterações que determinado domínio pode exigir.

A evolução também é conhecida como manutenção de software, e ela exige grande esforço da equipe desenvolvedora, mas esse esforço pode ser atenuado se determinados padrões de qualidade, como uma boa documentação, por exemplo, tenham sido seguidas ao longo da gestão do produto.

Outro ponto crítico aqui é a aferição da qualidade, pois serão verificadas se as características necessárias para satisfizer a necessidade dos usuários do negócio forma atendidas. Essas características estão relacionadas a todos os aspectos do sistema como complexidade, recursos envolvidos, metodologias e técnicas aplicadas, além de procedimentos associados a detecção e remoção de erros e, a relação custo-benefício. Essas medições podem ser quantitativas e qualitativas.

CONSIDERAÇÕES FINAIS

Nesta unidade abordamos algumas das atividades essenciais no processo de software. Vimos que processos de software são as fases indispensáveis para produzir e manter um software, onde cada atividade possui uma organização, técnicas e resultam em métodos, ferramentas, artefatos e regras que possibilitam sistematizar e organizar o desenvolvimento e manutenção de produtos de software.

As atividades que listamos nesta unidade foram especificação, desenvolvimento, validação e evolução, e cada uma delas exerce um papel importantíssimo na construção do software.

Na especificação cuidamos do entendimento do problema, bem como de toda a gestão de seus requisitos. Após esse levantamento inicial, damos sequência ao desenvolvimento, que é quando estabelecemos o projeto e iniciamos o processo de desenvolvimento do software.

Ao final deste desenvolvimento tratamos dos testes e validação do produto, para que enfim ele seja colocado em produção. Na evolução cuidamos da existência do software, cuidando de sua manutenção, no sentido de reagir às mudanças de requisitos, legislação ou no ajuste de suas funções.

MATERIAIS COMPLEMENTARES

Artigo: *Processos de Software: o que você precisa saber*. 2018. Disponível em: https://medium.com/@danielemsilva/processos-de-software-o-que-voc%C3%AA-precisa-saber-9b-89c359d3e7. Acesso em: 01 nov. 2022.

REFERÊNCIAS

KENNETH C. LAUDON; JANE P. LAUDON. *Sistemas de informação gerenciais*. 11ª edição. Editora Pearson, 2014.

PRESSMAN, R. G. *Engenharia de Software.* 9ª ed. Rio de Janeiro: McGraw-Hill, 2021.

SOMMERVILLE, I. *Engenharia de Software.* 10ª ed. São Paulo: Pearson Addison. Wesley, 2019.

AULA 3 – CICLO DE VIDA DO SOFTWARE

OBJETIVO DA AULA

Conhecer as definições do ciclo de vida do desenvolvimento do software, assim como discutir e comparar os benefícios e as desvantagens dos processos incrementais, iterativos, evolutivos e ágeis.

APRESENTAÇÃO

O processo do software, também conhecido como ciclo de vida do software, visa descrever os processos e atividades envolvidas na operação e na evolução do sistema, ao longo de toda a sua existência.

Para facilitar e conduzir esse processo de gestão do software existe modelos que definem como o software será elaborado, guiando o desenvolvedor ao logo de cada fase.

A definição de um modelo vai definir, por exemplo, a ordem de sequenciamento das atividades, dentre muitos quesitos ao longo do desenvolvimento do software.

Esta aula apresentará um pouco sobre esse processo, abordando os principais ciclos de vida existentes e praticados no mercado.

1. CICLO DE VIDA

O ciclo de vida de um software constitui-se na divisão do desenvolvimento em etapas. As etapas por sua são desmembradas em atividades ou disciplinas que possuem importantes passos para a elaboração do software.

Os ciclos de vida são aplicados para conduzir o desenvolvimento e vão produzir artefatos na finalização de cada etapa. Os ciclos de vida são definidos de acordo com vários fatores como familiaridade da organização, experiência em projetos anteriores, filosofia e, principalmente, visto a natureza do projeto, ou seja, sua criticidade de tempo, complexidade e tamanho.

Existem diversos modelos de ciclo de vida utilizados atualmente no mercado, e outros que se tornaram obsoletos ao longo do tempo. Atualmente costumamos dividir os modelos em incrementais e iterativos, mas também consideramos a filosofia de desenvolvimento ágil, que é amplamente adotada nas organizações.

O ciclo de vida de desenvolvimento de um sistema ou software define as etapas que deverão ser seguidas durante a construção do produto, e quais são artefatos que devem ser gerados em cada uma das etapas, desde o conhecimento do problema até a solução instalada num computador.

Consideramos como artefato de software todo produto originário a partir do desenvolvimento de um software. Alguns exemplos são documentações, códigos, planos de testes, diagramas, como os de casos de uso, diagramas de classes, diagramas entidade-relacionamento, e outros modelos da UML, lista de requisitos, protótipos, manuais, documentos de visão, além de uma infinidade de outros objetos oriundos de um esforço pessoal ou coletivo.

Apesar dos modelos de ciclos de vida possuírem a mesma proposta, ou seja, a elaboração de um sistema, o processo de execução difere um do outro. É importante lembrar, no entanto, que a abordagem a ser utilizada é a mesma: conhecer o problema, representar conceitualmente a solução, e desenvolver o produto, procedendo testes e validações para que ele chegue à produção e possa ser utilizado pelos usuários.

2. MODELO EM CASCATA

O modelo em cascata, ou *waterfall*, é um dos modelos mais antigos, ele foi criado em 1966, porém teve a sua formalização apenas em 1970 por Wiston Royce.

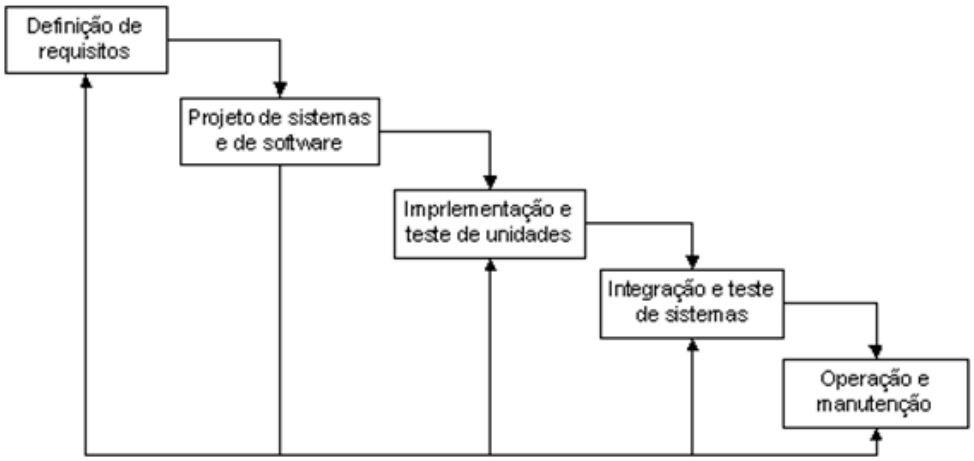
Neste modelo as etapas são executadas de maneira sequencial, na prática, isso quer dizer que uma fase só inicia após o término da fase anterior. Essa característica, no entanto, é uma das grandes desvantagens deste modelo, já que o torna inflexível quanto às mudanças de requisitos, algo comum a todo projeto atualmente se considerarmos a velocidade de mudança nos processos e nas regras de negócios.

No modelo cascata são produzidos muitos artefatos antes do processo de desenvolvimento do produto, isso faz com que o cliente e usuários tenham raros contatos com o sistema que está sendo produzido, aumentando as expectativas e incertezas.

Essa característica classifica este modelo como burocrático, pois mesmo que o projeto já tenha iniciado há um bom tempo, muitas vezes o que temos pronto são apenas documentação e diagramas.

Este modelo não oportuniza uma validação mais rápida por parte do usuário, que pode detectar erros de entendimento e de programação antecipadamente, o que ajudaria a reduzir o impacto, caso tais problemas sejam identificados somente no final do projeto.

FIGURA 1 | **O Modelo Cascata**



Fonte: SOMMERVILLE (2019, p.20)

Na Figura 1 observamos o modelo cascata com as suas respectivas atividades, do levantamento de requisitos até o processo de operação e manutenção. É possível observar os

feedbacks ao fim de cada fase, embora na maioria das organizações este modelo é tratado de forma estritamente linear.

Entre outros problemas cabe ressaltar a dificuldade ao necessariamente termos que levantar todos os requisitos e detalhes do projeto já na fase inicial, pois muitas vezes alguns pontos importantes podem ficar de fora desse levantamento, muitas vezes por conta do próprio usuário no momento de levantar as suas próprias necessidades.

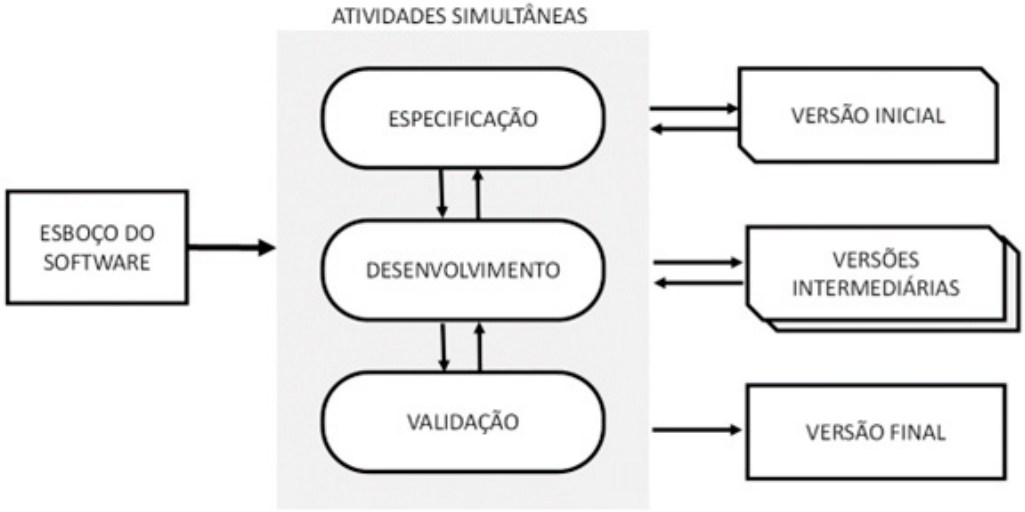
Em um ponto de vista mais otimista, podemos dizer que o cascata tem ao seu favor o fato de ele ser muito simples de ser elaborado e executado, pois a sua estrutura é extremamente simples.

Este modelo deve ser usado apenas em casos em que todos os requisitos são conhecidos e quando houver pouca chance de mudança ao longo do desenvolvimento do produto. Em termos práticos, este modelo favorece pequenos projetos, tornando fácil também as tarefas de testes e validação.

3. ENTREGA INCREMENTAL

Os modelos incrementais são grupos de processos de software que trabalham a partir da geração de incrementos a cada ciclo de desenvolvimento. O trabalho de desenvolvimento prevê a elaboração de uma implementação inicial, que é submetida aos comentários e críticas dos usuários. Após o feedback, são realizadas outras versões, até que uma versão final do sistema seja aprovada.

FIGURA 2 | **Desenvolvimento Incremental**

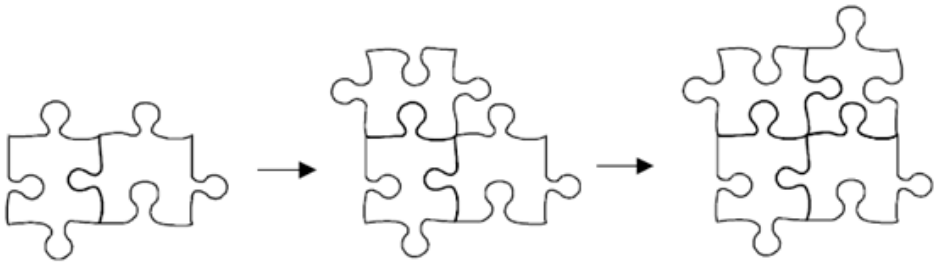


Fonte: SOMMERVILLE (2011, p. 22)

Na Figura 2 observamos o modus operandi do desenvolvimento incremental, em que há um esboço inicial do software e vemos as fases de especificação, desenvolvimento e validação intercaladas, e não separadas, como no cascata. Por fim, observamos as várias versões do software sendo construídas até chegarmos à versão final.

Esse tipo de desenvolvimento ganhou muita popularidade com a adoção dos métodos ágeis, já que ele é a base desse tipo de abordagem. Ele reflete a forma como normalmente resolvemos problemas me nosso dia a dia. No geral, este tipo de desenvolvimento é mais barato e facilita eventuais mudanças de requisitos ao longo do desenvolvimento do projeto.

FIGURA 3 | **Entrega Incremental**



Fonte: IFSC. Disponível em: wiki.sj.ifsc.edu.br/wiki/index.php/Ciclo_de_Vida_Iterativo_e_Incremental. Acesso em: 02 nov. 2022.

A Figura 3 mostra o produto sendo construído de maneira incremental, até chegar ao produto completo.

A ideia é que seja incrementadas novas funcionalidades a cada nova versão elaborada pela equipe de desenvolvimento, isso permite ao cliente priorizar aquelas atividades mais críticas, por exemplo, além de dar ao cliente uma visão exata de como está o processo de desenvolvimento, pelo fato de que ele precisa estar acompanhando de perto a equipe de desenvolvimento.

Apesar de apresentar visíveis melhorias se comparado ao modelo cascata, o modelo de entrega incremental apresenta alguns pontos de atenção.

Há alguma dificuldade no gerenciamento do projeto por parte do gerente, já que a documentação não acompanha de forma viável a elaboração das versões.

Em outro prisma, conforme os incrementos vão sendo gerados ao longo da vida do software, a sua estrutura tende a se degradar, ou seja, sistemas de vida muito longa não são favorecidos com este modelo de desenvolvimento.

4. MODELOS ITERATIVOS

Definimos iteração como o processo em que um conjunto de instruções ou estruturas são repetidas em uma sequência por um número específico de vezes ou até que uma condição predefinida seja alcançada.

Assim, os modelos ditos como iterativos possuem uma repetição de atividades de desenvolvimento até que o sistema esteja pronto. Um exemplo de desenvolvimento iterativo é a Prototipação, modelo no qual é gerado um novo protótipo a cada iteração realizada.

FIGURA 4 | **Ciclo de Vida Iterativo**

Fonte: IFSC. Disponível em: wiki.sj.ifsc.edu.br/wiki/index.php/Ciclo_de_Vida_Iterativo_e_Incremental. Acesso em: 02 nov. 2022.

A Figura 4 mostra um produto sendo construído iterativamente, tendo uma nova versão a cada ciclo.

A ideia central do modelo iterativo é a de refinar o sistema a cada iteração, ou seja, melhorá-lo pouco a pouco. Em cada iteração os desenvolvedores identificam junto ao cliente os requisitos relevantes, criando um projeto que utilize uma arquitetura que será escolhida como guia, o projeto é implementado então a partir destes componentes.

Assim que uma iteração atinge os seus objetivos, o desenvolvimento passa para a próxima iteração, caso contrário, a equipe volta ao ciclo para correções, realizando os devidos ajustes e submetendo o produto a uma nova avaliação. Dessa forma, podemos dizer que não é o sistema como um todo que é alterado, mas, sim, a sua composição, o seu detalhe, em cada iteração.

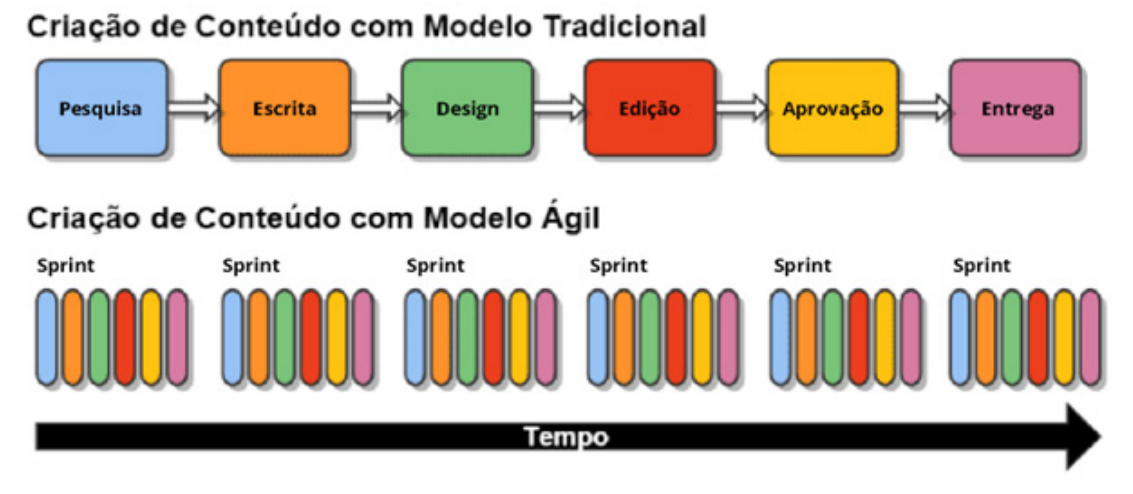
O modelo iterativo também é conhecido como evolucionário, e tem como exemplos os modelos Prototipação e Espiral. Ele é atualmente é empregado juntamente como complemento de outro modelo de ciclo de vida como o incremental.

5. DESENVOLVIMENTO ÁGIL

O desenvolvimento ágil é uma filosofia de desenvolvimento de software que utiliza normalmente os modelos iterativos e incrementais, mas especificado a partir de uma abordagem que visa uma postura de maior colaboração com o cliente e maior integração entre a equipe desenvolvedora.

Esta abordagem oferece algumas vantagens importantes ao processo de resolução de problemas de software. Uma delas é realizar o desenvolvimento de uma maneira que permita que os clientes possam ver os resultados mais rapidamente, e outra importante vantagem é a entrega do produto funcional de maneira mais rápida, tendo incrementos agregados, de forma flexível, a cada novo ciclo.

FIGURA 5 | **Comparação do Modelo Tradicional X Modelo Ágil**



Fonte: Blog Cdlfor. Disponível em: https://blog.cdlfor.com.br/dicas/metodologia-agil/. Acesso em: 02 nov. 2022.

Na Figura 5 distinguimos a diferença entre as abordagens, onde no modelo tradicional a construção se dá etapa por etapa, enquanto no modelo ágil cada nova iteração (Sprint) produz uma série de novos incrementos ao sistema.

Os métodos ágeis são focados em entregas incrementais contínuas, com redução considerável na documentação gerada ao longo do desenvolvimento do software. Os exemplos de modelos mais comuns nessa abordagem são o XP (*Extreme Programming*) e o Scrum.

CONSIDERAÇÕES FINAIS

O ciclo de vida estabelece um guia para a elaboração de software seguindo as atividades propostas por cada modelo. É importante para o analista conhecer o problema que deseja resolver para escolher a abordagem de construção ideal.

O modela mais antigo é o chamado cascata e, embora a sua utilização e implantação seja muito simples, ele possui diversos problemas que o tornam ultrapassado para a maioria dos casos.

Os modelos de entrega incremental e iterativos hoje são utilizados como base para outras abordagens de desenvolvimento como os métodos ágeis de desenvolvimento, trazendo alguns benefícios interessantes no ponto de vista de agilidade e otimização em relação aos artefatos produzidos ao longo do projeto.

MATERIAIS COMPLEMENTARES

Vídeo: *Ciclo de Vida do Sofware*. Disponível em: https://www.youtube.com/watch?v=6_fV-cpC0cxE. Acesso em: 02 nov. 2022.

REFERÊNCIAS

PRESSMAN, R. G. *Engenharia de Software*. 9ª ed. Rio de Janeiro: McGraw-Hill, 2021.

SOMMERVILLE, I. *Engenharia de Software*. 10ª ed. São Paulo: Pearson Addison. Wesley, 2019.

AULA 4 – MODELOS ITERATIVOS

OBJETIVO DA AULA

Compreender o modelo de desenvolvimento iterativo, apresentando as características, vantagens e desvantagens de modelos de desenvolvimento como Prototipação e Espiral.

APRESENTAÇÃO

Esta aula apresenta os modelos iterativos que tratam o desenvolvimento de software a partir de iterações, isto é, as atividades são repetidas para cada ciclo de desenvolvimento executado.

Dentre os modelos mais conhecidos desta fase estão o modelo chamado de prototipação e o modelo espiral.

No caso da prototipação, protótipos são construídos para a validação imediata junto ao usuário. Esta validação pode ajudar na definição da interface do software, na elicitação e validação de requisitos e também na construção de funcionalidades do software.

O modelo espiral define o desenvolvimento em quatro partes, definição de objetivos, avaliação e redução de riscos, desenvolvimento e validação, e planejamento. É um modelo que possui como destaque importante a validação e mitigação dos riscos do projeto.

1. OS MODELOS ITERATIVOS

Os modelos iterativos abordam a construção de software a partir da repetição das atividades do processo de software, conhecidas como iterações, até que o produto esteja disponível para ser disponibilizado para o uso do cliente.

É comum encontrarmos alguns modelos de desenvolvimento que juntam características dos modelos iterativos com os incrementais. Como o desenvolvimento e o gerenciamento de software é algo complexo, a ideia é dividir o trabalho em partes menores a partir de iterações, tendo como resultado de cada incremento.

2. PROTOTIPAÇÃO

O modelo conhecido como prototipação objetiva a construção de protótipos a cada ciclo iterativo, até que a repetição desse processo resulte na versão final do software.

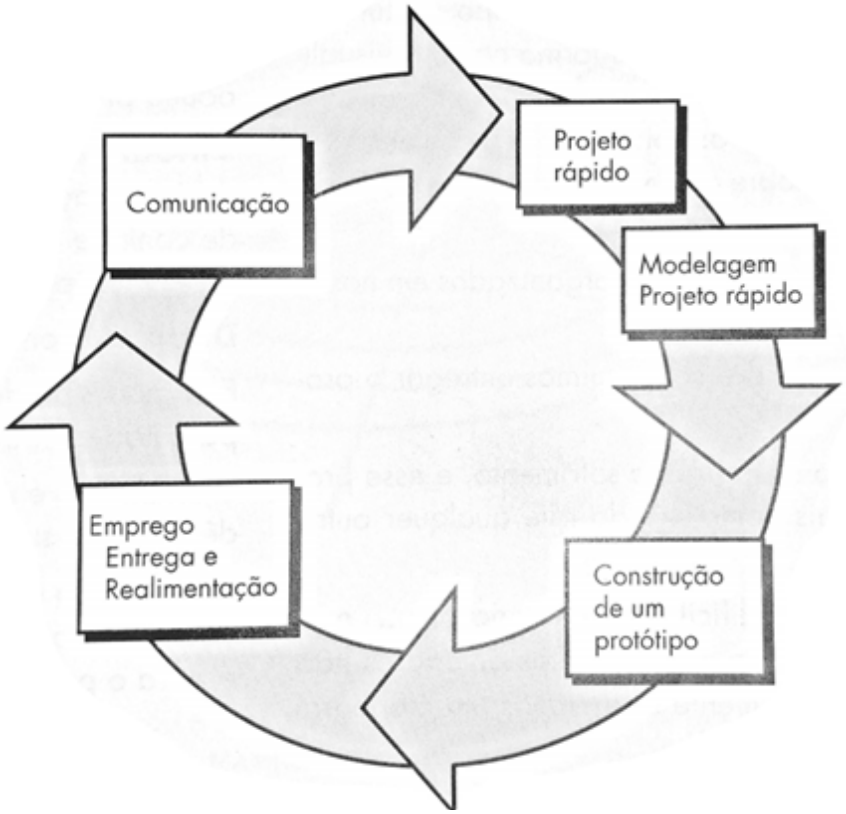
A ideia é que esse processo agilize o desenvolvimento do produto, mas o foco está nas funcionalidades em si, e não no acabamento do produto, o que pode gerar versões incompletas e bastante retrabalho.

O modelo apresenta alguns problemas visíveis. Em primeiro lugar, o usuário tem a impressão de que aquele protótipo está sempre próximo à versão final do produto, e isso pode gerar alguns descontentamentos, pois a produção do protótipo normalmente não considera questões relacionadas à usabilidade e interface.

Em outro prisma, o usuário pode querer já usar o protótipo para resolver problemas críticos, não entendendo que ainda há muito trabalho a ser realizado até que o sistema esteja completamente disponível. De certa forma, deve-se evitar o uso desta abordagem em sistemas com maior criticidade.

Em outro ponto, a prototipação apresenta alguns benefícios observáveis. O primeiro diz respeito à constante interação com o usuário, que deve participar ativamente do processo, para aprovar os protótipos. O segundo diz respeito ao processo de validação imediata que este modelo requer, o que pode acelerar o processo de desenvolvimento como um todo.

FIGURA 1 | **O Paradigma da Prototipação**



Fonte: PRESSMAN (2019, p. 27).

Na figura 1 verificamos o ciclo iterativo da prototipação, em que o produto vai sendo construído a partir da execução cada ciclo completo.

Apesar de notavelmente apresentar alguns problemas, a prototipagem de sistemas possui algumas boas aplicações. Em casos em que o domínio é completamente desconhecido para o desenvolvedor, a sua aplicação ajuda a quebrar essas barreiras de entendimento do domínio.

Em engenharia de requisitos os protótipos também são bastante utilizados para ajudar na elicitação e validação dos requisitos do sistema que será desenvolvido. E neste ponto estamos falando do processo inicial de análise e especificação de sistemas.

Em um momento posterior, este tipo de abordagem de construção de software também pode ser utilizado para a definição de interfaces e usabilidade do sistema, essa aplicação torna muito mais fácil para o cliente aprovar ou solicitar ajustes na interface do software.

FIGURA 2 | **Processo de Desenvolvimento do Protótipo**

Fonte: SOMMERVILLE (2011, p. 30)

Na Figura 2 observamos o fluxo do ciclo de vida do protótipo executado até sejam finalizadas todas as implementações levantadas na atividade de especificação de software.

Na fase inicial devem ser estabelecidos os objetivos do protótipo, que podem ser para validar a interface, validar requisitos ou para a demonstração da viabilidade do projeto de software.

A seguir deve-se estabelecer que requisitos serão incluídos o protótipo, no que chamamos de definição geral do protótipo. O desenvolvimento do protótipo vem logo a seguir, gerando então um projeto executável.

Por fim, há a necessidade de avaliação do protótipo. Nesta etapa um relatório deve ser construído visando apontar as falhas descobertas pelos usuários e os ajustes que devem ser feitos, como tempo de resposta e usabilidade, caso estas funcionalidades tenham sido definidas no plano de prototipação.

É importante destacar que os protótipos são ótimas soluções quando empregadas com outras técnicas. Há uma série de softwares disponíveis na web para a elaboração de protótipos funcionais e de interface.

LINK

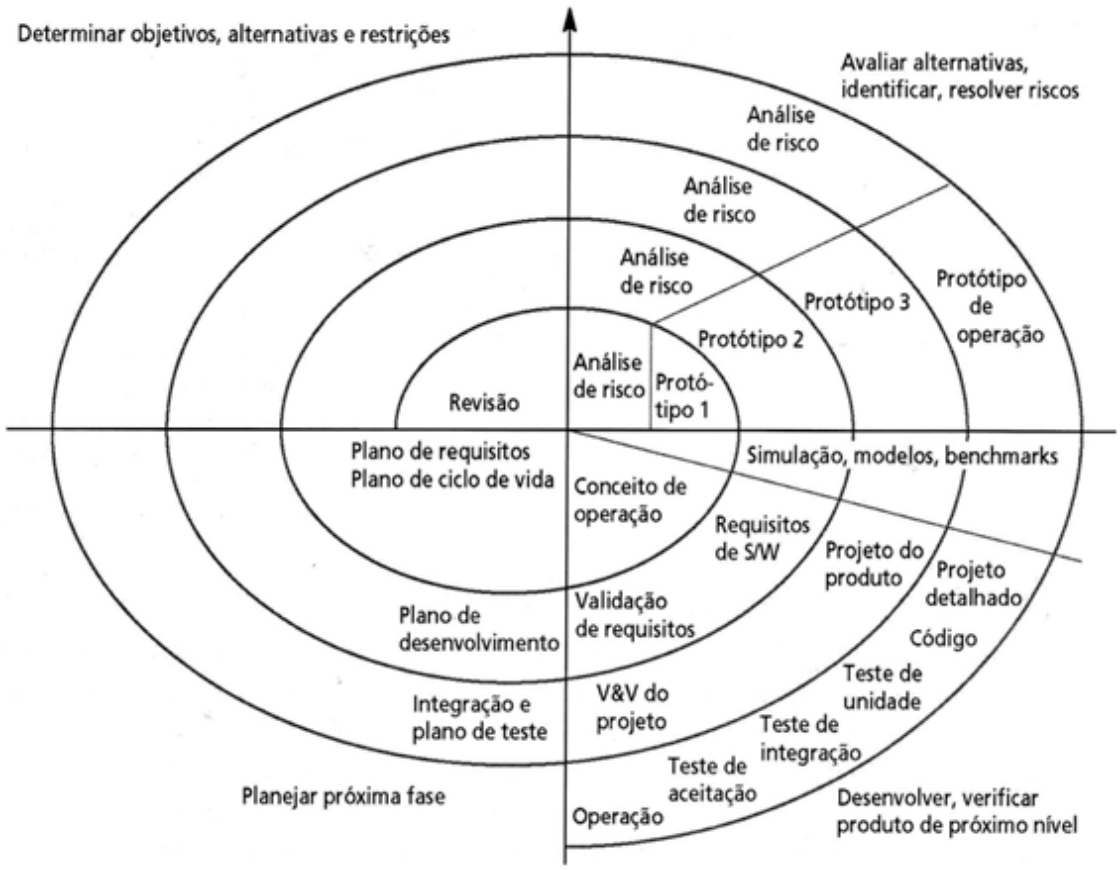
Prototipagem: o que é e 10 ferramentas para fazer o seu protótipo. Disponível em: https://49educacao.com.br/mvp/prototipagem/. Acesso em: 02 nov. 2022.

3. ESPIRAL

Alguns processos de software acabam agregando a união de paradigmas para a geração do seu próprio modelo. Este é o caso do modelo Espiral, concebido como uma tentativa da união do ciclo tradicional cascata com o modelo de protótipos, propondo o desenvolvimento em partes.

Este modelo possui quatro atividades: definição de objetivos, avaliação e redução de riscos, desenvolvimento e validação, e planejamento. É importante o entendimento de que estas fases possam ter nomes diferentes em algumas outras literaturas. A que abordaremos aqui se chama Modelo Espiral de Boehm.

FIGURA 3 | **Modelo Espiral de Boehm**



Fonte: SOMMERVILLE (2011, p. 33)

A Figura 3 especifica o modelo espiral considerando todas as atividades realizadas em cada uma das suas quatro fases.

Na fase de definição de objetivos, são definidos os objetivos, alternativas e são identificadas as restrições ao processo e produto. É necessária a construção de um plano de gerenciamento detalhado, assim como uma lista com os riscos do projeto que foram identificados.

Na fase de validação e redução de riscos, deve-se montar uma estratégica para mitigar cada risco encontrado na fase anterior. Em desenvolvimento e validação é definido um modelo de desenvolvimento do sistema.

A fase de Planejamento encerra o ciclo, e é o momento em que o projeto é revisado e o próximo giro no espiral é definido.

Apesar de ser um modelo realista, principalmente por conta da observação quanto aos riscos do projeto, o convencimento da sua adoção é difícil, pois o gerenciamento deste tipo de projeto não é trivial, justamente por conta das sucessivas evoluções. Deve-se considerar preocupar-se com uma análise de riscos muito bem elaborada.

CONSIDERAÇÕES FINAIS

Nesta aula vimos dois importantes modelos de elaboração de software chamados de Prototipação e Espiral. Esses modelos são baseados em iterações, ou seja, suas atividades são repetidas várias vezes até que o projeto seja finalizado.

O ciclo de vida prototipação trabalha com a implementação de protótipos que podem ser utilizados para diversos fins de validação junto ao cliente, como interfaces, requisitos e até na execução de funcionalidades.

Já o ciclo de vida em espiral requer dinamismo na implementação e experiência na definição, pois aborda o seu desenvolvimento com bases em iterações e incrementos. É um modelo importante por se preocupar em levantar e tratar os possíveis riscos de um projeto.

MATERIAIS COMPLEMENTARES

Vídeo: *Prototipação, saiba porque esta etapa é fundamental!* Disponível em: https://www.youtube.com/watch?v=8YbAHNCv9-w. Acesso em: 02 nov. 2022.

REFERÊNCIAS

PRESSMAN, R. G. *Engenharia de Software.* 9ª ed. Rio de Janeiro: McGraw-Hill, 2021.

SOMMERVILLE, I. *Engenharia de Software.* 10ª ed. São Paulo: Pearson Addison. Wesley, 2019.

AULA 5 – MODELO DE PROCESSO UNIFICADO

OBJETIVO DA AULA

Conhecer o modelo de processo unificado e as suas principais características.

APRESENTAÇÃO

Esta aula tem o objetivo de apresentar o modelo de processo unificado, abordando as suas principais características, vantagens, desvantagens, e quais são os melhores cenários de aplicação.

Além disso, abordamos as suas quatro fases, concepção, elaboração, construção e transição, elencando quais são os principais artefatos produzidos em cada uma das fases.

Também apresentaremos o modelo *Rational Unified Process*, conhecido como RUP, especificando os seus pontos fortes e fracos, bem como tratando das suas mais recomendáveis aplicações.

1. PROCESSO UNIFICADO

O modelo unificado (*Unified Process* – UP) é um exemplo de um modelo híbrido, que agrega características de outros modelos existentes, principalmente as características iterativas e incrementais.

Esse modelo surgiu para atender o desenvolvimento de softwares orientados a objetos, dessa forma ele permite que o desenvolvimento se pareça com a construção de vários microssistemas, formando um sistema de grande porte.

O processo unificado se baseia no paradigma evolucionário para tratar do desenvolvimento de softwares, considerando sucessivos refinamentos, com cada iteração incrementando um pouco mais o produto final. Cada iteração também será analisada e criticada pelo usuário, o que ajuda em diversos aspectos a melhoria do produto final.

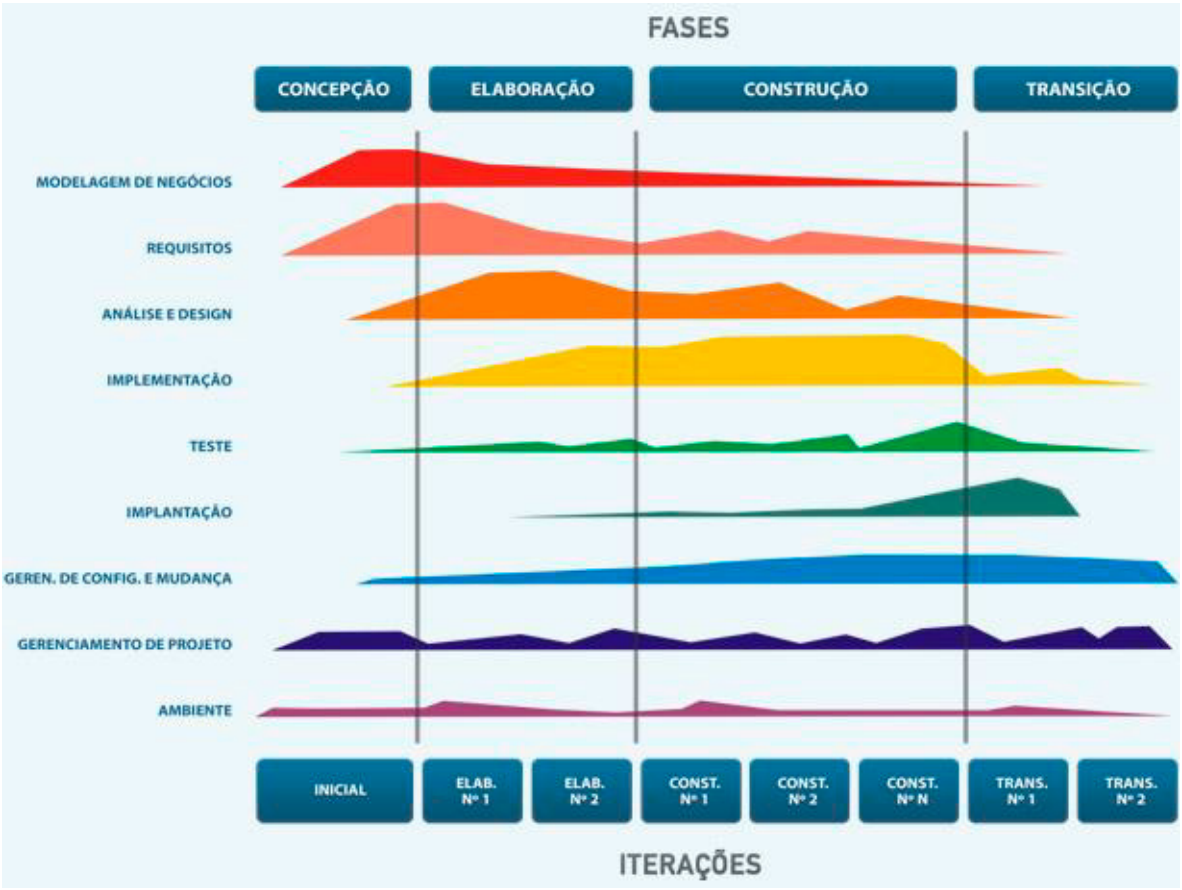
As iterações são vistas como pequenos projetos de duração determinada, e cada uma possui todo o conjunto de atividades especificadas no processo unificado, ou seja, Concepção, Elaboração, Construção e Transição. Diferentemente de outros paradigmas, no processo unificado você não tem um sistema pronto ou protótipo executável em cada iteração, apenas temos mais um conjunto de requisitos incrementado.

Algumas características são marcantes neste modelo de processo:

- Iterativo e Incremental – as fases de elaboração, construção e transição são normalmente divididas em uma série de iterações e cada iteração gera um incremento;
- Orientados por Casos de Uso – os casos de uso são utilizados como ferramenta de captura de requisitos e também para o refinamento das iterações;

- Focado na arquitetura – a arquitetura é criada na fase de elaboração e, após validada, torna-se base para todo o sistema;
- Componentização – o modelo auxilia no processo de construção de softwares a partir de componentes interligados a partir de interfaces bem definidas;
- Avaliação de risco – um dos focos deste modelo é a mitigação de riscos, pois a equipe desde o início do projeto trata de identificar e garantir que os riscos estejam documentados e gerenciados.

FIGURA 1 | **O Processo Unificado**



Fonte: Sist Gestão Blog. Disponível em: sistgestaoblog.wordpress.com/2015/10/24/rational-unified-process-rup/. Acesso em: 02 nov. 2022.

A Figura 1 mostra o modelo unificado, suas fases e subfases tendo como base as iterações que ocorrem ao longo do processo.

2. FASES DO PROCESSO UNIFICADO

Como já vimos, o processo unificado é dividido em quatro fases que possuem suas subfases e podem ser iteradas diversas vezes. No entanto, é importante que estas iterações possuam prazos determinados, ou então o projeto pode acabar se perdendo e atrasando. Costuma-se adotar que iterações pequenas tenham entre duas e seis semanas, mas cada projeto precisa obedecer a sua própria natureza.

Vamos destacar agora quais são os objetivos principais de cada fase:

- Concepção: nesta fase são levantados os requisitos e definido o escopo do projeto. Naturalmente não é um objetivo esgotar detalhadamente cada requisito, mas, sim, obter uma visão inicial de como o sistema resolverá o problema. Também podem ser definidos preços e prazos após a avaliação inicial dos riscos;
- Elaboração: a seguir detalhamos todos os requisitos, principalmente aqueles que forma o *core* do sistema. Há também o foco no tratamento dos principais riscos identificados e especificação das características da arquitetura do software;
- Construção: nesta fase ocorre a implementação iterativa dos elementos do sistema e a preparação para a implantação;
- Transição: por fimsão realizados os testes finais e a implantação do sistema. Aqui também são incluídas as fases de testes, aceitação e treinamento dos usuários.

O modelo unificado tem como característica a produção de vários artefatos ao longo do seu ciclo de vida. De acordo com Pressman (2006), os principais artefatos gerados em cada fase são:

a) Concepção:

- Documento de Visão;
- Modelo inicial de Casos de Uso;
- Glossário inicial do projeto;
- Caso de negócio inicial;
- Avaliação inicial de risco;
- Plano de projeto: fases e iterações;
- Modelo de negócio;
- Um ou mais protótipos.

b) Elaboração:

- Modelo de Casos de Uso;
- Requisitos suplementares, incluindo não funcionais;
- Modelo de análise;
- Descrição da arquitetura do software;
- Protótipo arquitetural executável;
- Modelo de projeto preliminar;

- Lista de riscos revisada;
- Plano de projeto: planos de iteração, fluxos de trabalho adaptados, marcos e produtos técnicos de trabalho;
- Manual preliminar do usuário.

c) Construção:

- Modelo de projeto;
- Componentes de software;
- Incremento integrado de software;
- Plano e procedimento de teste;
- Caso de teste;
- Documentação de apoio: manuais do usuário, manuais de instalação, descrição do incremento atual.

d) Transição:

- Incremento de software entregue;
- Relatório de teste beta;
- Realimentação geral do usuário (modificar ou adaptar a compreensão dos requisitos do projeto).

3. RUP

A partir da proposta do processo unificado surgiram algumas ferramentas para apoiar o desenvolvimento do software. O RUP é a mais notória delas, embora também existam outras como o Open-UP.

O *Rational Unified Process* (RUP) é um processo proprietário, pertencente à *Rational* e que, em 2003, foi adquirido pela IBM.

Como vantagens podemos dizer que ele é um dos modelos mais robustos e bem estruturados. O fato de ele ter como base a linguagem de modelagem UML ajuda na elaboração dos artefatos e diagramas.

Tem o foco na resolução dos riscos já no início do projeto, minimizando a chance de fracassos e retrabalhos ao durante a gestão do projeto. Em termos de gestão, o processo fica mais visível aos envolvidos, graças a sua estrutura bem definida.

Mas tem também o outro lado, assim como todos os modelos. Uma crítica ao RUP é justamente a sua robustez, que torna o processo trabalhoso, o que praticamente inviabiliza o seu uso para projetos pequenos.

Este modelo de projeto exige um gerente experiente e uma equipe que já possua algum conhecimento no funcionamento do modelo.

4. OPENUP

O Processo Unificado Aberto é considerada uma versão ágil do processo unificado, é considerado mais simples, pois elimina algumas práticas operacionais e também alguns artefatos, principalmente os de documentação.

LINK

Saiba mais sobre o OpenUp. Disponível em: https://www.devmedia.com.br/processo-unificado-aberto-net-magazine-75/17463. Acesso em: 02 nov. 2022.

O OpenUP é originário do *Eclipse Process Framework* (EPF), um framework de processo *open source* desenvolvido na *Eclipse Foundation*. Ele fornece as melhores práticas coletadas a partir de profissionais de desenvolvimento de software, e possui uma vasta comunidade de desenvolvimento de software que mantém um conjunto diverso de perspectivas e necessidades de desenvolvimento.

OpenUP é uma metodologia livre e mantém as características essenciais do RUP, como o desenvolvimento iterativo e incremental, aplicação de casos de uso e cenários na para apoio ao desenvolvimento, além de também ter como foco o gerenciamento de riscos.

CONSIDERAÇÕES FINAIS

Nesta aula apresentamos o modelo de processo unificado, um modelo que combina os ciclos iterativo e incremental para a construção de softwares.

Com forte base na orientação ao objeto, o processo unificado consegue apoiar a produção de um sistema de grande porte como a partir de vários pequenos sistemas, o que diminui o risco do projeto.

O processo unificado é dividido em 4 etapas: Concepção, Elaboração, Construção e Transição, que possui vários subetapas executadas em cada ciclo.

A partir do processo unificado surgiu o *Rational Unified Process*, conhecido como RUP, que refina no processo e estabelece o desenvolvimento de software organizado, e com especial atenção aos riscos desde as fases iniciais do projeto.

Embora seja muito bem estruturado e organizado, o processo sofre algumas críticas, como a de ser complexo e trabalhoso, principalmente no caso de projetos de pequeno porte.

MATERIAIS COMPLEMENTARES

O processo unificado. Disponível em: https://medium.com/contexto-delimitado/o-processo-unificado-d102b1fc9d00. Acesso em: 02 nov. 2022.

O processo unificado para o desenvolvimento web. Disponível em: https://www.devmedia.com.br/artigo-engenharia-de-software-o-processo-unificado-integrado-ao-desenvolvimento-web/8032. Acesso em: 02 nov. 2022.

REFERÊNCIAS

PRESSMAN, R. *Engenharia de Software*. 6ª Edição. McGraw-Hill, Nova York, EUA, 2006.

PRESSMAN, R. *Engenharia de Software*. 9ª ed. Rio de Janeiro: McGraw-Hill, 2021.

SOMMERVILLE, I. *Engenharia de Software*. 10ª ed. São Paulo: Pearson Addison. Wesley, 2019.