**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.

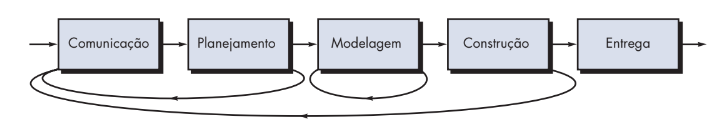
Além de fornecer um roteiro para desenvolvimento de software definindo as atividades necessárias a serem realizadas e a sequência delas, o modelo de processo também estabelece o fluxo de processo, neste aspecto, o modelo descreve como são organizadas as atividades metodológicas, bem como as ações e tarefas 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:



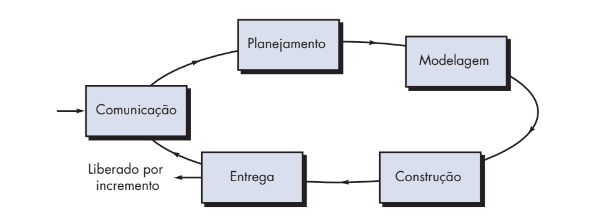
**Figura 1.1. Fluxo de processo linear**

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)



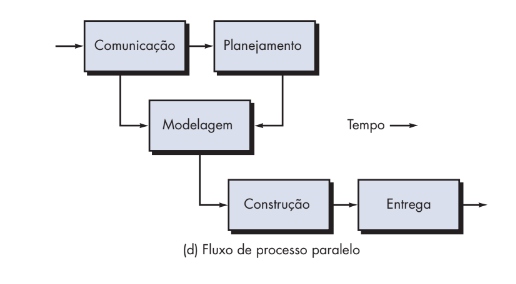
**Figura 1.2. Fluxo de processo iterativo**

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.

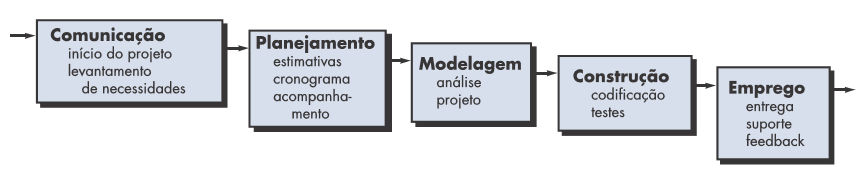
**1.5.4. Construção**

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**

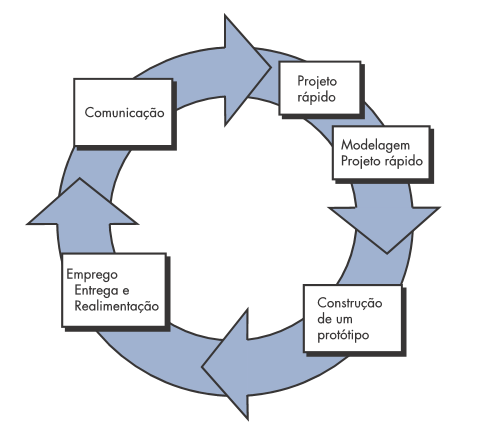
**1.5.7. Observações sobre o Modelo Cascata**

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**

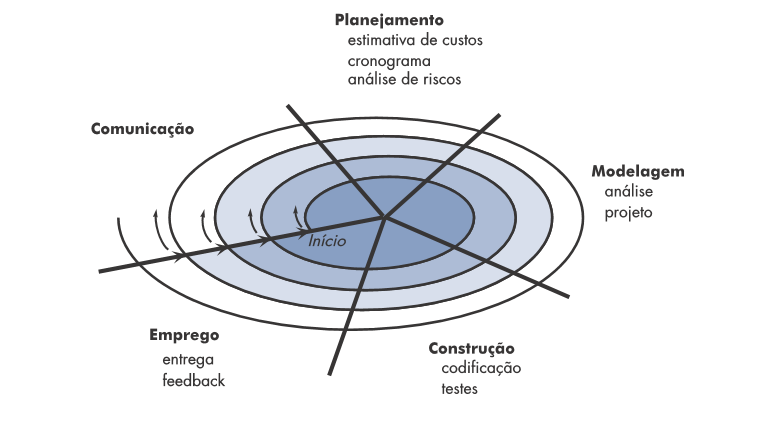
Alguns problemas desse modelo são que o cliente pode não entender que o software que ele vê não considerou, durante o desenvolvimento, a qualidade global e a manutenção a longo prazo. Ele pode não aceitar bem a ideia que a versão final do software vai ser construída e "força" a utilização do protótipo 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 4ª Geração**

De acordo com Pressman (2011) o desenvolvimento baseado em técnicas de 4ª 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.