

# Engenharia de Software



Educação a Distância  
Cruzeiro do Sul Educacional  
Campus Virtual



# Material Teórico



Processos de software

**Responsável pelo Conteúdo:**

Prof.<sup>a</sup> Dr.<sup>a</sup> Ana Paula do Carmo Marchetti Ferraz

**Revisão Textual:**

Prof.<sup>a</sup> Me. Luciene Oliveira da Costa Santos





- Introdução

- Modelos de desenvolvimento de software, chamados também de modelos de processo de software ou paradigmas de software



**Objetivo de APRENDIZADO**

- Compreender o que é um processo de software.
- Entender a importância do processo para o sucesso do produto desenvolvido.
- Conhecer e aplicar os principais paradigmas de desenvolvimento de software.

Organize-se de forma a não deixar para o último dia a realização das atividades (AS e AP), pois podem ocorrer imprevistos. Lembre-se de que, após o encerramento da Unidade, encerra-se a possibilidade de obter a nota relativa a cada atividade.

Para ampliar seu conhecimento, bem como aprofundar os assuntos discutidos, pesquise, leia e consulte os livros indicados nas Referências e/ou na Bibliografia.

As Referências estão indicadas ao final dos textos de conteúdo de cada Unidade.

A Bibliografia Fundamental para esta Disciplina é: SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011. A Bibliografia Complementar está indicada em item específico, em cada unidade.

Caso ocorram dúvidas, contate o professor tutor por meio do Fórum de Dúvidas, local ideal para esclarecê-las, pois assim a explicação poderá ser compartilhada por todos.

Existe, ainda, a possibilidade de contatar o professor pelo link mensagens, caso seja algum assunto relativo somente a você, e não uma dúvida sobre a matéria, que pode ser também dúvida de outros alunos.

Fique atento(a) quanto a possíveis dúvidas e problemas, lembrando-se de que o professor tutor está à sua disposição para resolver assuntos pedagógicos, isto é, dúvidas quanto ao conteúdo da matéria.

Se suas dúvidas ou problemas se referirem ao sistema, aos programas etc., contate o Suporte ou procure auxílio de um monitor no webclass de seu campus.

## Contextualização

A partir das dificuldades envolvidas na produção de software, iniciaremos a contextualização sobre desenvolvimento de software, métodos e técnicas envolvidos, além de conhecer alguns dos principais paradigmas para o desenvolvimento de software.

Durante esta unidade, você conhecerá os principais modelos de processo de software. Afinal, são muitos os modelos propostos pela Engenharia de Software. Portanto, não limite seu aprendizado apenas aos modelos aqui apresentados, estude, pesquise, busque novos conhecimentos!

Adquira o hábito da pesquisa, visite sites, pesquise em livros e periódicos. Sugerimos que você dê pequenas pausas na leitura e pesquise sobre os conceitos abordados nas unidades. E lembre-se de compartilhar suas descobertas com seus colegas de turma.

## 1. Introdução



Um processo de software é um conjunto de atividades relacionadas que levam à produção de um produto de software (SOMMERVILLE, 2011, p. 33).

Para Pfleeger (2004), é um conjunto de atividades cujo objetivo é o desenvolvimento ou evolução do software.

Existem muitos processos diferentes de desenvolvimento de software, entretanto, algumas atividades são genéricas a todos eles:

- **Especificação:** a funcionalidade do software e as restrições em sua operação devem ser definidas.
- **Desenvolvimento:** produção do software de modo que atenda a suas especificações.
- **Validação:** o software deve ser validado para garantir que ele faça o que o cliente deseja.
- **Evolução:** o software deve evoluir para atender às necessidades mutáveis do cliente.

### Glossário



**Cliente:** “[...] é a empresa, organização ou pessoa que está pagando para o sistema de software ser desenvolvido”. (PFLEEGER, 2004, p. 11)

Não existe, segundo Pressman (2006, p. 27), uma única abordagem ou método de produção de software mágico, que funcione para todos os tipos, o que existe é uma combinação de métodos abrangentes a todas as etapas relacionadas.

Além disso, é importante e desejável que esses métodos sejam suportados por um conjunto de ferramentas que permita automatizar o desenrolar das etapas, juntamente com uma definição clara de critérios de qualidade e produtividade de software. São esses aspectos que caracterizam de maneira mais influente a disciplina de **Engenharia de Software**.

Assim, podemos relembrar com propriedade que Engenharia de Software é uma disciplina que reúne **procedimentos** (metodologias), **métodos e ferramentas** a serem utilizados, desde a percepção do problema até o momento em que o sistema desenvolvido deixa de ser operacional (existir), visando resolver problemas inerentes ao processo de desenvolvimento e ao produto de software (PFLEEGER, 2004).

Segundo Sommerville (2009) e Pfleeger (2004):

- **Métodos:** proporcionam os detalhes de como fazer para construir o software (gerenciamento do projeto, análise de sistemas, análise de requisito, projeto do software, geração do código, teste, manutenção etc.). Muitos desses veremos nas próximas unidades.
- **Ferramentas:** dão suporte automatizado aos métodos. Existem ferramentas para sustentar cada um dos métodos. Quando as ferramentas são integradas, é estabelecido um sistema de suporte ao desenvolvimento de software chamado *Case - Computer Aided Software Engineering*.
- **Procedimentos:** constituem o elo entre os métodos e ferramentas. Estão relacionados à sequência em que os métodos serão aplicados aos produtos de software. Por meio dos procedimentos, ocorrem controles que asseguram a qualidade e coordenam as alterações, além de permitirem marcos de referência que possibilitam a administração do progresso do produto de software.

Nesse contexto, a Engenharia de Software apoia-se na tecnologia para produzir software de alta qualidade, a um baixo custo e num tempo menor. Em contrapartida, os engenheiros de software a utilizam para enfrentar os desafios do desenvolvimento de software.

Existem vários modelos de processo de software também chamados de **paradigmas de engenharia de software**. Cada modelo representa uma perspectiva particular de um processo, portanto, não fornece todas as informações. Por exemplo, um determinado modelo de atividade de processo pode mostrar as atividades envolvidas e suas sequências, mas não mostra os papéis dos agentes envolvidos.

## Glossário

**Processo de Software:** “[...] um arcabouço para as tarefas que são necessárias para construir softwares de alta qualidade”. (PRESSMAN, 2006, p. 16)



## 2. Modelos de desenvolvimento de software, chamados também de modelos de processo de software ou paradigmas de software



### Glossário



**Paradigma da Engenharia de Software:** “[...] representa a abordagem ou filosofia em particular para a construção de software”. (PFLEEGER, 2004, p. 3) Segundo Pressman (1996, p. 33): “[...] a engenharia de software compreende um conjunto de métodos, ferramentas e procedimentos [...] estas etapas são muitas vezes citadas como paradigmas de engenharia de software”.

O esforço de desenvolvimento deve resultar em um **produto**.

O **modelo de processo de desenvolvimento** corresponde ao conjunto e ao ordenamento de atividades de modo a que o produto desejado seja obtido (SOMMERVILLE, 1994).

Um **modelo de desenvolvimento** corresponde a uma representação abstrata do processo de desenvolvimento que vai, em geral, definir como as etapas relativas à criação do software serão conduzidas e inter-relacionadas para atingir o objetivo – que é a obtenção de um produto de software de alta qualidade a um custo relativamente baixo.

Vários são os modelos existentes:

- **Sequencial Linear**

- Modelo Cascata ou Ciclo de Vida Clássico \*
- Prototipação \*
- O Modelo RAD (*Rapid Application Development*)

- **Modelos Evolutivos de Processo de Software**

- Incremental \*
- Espiral \*
- De Montagem de Componentes \*
- De Desenvolvimento Concorrente

- **Modelos de Métodos Formais**

- **Técnicas de Quarta Geração**

- **Métodos Ágeis\***

- *Extreme Programming (XP)*

Na nossa disciplina, por uma questão de tempo, não veremos todos os modelos, mas apenas os que estão marcados com asteriscos. Por isso é importante que você faça uma pequena pesquisa sobre outros modelos para conhecê-los.

## 2.1 Modelos Lineares

### 2.1.1 O modelo cascata ou ciclo de vida clássico

Este é o modelo mais simples de desenvolvimento de software, estabelecendo uma ordenação linear no que diz respeito à realização das diferentes etapas.

Ele é o modelo mais antigo e o mais amplamente usado da Engenharia de Software por ter sido modelado em função do ciclo da Engenharia convencional.

Requer uma abordagem sistemática, sequencial ao desenvolvimento de software e o resultado de uma fase se constitui na entrada da outra. (Figura 1)

Existem inúmeras variações desse modelo, dependendo da natureza das atividades e do fluxo de controle entre elas.

Os estágios principais do modelo estão relacionados às atividades fundamentais de desenvolvimento.

O ponto de partida do modelo é uma etapa de **Engenharia de Sistemas**, onde o objetivo é ter uma visão global do sistema (incluindo hardware, software, equipamentos e as pessoas envolvidas). Em seguida, a etapa de **Análise de Requisitos** vai permitir uma clara definição dos requisitos de software, na qual o resultado será utilizado como referência para as etapas posteriores de **Projeto, Codificação, Teste e Manutenção**.

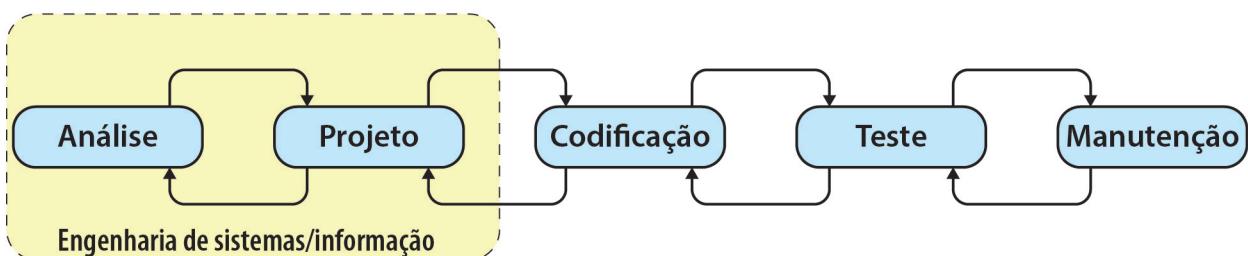


Figura 1 - Ilustração do modelo Cascata adaptado de Pressman (2006).

O modelo Cascata, por ter uma ordenação linear nas etapas de desenvolvimento, apresenta características úteis no processo de desenvolvimento de software, particularmente em razão da definição de um ordenamento linear das etapas de desenvolvimento.

Como forma de identificar o fim de uma etapa e o início da próxima, existem mecanismos implementados ao final de cada etapa que serve como balizador para início da próxima. Isso é feito normalmente através da aplicação de algum método de validação ou verificação, cujo objetivo é garantir que a saída de uma dada etapa seja coerente com a sua entrada (a qual já é a saída da etapa precedente).

Isso significa que, ao final de cada etapa realizada, deve existir um resultado (ou saída) a qual possa ser submetida à atividade de certificação.

Segundo Pressman (2006, p. 67), duas diretivas que norteiam o desenvolvimento, segundo o modelo Cascata, são:

1. Todas as etapas definidas no modelo devem ser realizadas, isto porque, em projetos de grande complexidade, a realização formal delas vai determinar o sucesso ou não do desenvolvimento.

*A realização informal e implícita de algumas dessas etapas poderia ser feita apenas no caso de projetos de pequeno porte.*

2. A ordenação das etapas na forma como foi apresentada deve ser rigorosamente respeitada.

Temos sempre que lembrar que os resultados de um processo de desenvolvimento de produto de software não é apenas o programa executável, mas também toda documentação associada.

Cada fase mostrada na Figura 1 pode gerar resultados que devem ser documentados (processo e resultados). Alguns desses documentos são: de especificação de requisitos, de projeto do sistema, de plano e relatório de testes, de codificação ou implementação, de utilização, relatórios de revisões etc.

Todo modelo tem suas vantagens e desvantagens na sua utilização, e no modelo Cascata não seria diferente. Vejamos algumas limitações desse modelo:

- Nenhum modelo segue rotinas tão sequenciais quanto almeja.
- No início do processo, não temos todos os requisitos mapeados ou definidos.
- No começo dos projetos, sempre existe uma incerteza. O cliente deve ter paciência, pois uma versão final do produto só fica pronta e disponível no final do processo.

Algumas contribuições do modelo:

- Esse modelo requer que o desenvolvedor e todos os envolvidos tenham disciplina, e realizem o planejamento e gerenciamento do processo.
- O desenvolvimento do produto deve ser adiado até que os objetivos tenham sido completamente compreendidos.

## 2.1.2 Modelo de prototipação

A Prototipação é um modelo de processo de desenvolvimento que busca contornar algumas das limitações existentes no modelo Cascata, inclusive apresentadas anteriormente. Isso é possível devido ao fato de que o foco principal é o desenvolvimento de um protótipo, com base no conhecimento dos requisitos iniciais para o sistema. Dessa forma, o cliente poderá ter uma ideia de como ficará seu produto, antes da etapa final.

Para desenvolver o protótipo, etapas devem ser seguidas (a análise de requisitos, o projeto, a codificação e os testes). Entretanto, elas não precisam ser realizadas com o rigor necessário (final) que o modelo Cascata define. Podemos criar o protótipo com apenas alguns requisitos definidos e, ao longo do processo, o protótipo vai se completando conforme a definição de outros.

Esse modelo tem como objetivo entender os requisitos do usuário para, assim, obter uma melhor definição dos requisitos do sistema e possibilitar que o desenvolvedor crie um modelo (protótipo) do software que deve ser construído.

É apropriado para quando o cliente já definiu um conjunto de objetivos gerais para o software, mas não identificou detalhadamente esses requisitos.

Após criar o protótipo, ele é colocado à disposição do cliente, que vai ajudar na melhor compreensão do que será o sistema desenvolvido. Além disso, através da manipulação desse protótipo, é possível validar ou reformular os requisitos para as etapas seguintes do sistema.

Segundo Pressman (2006), o modelo ilustrado na figura 1 apresenta algumas características interessantes, tais como:

- \* Através da construção de um protótipo do sistema, é possível demonstrar a reusabilidade do mesmo.
- \* É possível obter uma versão, mesmo que simplificada, do que será o sistema, com um pequeno investimento inicial.

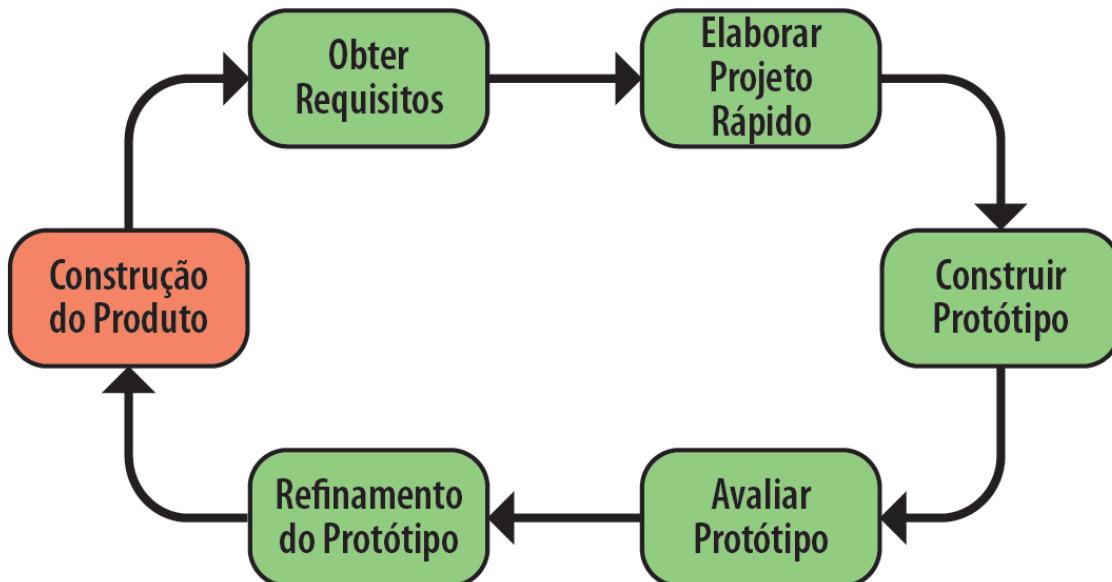


Figura 2 - Esquema de evolução da prototipação.

### Problemas com a prototipação:

- O cliente não sabe que o software que ele vê não considerou, durante o desenvolvimento, a qualidade global e a manutenibilidade em longo prazo.
- O desenvolvedor frequentemente faz uma implementação comprometida (utilizando o que está disponível), com o objetivo de produzir rapidamente um protótipo.
- Os protótipos não são sistemas completos e deixam, normalmente, a desejar em alguns aspectos. Um desses aspectos é normalmente a interface com o usuário.
- Os esforços de desenvolvimento são concentrados principalmente nos algoritmos que implementam as principais funções associadas aos requisitos apresentados, a interface, sendo, nesse nível, parte supérflua do desenvolvimento, o que permite caracterizar esta etapa por um custo relativamente baixo.

### Contribuições:

- Ainda que possam ocorrer problemas, a prototipação é um ciclo de vida eficiente.
- A chave é definir todas as regras (requisitos, processos, métodos, técnicas etc.) logo no começo do desenvolvimento.
- O cliente e o desenvolvedor devem concordar que o protótipo seja construído para servir como um mecanismo a fim de definir os requisitos.
- A experiência adquirida no desenvolvimento do protótipo vai ser de extrema utilidade nas etapas posteriores do desenvolvimento do sistema real, permitindo reduzir certamente o seu custo, resultando também num sistema melhor concebido.

Segundo Sommerville (2011), às vezes, os desenvolvedores são pressionados a entregar protótipos descartáveis, principalmente quando há atraso no prazo de entrega. Entretanto, isso não é aconselhável, pois, dentre outras coisas, pode ser impossível ajustar o protótipo para atender aos requisitos não funcionais como desempenho, robustez, segurança, confiabilidade etc. durante o desenvolvimento do protótipo.

## 2.2 Modelos Evolutivos de Processo de Software

Existem situações em que a Engenharia de Software necessita de um modelo de processo que possa acomodar um produto que evolui com o tempo.

São eles, segundo Pressman (2006, p. 87):

- Quando os requisitos de produto e de negócio mudam conforme o desenvolvimento procede.
- Quando um prazo de entrega é estreito (mercado), há impossibilidade de conclusão de um produto completo.
- Quando um conjunto de requisitos importantes é bem conhecido, mas os detalhes ainda devem ser definidos.

Os modelos evolutivos são iterativos e possibilitam o desenvolvimento de versões cada vez mais completas do software.

### 2.2.1 Modelo Incremental

Como no modelo de Prototipação, o modelo Incremental também foi concebido a partir da exploração das limitações do modelo Cascata, ao mesmo tempo em que combina as vantagens do modelo Cascata com as do de Prototipação.

A ideia principal desse modelo, ilustrada na Figura 3, é a de que um sistema deve ser desenvolvido de forma incremental, sendo que cada incremento vai adicionando ao sistema novas capacidades funcionais, até a obtenção do sistema final, ao passo que, a cada etapa realizada, modificações podem ser introduzidas.

Uma das vantagens dessa abordagem é a facilidade em testar o sistema durante cada uma das fases de desenvolvimento.

Um aspecto interessante desse modelo é a criação de uma lista de controle de projeto. Ela deve conter todas as etapas a serem realizadas, da concepção à obtenção do sistema final. Ela vai servir também para medir, num dado nível, o quanto distante está da última versão aquela que será disponibilizada ao cliente.

Esta lista de controle de projeto gerencia todo o desenvolvimento, definindo quais tarefas devem ser realizadas a cada iteração. Na lista de tarefas, podemos, inclusive, inserir as redefinições de componentes já implementados, em razão de erros ou problemas detectados numa eventual etapa de análise.

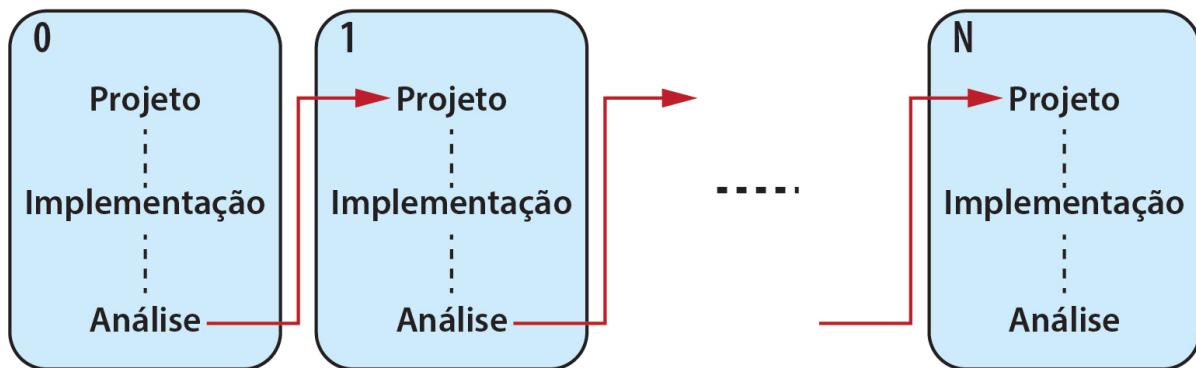


Figura 3 - O modelo de desenvolvimento incremental.

Considerações positivas sobre o modelo:

- A versão inicial é frequentemente o núcleo do produto (a parte mais importante).
  - O desenvolvimento começa com as partes do produto que são mais bem entendidas.
  - A evolução acontece quando novas características são adicionadas à medida que são sugeridas pelo usuário.
- O desenvolvimento incremental é importante quando é difícil, ou mesmo impossível, estabelecer *a priori* uma especificação detalhada dos requisitos.
- As primeiras versões podem ser implementadas com poucas pessoas se o núcleo do produto for bem recebido, novas pessoas podem participar do desenvolvimento da nova versão.
- As novas versões podem ser planejadas de modo que os riscos técnicos possam ser administrados.

Exemplo:

- O sistema pode exigir a disponibilidade de um hardware que está em desenvolvimento e cuja data de liberação é incerta.
- Podem ser planejadas pequenas versões de modo a evitar o uso desse hardware.
- O software com funcionalidade parcial pode ser liberado para o cliente.

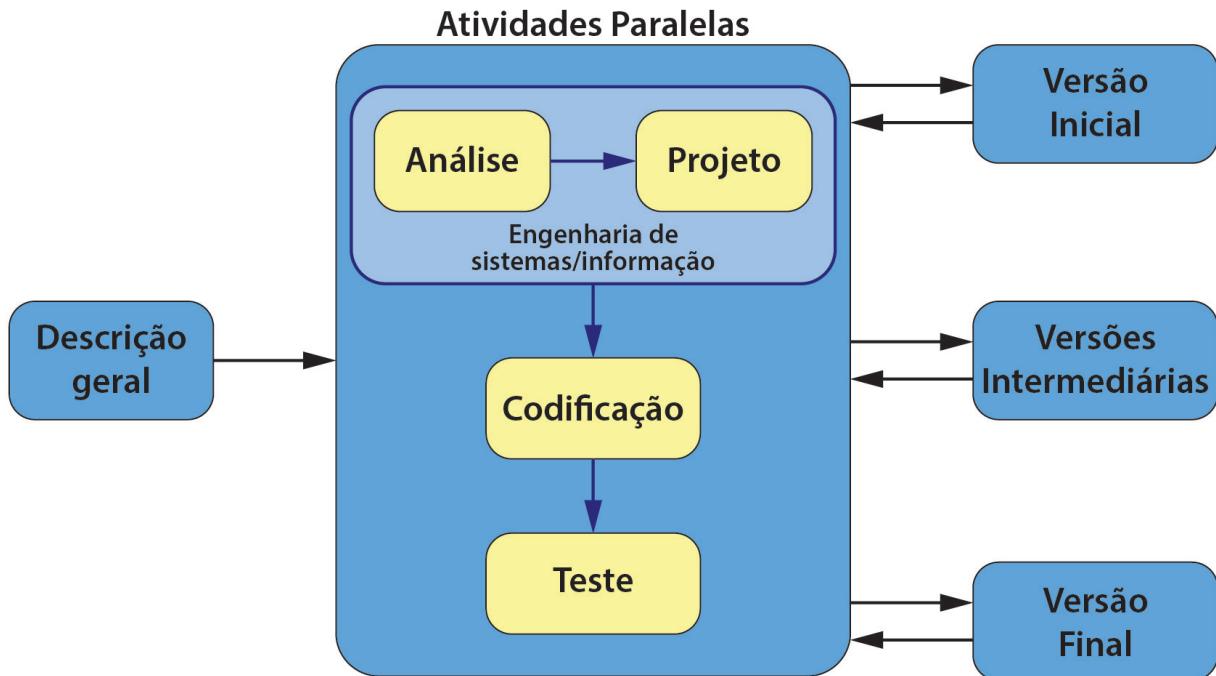


Figura 4 - Modelo Incremental, com a exemplificação das atividades paralelas.

### 2.2.2 O Modelo Espiral

Esse modelo foi proposto em 1988 e sugere uma organização de desenvolvimento em espiral, o que deu origem ao nome Modelo de Desenvolvimento Espiral.

Ele reúne a natureza iterativa da prototipação com os aspectos controlados e sistemáticos do modelo Cascata.



#### Explore

Faça uma pesquisa em algum dicionário sobre a diferença entre interação e iteração. O conhecimento e reconhecimento dessa diferença, para você que está estudando sobre Engenharia de Software, é muito importante.

O modelo Espiral é dividido em uma série de atividades de trabalho ou regiões de tarefa.

Existem tipicamente de 3 a 6 regiões ou setores de tarefa. Na Figura 5 temos um exemplo de 4 regiões ou setores definidos pelos quadrantes.

Segundo Sommerville (2011), em cada uma delas, temos:

**Definição dos objetivos:** os objetivos específicos para esta fase são definidos, restrições ao processo e ao produto são identificadas e um plano de gerenciamento é detalhado e elaborado; **riscos são identificados.**

Avaliação e redução de riscos: para cada um dos riscos identificados é feita uma análise detalhada e ações para eliminá-los ou reduzi-los são tomadas. A continuidade do processo de desenvolvimento é definida como função dos riscos que ainda não foram resolvidos, inclusive

em ordem de prioridade. Um bom exemplo disso são os riscos relacionados ao desempenho ou à interface que são considerados mais importantes do que aqueles relacionados ao desenvolvimento do programa. A partir das decisões tomadas na avaliação e redução de riscos, um próximo passo pode ser o desenvolvimento de um protótipo que elimine os riscos considerados para novamente avaliar o produto, o processo e identificarmos os riscos.

Por outro lado, se os riscos de desenvolvimento de programa forem considerados os mais importantes, e se o protótipo construído já resolver alguns dos riscos ligados ao desempenho e à interface, podemos seguir à implementação utilizando o modelo Cascata na nossa fase do projeto.

**Desenvolvimento e Avaliação:** após a avaliação dos riscos, é selecionado um modelo para o aprimoramento do sistema, que pode ser Prototipação, Incremental, Cascata etc.

**Planejamento:** o projeto é revisado e uma decisão é tomada a respeito da continuidade do modelo com mais de uma volta da espiral. Caso se decida pela continuidade, novos planos são elaborados para a próxima fase do projeto.

Como se pode ver, o elemento que conduz esse processo é a consideração sobre os riscos existentes ou iminentes, o que permite, em alguns casos, adequação e escolha de qualquer política ou modelo de desenvolvimento, como a baseada em especificação, baseada em simulação, baseada em protótipo, dentre outras.

Esse modelo, por ter sido desenvolvido há pouco tempo, não é muito utilizado, embora seja muito adequado ao desenvolvimento de sistemas complexos.

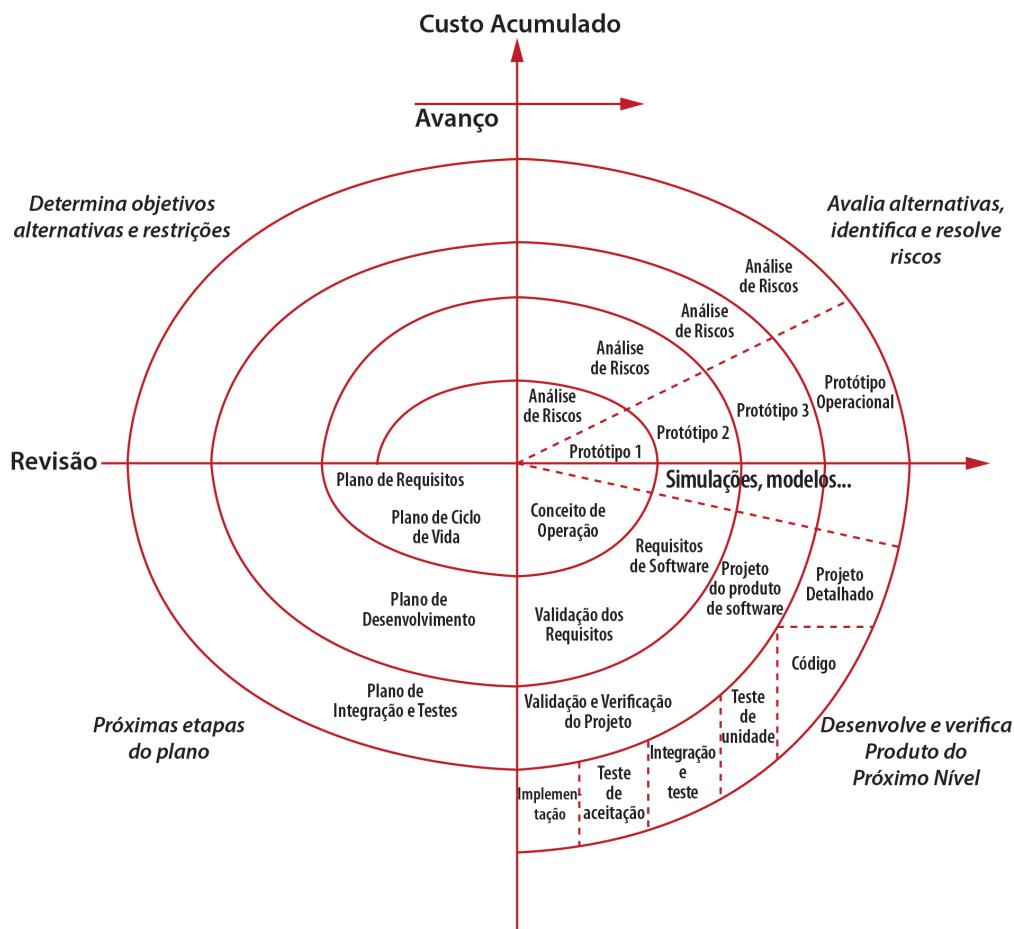


Figura 5 - O modelo Espiral. (SOMMERVILLE, 2011, p. 33).

Observações sobre esse modelo, segundo Sommerville (2006, 2001) e Pfleeger (2004):

- Cada loop no espiral representa uma fase do processo de software. O mais interno está concentrado nas possibilidades do sistema.
- O próximo loop está concentrado na definição dos requisitos do sistema.
- O loop um pouco mais externo está concentrado no projeto do sistema.
- Engloba as melhores características do ciclo de vida Clássico e da Prototipação, adicionando um novo elemento: a Análise de Risco.
- Esse modelo segue a abordagem de passos sistemáticos do Ciclo de Vida Clássico, incorporando-os numa estrutura iterativa que reflete mais claramente o mundo real.
- Usa a Prototipação, em qualquer etapa da evolução do produto, como mecanismo de redução de riscos.
- É, atualmente, a abordagem mais realística para o desenvolvimento de software em grande escala.
- Usa uma abordagem que capacita o desenvolvedor e o cliente a entender, perceber e resolver os riscos em cada etapa evolutiva.
- Pode ser difícil convencer os clientes de que uma abordagem evolutiva é controlável.
- Exige considerável experiência na determinação de riscos e depende dessa experiência para ter sucesso.

Uma característica importante desse modelo é o fato de que cada ciclo é encerrado por uma atividade de revisão, na qual todos os produtos do ciclo são avaliados, incluindo o plano para o próximo passo (ou ciclo).

### 2.2.3 O Modelo de Montagem de Componentes

As tecnologias de objeto fornecem o arcabouço técnico para o modelo de processo baseado em componentes. Esse modelo é orientado a objeto e enfatiza a criação de classes que encapsulam tanto os dados quanto os algoritmos que são usados para manipular os dados.

Quando projetadas e implementadas de forma adequada, todas as classes orientadas a objeto são reutilizáveis em diferentes aplicações e arquiteturas de sistema.

Esse modelo incorpora características de tecnologias orientadas a objetos no modelo Espiral, é de natureza evolutiva e demanda uma abordagem iterativa para a criação do software.

## 2.3 Métodos Ágeis

Entre as décadas de 1980 e 1990, havia uma visão generalizada de que a melhor forma de produzir um software era por meio de planejamento cuidadoso do projeto, qualidade de segurança formalizada do uso de métodos de Engenharia de Software e projeto apoiado por ferramentas CASE e de processo de desenvolvimento de software rigorosamente controlado.

## Glossário



**Métodos de engenharia de software:** Segundo Pressman (1996, p. 31): “[...] proporcionam detalhes de ‘como fazer’ para construir o software. Os métodos envolvem um grande número de tarefas que incluem: planejamento e estimativa de projeto, análise de requisitos de software e de sistema, projeto da estrutura de dados, arquitetura de programa e algoritmo de processamento, codificação, teste e manutenção.”

Entretanto, foi percebido que essa abordagem pesada de desenvolvimento, num ambiente dinâmico e conectado não era adequada. Muito tempo era gasto no processo de planejamento e análise de requisitos e isso prejudicava algumas etapas, sem contar o atraso no prazo de entrega.

A insatisfação com essa abordagem na década de 1990 foi tão grande que um grupo de engenheiros de software idealizou uma nova abordagem, a qual foi chamada de Métodos Ágeis. Esses métodos permitiram que a equipe de desenvolvimento focasse no software em si e não na documentação do processo. Assim, esse modelo mostrou-se adequado para aqueles cujos requisitos mudam constantemente durante o processo de desenvolvimento.

Com esse método foi possível executar a entrega de sistemas complexos em tempos menores. Além disso, quaisquer novos requisitos solicitados têm a possibilidade de inclusão como alterações de sistema, o que significa a entrega de uma nova versão com tais adequações.

Segundo Sommerville (2011), práticas ágeis enfatizam a importância de se escrever códigos bem estruturados e investir na sua melhoria. A falta de documentação não deve ser um problema na manutenção do sistema por meio de uma abordagem ágil.

Entretanto, talvez esse seja o principal problema dessa abordagem, pois manutenção, testes e alterações de sistemas se tornam extremamente complexos quando não existem documentações disponíveis.

À medida que as características dos softwares, dos desenvolvedores e dos clientes mudam, a Engenharia de Software procura adaptar seus modelos à nova realidade.

A metodologia Extreme Programming, também conhecida como XP, ou Programação Extrema, proposta por Kent Beck, em 1998 (TELES, 2004), é um dos mais novos métodos da Engenharia de Software e é um exemplo de metodologia ágil para desenvolvimento de software. Por ser um modelo novo e bastante aceito, vejamos a seguir alguns de seus detalhes:

O XP é um processo de desenvolvimento que busca assegurar que o cliente receba o máximo de valor de cada dia de trabalho da equipe de desenvolvimento. Ele é organizado em torno de conjunto de valores e práticas que atuam de forma harmônica e coesa para assegurar que o cliente sempre receba um alto retorno do investimento em software. (TELES, 2004, p. 21)

Essa abordagem foi desenvolvida para impulsionar práticas reconhecidamente boas, como desenvolvimento interativo, em níveis extremos. Por exemplo, em um XP, várias novas versões de um sistema podem ser desenvolvidas, integradas e testadas em um único dia por programadores diferentes (SOMMERVILLE, 2011).

O ciclo de vida do XP tem quatro atividades básicas: codificar, testar, escutar e modelar, demonstradas através de quatro valores: **a comunicação, a simplicidade, o feedback e a coragem** (SOMMERVILLE, 2006).

A prática do XP tem **comunicação** contínua entre o cliente e a equipe, com o objetivo de criar o melhor relacionamento entre o cliente e desenvolvedor, dando preferência às conversas pessoais em vez de manter contato através de outros meios. É incentivada também a comunicação intensa entre desenvolvedores e gerente do projeto. Essa comunicação entre o cliente e a equipe permite que todos os detalhes do projeto sejam tratados com a atenção e a agilidade que merecem, ou seja, a equipe pode codificar uma funcionalidade preocupando-se apenas com os problemas de hoje e deixar os problemas do futuro para o futuro.

A **simplicidade** foca sempre no mais simples que possa funcionar, visando minimizar o código, desprezando funções consideradas necessárias. O importante é ter em mente que os requisitos são mutáveis. Sendo assim, o interessante no XP é implementar apenas o que é estritamente necessário.

O contato incessante com o cliente a respeito do projeto é o que se pode chamar de **feedback constante**. As informações sobre o código são dadas por teste periódico, os quais indicam erros tanto individuais quanto do software integrado. O cliente terá sempre uma parte do software funcional para avaliar.

A grande maioria dos princípios do XP corresponde a práticas de senso comum e que fazem parte de qualquer processo disciplinado. Por exemplo, simplicidade significa foco nas partes do sistema que são de alta prioridade e, somente depois, pensar em soluções para problemas que, até então, não são relevantes (e talvez nunca sejam devido às grandes modificações no sistema), pois o cliente aprende com o sistema que utiliza e reavalia as suas necessidades, gerando o feedback para a equipe de desenvolvimento.

As equipes XP acreditam que errar é natural e que falhas no que estava funcionando acontecem cedo ou tarde. É necessário ter coragem para lidar com esse risco, o que em XP se traduz em confiança nos seus mecanismos de proteção. A **coragem** encaixa-se na implantação dos três valores anteriores. São necessários profissionais comunicativos e com facilidade de se relacionar. A coragem auxilia a simplicidade, quando a possibilidade de simplificar o software é detectada. Desse modo, a coragem é necessária para garantir que o feedback do cliente ocorra com frequência, pois essa abordagem implica a possibilidade de mudanças constantes do produto.

Enfim, no sistema desenvolvido de forma incremental, a equipe está continuamente fazendo a manutenção do software e criando novas funcionalidades. Por isso a equipe precisa ser corajosa e acreditar que, utilizando práticas e valores do XP, será capaz de fazer o software evoluir com segurança e agilidade.

A abordagem tradicional é mais arriscada que a do XP, pois os usuários não têm a chance de avaliar o software sempre, e o projeto só começa a gerar receita após a conclusão.

Segundo Brooks (1995), é comum o cliente fazer um grande investimento na construção de um software e este não atender às suas necessidades, levando a um novo projeto para a construção de outro software que atenda aos anseios dos usuários.

Por todas essas razões, os projetos em XP trabalham com releases de, no máximo, dois meses. Isso permite incorporar uma boa quantidade de funcionalidade em cada release e permite que os usuários aprendam com o software com bastante frequência.

O XP é aplicado em larga escala em vários projetos no mundo todo, mas ainda há muito a evoluir em sua compreensão e aplicação. Notamos isso principalmente em pontos polêmicos como testes unitários, programação em dupla, rodízio de pessoas, propriedade coletiva do código e otimização de jornadas, práticas que, se mal utilizadas, podem realmente trazer aumentos no custo e no prazo de projetos. Ou seja, é de extrema importância que entendamos bem a essência em XP e, principalmente, que tenhamos disciplina e criatividade, duas qualidades básicas em quem pretende desenvolver projetos. Somente a partir de uma visão criativa sobre a metodologia e disciplina para cumprimento de tarefas é que todos poderão usar e ter benefícios em XP.

## 2.4 As fases genéricas da Engenharia de Software e suas composições

Agora podemos definir e agrupar alguns dos assuntos abordados até o momento com o intuito de dar uma visão geral do papel do Engenheiro de Software.

Sabemos que a Engenharia engloba métodos, técnicas e procedimentos com o objetivo de desenvolver um bom produto de software.

Entretanto, é importante lembrarmos que, independentemente da área de aplicação, tamanho ou complexidade do projeto/produto, o trabalho associado à Engenharia de Software pode ser categorizado em 3 fases genéricas: fase de definição de projeto, fase de desenvolvimento e fase de manutenção, sendo que todas são complementadas por atividades de apoio.

A fase de **definição de processo** focaliza **o que** será desenvolvido: que informação vai ser processada, que função e desempenho são desejados, que comportamento pode ser esperado do sistema, que interfaces vão ser estabelecidas, que restrições de projeto existem, que critérios de validação são exigidos para definir um sistema bem-sucedido. Nessa fase, acontece a Engenharia de Sistemas, planejamento do projeto de software e análise de requisitos.

A fase de **desenvolvimento de processos** de software focaliza como o software será desenvolvido: como os dados vão ser estruturados, como a função vai ser implementada como uma arquitetura de software, como os detalhes procedimentais vão ser implementados, como as interfaces vão ser caracterizadas, como o projeto será traduzido em uma linguagem de programação, como os testes serão efetuados. Nessa fase, acontece o projeto do software, geração do código e o teste do software.

A última fase é a da **manutenção**, que focaliza as **mudanças** que ocorrerão depois que o software for liberado para uso operacional. Essa etapa reaplica os passos das fases de definição e desenvolvimento, mas faz isso no contexto de um software existente.

As fases do processo de software são complementadas por uma série de atividades de apoio. Essas atividades são aplicadas durante toda a Engenharia do Software. São elas: Métricas, Gerenciamento de Riscos, Acompanhamento e Controle do Projeto de Software, Garantia de Qualidade de Software, Gerenciamento de Configuração de Software.

Nas nossas próximas unidades, abordaremos vários dos itens existentes em cada uma das fases.



## Explore

**Dica:** visite sempre o site: <http://www.devmedia.com.br/revista-engenharia-de-software-magazine>.

Essa revista sobre Engenharia de Software apresenta relatos, casos e aplicações práticas dos assuntos abordados nesta disciplina.

## Material Complementar

O objetivo do material complementar é lhe ajudar a entender, sob uma ótica diferente daquela da professora conteudista, assuntos abordados nas unidades teóricas.

É fundamental a leitura deste material para o melhor entendimento sobre o assunto.



### Explore

Como nesta unidade abordamos os conceitos gerais da Engenharia de Software, nossa sugestão de material complementar é o capítulo intitulado Processo de software, no seguinte livro:

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011. p. 18-37.

## Referências

### Bibliografia Fundamental

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

### Bibliografia Básica

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informação**. 4. ed. Rio de Janeiro: LTC, 1999.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informação Gerenciais. Administrando a empresa digital**. 5. ed. São Paulo: Pearson Education do Brasil, 2006.

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

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 2006.

SOMMERVILLE, I. **Engenharia de Software**. 6. ed. São Paulo: Pearson Addison Wesley, 2005.

### Bibliografia Complementar

ALCADE, E.; GARCIA, M.; PENUELAS, S. **Informática Básica**. São Paulo: Makron Books, 1991.

FAIRLEY, R. E. **Software engineering concepts**. New York: McGraw-Hill, 1987.

**IEEE Software Engineering Standards (2013)** Disponível em: <[http://www.ieee.org/portal/innovate/products/standard/ieee\\_soft\\_eng.html](http://www.ieee.org/portal/innovate/products/standard/ieee_soft_eng.html)>. Acesso em: 10 dez. 2013.

LUKOSEVICIUS, A. P.; CAMPOS FILHO, A. N.; COSTA, H. G.. **Maturidade Em Gerenciamento De Projetos E Desempenho Dos Projetos**. Disponível em: <[www.producao.uff.br/conteudo/rpep/.../RelPesq\\_V7\\_2007\\_07.doc](http://producao.uff.br/conteudo/rpep/.../RelPesq_V7_2007_07.doc)>. Acesso em 12 nov. 2013.

MAFFEO, B. **Engenharia de software e especialização de sistemas**. Rio de Janeiro: Campus, 1992.

MICHAELIS. **Moderno dicionário da língua portuguesa**. São Paulo: Cia. Melhoramentos, 1998.

PARREIRA JÚNIOR, W. M. O. P. **Apostila de Engenharia de software**. Disponível em: <[http://www.waltenomartins.com.br/ap\\_es\\_v1.pdf](http://www.waltenomartins.com.br/ap_es_v1.pdf)>. Acesso em: 13 nov. 2013.

PAULA FILHO, W. P. **Engenharia de Software: fundamentos, métodos e padrões**. 2. ed. Rio de Janeiro: LTC, 2001.

**Revista Engenharia de Software.** Disponível em: <<http://www.devmedia.com.br/revista-engenharia-de-software-magazine>>. Acesso em 12 nov. 2013.

VONSTA, A. **Engenharia de programas.** Rio de Janeiro: LTC, 1983.

WIENNER, R.; SINCOVEC, R. **Software engineering with Modula 2 and ADA.** New York: Wiley, 1984.

WIKIPEDIA (2007a). **ISO 9000.** Disponível em: <[http://pt.wikipedia.org/wiki/ISO\\_9000](http://pt.wikipedia.org/wiki/ISO_9000)>. Acesso em: 22 jun. 2007.

WIKIPEDIA (2007b). **Melhoria de processo de software brasileiro.** Disponível em: <<http://pt.wikipedia.org/wiki/MPS.BR>>. Acesso em: 22 jun. 2007.

WIKIPEDIA (2007c). **CMMI.** Disponível em: <<http://pt.wikipedia.org/wiki/Cmmi>>. Acesso em: 22 jun. 2007.

WIKIPEDIA (2007d). **Engenharia de Software.** Disponível em: <[http://pt.wikipedia.org/wiki/Engenharia\\_de\\_software](http://pt.wikipedia.org/wiki/Engenharia_de_software)>. Acesso em: 1 fev. 2007.

# Anotações







**Educação a Distância**  
Cruzeiro do Sul Educacional  
*Campus Virtual*

www.cruzeirodosulvirtual.com.br  
Campus Liberdade  
Rua Galvão Bueno, 868  
CEP 01506-000  
São Paulo SP Brasil  
Tel: (55 11) 3385-3000



Universidade  
**Cruzeiro do Sul**



**UNICID**  
Universidade  
Cidade de S. Paulo



**UNIFRAN**  
Universidade  
de Franca



**UDF**  
Centro  
Universitário



**Módulo**  
Centro  
Universitário