**Processos de Software**

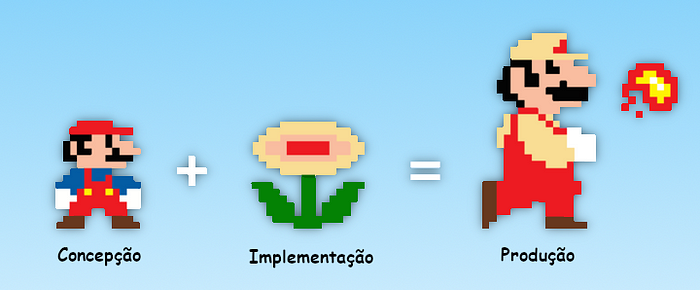
Quando um novo projeto de software se inicia, seu planejamento é um desafio e nem sempre é possível conseguir preencher todas as lacunas existentes no momento de sua elaboração. Dificuldades, imprevistos e retrabalho são comuns em muitos projetos, gerando desconforto para a empresa e para o cliente.

Embora não seja possível extinguir completamente esses fatores, é possível reduzi-los a um nível mínimo se, na estratégia utilizada, adotarmos um processo adequado para a realidade específica e o tipo de projeto a ser desenvolvido.

Neste artigo, tentarei explicar como funcionam os processos.

**O que é um processo?**

O *Processo de Software* é um conjunto de atividades usadas para a produção de sistemas computacionais, envolvendo o desenvolvimento do início ao fim.



Segundo JALOTE (2005), o *Processo de Software* é um conjunto de atividades, ligadas por padrões de relacionamento entre elas, pelas quais, se as atividades operarem corretamente e de acordo com os padrões requeridos, o resultado desejado é produzido. O resultado desejado é um software de alta qualidade e com baixo custo. Obviamente , um processo que não aumenta a produção ou não pode produzir software com boa qualidade não é um processo adequado.

Existem vários ***modelos de processo de software***, porém todos possuem algumas etapas em comum (SOMMERVILLE, 2011), como se seguem:

**1. Especificação**

É onde acontece a definição das funcionalidades do software e as restrições em seu funcionamento. Envolve:

* ***Engenharia de Sistema:***estabelece uma solução geral para o problema, envolvendo questões extra software;
* ***Análise de Requisitos:***levantamento das necessidades do software produzindo um documento com a especificação de requisitos;
* ***Especificação de Sistema:***descrição funcional do sistema, incluindo um plano de testes para verificar adequação.

**2. Projeto**

O planejamento do software acontece neste etapa, que envolve:

* ***Projeto Arquitetural:***desenvolve um modelo conceitual, composto de módulos mais ou menos independentes;
* ***Projeto de Interface:***onde cada módulo tem sua interface de comunicação estudada e definida;
* ***Projeto Detalhado:***onde os módulos em si são definidos, e possivelmente traduzidos para pseudocódigo.

**3. Implementação e Validação**

Nesta etapa o software deve ser produzido de modo que cumpra sua especificação. É onde acontece a chamada “programação” ou “codificação”.

Após a programação, o código precisa ser validado para garantir que esteja, de fato, fazendo o que o cliente solicitou. Envolve:

* ***Teste de Unidade e de Módulo:***são feitos testes para verificar a presença de erros e comportamento adequado a nível das funções e módulos básicos do sistema;
* ***Integração:*** a união de diferentes módulos implementados em um único produto e os testes da interação entre eles quando operando em conjunto.

**4. Manutenção e Evolução**

Esta etapa é muito importante, pois o software precisa evoluir para atender a novas necessidades ou mudanças solicitadas pelo cliente. O software em geral entra em um ciclo iterativo que abrange todas as fases anteriores.



**Modelos de processo**

O jargão “cada caso é um caso” é bem adequado aqui, pois para cada tipo de projeto, um *Modelo de Processo* será o mais adequado. Assim sendo, não existe o “*modelo melhor de todos*”, mas sim o “***modelo melhor para este projeto***”. Cada modelo possui vantagens e desvantagens que precisam ser consideradas na hora de serem adotados.

A escolha de um modelo deve ser embasada no custo-benefício que ele pode trazer no momento do desenvolvimento. Ou seja:

* Se for um projeto pequeno, **não serão necessárias muitas tratativas de controle**, podendo talvez ser desenvolvido por um ou dois programadores;
* Ao contrário, em projetos maiores, **um controle mais minucioso de variantes será necessário**, por causa da maior tacha de mudanças durante o processo.

Uma escolha acertada do melhor modelo é determinante e pode reduzir imensamente o desperdício de tempo e de dinheiro, aumentando o lucro obtido na entrega do projeto.

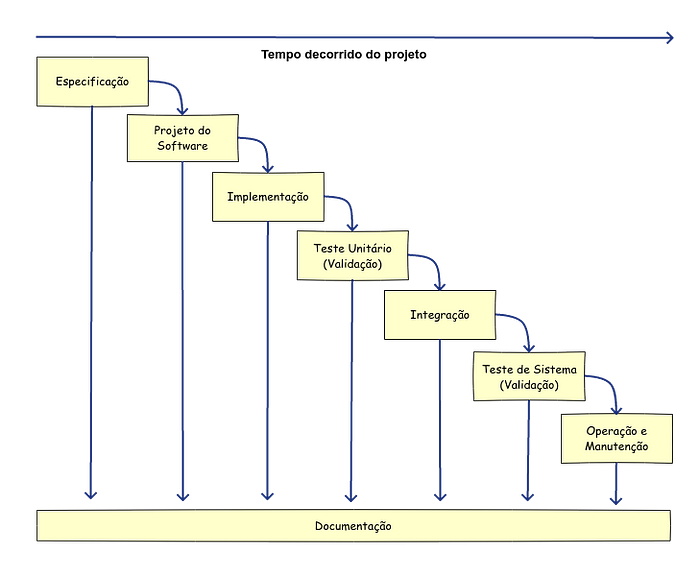
É sempre difícil explicar de maneira concisa os diversos processos de software existentes. Na literatura, alguns autores falam de uns modelos enquanto outros enfatizam pontos de forma diferente, conforme suas próprias experiências. Mas não deixa de ser um assunto importantíssimo e que merece uma atenção especial por parte de empresas que queiram se profissionalizar, entregar mais qualidade em menos tempo e lucrar com isso.

Espero que o conteúdo tenha sido útil até agora. Em artigos posteriores, abordarei os principais modelos de processo de software. Um grande abraço e até a próxima!

# Modelo Cascata

Criando em 1970 por **Winston Walker Royce,**o Modelo em Cascata (também conhecido como Modelo Sequencial Linear) é o mais antigo de todos os processos e possui esse nome devido a sua forma sequencial cascateada que acontece de uma fase para a outra.

Considerado o “modelo clássico de desenvolvimento”, o Modelo em Cascata é “sugerido para projetos pequenos” e sua ideia defende que “para uma fase iniciar, a anterior deve estar totalmente finalizada” (ENGHOLM, 2010) e que o “resultado de cada etapa é a aprovação de um ou mais documentos ‘assinados’” (SOMMERVILLE, 2011).



No modelo cascata basicamente, o desenvolvimento é dividido em cinco etapas, que começam assim que a anterior termina:

1. **Especificação:** é feita a análise e captação das necessidades do cliente e definidos os “requisitos do sistema” com suas funcionalidades;
2. **Projeto do software:** é elaborado o desenho da arquitetura geral do sistema, seguindo as informações coletadas na etapa anterior. Tudo é separado em partes chamadas de “unidades de programa” para poder agilizar o desenvolvimento. Cada unidade terá seu próprio “requisito de unidade”;
3. **Implementação:** nesta etapa, com o projeto definido, cada “unidade de programa” é implementada, podendo alocar vários programadores simultaneamente.
4. **Teste unitário:** no final da implementação, são feitos os “testes de unidade” para certificar (validação) que as implementações atendem aos “requisitos de unidade”;
5. **Integração:** com as unidades testadas, a próxima tarefa é integrar todas elas para compor um sistema completo.
6. **Teste de Sistema:** no final da integração, é feito um teste geral para certificar (validação) que os “requisitos do sistema” foram completamente atendidos. Por fim, o software é entregue ao cliente;
7. **Operação e manutenção:**Ao operar, ou seja, utilizar o sistema, o cliente poderá encontrar erros ou necessidades que não foram identificadas anteriormente. Esta fase se responsabiliza pela correção dos erros ou melhorias do sistema (evolução).

## Vantagens

* Por causa de sua natureza **separada em unidades**, o modelo em cascata possibilita a implementação concorrente entre vários programadores, agilizando a entrega;
* É útil quando o pessoal envolvido no projeto é fraco tecnicamente. (McCONNEL, 1996);
* O modelo apresenta uma grande vantagem quando o escopo do trabalho é claramente definido. Se as especificações estiverem corretas (o que é muito difícil em projetos de médio e grande porte), um software pode ser desenvolvido de forma muito rápida.

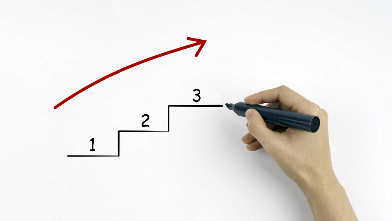
## Desvantagens

* O próprio criador do método (Royce) afirmou que, se a especificação não for bem definida, ele pode ser tornar um risco e um convite para falhas (TORRES, 2014). Afirmou também que o modelo tratava-se era um conceito inicial e ainda defeituoso;
* Como o modelo não possui flexibilidade, seguindo um sequencia engessada, é considerado um modelo frágil, principalmente nos dias de hoje que os projetos mudam muito no decorrer do desenvolvimento;
* Também é difícil estabelecer explicitamente todos os requisitos logo o início do projeto, pois no começo, sempre existe uma incerteza natural;
* O cliente deve ter muita paciência, pois uma versão executável do software só fica disponível numa etapa avançada do desenvolvimento;
* Projetos grandes raramente seguem a sequencia proposta pelo modelo. A produção e aprovação de documentos durante as etapas costumam se caras e geradoras de muito retrabalho;
* Durante o processo, é comum o cliente solicitar o congelamento de partes do desenvolvimento para dar continuidade aos estágios posteriores, tornando a solução dos problemas mais difícil, pois acaba sendo programada para mais tarde e, dependendo da situação, acaba sendo parcial ou totalmente ignorada;
* O congelamento prematuro de partes da implementação pode resultar em um sistema que não faça o que o usuário quer. Também pode “levar a sistemas mal estruturados, quando os problemas de projeto são contornados por artifícios de implementação” (SOMMERVILLE, 2011);
* EVANS (2017), defensor de uma linguagem de arquitetura que reflita as regras de negócio do cliente, afirma que no Modelo em Cascata, os especialistas de um negócio conversam com os analistas, […] que passam o resultado para os programadores […] No entanto, essa abordagem falha porque lhe falta feedback. Em outra palavras, o software resultante pode atender às necessidades do cliente, mas o código não reflete as regras de negócio, uma vez que os programadores “só programam”, não sendo possível identificar os termos de negócio olhando no código;
* A natureza linear do modelo leva a “estados de bloqueio”, nos quais alguns membros da equipe de projeto precisam esperar que outros membros completem as tarefas dependentes (PRESSMAN, 2006);

Segundo ENGHOLM (2010), as desvantagens podem ser resumidas aos seguintes itens:

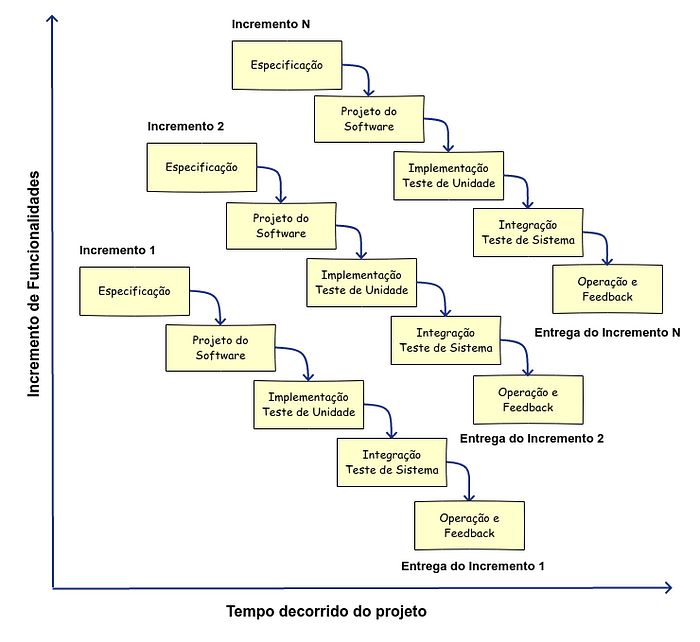
* É muito difícil retornar para as fases anteriores para corrigir problemas detectados posteriormente;
* Muito tempo é gasto para garantir que as fases sejam executadas corretamente;
* Demora muito para o cliente ver algum resultado;
* O cliente se decepciona quando recebe o produto;
* A equipe de desenvolvimento está sempre “quase acabando”;
* Esconde os riscos por muito tempo, retardando sua resolução;

Resumindo, o Modelo em Cascata “não foi elaborado para lidar com mudanças” (ENGHOLM, 2010).

**O Modelo Incremental**

CadaO Modelo Incremental surge como uma melhoria do Modelo em Cascata. Ao invés de especificar e desenvolver tudo de uma só vez, este modelo trabalha com incrementos, ou seja, pequenos pedaços de software entregues de cada vez. Este modelo combina elementos do Modelo em Cascata aplicados de maneira iterativa, ou seja, de forma que o progresso aconteça através de sucessivos refinamentos, melhorados a cada iteração. (PRESSMAN, 2006).

Cada pedaço (incremento) é desenvolvido de forma linear, como no Modelo em Cascata, e em seguida exposto *aos comentários dos clientes*(SOMMERVILLE, 2011). Caso seja necessário alterar algo nessa implementação, é desenvolvido um novo incremento e o resultado é novamente apresentado.



**Modelo incremental:**

O primeiro incremento é frequentemente chamado de “núcleo do produto” (PRESSMAN, 2006) e contém a implementação dos requisitos básicos para que o sistema possa funcionar e atender minimamente as necessidades do cliente.

Cada aprimoramento é lançado como uma versão. Novas versões são criadas até que o sistema fique completo e adequado, para então, ser lançada a versão final.

Todavia, diferente do *Modelo em Cascata*, onde cada etapa tem sua vez para acontecer e, ao término de todas, o projeto termina, no *Modelo Incremental* as atividades de Especificação, Projeto, Implementação e Validação são intercaladas, acontecendo **em cada nova versão**, com rápido feedback entre todas as atividades (SOMMERVILLE, 2011).

Os clientes podem estabelecer as prioridades das partes do sistema a serem desenvolvidas, especificando as mais úteis primeiro. Após o cliente estabelecer as funcionalidades necessárias, são criados os estágios (incrementos) de entrega, onde cada estágio fornece um conjunto de funcionalidades do sistema.

**Vantagens**

* É particularmente útil quando não há mão-de-obra disponível para uma implementação completa, dentro do prazo comercial de entrega estabelecido pelo projeto (PRESSMAN, 2006);
* O cliente não precisa receber todo o sistema para poder usá-lo. Como a implementação é feita por pedaços ordenados por prioridades, o cliente pode ter acesso às partes mais importantes antes, podendo utilizá-las imediatamente;
* *“O custo de acomodar as mudanças nos requisitos do cliente é reduzido”.* (SOMMERVILLE, 2011) A quantidade de análise e documentação a ser refeita é muito menor do que o necessário no modelo em cascata.
* É mais fácil obter feedback dos clientes sobre o desenvolvimento que foi feito. Os clientes podem fazer comentários sobre as demonstrações do software e ver o quanto foi implementado. Os clientes têm dificuldade em avaliar a evolução por meio de documentos de projeto de software.
* Para PRESSMAN (2006) e SOMMERVILLE (2011), é possível obter entrega e implementação rápida de um software útil ao cliente, mesmo se todas as funcionalidades não forem incluídas. Os clientes podem usar e obter ganhos a partir do software inicial antes do que seria possível com um processo em cascata.

**Desvantagens**

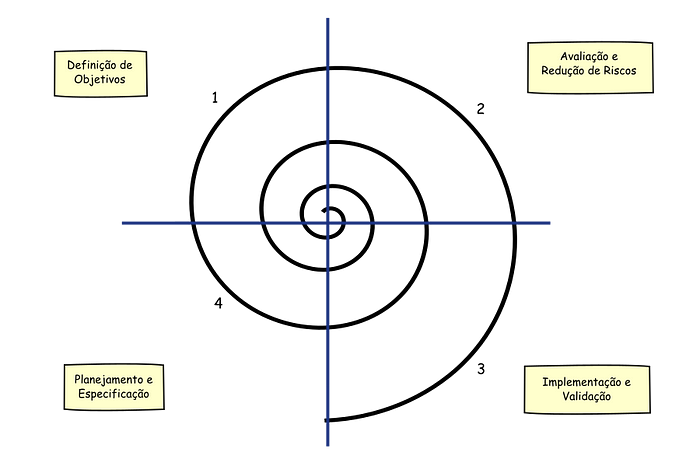
* Os problemas são particularmente críticos para os sistemas grandes, complexos e com vida-longa, onde várias equipes desenvolvem diferentes partes do sistema;
* Sistemas de grande porte necessitam de um framework ou arquitetura estável, e as responsabilidades das diferentes equipes de trabalho do sistema precisam ser claramente definidas, respeitando essa arquitetura. Segundo SOMMERVILLE (2011), essa parte deve ser planejada com antecedência, e **não pode ser desenvolvida de forma incremental;**

SOMMERVILLE (2011) aponta dois problemas no modelo incremental.

* O primeiro problema é que o **progresso não é visível** e os gerentes precisam de entregas regulares para mensurar o progresso. Se os sistemas forem desenvolvidos com rapidez, não será economicamente viável produzir documentos que reflitam cada uma das versões do sistema;
* O segundo problema é que **a estrutura do sistema tende a se degradar com a adição dos novos incrementos**. Tempo e dinheiro devem ser direcionados para refatoração e melhorias do software, pois as constantes mudanças sem previsão do futuro tendem a corromper sua estrutura. A incorporação de mudanças do software torna-se cada vez mais difícil e custosa.

# O Modelo em Espiral de Boehm

Criado por Barry Boehm em 1988, o Modelo em Espiral é uma melhoria do Modelo Incremental e possui esse nome por causa de sua representação, onde cada volta no espiral percorre todas as fases do processo de software. As voltas devem ser repetidas quantas vezes forem necessárias até que o software possa ser completamente entregue.



O modelo em espiral e as fases do processo

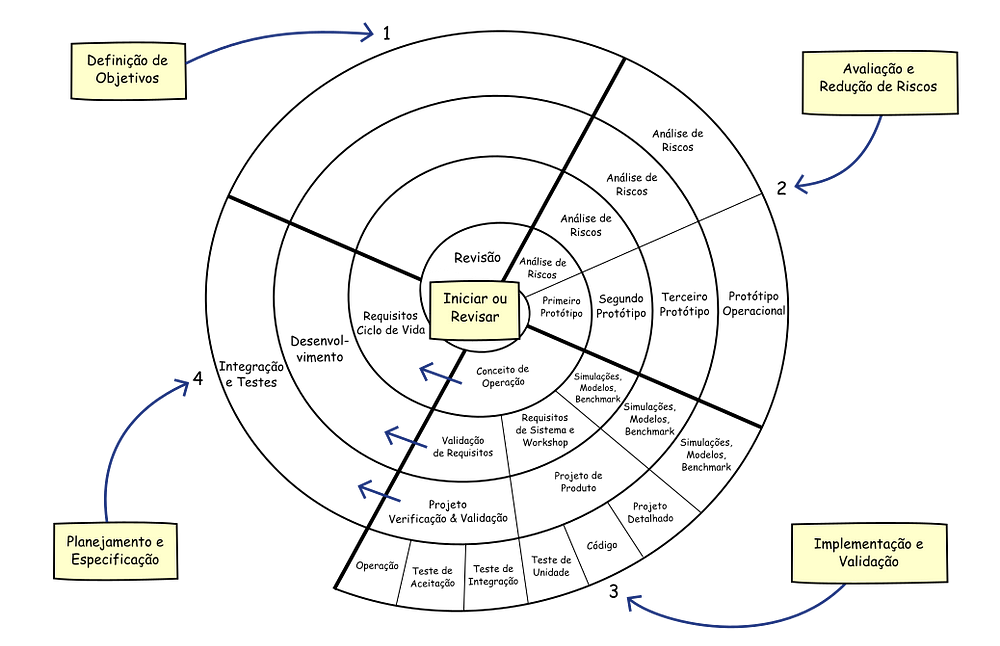
É um processo evolucionário, ou seja, adequado para softwares que precisam passar por inúmeras evoluções na medida que o desenvolvimento acontece.

Diferente do Modelo Incremental, que entrega partes prontas uma de cada vez, o Modelo Espiral é mais iterativo e tenta fazer sucessivos refinamentos. Outras novidades são os novos conceitos de **Prototipagem e Gerenciamento de Riscos**.

Segundo SOMMERVILLE (2011), o Modelo em Espiral “combina prevenção e tolerância a mudanças, assume que mudanças são um resultado de riscos de projeto e inclui atividades explícitas de gerenciamento de riscos para sua redução”.

PRESSMAN (2006) também diz que o modelo é “uma abordagem realista do desenvolvimento de sistemas e softwares de grande porte … usando a prototipagem como mecanismo de redução de riscos”.

No espiral que representa o modelo, a volta mais interna pode preocupar-se com a viabilidade do sistema; o ciclo seguinte, com definição de requisitos; o seguinte, com o projeto do sistema, e assim por diante (SOMMERVILLE, 2011).



O modelo em espiral contendo o processo inteiro

Na representação o desenvolvimento é dividido em quatro setores (SOMMERVILLE, 2011):

1. **Definição de objetivos:** onde são definidos os objetivos para essa fase do projeto, identificando as restrições e preparando um plano de gerenciamento detalhado que inclui todos os possíveis riscos do projeto;
2. **Avaliação e redução de riscos:** para cada risco identificado é feita uma “Análise de Risco” detalhada com o objetivo de identificar estratégias para reduzi-lo ou evitá-lo. Por exemplo, caso exista uma dificuldade em especificar claramente um requisito, isso significa que existe um “risco de requisitos inadequados” e para amenizá-lo será preciso desenvolver um protótipo para apresentar ao cliente a fim de colher sugestões para refinar os requisitos. (Um protótipo é uma simulação do sistema, podendo ser desenhado em papel ou programado de forma simples e não funcional para que o cliente tenha uma ideia melhor do que o engenheiro entendeu de seus requisitos. Mais informações sobre prototipagem será fornecida em um artigo exclusivo sobre Interface Humano-Computador);
3. **Implementação e validação:** com as estratégias definidas, é escolhido um modelo de desenvolvimento, como por exemplo, o “Modelo em Cascata”, “Modelo Incremental”, etc. Pode-se utilizar modelos diferentes em cada volta de implementação, conforme a necessidade;
4. **Planejamento e Especificação:** o projeto todo é analisado para verificar o que foi realizado e planejar quais serão os próximos passos para iniciar novas voltas do espiral ou concluir o sistema.

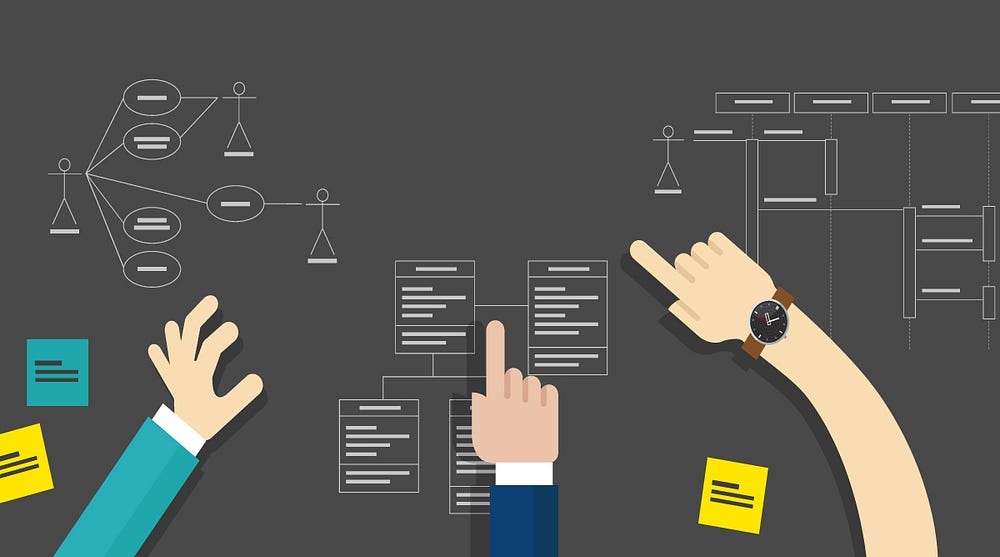
## Vantagens

* Como o modelo exige a consideração dos riscos técnicos em todos os estágios de evolução, se for aplicado adequadamente, reduzirá os riscos antes que se tornem problemáticos;
* As estimativas se tornam mais realistas e o tempo de implementação é reduzido;
* Mais versátil para testar e lidar com mudanças;
* Não faz distinção entre desenvolvimento e manutenção.

## Desvantagens

* Segundo PRESSMAN (2006), pode ser difícil convencer os clientes que o processo de evolução é controlável, pois ele exige competência considerável na avaliação dos riscos e depende dessa competência para ser bem sucedido;
* Se um risco importante não for descoberto e gerenciado corretamente, fatalmente ocorrerão problemas;
* A avaliação dos riscos exige um analista com experiência;
* Aplica-se melhor a sistemas de grande porte;
* Erros na avaliação de riscos podem impactar o projeto.

# O Processo Unificado



O Processo Unificado (Unified Process) foi criado em 1995 por Jim Rumbaugh, Grady Booch e Ivar Jacobson para suportar o paradigma de Orientação a Objetos. Os “três amigos”, como passaram a ser chamados, faziam parte da Rational Software Corporation, onde concluíram a elaboração da UML (Unified Modeling Language) e a lançaram em 1996 (GUEDES, 2018).

Em 2003, a IBM adquiriu o **Rational Unified Process ou RUP**, como passou a ser chamada a versão melhorada do Processo Unificado, que fornecia “uma forma sistemática para se obter vantagens no uso da UML” (KRUTCHEN, 2003).

A principal novidade é o acréscimo da UML para modelagem do sistema.

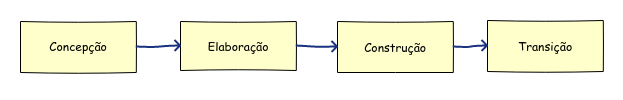
Segundo PEREIRA (2006), esta metodologia “objetiva realizar um maior controle sobre os resultados obtidos, gerenciar mudanças e fomentar um produto de qualidade e estável”.

Segundo KROLL e KRUNCHTEN (2003, p34), conforme citado por PEREIRA (2006), os princípios essenciais do RUP são: “atacar os riscos cedo e continuamente, entregar algo de valor ao cliente, focar em um software que possa ser utilizado o quanto antes, realizar mudanças cedo, liberar protótipos da aplicação, utilizar componentes reutilizáveis, trabalhar como um time e fazer da qualidade um estilo de vida”.

Os criadores do processo discutem sua necessidade (JACOBSON, 1999):

*Hoje, a tendência em software é em direção a sistemas maiores, mais complexos. Isso se deve, em parte, ao fato de que os computadores têm se tornado mais potentes a cada ano, levando os usuários a esperar mais deles. Essa tendência tem também sido influenciada pelo uso da internet, que está se expandindo, para trocar toda espécie de informação… Nosso apetite por softwares cada vez mais sofisticados cresce à medida que aprendemos de uma versão de produto para a seguinte como o produto poderia ser aperfeiçoado. Desejamos softwares que sejam melhor adaptados às nossas necessidades, mas que, por sua vez, não torne o software somente mais complexo. Em resumo, desejamos mais.*

O RUP é uma tentativa de “unificar” o melhor de todos os modelos convencionais (por isso o nome Processo Unificado) de forma que possam se adequar ao Desenvolvimento Ágil de software. Como os outros métodos não possuíam uma maneira satisfatória para lidar com o paradigma de Orientação a Objetos, o RUP introduziu a UML, uma notação gráfica para especificação de objetos capaz de demonstrar visualmente as funcionalidades e os fluxos de um software. Segundo PRESSMAN (2006), “o Processo Unificado e a UML são amplamente usados em projetos de OO de todas as naturezas”.



As fases do RUP

## Características

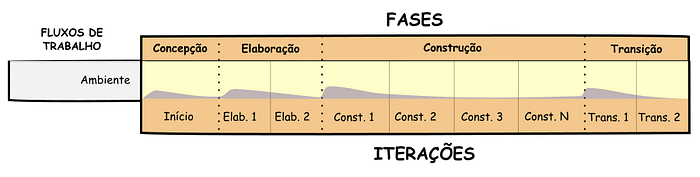
O modelo possui as seguintes características:

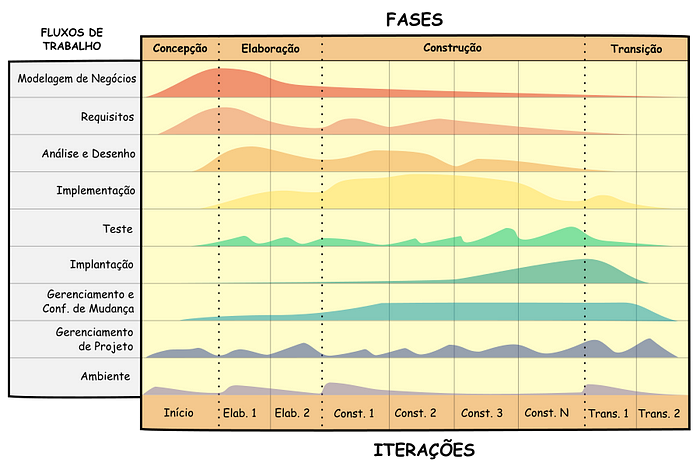
* **Iterativo e incremental:** os fases de Elaboração, Construção e Transição são divididas em uma série de interações. Em projetos grandes, a fase de Concepção também pode ser dividida em iterações. Cada iteração resulta em um incremento, que é uma versão do sistema contendo funcionalidades adicionais ou melhoradas em comparação com a versão anterior. Essa divisão de incrementos deve ser feita no inicio do projeto, definindo todas antes de iniciar a programação;
* **Dirigido por Casos de Uso:** na captura de requisitos, eles são refinados e desenhados na forma de Casos de Uso da UML. Cada iteração tem um conjunto de casos de uso ou cenários de requisitos durante todo o tempo de implementação, teste e desenvolvimento;
* **Centrado na Arquitetura:** o RUP defende que, para dar forma ao sistema, a Arquitetura deve estar no centro dos esforços da equipe do projeto. Uma vez que não existe um modelo único suficiente para cobrir todos os aspectos do sistema, o RUP suporta múltiplas visões e modelos arquiteturais.
* **Focado no Risco:** o modelo requer que a equipe do projeto concentre-se em enfrentar os riscos mais críticos no início do ciclo de vida do projeto. As entregas de cada iteração, especialmente na fase de Elaboração, devem ser selecionadas de forma a garantir que os maiores riscos sejam tratados em primeiro lugar.

## As fases do processo

O modelo se divide em quatro fases (GUEDES, 2018):

* Concepção
* Elaboração
* Construção
* Transição



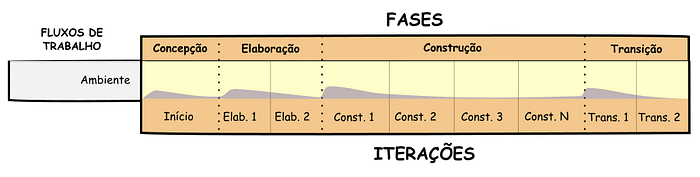


O processo unificado

* **Concepção:** objetivo da fase é fazer a elicitação de requisitos inicial e determinar a viabilidade de se desenvolver o sistema (GUEDES, 2018). São identificadas todas as entidades externas (pessoas ou sistemas) que vão interagir com a aplicação. Com essas informações, avalia-se a viabilidade do sistema para o negócio e, se for pequena, o projeto pode ser cancelado aqui;
* **Elaboração:** nesta fase a maioria dos Casos de Uso são elaborados e especificados. A arquitetura do sistema é projetada, gerando diversos artefatos (Documentação, Diagramas, Planilhas, etc) e uma “baseline” completa é apresentada incluindo os componentes estruturados para posterior formação da equipe de desenvolvimento. No final dessa fase os envolvidos devem estar aptos a planejar a fase de construção em detalhes;
* **Construção:**esta fase envolve a implementação do software e seus testes (GUEDES, 2018). É onde utilização dos vários artefatos possibilita que o sistema seja implementado quase completamente. Tem-se uma visão geral de como o “baseline” do projeto está sendo seguido. Na conclusão dessa fase, você deve ter um sistema de software já funcionando, bem como a documentação associada pronta para ser entregue aos usuários;
* **Transição:** nesta fase o software será implantado garantindo que todos os requisitos do projeto foram atendidos e implementados corretamente. O produto final pode ser liberado em uma versão beta. Outras atividades desta fase, dependendo do projeto, podem ser os testes no ambiente real (onde o software irá funcionar), a conclusão do manual do usuário, a identificação e correção de defeitos, etc. No final, deve-se tirar uma conclusão geral do projeto, levantando e documentando os pontos positivos e negativos para serem ser utilizados nas decisões em projetos futuros.

Essas fases servem de apoio para a execução das tarefas de cada um dos Fluxos de Trabalho, sendo seis da**Engenharia de Software** e de três para**Apoio e Suporte**.

* **Modelagem de negócios (ES)**: os processos de negócio são modelados por meio de Casos de Uso de negócios (SOMMERVILLE, 2011). O objetivo principal é que o analista entenda muito bem o problema a ser resolvido, elaborando se necessário uma análise de risco e de viabilidade para o projeto como um todo. É preciso uma grande interação entre o analista e o cliente, para que seja possível a criação dos Casos de Uso e consequentemente a extração dos requisitos. Entender o modelo de negócio do cliente é fundamental antes que um requisito possa ser definido;
* **Requisitos (ES)**: nesse fluxo de trabalho, procura-se extrair os requisitos de forma que o cliente possa entender claramente a proposta do sistema. O alicerce é que o analista entenda o domínio do problema e consequentemente construa um bom modelo de Caso de Uso. A extração dos requisitos a partir de um Caso de Uso gerará um artefato que será evoluído durante todo o projeto;
* **Análise e Desenho (ES)**: um modelo de projeto será criado e documentado em artefatos, como Descrição da Arquitetura Básica do sistema, Protótipos de Funcionalidade e Interface, Diagrama de Classes, Diagrama de Estado, Diagrama de Iteração, Diagrama de Seqüência, etc (SOMMERVILLE, 2011). Durante o desenvolvimento do projeto alguns artefatos poderão sofrer ajustes de acordo com as implementações realizadas. O objetivo aqui é compreender os Casos de Uso mais importantes, que serão úteis para a elaboração dos artefatos necessários, lembrando que não é necessária a utilização de todos os artefatos, mas apenas aqueles que sejam relevantes para o cliente entender perfeitamente o que será construído. Com artefatos bem elaborados, a equipe de desenvolvimento terá grande facilidade em realizar a implementação;
* **Implementação (ES)**: os desenvolvedores poderão fazer uso de componentes (funções) que foram utilizados em outro sistema ou projeto. Ainda na fase de Concepção, pode-se ter um protótipo de funcionalidade. No decorrer deste fluxo de trabalho, tenta-se gerar um sistema executável a cada iteração, além da implementação baseada nos artefatos criados no fluxo de Análise e Desenho. O conceito de componentes deve ser sempre consideração, para que seja possível aproveitar esses “pedaços de código” em outros projetos;
* **Testes (ES)**: um plano de testes é elaborado, definindo quais tipos de testes serão realizados. Esse plano poderá ser alterado de acordo com as melhorias ou alterações nos requisitos do sistema, impactando no número de testes a serem realizados. Nas fases de Concepção e Elaboração são feitos os testes de módulos e na fase de Construção, os testes de integração;
* **Implantação (ES)**: neste fluxo de trabalho, um release do produto será criado, distribuído aos usuários e instalado no ambiente do cliente, ou seja, em seu local de trabalho (SOMMERVILLE, 2011). Durante toda a fase de Elaboração, até o meio da fase de Construção, poderá ser criado um documento especificando, de forma simples, algumas características do ambiente do cliente, contendo especificações técnicas sobre a infra-estrutura de rede, os sistemas operacionais, sistemas integrados, etc. Também é interessante adicionar algumas dicas de instalação para reduzir, no futuro, os erros de instalação e o tempo de testes. No final da fase de Construção, começa a migração do sistema para o “**ambiente de testes**” do cliente e posteriormente, na fase de Transição, o sistema é configurado no “**ambiente de produção**” do cliente.
* **Gerência de configuração e mudança (AS)**: nesse fluxo de trabalho são controlados todos os artefatos do projeto e suas versões. Antes de realizar uma mudança corretiva ou evolutiva, deve-se fazer uma análise sobre o que deve ser modificado e quais artefatos serão afetados. **Um bom controle de mudança é crucial para garantir o sucesso e a qualidade do projeto**. Conforme o projeto entra na fase de construção, aumenta-se a dificuldade em controlar as mudanças e gerenciar a configuração. Quanto maior o projeto se torna, com mais requisitos implementados, maior será a chance de uma alteração afetar outras áreas do sistema. Por isso, saber rastrear e relacionar requisitos é uma importante tarefa do engenheiro de software. Após uma modificação, é necessário executar novos testes em várias áreas do sistema, garantindo que a mudança foi implementada corretamente e que nada foi quebrado. Igualmente importante é a atualização da documentação para que reflita perfeitamente o que foi implementado;
* **Gerenciamento de projeto (AS)**: aqui se escolhe os artefatos a serem utilizados no desenvolvimento, de acordo com o tipo do projeto e o entendimento do cliente. O gerente de projeto deve ter uma visão clara do que o cliente deseja, do que está documentado e do que está sendo implementado. A atividade de gerenciamento de projeto é constante durante todo o ciclo de vida do software, onde se deve: a) elaborar reuniões para entrega de versões; b) criar documentos de RTF (Revisão Técnica Formal) para cada reunião; c) garantir a correta mudança dos artefatos; d) manter um bom relacionamento com o cliente;
* **Ambiente (AS)**: Esse fluxo de trabalho está relacionado com a disponibilização de ferramentas apropriadas para a equipe de desenvolvimento de software (SOMMERVILLE, 2011). Essa disponibilização pode ser referente ao tipo de plataforma usada, a velocidade da internet e da rede, a organização dos diretórios/repositórios onde serão armazenados os artefatos e os códigos fonte, o sistema de backup etc. No final de cada fase podem acontecer ajustes no ambiente, podendo ser: criação de diretórios, backup de versões do software, etc.



## Os artefatos gerados no processo

Durante os “fluxos de trabalho”, podem ser produzidos vários “produtos de trabalho”, também chamados de “artefatos”, contendo informações sobre todo o processo. Segundo PRESSMAN (2006), “do ponto de vista dos Engenheiros de Software, o produto de trabalho mais importante, produzido durante a concepção, é o Modelo de Casos de Uso, uma coleção de Casos de Uso que descreve como atores externos (‘usuários’ humanos e não-humanos do software) interagem com o sistema e obtém valor dele”.

A seguir, uma possível lista de artefatos como propõe PRESSMAN (2006):

## Os artefatos gerados na 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 (se necessário);
* Um ou mais protótipos

## Os artefatos gerados na 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.

## Os artefatos gerados na 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.

## Os artefatos gerados na Transição

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

## Vantagens

* É um dos modelos mais usados em projetos modernos de desenvolvimento de software;
* É extremamente robusto e bem estruturado;
* Por fazer uso da UML, a facilidade de compreensão dos requisitos aumenta, tanto para os clientes, como para os analistas e desenvolvedores;
* Os riscos que poderiam ser mais preocupantes são resolvidos no inicio, minimizando o retrabalho e, consequentemente, a chance de fracasso do projeto.

## Desvantagens

* Exige uma equipe experiente e alinhada com os processos;
* Exigem um bom gerente de projetos;
* Complexo e trabalhoso para projetos pequenos.

# Modelo Ágil

Os principais modelos ágeis incluem: Scrum, Kanban, Lean, XP (eXtreme Programming) e FDD (Feature Driven Development).

A metodologia ágil é um modelo de gestão de projetos que prioriza a flexibilidade e a colaboração, com foco em entregas rápidas e frequentes que atendam às necessidades do cliente. Em vez de seguir um plano detalhado e rígido, o modelo ágil permite adaptações e melhorias contínuas ao longo do projeto.

**Principais características da metodologia ágil:**

**Flexibilidade:** A metodologia ágil permite que as equipes se adaptem às mudanças e aos novos desafios que surgem durante o projeto.

**Colaboração:** A comunicação e a colaboração entre os membros da equipe, o cliente e outros stakeholders são essenciais para o sucesso de um projeto ágil.

**Entrega contínua:** O modelo ágil busca entregar resultados de forma rápida e frequente, permitindo que o cliente veja o progresso do projeto e possa fornecer feedback.

**Melhoria contínua:** A metodologia ágil incentiva a reflexão e a melhoria contínua do processo, buscando identificar oportunidades de aprimorar a qualidade e a eficiência do projeto.

**Benefícios da metodologia ágil:**

**Redução de riscos:** A entrega contínua e a colaboração com o cliente ajudam a identificar e corrigir erros mais rapidamente, reduzindo o risco de falhas no projeto.

**Aumento da produtividade:** A flexibilidade e a eficiência do modelo ágil permitem que as equipes entreguem mais valor ao cliente em menos tempo.

**Maior satisfação do cliente:** A entrega contínua e a colaboração com o cliente garantem que o projeto atenda às suas necessidades e expectativas.

**Melhora da qualidade:** A melhoria contínua e a colaboração com o cliente ajudam a garantir que o produto ou serviço final seja de alta qualidade.

**Exemplos de metodologias ágeis:**

**Scrum** é um framework popular para gerenciamento de projetos ágeis que é amplamente utilizado em desenvolvimento de software. Ele se baseia em ciclos de desenvolvimento curtos, chamados sprints, e enfatiza a colaboração e comunicação constante entre a equipe.

**Kanban** é outro modelo ágil que se concentra em melhorar o fluxo de trabalho e aumentar a eficiência. Ele utiliza um quadro visual para visualizar o progresso do projeto e gerenciar tarefas.

Lean é uma abordagem baseada em princípios de produção enxuta, que busca eliminar desperdícios e aumentar a eficiência. Ele é comumente usado em indústrias de fabricação, mas também pode ser aplicado a outros setores.

**XP** (eXtreme Programming) é um modelo ágil que enfatiza a programação em equipe, a comunicação constante e a simplicidade. Ele é amplamente utilizado em projetos de desenvolvimento de software.

**FDD** (Feature Driven Development) é um modelo ágil que se concentra em desenvolver recursos de maneira incremental e iterativa. Ele é amplamente utilizado em projetos de desenvolvimento de software e engenharia de sistemas.

**Referências:**

PRESSMAN, Roger. S. Engenharia de Software, 6ª Edição. McGrawHill, Nova York, EUA, 2006

SOMMERVILLE, Ian. Engenharia de Software, 9ª Edição. Pearson. São Paulo, Brasil, 2011.

Link: https://medium.com/contexto-delimitado/o-modelo-incremental-b41fc06cac04

Link: https://medium.com/contexto-delimitado/os-processos-de-software-56a2e70fddfb

Link: https://medium.com/contexto-delimitado/o-processo-unificado-d102b1fc9d00

**Referências para aprofundamento**

ENGHOLM, Hélio. Engenharia de Software na Prática. Novatec Editora. São Paulo, Brasil, 2010.

EVANS, Eric. Domain-Driven Design Atacanto as Complexidades no Coração do Software. Altabooks Editora. Rio de Janeiro, Brasil, 2017.

McCONNEL, Steve. Rapid development. Redmond, WA: Microsoft Press, 1996.

PRESSMAN, Roger. S. Engenharia de Software, 6ª Edição. McGrawHill, Nova York, EUA, 2006

SOMMERVILLE, Ian. Engenharia de Software, 9ª Edição. Pearson. São Paulo, Brasil, 2011.

TORRES, Luis Fernando (2014). Fundamentos de gerenciamento de processos. Rio de Janeiro: Elsevier.