**2 –** Fatores: projeto e programação estruturada, modularização de software, acoplamento e coesão. Os anos 70 foram os anos da *síntese* e da *antítese* dos Processos de Formalidade e do Waterfall (cascata), ouve a adoção deste modelo dentro da formalização de processos. Houve adoção de fases ao invés de “codificar e corrigir”. As principais reações ao enfoque do codifique e resolva envolveram processos em que a codificação era mais cuidadosamente organizada e era precedida pelo projeto, e o projeto era precedido pela engenharia dos requisitos.  Este movimento fez emergir dois campos temáticos: a) um foi o dos “métodos formais”, que focava na corretude dos programas, seja por prova matemática, ou por construção via cálculo de programação; e. b) o outro, menos formal, misturava métodos técnicos com gerenciais, “estrutura de programação top-down com times de programação chefiados. O sucesso da programação estruturada levou a muitas outras abordagens "estruturadas" aplicadas ao design de software. Princípios de modularidade foram reforçadas pelos conceitos de acoplamento de (a minimizar entre módulos) e a coesão (a maximizar dentro de módulos), por técnicas cada vez mais fortes de formas de esconder informações, e por tipos de dados abstratos. Conceitos sobre iteração e incrementos foram adicionados.

Este movimento tinha dois ramos primários. Um deles era um Métodos "focada na correção do programa, seja por Matemática, ou por construção através de uma "programação de

Cálculo. O outro ramo era uma combinação menos formal de e métodos de gestão, "programação estruturada top-down com equipes de programadores-chefe". O sucesso da programação estruturada levou a muitos outros abordagens "estruturadas" aplicadas ao design de software. Princípios de Modularidade foram reforçadas pelos conceitos de acoplamento de (A minimizar entre módulos) e a coesão (a maximizar Dentro de módulos), por técnicas cada vez mais fortes de formas de esconder, encapsular informações, e por tipos de dados abstratos. O sucesso da programação estruturada levou a muitos outros

Abordagens "estruturadas" aplicadas ao design de software. Uma série de ferramentas e métodos de conceitos estruturados foram desenvolvidos, tais como o projeto e a programação estruturados de Jackson, enfatizando considerações de dados, e programa estruturado em

linguagem.

3 - A procura da produtividade foi o marco de engenharia de software na década de 80, a mesma teve uma série de iniciativas para resolver os problemas de 1970. O aumento dos métodos quantitativos no final dos anos 70 ajudou a identificar os principais pontos de alavancagem para melhorar a produtividade do software. Começou a usar a tática de reutilização de código e o reforço do modelo em cascata. As empresas de software focaram em “Produtividade e Escalabilidade”, bem junto a distribuição de esforços e defeitos por fase e atividade ativada - melhor priorização das áreas de melhoria. Modelos de processos mais formais foram propostos e adotados. Ferramentas mais elaboradas para testes (analisadores de trajetórias e de cobertura de teste, geradores automatizados de casos de teste, analisadores de dados de teste, simuladores de teste e auxiliares de teste operacional) e gerenciamento de configuração foram criadas. A utilização de modelos [orientados a objetos](https://pt.wikipedia.org/wiki/Orienta%C3%A7%C3%A3o_a_objetos) começaram a ser utilizados como ferramenta para a reutilização de código. Surgiu também o conceito de [Fábrica de Software](https://pt.wikipedia.org/wiki/F%C3%A1brica_de_software) visando agregar reuso, performance e qualidade ao desenvolvimento de software, tornando o mesmo um processo formal dentro da fábrica.

- Para o Prof. Boehm a *tese* dos anos 50 era que a *Engenharia do Software era como a Engenharia do Hardware*.  Ou seja, naquela época o entendimento prevalecente era:“produza software como você produz hardware”.  A *antítese* veio nos anos 60 (a do *Artesanato do Software*), quando as pessoas descobriram que a fenomenologia do software diferia da fenomenologia do hardware.  Como o software era mais fácil de modificar do que hardware, ele não requeria linhas de produção custosas para fazer cópias de produtos.  Logo, emergiu o enfoque “*codeandfix” (codifique e resolva)* para o desenvolvimento do software.

Os anos 70 foram os anos da *síntese* e da *antítese* dos Processos de Formalidade e do Waterfall.  As principais reações ao enfoque do codifique e resolva envolveram processos em que a codificação era mais cuidadosamente organizada e era precedida pelo projeto, e o projeto era precedido pela engenharia dos requisitos.  Este movimento fez emergir dois campos temáticos: a) um foi o dos “métodos formais”, que focava na corretude dos programas, seja por prova matemática, ou por construção via cálculo de programação; e. b) o outro, menos formal, misturava métodos técnicos com gerenciais, “estrutura de programação de cima para baixo com times de programação chefiados”.

A *síntese* dos anos 80 (com as questões de Produtividade e Escalabilidade) foi representada pelo número de iniciativas desenvolvidas para enfrentar os problemas dos anos 70.  E neste momento o Prof. Boehm apresenta uma figura representativa da linha do tempo da área de engenharia de software dos anos 50 até os atuais.

A *antítese* dos anos 90 (a dos Processos Concomitantes versus Sequenciais) observou um forte momentum dos métodos *orientados-a-objeto*, os quais foram fortalecidos por avanços tais como os padrões de projeto, as arquiteturas de software e as linguagens de descrição de arquiteturas, e o desenvolvimento da UML (**UnifiedModelingLanguage,**uma linguagem de modelagem não proprietária de terceira geração).  A expansão continuada da Internet e a emergência da World-Wide-Web fortaleceram tanto os métodos orientados-a-objeto quanto a importância crítica do software para a competição no mercado.  Algumas questões ganharam ênfase, como o time-to-market, a interatividade com o usuário, o controle da concomitância, o desenvolvimento do open-source, e a usabilidade e a interação home-computador.

A *antítese* e a *síntese* parcial dos anos 2000, na opinião do Prof. Boehm, são a *Agilidade* e o *Valor*.  O rápido passo das mudanças em tecnologias de informação (com Google, suporte colaborativo baseado em Web), e em organizações (fusões, aquisições, startups), causou crescente frustração com plantas pesadas, especificações, e outras documentações impostas pela inércia contratual, bem como cumprimento de critérios de modelos maturidade.  Estes fatos deram margem à emergência de um número de *métodos ágeis* ([1]) bem como às questões de valor (preferências do consumidor e custos) no desenvolvimento de software.

Finalmente, a visão dos anos 2010 e adiante do Prof. Boehm é principalmente permeada pela emergência pelos fenômenos da conectividade global e pela existência dos massivos sistemas (intensivos de software) de sistemas.

**5 –** Um engenheiro de software deve ter como perfil: é proativo, pensa não só no curto prazo, mas também no longo prazo; aceita riscos e críticas (importante para o crescimento profissional); é comunicativo, sabe ouvir e interagir com o cliente. É necessário ser focado, ter liderança, estar alinhado com o negócio. É preciso ser curioso, dinâmico e pragmático, adaptando-se às novas tendências, tecnologias de software, banco de dados e as novas metodologias de desenvolvimento que surgem (qualificação e certificação). Evolui constantemente com a necessidade.

**6 –** Um processo de software é um conjunto de atividades que leva a produção de um produto de software. Essas atividades podem envolver o desenvolvimento de software propriamente dito, usando uma linguagem de programação como Java ou C.  Na definição de um processo de software devem ser consideradas as seguintes informações: atividades a serem realizadas, recursos necessários, artefatos requeridos e produzidos, procedimentos adotados e o modelo de ciclo de vida utilizado.

**7 –** Ainda existindo muitos modelos de processo de desenvolvimento de softwares diferentes, algumas atividades fundamentais são comuns a todos eles, são elas:

* Especificação de software;
* Processo e implementação de software;
* Validação de software;
* Evolução de softwares.

Especificando cada uma temos:

Na fase de especificação de software as funcionalidades do software e as restrições sobre sua operação devem ser definidas. É justamente a fase do levantamento de requisitos onde deve haver o envolvimento do cliente usuário com o desenvolvedor (e falo desenvolvedor não apenas uma só pessoa, mas podendo ser e geralmente é, uma equipe de desenvolvimento) para o entendimento do problema/negócio. Erros nesta fase podem comprometer inteiramente todo o processo de software, independentemente do modelo usado. É considerado o ponto, a etapa mais crítica do desenvolvimento.

Em projeto e implementação o software que atenda à especificação deve ser produzido. A partir da fase de especificação de requisitos o projeto passa para a próxima etapa – a de desenvolvimento. É a etapa justa da codificação. É o processo de conversão, formalmente falando, de uma especificação de um sistema em um sistema executável onde toda a equipe de desenvolvimento ou o desenvolvedor irá satisfazer todos as restrições do domínio de aplicação especificado na etapa anterior juntamente com todos as demais preferencias do cliente.

Em validação o software deve ser validado para garantir que ele faça o que o cliente deseja. É o sistema de testes que predomina nesta etapa. O software é testado inúmeras vezes para se ter a aprovação, seja do cliente ou do processo de desenvolvimento. É dividido em três categorias de testes: teste de componente: as unidades , que são as partes do sistema, são testadas individualmente para se ter um resultado de sucesso/fracasso entre as partes; teste de sistema: todos os componentes são integrados e o teste agora é feito no sistema completo, todas as partes interagindo e intercaladas atuando sobre um só produto; e teste de aceitação ou também, conhecido por teste alfa: é o teste de cliente, é onde se sabe que o levantamento de requisitos foi bem preciso e o cliente aprova todo o sistema.

Evolução: O software deve evoluir para atender às necessidades mutáveis do cliente. Tais mudanças podem ser onerosas. O software deve ser desenvolvido inicialmente com um conceito de flexibilidade. Nesta fase, no software é adicionado outras funções de acordo com os novos requisitos de implementação do cliente. Pode ser necessário reprovar todo o software dependendo do novo requisito, sendo ele de alta criticidade e indispensabilidade.

**8 –** Os principais modelos de software, os considerados genéricos, são:

* Modelo em cascata;
* Desenvolvimento evolucionário;
* Engenharia de software baseado em componentes (modelo de reuso);
* Desenvolvimento formal.

O *modelo em cascata* é um modelo de desenvolvimento de software sequencial no qual o desenvolvimento é visto como uma rota constante para frente através das fases de análise de requisitos, projeto, implementação, testes (validação), integração, e manutenção de software. Uma atividade posterior só pode ser executada quando uma anterior for terminada. É um modelo bem rígido. Quando há alguma deficiência em uma dada etapa do processo do modelo, deve-se retornar a etapa anterior e recomeçar todo o processo. Erros podem ser identificados em cada fase e dependendo do erro o processo é retornável a etapa anterior inicial de todo o planejamento (levantamento de requisitos). As vantagens do modelo em cascata consistem na documentação produzida em cada fase e sua aderência a outros modelos de processo de engenharia. Seu maior problema é a divisão inflexível do projeto em estágios distintos. Esse modelo deve ser usado apenas quando os requisitos forem bem compreendidos e houver pouca probabilidade de mudanças radicais durante o desenvolvimento do sistema.

O *desenvolvimento evolucionário* baseia-se na ideia de desenvolvimento de uma implementação inicial, expondo o resultado dos comentários do usuário e refinando esse resultado por meio de várias versões até que seja desenvolvido um sistema adequado. Neste modelo as atividades de especificação, desenvolvimento e validação são intercaladas, em vez de serem separadas, com feedback rápido que permeia as atividades. É subdividido em dois tipos: desenvolvimento exploratório e prototipação throwaway. Este modelo é ideal para a produção de sistemas que atendam às necessidades imediatas dos clientes. A vantagem de um processo de software baseado na abordagem evolucionária é que a especificação pode ser desenvolvida de forma incremental. À medida que os usuários compreendem melhor seu problema, isso pode ser refletido no sistema de software. No entanto, do ponto de vista da engenharia e do gerenciamento, a abordagem evolucionária tem dois problemas: o processo não é visível e os sistemas são frequentemente mal estruturados. Respectivamente; os gerentes precisam de produtos regulares para medir o progresso, se os sistemas são desenvolvidos rapidamente, não é viável economicamente produzir documentos que reflitam cada versão do sistema. Em se tratando do segundo problema: a mudança contínua tende a corromper a estrutura do software, a incorporação de mudanças de software torna-se cada vez mais difícil e onerosa.

O *modelo de reuso* baseia-se na abordagem da existência de um número significativo de componentes reusáveis. O processo de desenvolvimento do sistema enfoca a integração desses componentes, em vez de desenvolvê-los a partir do zero. Essa abordagem orientada a reuso depende de uma grande base de componentes de software reusáveis e algum framework de integração desses componentes. Algumas vezes, esses componentes são sistemas comerciais independentes COTS ou Comemercial Off-The-Shelf Systems) que podem fornecer funcionalidade específica, como a formação de texto ou um cálculo numérico. Neste tipo d e processo de desenvolvimento tem a vantagem óbvia de reduzir a quantidade de software a ser desenvolvido e, dessa maneira, reduzir os custos e riscos - gerando geralmente uma entrega mais rápida do produto de software. Deve haver uma vasta gama de códigos reutilizáveis que atenda aquelas certas atividades de requisitos (os requisitos são inevitáveis) e deve existir algum controle sobre a evolução do sistema se as novas versões dos componentes reusáveis não estiverem sob controle da organização que as utiliza.

O modelo de processo de engenharia de software de *desenvolvimento formal* compreende um conjunto de atividades que determinam uma especificação matemática para o software. É muito difícil garantir a qualidade de software a partir de inspeções informais, tanto pelo tamanho do software como pela complexidade das linguagens e paradigmas de programação. Métodos Formais são “técnicas baseadas na matemática para descrever propriedades de um sistema”. São usados para a especificação, construção por refinamento e verificação de sistemas de software e hardware, com o objetivo de atingir níveis de qualidade mais elevados e aumentar a confiança no desenvolvimento e na correção do software através de provas formais, refinamentos e testes. Elimina muitos problemas encontrados nos outros modelos: ambiguidade; inconsistência; incompletude. Favorece uma compreensão mais profunda dos requisitos. No entanto, atualmente é muito lento e dispendioso. Exige treinamento extensivo para dar capacitação aos desenvolvedores. Como mencionado no item anterior, falta de profissionais com treinamento necessário.

**09 –** O RUP é uma junção de várias práticas de processos de modelos de engenharia de software com o intuito de desenvolver o software iterativamente, gerenciar os requisitos da melhor forma, usar arquiteturas baseadas em componentes, modelar visualmente o software (UML), etc. É estruturado em 4 fases: iniciação, elaboração, construção e transição. As fases englobam nove disciplinas, divididas em disciplinas do processo e de suporte. As disciplinas de processo são: modelagem de negócios, requisitos, análise e projeto, implementação, teste e implantação. As de suporte são: configuração e gerenciamento de mudanças, gerenciamento de projeto, e ambiente. Explicando o modelo, segundo o gráfico da baleia, em cada fase há a concentração de atividades de uma ou mais disciplinas. Por exemplo, na fase de iniciação há uma maior concentração de trabalho (atividade) das disciplinas de modelagem de negócios e requisitos.

**10 -** Faz uso das melhores práticas de software; é um framework de processo; utiliza UML para especificar, modelar e documentar artefatos; guiado por casos de uso; centrado na arquitetura - baseado em componentes; iterativo e incremental; focado em riscos; permite a customização e autoria de processos, ou seja uma vasta variedade de processos, ou configuração de processos, podem ser obtidas a partir dele; é configurável: pode ser adaptado dependendo do tipo de software sendo desenvolvido, de características do ambiente de desenvolvimento (tamanho da equipe, técnicas usadas, etc.).