UNIDADE 1 – DESENVOLVIMENTO DE SOFTWARE PARA VALORIZAÇÃO DO NEGÓCIO

MÓDULO 1 – INTRODUÇÃO A ENGENHARIA DE SOFTWARE

01

1- INTRODUÇÃO A SOFTWARE

No início da década de 1980, as reportagens que apareciam nas revistas e jornais tratavam os softwares e hardwares como uma nova força propulsora. Era a desconfiança do momento, filmes e revistas tratavam o futuro como uma coisa incerta e com muitas possibilidades. Como exemplo temos o filme "De volta para o futuro". O filme trata a descoberta da máquina do tempo, onde possibilitava a mudança do passado e consequentemente do futuro. Tínhamos outros filmes mais antigos, como "Viagem no túnel do tempo". Todos esses filmes abordavam o uso da tecnologia como a possibilidade de realizar o impossível, soltando a imaginação.

Dava-se essa importância à tecnologia naquela época porque o assunto software estava amadurecendo, e consequentemente as possibilidades de construção de aplicativos também. O software se tornara um tema de preocupação administrativa.

Dava-se essa importância à tecnologia naquela época porque o assunto software estava amadurecendo, e consequentemente as possibilidades de construção de aplicativos também. O software se tornara um tema de preocupação administrativa.

Fonte: http://www.zoomdigital.com.br

02

No início dos anos 1990 surgia então um temor nas empresas acerca do assunto "automatização". Profissionais estavam assustados com a possibilidade de perderem os empregos, o que gerava uma

desconfiança em torno do futuro do software. De fato, algumas profissões foram extintas – como o datilógrafo – inúmeras fábricas que não mudaram seu negócio, acabaram fechando. Hoje não há mais treinamento em máquinas de escrever e estas só são encontradas em casas de colecionadores. Algumas empresas da área, consideradas sólidas na época, foram fechadas. A FACIT, cuja especialização era a construção de máquinas de escrever, foi fundada em 1922 e fechou suas portas em 1998.

Fonte: https://pt.wikipedia.org/wiki/Facit

O que aconteceu na verdade foi uma evolução das profissões e empresas. O datilógrafo virou digitador, hoje o digitador provavelmente nem existe mais, temos então os operadores de sistemas.

O software vem puxando a evolução do hardware. Sendo a chave para o sucesso de muitos sistemas baseados em computador. Seja o computador usado para dirigir um negócio, controlar um produto ou capacitar um sistema, o software é um fator que diferencia. O que diferenciam os sistemas dos outros é a capacidade de ser amigável ao ser humano. Sistemas que fazem a mesma coisa, mas com interfaces diferentes. A inteligência e as funções oferecidas pelos softwares muitas vezes diferenciam dois produtos de consumo ou indústrias idênticas. E o software pode fazer a diferença.

03

Mas, o que é um software? Vejamos algumas definições, segundo dicionários:

Michaelis	Priberam
Qualquer programa ou grupo de programas que instrui o *hardware* sobre a maneira como ele deve executar uma tarefa, inclusive sistemas operacionais, processadores de texto e programas de aplicação; programa que funciona em um computador local e um computador remoto, permitindo que um usuário controle o computador remoto. Conjunto de programas, processos, regras e,	Qualquer programa ou grupo de programas que instrui o hardware sobre a maneira como ele deve executar uma tarefa, inclusive sistemas operacionais, processadores de texto e programas de aplicação; programa que funciona em um computador local e um computador remoto, permitindo que um usuário controle o computador

eventualmente, documentação, relativos ao funcionamento de um conjunto de tratamento de informação (por oposição a hardware).	remoto. Conjunto de programas, processos, regras e, eventualmente, documentação, relativos ao funcionamento de um conjunto de tratamento de informação (por oposição a *hardware*).

Há 20 anos, menos de 1% do público poderia apresentar de forma inteligível o que significa "software". Hoje, a maioria dos profissionais de TI bem como profissionais de outras áreas tem alguma ideia do que seja software. Vale destacar algumas características importantes:

- O software é um elemento de sistema lógico, e não físico, portanto, o software tem características que são consideravelmente diferentes das do hardware.
- O software é desenvolvido e projetado por engenharia, e não construído com peças, como se fosse um carro.
- O software não se desgasta.
- A maioria dos softwares é feita sob medida em vez de ser montada a partir de componentes existentes.

04

O software pode ser aplicado a qualquer situação e momento, desde que esteja previamente especificado com passos procedimentais e se um algoritmo tiver sido definido e construído. O conteúdo de informações (requisitos) e a previsibilidade são fatores importantes na determinação da natureza de um aplicativo.

Desenvolver categorias genéricas para as aplicações de softwares é uma tarefa muito difícil. Quanto mais difícil e complexo é o sistema, mais complicado é determinar de forma clara quais os vários componentes do software serão construídos. Podem-se dividir as aplicações em:

- Software Básico;
- Software de Tempo Real;
- Software Comercial;
- Software Científico e de Engenharia;
- Software Embutido;
- Software de Computador Pessoal;
- Software de Inteligência Artificial.

***Software* Básico**
É um conjunto de programas para dar apoio a outros programas. Tem como característica uma forte interação com o *hardware*, operações concorrentes, compartilhamento de recursos, uso por múltiplos usuários e múltiplas interfaces.

***Software* de Tempo Real**
São programas que monitora, analisa e controla eventos do mundo real, devendo responder aos estímulos do mundo externo com restrições de tempo pré-determinadas.

***Software* Comercial**
É a maior área de aplicação de *softwares*, são aplicações que gerenciam as operações comerciais de modo a facilitar o gerenciamento comercial do negócio da empresa, permitindo também a tomada de decisões.

***Software* Científico e de Engenharia**
São caracterizados por algoritmos de processamento numérico, dependentes da coleta e processamento de dados para as mais variadas áreas do conhecimento.

***Software* Embutido**
São desenvolvidos para executar atividades muito específicas e inseridos em produtos inteligentes tanto para atividades comerciais como para atividades domesticas.

***Software* de Computador Pessoal**
São produtos desenvolvidos para o uso pessoal do computador, tais como planilhas eletrônicas, processadores de textos, jogos etc.

***Software* de Inteligência Artificial**
Faz uso de algoritmos não numéricos para resolver problemas complexos que não apresentam facilidades computacionais numéricas ou de análise direta.

05

2 - A EVOLUÇÃO DO *SOFTWARE*

Durante as três primeiras décadas da era do computador, o principal desafio era desenvolver um *hardware* que reduzisse o custo de processamento e armazenagem de dados. Ao longo da década de 80, avanços na microeletrônica resultaram em maior poder de computação a um custo cada vez mais baixo. Podemos dizer que o "poder da computação" foi o impulso da tecnologia para os dias atuais, pois começaram os desenvolvimentos para os PCs. Isso resultou o desenvolvimento de aplicações mais voltadas para a apresentação (aparência) dos sistemas. A evolução das máquinas acompanhava a evolução das linguagens. Hoje, o problema é diferente. O principal desafio após os anos 90 foi melhorar a qualidade da máquina reduzindo os custos de soluções baseadas em computador e, sobretudo, evoluindo para soluções que são implementadas com softwares.

O poder de um computador *mainframe* da década de 80 agora está à disposição em uma loja de departamento. A assombrosa capacidade de processamento e armazenagem do moderno *hardware* representa um grande potencial de computação.

O *software* é o mecanismo que possibilita ao usuário aproveitar e dar vazão a esse potencial. Evoluiu tanto o *hardware* quanto o *software*, um puxando o outro. A cada atualização de *software* e a cada nova aplicação, exige-se muito mais do *hardware*. É um ciclo de crescimento que não tem fim.

Figura: evolução do *hardware* e do *software*

06

O contexto de evolução do *software* está estritamente ligado a quase cinco décadas de evolução dos sistemas computadorizados. Um melhor desempenho de *hardware*, menor tamanho físico e custo mais baixo precipitaram o aparecimento de sistemas baseados em computadores mais sofisticados.

O desenvolvimento do *software* era feito virtualmente, sem gestão, até que os prazos começassem a se esgotar e os custos a subir abruptamente. Durante esse período, era usada uma orientação batch (em lote) para a maioria dos sistemas. Notáveis exceções foram os sistemas interativos, tais como o primeiro sistema de reservas da American Airlines. Na maior parte, entretanto, o *hardware* dedicava-se a execução de um único programa que, por sua vez, dedicava-se a uma aplicação específica.

Durante os primeiros anos, o *hardware* de propósito geral tornara-se lugar comum. O *software*, por outro lado, era projetado sob medida para cada aplicação e tinha uma distribuição relativamente limitada. O produto *software* estava em sua infância. A maior parte dos softwares era desenvolvida e, em última análise, usada pela própria pessoa ou organização. O técnico programava o sistema, disponibilizado para utilização da empresa, caso apresentasse algum problema, o mesmo técnico consertava. Naquela época a rotatividade de emprego era baixa, os chefes podiam dormir tranquilos com a certeza de que seu técnico estaria lá, para que os defeitos fossem encontrados e corrigidos.

Devido a esse ambiente de *software* personalizado, o projeto era um processo implícito realizado no cérebro de alguém, e a documentação muitas vezes não existia, ou era realizada de maneira muito diferente dos dias atuais. Durante os primeiros anos, aprendemos muito sobre a implementação de sistemas baseados em computador, mas relativamente pouco sobre **engenharia de sistemas de computador**.

A evolução trouxe novos produtos, porém há inúmeros softwares, que foram produzidos há décadas, em plena utilização. Para se ter uma ideia, sistemas importantes da rede bancária ainda estão utilizando a linguagem COBOL.

07

De acordo com a necessidade, as aplicações começaram a utilizar multiprogramação e os sistemas multiusuários, onde introduziram novos conceitos de interação homem-máquina. As técnicas interativas abriram um novo mundo de aplicações e níveis elevados de sofisticação de *software* e hardware. Sistemas passaram a funcionar em tempo real e fazer operações complexas como coletar, analisar e transformar dados de múltiplas fontes, controlando processos e produzindo saídas em milissegundos, e não mais em minutos.

Os avanços da armazenagem on-line levaram à primeira geração de sistemas de gerenciamento de bancos de dados. O *software* passou a ser desenvolvido para ampla distribuição num mercado interdisciplinar. Programas para *mainframes* e minicomputadores eram distribuídos para centenas e, às vezes, milhares de usuários. Empresários da indústria, governos e universidades puseram-se a desenvolver “pacotes de *software*" e a ganhar muito dinheiro.

À medida que o número de sistemas baseados em computador crescia, bibliotecas de *software* de computador começaram a se expandir. Projetos de desenvolvimento internos nas empresas produziram dezenas de milhares de instruções de programa. Os produtos de *software* comprados no exterior acrescentaram centenas de milhares de novas instruções.

De repente, diante desse turbilhão de novidades, uma nuvem negra surge no horizonte. Todos esses programas, todas essas instruções tinham de ser corrigidos ao se detectar falhas, ou alterados conforme as exigências do usuário, com o objetivo de se adaptar a um novo hardware que fosse adquirido. Essas atividades foram chamadas coletivamente de **manutenção de *software***. O esforço dispendido na manutenção de *software* começou a absorver recursos em índices alarmantes. E, ainda pior, a natureza personalizada de muitos programas tornava-os virtualmente impossíveis de sofrer manutenção. Os sistemas distribuídos – múltiplos computadores, cada um executando funções concorrentemente e comunicando-se um com o outro – aumentaram intensamente a complexidade dos sistemas baseados em computador.

Uma crise assombrou e agigantou no horizonte, a chamada crise de *software*.

08

3 - A CRISE DO SOFTWARE

A crise do software foi um termo que surgiu nos anos 70, quando a engenharia de software praticamente não existia. O termo expressava as dificuldades do desenvolvimento de software devido ao rápido crescimento da demanda por softwares, a complexidade dos problemas a serem resolvidos e a inexistência de técnicas e métodos para o desenvolvimento.

No desenvolvimento de softwares eram utilizadas as linguagens estruturadas e modulares. As empresas de softwares começaram a falhar constantemente nos prazos de entrega, apresentando resultados insatisfatórios, além de orçamentos que ultrapassavam o predeterminado. Percebe alguma semelhança com os problemas dos dias atuais? É fácil identificar: tem-se pouco dinheiro para investimento, tempo curto para desenvolvimento e exige-se uma qualidade de excelência.

Um relatório, em 1969, diversos pesquisadores independentes americanos evidenciaram que cerca de 50% a 80% dos projetos foram cancelados e outros fracassados por não terem atingindo os objetivos esperados. Restaram, portanto, apenas 20% de projetos finalizados, mas que foram entregues acima do prazo estipulado e orçamento acima do determinado no início do projeto.

Estes problemas estavam relacionados principalmente com a forma de trabalho da equipe, envolvendo também dúvidas em relação aos requisitos do sistema. Todos estes fatores exigiam novos métodos ou o aprimoramento dos métodos já existentes e foi então que surgiu a **Engenharia de Software**, que amenizou de maneira considerável alguns problemas citados, a partir da adoção de métodos e principalmente de modelos de processos de software.

09

Com a adoção dos modelos de processos de software, a tarefa de desenvolvimento foi dividida em etapas, possibilitando a coleta de dados, a interação com o cliente, a apresentação de protótipos do sistema, a estipulação dos prazos de entregas com margem de erros e finalmente a apresentação de um sistema robusto, eficiente e que viria satisfazer o cliente apresentando de fato uma solução para o problema existente.

Dentre alguns dos problemas que atingiam o desenvolvimento, podemos citar:

- Estimativas de prazos e de custos imprecisos;
- Produtividade dos profissionais que desenvolviam software;
- A qualidade do software desenvolvido era precária.

Observando os itens acima é possível notar que pouca coisa mudou nos dias atuais. Ainda há muitos problemas e desafios, embora cada vez mais se invista em novos métodos de desenvolvimento.

10

4 - ENGENHARIA DE *SOFTWARE*

Uma primeira definição de engenharia de *software* foi proposta por Fritz Bauer, alemão cientista da computação, na primeira grande conferência dedicada ao assunto:

Engenharia de software é a criação e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que funcione de forma eficiente.

A engenharia de software é uma derivação da engenharia de sistemas e de hardware. Ela abrange um conjunto de três elementos fundamentais:

- métodos,
- ferramentas e
- procedimentos.

Essas ferramentas possibilitam ao gerente o controle do processo de desenvolvimento do software e oferece ao profissional uma base para a construção de software de alta qualidade produtivamente.

11

- **Métodos**

Um método é uma prescrição explícita de como chegar a uma atividade requerida por um modelo de ciclo de vida, visando otimizar a execução das atividades que foram especificadas.

Os métodos de engenharia de software proporcionam os detalhes de como fazer para construir o software. Os métodos envolvem um amplo conjunto de tarefas que incluem:

- planejamento e estimativa de projeto,
- análise de requisitos de software e de sistemas,
- projeto da estrutura de dados,
- arquitetura de programa e
- algoritmo de processamento,
- codificação,
- teste,
- manutenção.

Muitas vezes são utilizadas notações gráficas ou orientadas a linguagens e introduzem um conjunto de critérios para a qualidade do software. As ferramentas de engenharia de software podem proporcionar apoio automatizado nos processos.

12

- **Ferramentas**

As **ferramentas** proporcionam apoio automatizado ou semi automatizado aos métodos. Os Métodos de Desenvolvimento de Sistema se diferenciam pela maneira como o problema deve ser visualizado e como a solução do problema deve ser modelada. São métodos baseados em decomposição, estruturas de dados e método análise baseado na orientação a objeto.

- **Procedimentos**

Os **procedimentos** da engenharia de software constituem o elo que mantém juntos os processos e as ferramentas e possibilita o desenvolvimento racional e oportuno do software de computador. Eles definem a sequência em que os processos serão aplicados, os produtos a serem entregues (documentos, códigos, pacotes, relatórios, formulários etc.), os controles que ajudam a assegurar a qualidade, o gerenciamento das mudanças, o gerenciamento do projeto e o seu progresso.

Como vimos, a engenharia de software compreende um conjunto de etapas que envolvem métodos, ferramentas e os procedimentos. Essas etapas muitas vezes são citadas como paradigmas da engenharia de software, e principalmente como base na complexidade e origem do projeto e da aplicação. Conforme característica do projeto é definido o melhor caminho, como métodos, ferramentas, controles e os produtos que precisam ser entregues.

13

5 - DESAFIOS DA EXPANSÃO DO *SOFTWARE* E DO *HARDWARE*

As redes globais e locais, as comunicações digitais, a grande e elevada transferências de dados e a crescente demanda de acesso imediato (*on-line*) a dados exigem muito dos desenvolvedores de *software* e do *hardware*. O microprocessador gerou um amplo conjunto de produtos inteligentes: de automóveis a fornos de micro-ondas, de robôs industriais a equipamentos para diagnostico de soro sanguíneo. Em muitos casos, a tecnologia de *software* está sendo integrada a produtos por equipes técnicas que entendem de *hardware*, mas que frequentemente são principiantes em desenvolvimento de *software*.

Figura: tecnologia que permite alteração de potência ou de combustível do carro

O computador pessoal foi o grande responsável pelo crescimento de muitas empresas de *software*. Os computadores das empresas, onde cada profissional tem o seu, se tornou um produto primário e essencial, sendo o *software* o diferencial. Não se faz nada sem *software* específico, assim, o *software* oferece as características que diferenciam do *hardware*. Como um bem primário, a venda de computadores pessoais se estabilizou, e as vendas de *software* continuaram a crescer. Na indústria ou em âmbito doméstico, muitas pessoas gastaram mais dinheiro com *software* do que com computadores.

14

Os *softwares* em geral estão evoluindo, os sistemas especialistas e o *software* de inteligência artificial finalmente saíram do laboratório para a aplicação prática em problemas de amplo espectro do mundo real.

O *software* de rede neural artificial abriu possibilidades para o reconhecimento de padrões e para capacidades de processamento de informações semelhantes às humanas. Temos agora problemas associados ao *software* de computador e continuam a se intensificar:

- A sofisticação do *software* ultrapassou nossa capacidade de construir um *software* que extraia o potencial do *hardware*.
- A capacidade de construir programas não pode acompanhar o ritmo da demanda de novos programas.
- A capacidade de manter os programas existentes é ameaçada por projetos ruins e recursos inadequados.

Em resposta a esses problemas, estão sendo adotadas práticas de engenharia de *software* em toda a indústria.

15

Em poucas palavras, é possível afirmar que os principais desafios para a engenharia de software são:

O **desafio do legado**, que é fazer a manutenção e a atualização dos *softwares* atuais, sem apresentar grandes custos e ao mesmo tempo prosseguir com a prestação dos serviços corporativos essenciais.	O **desafio da heterogeneidade**, que se refere a desenvolver técnicas para construir *softwares* confiáveis e que sejam flexíveis para lidar com diferentes tipos de equipamentos e sistemas.	O **desafio do fornecimento**, que deve apresentar uma redução do tempo gasto no desenvolvimento e implantação do *software* sem comprometer a qualidade.

Esses desafios são referentes à evolução do *software* e do *hardware*.

16

RESUMO

A engenharia de *software* iniciou com a crise do *software*. Com a evolução dos computadores e dos softwares, os problemas começaram a apresentar de forma desordenada para todos. A engenharia de *software* veio para padronizar o entendimento e definir vários processos para o desenvolvimento. O cientista da computação Fritz Bauer foi um dos primeiros a se definir um conceito e a querer padronizar a forma de desenvolver. Segundo Bauer, **engenharia de *software*** é a criação e uso de sólidos princípios de engenharia para que se possa obter economicamente um *software* que funcione de forma eficiente.

Os problemas atuais de desenvolvimento de *software* são bem parecidos com o que tínhamos há décadas: tempo escasso, recursos limitados e complexidade alta da solicitação. A engenharia de *software* veio para esclarecer e evoluir a melhor forma de conseguir entregar um *software*. E tem como grande desafio tratar os problemas do desenvolvimento do *software*.

UNIDADE 1 – DESENVOLVIMENTO DE SOFTWARE PARA VALORIZAÇÃO DO NEGÓCIO

MÓDULO 2 – TENDÊNCIA DA ENGENHARIA DE SOFTWARE PARA VALORIZAÇÃO DO NEGÓCIO

01

1 - Tendências da Engenharia de Software

Durante a história recente da engenharia de *software*, profissionais e estudiosos desenvolveram vários modelos de processos, métodos e ferramentas para automatizar e diminuir esforços. A história mostra que os projetos fracassam na sua maioria, não cumprindo os prazos, os custos estouram ou a qualidade não é satisfatória.

Novas técnicas nascem regularmente apresentadas como a solução dos problemas que os engenheiros de *softwares* enfrentam nos projetos grandes, médios e pequenos. Quando surge essa nova tecnologia há sempre uma tendência de uso por uma comunidade que defende e adota as práticas novas, mas a decepção aparece quando o resultado não é satisfatório.

Pressman cita em seu livro "Engenharia de *Software*" o comentário de dois estudiosos, Mili e Cowan. Eles comentam sobre os desafios que enfrentamos ao tentarmos isolar tendências significativas na tecnologia:

Que fatores determinam o sucesso de uma tendência?

- São as características e as tendências bem sucedidas, que são seus méritos técnicos, suas habilidade de abrir novos mercados e suas habilidade para alterar a economia dos mercados existentes.

Qual é o ciclo de vida de uma tendência?

- Embora a visão tradicional de que as tendências evoluem ao longo de um ciclo de vida bem definido e previsível, que vai à pesquisa a um produto acabado, por meio de um processo de transferência, descobrimos que muitas encurtaram esse ciclo ou seguiram outro.

Com que antecedência pode uma tendência bem sucedida ser identificada?

- Se soubéssemos identificar fatores de sucesso e se entendermos o ciclo de vida de uma tendência, podemos detectar sinais precoces de êxito. Retoricamente buscamos a habilidade para reconhecer a próxima tendência antes dos outros.

02

Segundo Pressman ninguém pode prever o futuro com certeza absoluta. Mas é possível avaliar tendências na área da engenharia de *software* e a partir dessas tendências sugerir direções possíveis para a tecnologia. Qualquer um que deseja dedicar seu tempo para se manter atualizado sobre os problemas de engenharia de *software* pode tentar prever a direção futura da tecnologia.

Atualmente grandes empresas contratam consultoria e se preparam segundo as previsões. É claro que todos nós queremos saber o que vai acontecer para que possamos nos preparar. Mas não há uma fórmula para prever o caminho correto. Tentamos fazer isso coletando dados, organizando esses dados para que nos forneçam informações úteis. Analisamos e associamos informações gerando conhecimento, o qual pode sugerir prováveis tendências para um tempo futuro. Essa visão pode ou não ser correta.

Pressman afirma que **prever o caminho é uma arte, não uma ciência**. De fato, é muito raro que uma previsão esteja totalmente correta, sempre há alguma exceção nas previsões, por isso falamos em **tendência**, ou seja, há uma tendência de acontecer, ou de ser da maneira que estamos prevendo. Não é uma certeza de 100% de acontecer.

Os filmes dos anos 80 mostravam que no ano 2000 teríamos carros sem rodas, ou seja, carros sem necessidade de gasolina e flutuantes. A tecnologia gerava uma incógnita e alguns afirmavam que era uma tendência de acontecer. Passamos pelo ano 2000 e estamos muito atrasados e escravizados aos combustíveis fósseis.

03

O Gartner Group, uma consultoria que estuda as tendências da tecnologia em muitos ramos de atividades, desenvolveu um ciclo para tecnologias emergentes. Esse ciclo possui cinco fases:

Clique nas cinco fases.

Disparo de tecnologia
Avanço na pesquisa ou lançamento de um produto inovador que leva a grande cobertura da mídia e entusiasmo popular.

Pico de expectativas inflacionadas
Entusiasmo demasiado e projeções extremamente otimistas do impacto com a base em um sucesso

limitado, mas muito bem publicado.

Desilusão
Projeções demasiado otimistas do impacto não se confirmam e os críticos começam a se manifestar. A tecnologia torna-se sem atrativos entre os conhecedores;

Ladeira do iluminismo
Utilização cada vez maior por uma ampla variedade de empresas leva a um melhor entendimento do verdadeiro potencial da tecnologia, aparecem métodos e ferramentas prontos para uso para suportar a tecnologia.

Platô de produtividade
Os benefícios do mundo real agora são óbvios e o uso atinge uma porcentagem significativa do mercado em potencial

04

Segundo Pressman, nem toda a tecnologia de engenharia de *software* segue esse caminho ao longo do ciclo da excelência, em alguns casos a decepção é justificada e a tecnologia fica relegada à obscuridade.

Ciclo de excelência – Gartner Group para tecnologias emergentes

05

O mundo é demasiado grande, cada país pode ter a sua tendência tecnológica de acordo com as características que definem os negócios e as políticas organizacionais. Essas tendências surgem dentro das empresas, de acordo com as necessidades e evolução cultural. Os aspectos locais, clientes e a própria cultura do país determinam a evolução da tecnologia.

Segundo Pressman, a globalização leva à força de trabalho diversificada (em termos de linguagem, cultura, solução de problemas, filosofia de administração, prioridades de comunicação e interação entre as pessoas). Uma equipe colabora com outras que estão separadas por uso horário, linguagem e cultura.

A engenharia de software deve responder com um modelo de processo abrangente para **equipes distribuídas**, que seja ágil o bastante para atender às demandas da urgência, mas disciplinado o suficiente para coordenar diferentes grupos.

Isso exige uma estrutura organizacional flexível, equipes diferentes em países diferentes devem responder a problemas de engenharia de maneira que melhor atenda suas necessidades especiais, enquanto proporciona uniformidade para a execução de um projeto global ou local.

06

Empresas internacionais podem utilizar ou não metodologias, conforme definição do país e empresa que está sendo atendida (cliente). Empresas de grande porte já possuem técnicas próprias e processos definidos, preferindo na maioria das vezes não utilizar os processos da empresa contratada, o que obriga as empresas a se adaptarem aos seus processos.

Isso pode ser bom e ruim, tudo depende de que lado você está. Se a multinacional contratada tiver bem definidos seus processos e técnicas de desenvolvimento, e assinam contratos com empresas nacionais que também possuem processos e técnicas estabelecidas, isso não é bom. Ninguém quer mudar para se adaptar ao outro, pois mudanças mexem com a cultura das pessoas. Os técnicos estão acostumados a executar determinados processos, e quando esses processos mudam, muitas vezes há resistências, pois

é difícil modificar aquilo que está dando certo, ainda que haja uma boa perspectiva de melhoria do processo.

Pressman afirma que a própria cultura humana irá em direção à engenharia de *software*. Toda geração deixa sua própria marca na cultura local, e a nossa não será diferente. Segundo Faith Popcorn, um especialista em tendências culturais, nossas tendências não são apenas modismos, elas permanecem e evoluem.

As tendências representam forças ocultas, causas principais, necessidades humanas básicas, atitudes, aspirações. Elas nos ajudam a navegar pelo mundo, entender o que está acontecendo e por que, e nos preparar para aquilo que ainda virá.

A natureza de uma equipe de engenharia de *software* pode mudar à medida que os sistemas baseados em *software* ficarem mais complexos, as comunicações e colaborações entre equipes globais transformaram em lugar comum, afirma Pressman. Cada equipe de projetos de *softwares* deve contribuir para que o talento criativo e habilidades de seus membros sejam incentivados e preservados. A equipe deve possuir pessoas com perfis distintos. Para cada fase do projeto você precisará de um determinado perfil.

07

2 - TERMOS QUE ABRANGEM OS AVANÇOS DA TECNOLOGIA

Para compreendermos as possibilidades de avanço da tecnologia, é importante entendermos os significados de certas expressões que escutamos diariamente no nosso dia a dia. Nomes como código aberto, *software* livre, *software* gratuito, código fonte, todos são termos que em algum momento é usado erroneamente, mas são definições importantes para as empresas que trabalham com desenvolvimento de *softwares*.

A seguir trataremos de esclarecer algumas dúvidas sobre esses termos.

- ***Software* livre**

Segundo o site do infowester, *software* livre é um conceito de extrema importância no mundo da computação. Para estar nesta condição, o *software* precisa ter características vinculadas a aspectos de liberdade. Pode-se dizer, portanto, que o *software* livre é um movimento social, que defende uma causa.

Quando falamos em *software* livre, portanto, tratamos da liberdade que o usuário tem para não só utilizar, mas também para copiar, distribuir, modificar e estudar o *software*. É isso o que o movimento do *software* livre defende.

Para tanto, a Free *Software* Foundation definiu quatro "fundamentos" como base:

1. liberdade de executar o programa, para qualquer propósito (liberdade 1);

2. liberdade de estudar como o programa funciona e adaptá-lo às suas necessidades (liberdade 2), sendo o acesso ao código fonte* um pré-requisito para esta aspecto;
3. liberdade de distribuir cópias de forma que você possa ajudar ao seu próximo (liberdade 3);
4. liberdade de melhorar o programa e liberar os seus aperfeiçoamentos, de modo que toda a comunidade se beneficie (liberdade 4). Novamente, aqui o acesso ao código fonte é um pré-requisito.

***software* livre**

A ideia começou a tomar forma em 1983, pelas mãos de Richard Stallman, que na época criou o GNU e, cerca de dois anos depois, fundou a Free *Software* Foundation (FSF). O GNU é um projeto para o desenvolvimento de um sistema operacional livre, isto é, já apoiado nos objetivos da liberdade. A Free *Software* Foundation, por sua vez, é uma entidade sem fins lucrativos criados justamente para servir de base para o movimento do *software* livre.

liberdade 1

No caso da primeira liberdade, um indivíduo ou uma organização pode utilizar o *software* em qualquer atividade, em qualquer quantidade de computadores.

liberdade 2

A segunda liberdade, por sua vez, dá acesso ao código fonte do programa para que a pessoa possa estudá-lo ou alterá-lo conforme a sua necessidade.

liberdade 3

A terceira liberdade diz respeito à permissão dada ao usuário de distribuir quantas cópias quiser do programa, mesmo porque esta é uma forma de torná-lo acessível a um número maior de pessoas. O mesmo vale para o código fonte do *software*.

liberdade 4

Por fim, a quarta liberdade consiste na permissão que o usuário tem para alterar um *software* ou mesmo colaborar com o seu desenvolvimento, permitindo que outras pessoas ou organizações tirem proveito de algo que ele aperfeiçoou.

08

- **Código fonte**

Código fonte são as instruções que formam um programa.

O código fonte se baseia nas linguagens de programação. Depois de concluído, esse código deve ser transformado em linguagem de máquina para que o computador efetivamente faça das instruções um *software*. Tendo acesso ao código fonte, uma pessoa com conhecimentos para isso pode estudá-lo ou mesmo alterá-lo conforme sua necessidade ou interesse.

- ***Software* grátis (gratuito)**

Quando nos referimos a um *software* meramente gratuito (freeware), estamos falando de um programa que você pode utilizar sem pagar.

Perceba, com isso, que um *software* pode ser gratuito e livre, por outro lado, pode ser também gratuito e fechado. Um *software* nesta condição é restrito, isto é, somente o autor ou a entidade que o desenvolve tem acesso ao código fonte, portanto você não pode alterá-lo ou simplesmente estudá-lo, somente usá-lo da forma como foi disponibilizado. Muitas vezes, há limitações também em sua distribuição.

Antes de utilizar um *software* grátis na sua empresa verifique com a equipe da Qualidade ou o suporte se não poderá ter problemas posteriores com manutenção.

Software livre e *software* gratuito não é a mesma coisa.

09

- **Todo *Software* é gratuito?**

Você já sabe que o *software* livre consiste na ideia de que você possa utilizar, distribuir, estudar o código fonte e até modificá-lo, sem necessidade de pedir autorização ao seu desenvolvedor. Softwares nestas condições geralmente não requerem pagamento, mas isso não é regra: **um programa pode ser livre, mas não necessariamente gratuito**.

Uma pessoa pode pagar para receber um *software* livre ou cobrar para distribuir um programa nesta condição, por exemplo, desde que esta ação não entre em conflito com as liberdades apontadas pela *Free Software Foundation*.

Como exemplo, um programador pode desenvolver um aplicativo, disponibilizá-lo como *software* livre e vendê-lo em seu site, desde que não impeça o comprador de acessar o código fonte, fazer alterações, redistribuir e assim por diante.

Perceba que esta é mais uma diferença entre *software* livre e um *software* meramente gratuito.

- **GNU Public License (GPL)**

Quando um *software* é criado, o desenvolvedor o associa a um documento que determina quais ações o utilizador pode ou não executar. Esta é a **licença de software**. Por exemplo, você pode adquirir uma solução de ERP que só é possível de ser implementada em um número limitado de máquinas. Esta e outras condições devem ficar explícitas na licença.

A GNU Public License (GPL) é uma licença criada pela *Free Software Foundation* baseada nas liberdades que a entidade defende. Ou seja, quando um programa possui licença GPL, significa que é, de fato, um *software* livre.

É importante frisar que um programa não necessita obrigatoriamente de uma licença GPL para ser um *software* livre. É possível o uso de outras licenças, desde que compatíveis com as liberdades em questão. Saiba+

10

- **O que é copyleft?**

Não é raro encontrarmos a expressão **copyleft** em evidência quando tratamos de *software* livre. Isso tem um motivo: ambos os conceitos estão intimamente relacionados.

Talvez você tenha percebido que a expressão copyleft (*copy* + *left*) é um trocadilho com o termo *copyright* (*copy* + *right*), que se refere aos direitos de uso ou cópia de uma propriedade intelectual. No caso, a palavra *left* faz alusão a um contexto mais generoso: **enquanto o copyright dá mais foco nas restrições, o copyleft se baseia nas permissões**. Mas, no que exatamente o *software* livre está relacionado ao copyleft?

No caso do *software* livre, o desenvolvedor poderia deixar seu programa em *domínio público*, isto é, sujeito a toda e qualquer forma de utilização, alteração e distribuição. Porém, esta situação pode fazer com que indivíduos ou entidades modifiquem este *software* e o disponibilizem mediante uma série de restrições, ignorando as liberdades que o tornariam livre.

É para evitar problemas desse tipo que o copyleft entra em cena: com ele, as **liberdades de modificação e distribuição são garantidas**, tanto em um projeto original quanto em um derivado. Isso significa que uma pessoa ou uma organização não poderá obter um *software* livre, modificá-lo e distribuí-lo de maneira restrita, devendo compartilhar o programa - seja ele alterado ou não pelas mesmas condições em que o obteve (compartilhamento pela mesma licença).

Este cenário é válido para as licenças compatíveis com tais condições, como é o caso da GPL. Vale frisar, no entanto, que há licenças para *software* livre que não contemplam as características do copyleft.

11

- **Código aberto (Open source)**

É comum ver *Software* Livre e Código Aberto (Open Source) sendo tratados como se fossem a mesma coisa. De igual maneira, não é difícil encontrar a expressão "código aberto" como mero sinônimo de "código fonte aberto". Não há, necessariamente, erros aqui, mas há diferenças.

O Open Source é um movimento que surgiu em 1998 por iniciativa principal de Bruce Perens, mas com o apoio de várias outras pessoas que não estavam totalmente de acordo com os ideais filosóficos ou com outros aspectos do *Software* Livre, resultando na criação da Open Source Initiative (OSI).

A Open Source Initiative não ignora as liberdades da *Free Software Foundation*, por outro lado, tenta ser mais flexível. Para isso, a organização definiu **dez quesitos para que um *software* possa ser considerado Open Source:**

1. Distribuição livre;
2. Acesso ao código fonte;
3. Permissão para criação de trabalhos derivados;
4. Integridade do autor do código fonte;
5. Não discriminação contra pessoas ou grupos;
6. Não discriminação contra áreas de atuação;
7. Distribuição da licença;
8. Licença não específica a um produto;
9. Licença não restritiva a outros programas;
10. Licença neutra em relação à tecnologia.

A descrição de cada um desses critérios pode ser encontrada em*software*livre.org/open-source-codigo-aberto ou em www.opensource.org/docs/definition.php (em inglês).

12

Analisando as características da Free *Software* Foundation e da Open Source Initiative, percebemos que, em muitos casos, um *software* livre pode também ser considerado código aberto e vice-versa.

A diferença está, essencialmente, no fato de a OSI ter receptividade maior em relação às iniciativas de *software* do mercado. Assim, empresas como Microsoft e Oracle, duas gigantes

do *software* proprietário, podem desenvolver soluções de código aberto utilizando suas próprias licenças, desde que estas respeitem os critérios da OSI. No *Software* Livre, empresas como estas provavelmente enfrentariam algum tipo de resistência, uma vez que suas atividades principais ou mesmo os programas oferecidos podem entrar em conflito com os ideais morais da Free *Software* Foundation.

Apesar disso, Free *Software* Foundation e Open Source Initiative não são inimigas. Embora não concordem em alguns pontos, ambas as entidades se respeitam e reconhecem a importância da atuação de cada uma. De certa forma, pode-se dizer inclusive que o trabalho das duas organizações se complementa: a FSF atuando mais pelo **lado social**; a OSI, pelos **contextos técnicos e de mercado**.

13

Para Pressman, há uma tendência inegável, que os sistemas baseados em *software* sem dúvida se tornarão maiores e mais complexos, independentemente da plataforma ou do domínio de aplicação, que apresenta o grande desafio para os engenheiros de *softwares*. Conforme a comunidade de engenharia de *software* desenvolve novas abordagens motivadas por modelos mais atuais focados em representação de requisitos de sistemas e de projetos.

A evolução da engenharia de *software* acontecerá se a comunidade desenvolver uma filosofia de engenharia de *software* mais distribuída e colaborativa. A comunicação e a divulgação de novas experiências com novos modelos são importantes para o crescimento do desenvolvimento dos *softwares*.

14

RESUMO

O módulo apresenta tópicos sobre as tendências da engenharia de *software*. Vimos que a evolução da engenharia de *software* ocorre em diversos países e cada empresa pode ter uma cultura de desenvolvimento que define como os projetos e sistemas serão desenvolvidos e implementados. A cultura é um fator importante em uma evolução dos processos da engenharia de *software*, sendo um dos fatores mais difíceis de mudar. A aparição de um novo modelo ou tecnologia emergente pode ocorrer em fases, que demonstram o ápice do uso ao declínio da utilização e motivação.

Vários conceitos são importantes a destacar no mundo da engenharia de *software*, pois muitas pessoas confundem *software* livre, código aberto, *software* gratuito. Todos estão no dia a dia do desenvolvimento e nos escritórios e casas mundo afora.

Mas o fator importante a destacar é do futuro da engenharia de *software*, o qual muito provavelmente continua a depender de pessoas. Especialistas que estudam, que desenvolvem projetos, analistas da qualidade e equipes de manutenção de sistemas, todos estão no mundo da engenharia de *software*. E para que o assunto avance a comunicação deve ser o ponto principal.

UNIDADE 1 – DESENVOLVIMENTO DE SOFTWARE PARA VALORIZAÇÃO DO NEGÓCIO

MÓDULO 3 – PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

01

1 - DEFINIÇÃO DE PROCESSO

A Engenharia de Software nos faz revisitar assuntos já tratados, pois estamos sempre querendo evoluir os processos que estamos seguindo. Sendo assim entraremos no assunto processo que, de agora em diante, sempre fará parte de sua vida.

Recapitulando rapidamente a definição de processo temos, do dicionário, que

> Processo é um conjunto sequencial e peculiar de ações que objetivam atingir uma meta. É usado para criar, inventar, projetar, transformar, produzir, controlar, manter e usar produtos ou sistemas.

Analisando essa definição podemos perceber que praticamente todas as atividades que executamos podem ser vistas como processo.

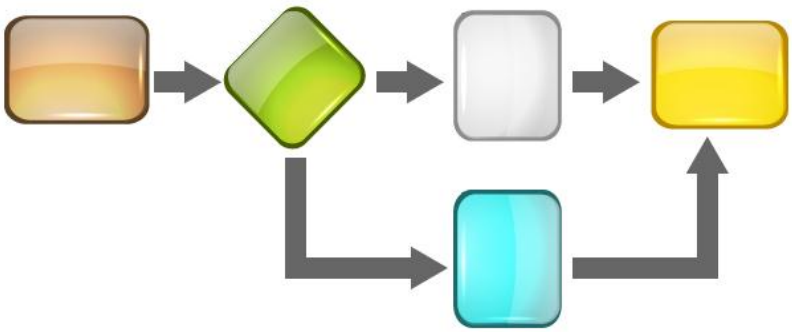


FIGURA: EXEMPLO DE FLUXO DE PROCESSO

Na definição de processos, uma série de parâmetros é considerada:

- Evento que inicia o processo.
- Matriz de responsabilidades.
- Atividades a serem executadas e as respectivas sequências.
- Entradas e saídas de cada atividade.
- Políticas e regras a serem utilizadas na execução das tarefas.
- Infraestrutura necessária.
- Resultado final gerado na execução do processo.

02

Enumeramos a seguir uma série de **características relacionadas a processo**:

- É disparado/inicializado por um evento externo.
- Engloba todas as atividades necessárias para prover os resultados apropriados em resposta ao evento que o inicializou.
- Transforma entradas (de qualquer tipo) em saídas (de qualquer tipo), de acordo com regulamentos (políticas, padrões, procedimentos, regras etc.), empregando recursos de qualquer tipo.
- Contêm atividades que, normalmente, cruzam funções e unidades organizacionais.
- Possui indicadores de performance para os quais objetivos mensuráveis podem ser estabelecidos e a performance avaliada.
- Entrega um produto ou serviço a um *stakeholder* externo ou a um processo interno.
- Constitui-se de um conjunto de atividades de objetivo comum que atende a uma necessidade.

Podemos citar uma série de benefícios na utilização de processos definidos na execução de tarefas nas empresas. Imagine que em uma lanchonete famosa que você frequenta há anos com dezenas de unidades por todo o país, determinado sanduíche seja sempre o mesmo, independente da unidade que você vá e com o mesmo sabor de anos atrás. Agora pense que são vários cozinheiros diferentes, em localidades diferentes, que fazem o mesmo lanche. Como o gosto pode ser sempre o mesmo independente do local e durante todo este tempo? A resposta é: **existe processo definido**!

Com processo definido, podemos trocar o cozinheiro diariamente ou comer o mesmo em qualquer unidade da lanchonete, que o lanche será sempre igual.

A definição e utilização de processos constitui uma grande preocupação das empresas, a retenção do conhecimento. No caso da utilização de processos definidos, este conhecimento está embarcado e preservado dentro dos próprios processos.

03

De uma maneira geral, podemos dizer que **processo**:

- Padroniza a geração de serviços e produtos.
- Garante a repetitividade da geração de serviços e produtos.
- Mantém conhecimento na empresa permitindo que novos funcionários possam executar tarefas, seguindo procedimentos definidos e conhecidos na empresa.
- Oferece um guia para definir as atividades.
- Especifica todo o processo para construção e desenvolvimento de artefatos.
- Direciona as tarefas individuais e da equipe como um todo.
- Permite definir e fornecer critérios para monitoração e medição dos produtos e atividades do projeto.
- Provê linhas gerais para os usuários e desenvolvedores.

- Reduz riscos e torna resultados dos projetos mais previsíveis.
- Provê visões comuns a equipe de desenvolvimento.
- Serve como um template que pode ser reutilizado.

Vale lembrar que **template** pode ser um modelo de um documento como um modelo de processo definido, que possui várias atividades e tarefas para serem desempenhadas pela equipe de desenvolvimento de *software*.

04

2 - DEFININDO PROCESSO PARA DESENVOLVIMENTO DE SISTEMAS

Toda empresa que desejar obter pelo menos parcialmente os benefícios fornecidos por processos, deverá institucionalizar processos definidos na execução de suas tarefas relacionadas ao negócio. Em nosso caso, o negócio é o desenvolvimento de *software* com qualidade e menores custos, atingindo os prazos do projeto. Podemos dizer que a qualidade de um sistema é fortemente influenciada pela qualidade do processo utilizado no desenvolvimento e manutenção do mesmo.

Ao se pensar a implantação de processos para desenvolvimento de *software*, pode-se criar internamente ou utilizar propostas já existentes no mercado. Uma equipe de desenvolvimento pode criar seu próprio processo de desenvolvimento, o que pode ou não ser eficaz.

O que precisamos avaliar nesta hora é que já existem muitas propostas de mercado prontas e testadas há anos por grandes empresas. Isto sugere que antes de criar uma própria, devemos analisar as prontas e verificar se alguma atende à realidade de nossa empresa.

Independente se a empresa decide utilizar uma metodologia de mercado com seus processos embarcados ou criar uma própria, a mesma deveria considerar ao menos os seguintes pontos:

- **Cultura** da empresa.
- **Tamanho** da organização.
- **Prioridades relativas**, facilidades a oferecer, prazo de entrega.
- **Formalidade**: determinar o grau de formalidade e de acordo com o tamanho da equipe, a velocidade com que a equipe se modifica e complexidade e tamanho do projeto.
- **Criação de *road maps*** específicos para tratar situações diferentes.
- **Complexidade do sistema**: no caso de sistemas com grande complexidade torna-se interessante a utilização de processo de desenvolvimento iterativo, com entregas modulares.
- **Equipe**: além do tamanho da equipe de desenvolvimento devem ser consideradas as áreas de especialidade necessárias a cada membro e o grau de conhecimento a ser exigido da equipe no domínio do problema e da tecnologia a ser utilizada.

Grau de formalidade
A definição do grau de formalidade impede que requisitos de sistemas, compromissos assumidos etc. não sejam desconsiderados. Quanto mais formal o processo, menos possibilidades de erros, porém deve-se cuidar para que não se tenha um processo engessado.

05

Todo o ciclo de vida de um projeto de desenvolvimento de sistemas se inicia com uma demanda, a solicitação do cliente interno ou externo à empresa ou de alguma área interna. Essa solicitação pode ser apresentada de várias maneiras, tais como:

- solicitação da área de negócio;
- estudo de viabilidade;
- solicitação da área técnica;
- melhoria nos processos internos.

Esse evento é responsável por iniciar uma série de processos encadeados, precisando de aprovações para efetivamente se tornar um projeto de desenvolvimento ou apenas um atendimento normal de uma demanda.

Macroprocesso de atendimento de demanda

Vejamos cada uma dessas fases a seguir.

06

a) Solicitação de demanda

Toda a demanda e todo o projeto começam com uma **solicitação**, essa solicitação pode ser de algum cliente interno ou externo, de uma necessidade interna de atualização e também pode ser uma demanda legal (necessidade de atualização dos aplicativos de acordo com alterações de leis ou regras governamentais).

As **solicitações de demandas** podem ocorrer de diversas formas:

- via sistema de solicitação de demandas;

- proposta ou uma declaração de trabalho.

Ambas especificam preliminarmente as necessidades da área de negócios com suas expectativas para a área técnica.

Algumas empresas padronizam a entrada de demandas. Isso é importante para que se ganhe agilidade no atendimento. Para se ter uma ideia, grandes bancos possuem áreas específicas que pensam apenas em negócios que podem gerar lucro e melhor atendimento para o usuário final.

07

b) Avaliação técnica

Após a área de negócio ou o cliente realizar uma solicitação, a área técnica avalia a **necessidade de alteração ou criação** junto com as particularidades do pedido. Essas particularidades envolvem as dificuldades, custos, riscos, prazos, urgência e aspectos de qualidade. Outro ponto que a área técnica deve avaliar é se a solicitação interfere em alguma aplicação da empresa. Essa avaliação evita estragar o que já está funcionando. É importante ter controle dos códigos desenvolvidos na empresa, uma solicitação pode já estar construída (código desenvolvido) em outro sistema.

Outro ponto importante na avaliação técnica é a definição de **quem vai construir a demanda**. Empresas podem possuir equipes de desenvolvimento e também podem ter fábricas externas (terceirizadas). Geralmente fábricas de desenvolvimento de *software* produzem *softwares* mais rapidamente.

Atualmente existem algumas técnicas para definir o valor de uma solicitação, que chamamos de **métricas de *software***. Com a finalidade de atender às estimativas de prazo e custo para o desenvolvimento de *software*, foram criadas várias técnicas de métricas relacionadas a solicitações de alteração ou criação de *softwares*. Essas técnicas propõem realizar estimativas relacionadas a prazo e custo. Podemos citar as seguintes métricas:

- **APF** (Análise de Pontos de Função);

- **SNAP** (Software Non-Functional Assessment Process).

As métricas de *software* atualmente são padronizadas pelo órgão IFPUG (International Function Point Users Group), que é grupo de utilizadores do ponto da função internacional sem fins lucrativos. A missão do IFPUG é ser um líder reconhecido em promover e incentivar a gestão eficaz das atividades de manutenção com o uso de análise de pontos de função e outras técnicas de medição de *software* e desenvolvimento de *software* aplicativo.

Análise de Pontos de Função (APF)
A APF (análise dos pontos de função) é uma técnica de medição das funcionalidades fornecidas por um software do ponto de vista de seus usuários. Ponto de função (PF) é a sua unidade de medida, que tem por objetivo tornar a medição independente da tecnologia utilizada para a construção do software. Ou seja, a APF busca medir o que o software faz, e não como ele foi construído. É o Método mais utilizado

nas empresas governamentais, foi criado nos anos 70 para medir o *software* pela quantificação das funcionalidades (requisitos funcionais) oferecidas para os usuários.

SNAP
O processo de dimensionamento de SNAP é muito semelhante ao ponto de função (APF). Possibilita a contagem de requisitos não funcionais do *software* aplicativo. Os SNAP Points (SP) constituem a unidade de medida no SNAP e são apurados a partir da análise do projeto (design) dividido em componentes, processos ou atividades usados no atendimento de requisitos não funcionais.

08

c) Desenvolvimento

Após a avaliação técnica da solicitação é escolhido quem vai desenvolver o *software*. O desenvolvimento é responsável por transformar a ideia do cliente em realidade. O desenvolvimento constrói o código de acordo com as especificações técnicas dos requisitos que foram levantadas e aprovadas pelos clientes. Esse desenvolvimento pode ser interno (na empresa) ou por fábricas de *software*. Algumas fábricas de *softwares* possuem certificados de maturidade em seus processos de desenvolvimento de *software*, como CMMI e MPS-BR.

d) Testagem e Homologação

Fase importante no desenvolvimento do *software*. A aplicação deve ser testada por quem construiu e por quem solicitou. É o formalismo que tudo que foi pedido está de acordo. Caso o aplicativo tenha algum problema técnico, será revelado nos testes da aplicação. Caso esteja tudo certo nos testes a equipe solicita ao cliente que homologue a solicitação realizada. O cliente avaliará se tudo que foi construído está de acordo com a sua solicitação e requisitos aprovados. Se a aplicação não estiver como o cliente pediu, a equipe de desenvolvimento deve corrigir o mais rápido possível. Caso esteja tudo como o cliente solicitou, a aplicação é homologada.

CMMI
O CMMI - Capability Maturity Model Integration - é um conjunto de modelos integrados de maturidade e capacidade para diversas disciplinas, tais como: engenharia de software e sistemas, fontes de aquisição e desenvolvimento integrado do produto. Está dividido em 5 níveis de maturidade que atestam o grau de evolução em que uma organização se encontra num determinado momento, o que permite que o software seja produzido de forma sistemática, dentro dos prazos pré-definidos e com níveis de qualidade que também são preestabelecidos e controlados.

MPS-BR

MPS-BR - Melhoria do Processo de Software Brasileiro - É simultaneamente um movimento de melhoria do software brasileiro e um modelo de qualidade de processos voltados para a realidade brasileira. Possui sete níveis de maturidade, onde a implantação é mais gradual e adequada a pequenas e médias empresas.

09

e) Implantação

A solicitação construída e aprovada pelo cliente significa que ela pode ser colocada em produção, ou que a aplicação pode ser implantada.

Isso também significa que a solicitação será adaptada ao *software* existente e outras pessoas passarão a usufruir. Como exemplo, imagine que o Banco Central solicite aos bancos a possibilidade de o cliente bancário ter acesso ao seu extrato do cartão de crédito desde o primeiro dia de aquisição. Essa demanda, caracterizada como legal, será avaliada e em seguida construída. Logo após o término, os incluindo testes e a homologação, ela é colocada em produção. Nesse momento, todos os clientes bancários que possuam cartões terão a possibilidade de emitir seu extrato, desde o primeiro dia de utilização.

Sob o ponto de vista de avaliação técnica, a demanda parece simples, mas provavelmente teremos alguns riscos. Como por exemplo, imagine clientes com mais de 20, 30 anos de cartão de crédito no mesmo banco. O cliente poderá tirar o extrato desses anos todos. Muito provavelmente teremos uma análise bem detalhada para não prejudicar o funcionamento dos sistemas, que certamente sofrerão impactos com essa mudança.

10

3 - PROCESSOS DE DESENVOLVIMENTO DE *SOFTWARE*

Embora pareça que estamos sempre revisitando os mesmos assuntos, vale salientar que a evolução dos processos depende de sua repetição constante. Os projetos sempre estarão utilizando algum tipo de processo, mesmo que esse processo não esteja documentado ou seguindo algum método já conhecido e consolidado.

Cada projeto executado, ainda que utilize um mesmo método, terá comportamentos diferenciados. Por isso vale sempre lembrar a essência de cada processo.

Nesse contexto entramos novamente nos processos de desenvolvimento de *software* já descritos no curso, como o processo Orientado ao Objeto, processo de desenvolvimento Cascata e o Processo Unificado.

- **Processo orientado a objetos**

Sendo assim, o desenvolvimento de *software* orientado a objetos refere-se à organização mais alta de um projeto de software, podendo ser decomposto em fases intermediárias que, por sua vez, podem ser decompostas em *workflows* e depois em atividades.

Artefatos de um workflow são insumos para outros *workflows*, isto quer dizer que, na prática, sai de um workflow para entrar em outro. O produto final deve ser o *software* construído ou alterado, tudo de acordo com a solicitação do cliente.

Para o processo de desenvolvimento de *software* OO é necessário que executemos alguns sub processos, tais como:

- Levantamento de requisitos;
- Análise de requisitos;
- Arquitetura;
- Design;
- Implementação;
- Testes;
- Implantação.

11

- **Processo de desenvolvimento Cascata**

Já no modelo cascata tem os processos tradicionais de desenvolvimento de *software*, sendo executados sequencialmente. Sequencialmente significa iniciar um bloco de tarefas seguintes apenas quando o bloco de tarefas anteriores estiver totalmente finalizadas.

O modelo cascata ainda pode ser sugerido, mas para projetos pequenos, onde não existe possibilidade do gestor alterar o escopo de sua solicitação, o que na prática é totalmente improvável.

- **Processo Unificado de desenvolvimento de *software***

O processo Unificado é um processo de desenvolvimento de *software* que utiliza como framework o RUP (Processo Unificado Rational). O RUP é a junção de vários *workflows* e templates para a empresa utilizar no seu desenvolvimento de demandas.

Como visto anteriormente no curso, sobre o processo unificado podemos citar as seguintes características:

- Criado para superar as dificuldades do modelo cascata;
- grandes projetos podem ser quebrados em projetos pequenos com ciclo de vida próprios;
- Utiliza os conceitos de iterativo e incremental;

- Cada pedaço do sistema é feito em uma iteração;
- As iterações segue o modelo sequencial e possui características próprias do projeto todo;
- Cada iteração é incrementada até a última, caracterizando que o *software* está pronto.

12

RESUMO

A palavra processo nos remete a pensar em várias coisas, mas como definida sua finalidade é para atingir uma meta estipulada, de acordo com o seu caminho a ser seguido. O benefício de utilizar um processo definido é enorme, pois define o momento exato de cada ação, de cada membro da equipe.

Na tecnologia temos vários processos de desenvolvimento de sistemas, orientado a objetos, cascata, processo unificado, dentre outros. Ainda temos desenvolvimento sem seguir processo nenhum, que pode ser um desastre.

Todo desenvolvimento começa com uma solicitação de alguém, é a solicitação de demanda. Essa solicitação define o que deve ser feito no projeto ou na manutenção. E a construção dessa demanda segue os ritos de um desenvolvimento: avaliação técnica, desenvolvimento, teste e homologação. São essas fases/etapas que servem para qualquer processo, sendo que cada um tem sua particularidade em si.

UNIDADE 1 – DESENVOLVIMENTO DE SOFTWARE PARA VALORIZAÇÃO DO NEGÓCIO

MÓDULO 4 – ROTEIRO DE DESENVOLVIMENTO DE SOFTWARE

01

1 - PROCESSOS DE DESENVOLVIMENTO DE *SOFTWARE*

Existe no mercado uma série de sugestões de roteiros para desenvolvimento de *software*. Trataremos nesse módulo do roteiro baseado em desenvolvimento de *software* orientado a objetos.

Segundo o professor e escritor Hélio Enghotm Jr., o desenvolvimento de *software* pode ser visto como uma série de transformações que se iniciam no modelo mental dos envolvidos no projeto.

O ponto de partida de alguns projetos é um artefato conhecido no mercado como **Declaração de Trabalho**, que mostra em alto nível os objetivos e macrorrequisitos do projeto. Esse documento é originado pelo patrocinador ou iniciador do projeto e normalmente descreve as necessidades, os produtos ou serviços que deverão ser providos como resultado final do projeto.

Em projetos onde temos relacionamento com mais de uma empresa, esse documento pode ser recebido pelas empresas de desenvolvimento de *software* como parte de um processo de contratação. Em alguns casos a Declaração de Trabalho pode ser vista como uma proposta de melhoria.

Se existente esse documento elaborado antes de o projeto ser iniciado, ele deve ser considerado como artefato de entrada no processo de levantamento e refinamento de requisitos.

Assim que o documento for aprovado e todos os aspectos de contratação forem formalizados, é possível iniciar o trabalho, ou seja, dá-se início ao processo de desenvolvimento do *software*.

02

Como já visto, os processos de desenvolvimento de *software* tradicionalmente incluem os seguintes subprocessos:

a. Elicitação de requisitos;
b. Análise dos requisitos;
c. Arquitetura;
d. Design;
e. Implementação;
f. Testes;
g. Implantação.

Roteiro para o desenvolvimento de *softwares*

Revisaremos, a seguir, alguns pontos importantes de todas essas etapas.

03

a) Elicitação de requisitos

O levantamento de requisitos ou elicitação está na fase inicial de qualquer sistema, e é extremamente importante para o sucesso do projeto. Como vimos em módulos anteriores, essa fase determina o que deve ser feito, esclarece e dá clareza no entendimento de todos. Com isso é possível realizar um planejamento mais próximo da realidade. Além do planejamento, o levantamento de requisitos possibilita a realização de um estudo de viabilidade com as estimativas de custos. É fundamental para todos do projeto o entendimento e controle desses requisitos para que o projeto possa obter sucesso e atender aos objetivos e as necessidades do cliente.

A elicitação de requisitos pode ser uma árdua tarefa quando o cliente não tem tempo ou não é dada certa importância nessa atividade pela empresa. Como vimos anteriormente no curso, existem várias técnicas de levantamento que os analistas podem utilizar para conseguir extrair as informações necessárias para prosseguir com o andamento do projeto.

No levantamento dos requisitos descobrimos os **requisitos funcionais e não funcionais**. Os requisitos funcionais constituem as funcionalidades que o sistema terá e estão escritos nos casos de uso. Os requisitos não funcionais têm o objetivo de determinar a complexidade e o tamanho do que vamos construir. O documento que registra os requisitos não funcionais em alguns locais é chamado de especificação suplementar.

O analista de requisitos deve atualizar os atributos de requisitos e matrizes de rastreabilidade definidas na estratégia de gerenciamento, conforme o plano de gerenciamento de requisitos.

Casos de uso
Os casos de uso possuem vários itens no seu artefato como:

- Pré-condições e pós-condições;
- Fluxos de operação;
- Entradas e saídas;
- Interação do sistema com as interfaces internas, externas e usuários;
- Regras de negócio e de segurança;
- Exceções.

Esses artefatos com esses itens são chamados de **especificação de caso de uso**.

Especificação suplementar
A especificação suplementar, que abrange os requisitos não funcionais, deve conter os tópicos abaixo:

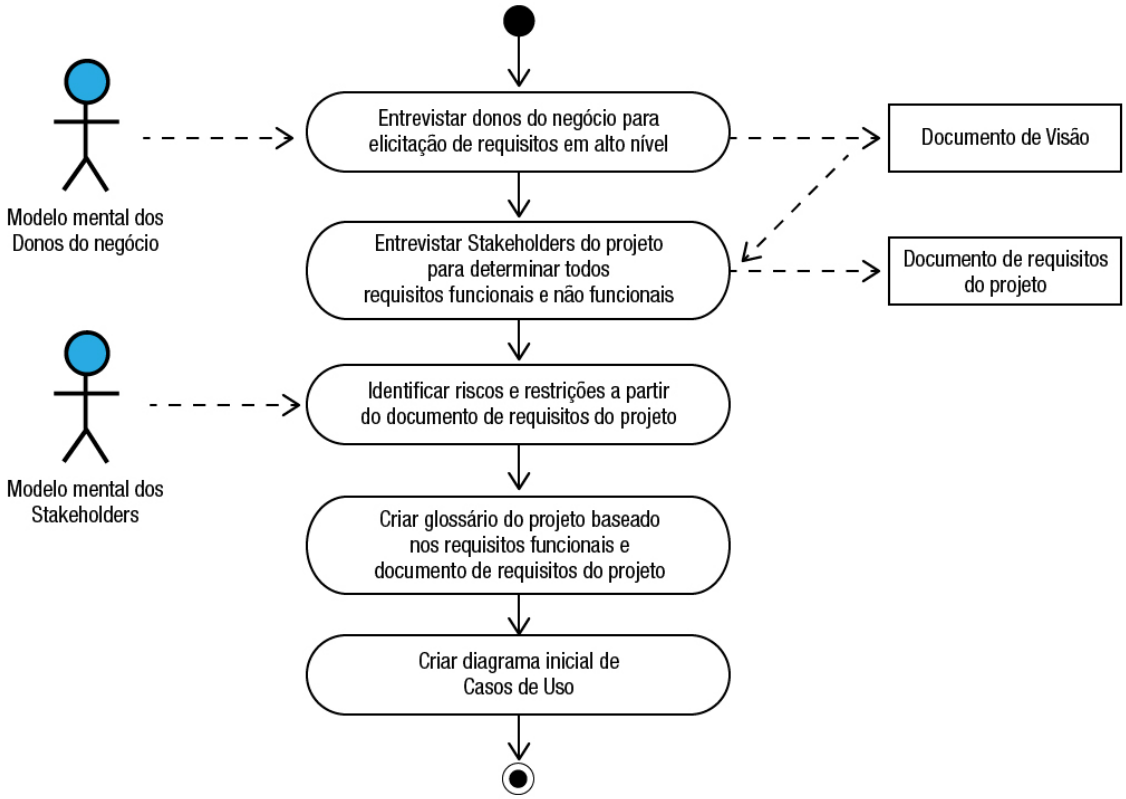
- **Usabilidade**: existência de padrões especiais para a interface, requisitos que enfatizam o aumento da eficiência do usuário final;
- **Confiabilidade**: o sistema deve prover facilidades de operação, como recuperação automática de erros, e apresentar dados fidedignos;
- **Desempenho**: descrever em que nível os requisitos estabelecidos pelo usuário influenciam o projeto, o desenvolvimento, a instalação e o suporte da aplicação;
- **Suportabilidade**: existência de processamento distribuído, diferentes sistemas operacionais ou

plataformas, especificação de diferentes protocolos de comunicação;

- **Restrições de projeto**: aplicação com módulos que precisam ser reutilizadas em outros sistemas, causando esforços adicionais ao projeto. É comum em grandes empresas o reaproveitamento de códigos ou a padronização de componentes de *software*;
- **Requisitos de implementação**: restrição do código ou da construção do sistema. É comum em grandes empresas a padronização da construção dos códigos, como palavras chaves, padronização de nomes de campos, atributos dos bancos de dados;
- **Requisitos de interface**: especificar características especiais para a integração com outros sistemas. Sistemas de grandes empresas geralmente se relacionam, isso sempre é um fator complicador em um projeto;
- **Requisitos físicos**: esse tipo de requisito pode ser usado para representar requisitos de hardware, como as configurações físicas de rede obrigatórias. Questões de segurança podem influenciar as configurações físicas de rede. Grandes empresas separam a rede interna da rede dos fornecedores, isso geralmente causa transtornos nas entregas de produtos de *software*s. Quando a empresa tem esse problema de comunicação devido a redes diferentes, ela acaba indo desenvolver dentro da empresa do cliente.

04

Segundo Enghotm Jr., a etapa de elicitação de requisitos pode ser sintetizada nos seguintes passos:



05

b) Análise de requisitos

Nessa fase procura-se entender o problema a partir da perspectiva do cliente, sem preocupações relacionadas à tecnologia que será utilizada. Determina-se nessa fase o *que* deve ser feito sem se preocupar com o *como* será feito. Utilizando os casos de uso conseguimos verificar o que deve ser feito. Não se considera nessa fase a linguagem de desenvolvimento que será utilizada para construir o sistema.

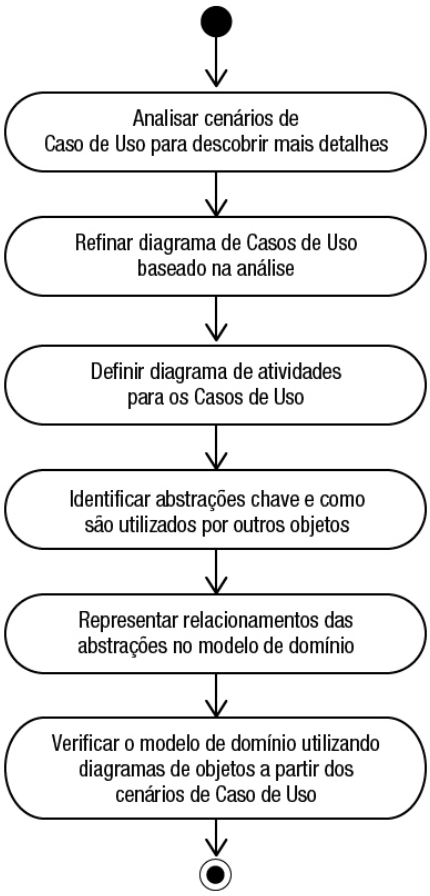
Podemos definir os seguintes *objetivos* para essa fase:

- Estabelecer uma visão clara do problema do cliente;
- Verificar as tarefas que o sistema deve contemplar;
- Entender o vocabulário do cliente;
- Levantar os recursos existentes no cliente;
- Verificar a melhor solução para o problema do cliente;
- Definir as tarefas que devem ser realizadas.

Utilizamos o documento especificação de caso de uso para detalharmos e realizarmos essa fase de análise.

Na fase de análise atualizaremos as informações relacionadas aos cenários de caso de uso. Nessa fase é realizado um teste nos cenários levantados, o qual possibilita perceber o comportamento de cada caso de uso de acordo com o cenário. Por exemplo, imagine um caso de uso realizar uma locação, para que essa funcionalidade funcione deve haver o cenário que tenha o livro para ser alocado e o cenário que não tenha o livro para ser locado. Temos, assim, que mapear cada cenário que o sistema terá.

De acordo com Enghotm Jr., a etapa de análise dos requisitos pode ser resumida nos seguintes passos:



06

c) Processo de arquitetura

As mudanças relacionadas à engenharia de *software* e à evolução de tecnologia aumentou a importância da arquitetura do sistema, a qual possibilita a garantia de escalabilidade e do desempenho da aplicação a ser construída. Lembrando que escalabilidade é a capacidade que o sistema possui de crescer atendendo às demandas sem perder as qualidades que lhe agregam valor. Essa característica se tornou essencial para as empresas que trabalham com tecnologia da informação.

A arquitetura se preocupa em projetar os requisitos funcionais, mas principalmente os requisitos não funcionais. Nessa fase, o arquiteto de *software* deve propor uma arquitetura que suporte o sistema, olhando em um possível crescimento. Mas, para isso, ele deve considerar alguns aspectos:

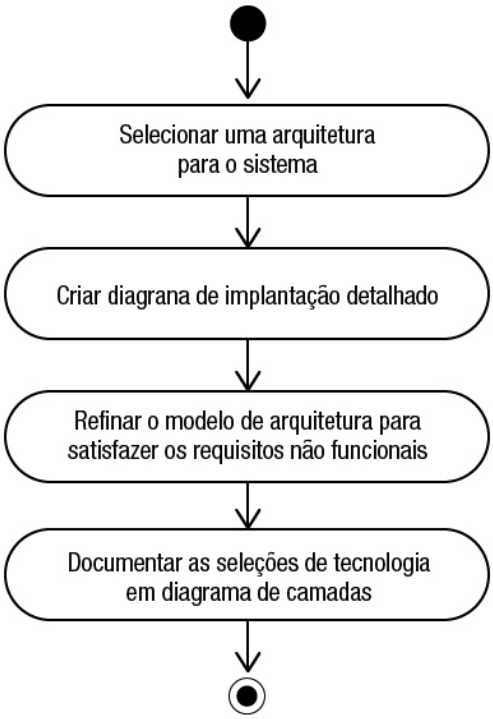
- Onde a empresa pretende hospedar o sistema;
- Plataformas utilizadas pela empresa;
- Tipo de plataforma onde se deseja utilizar no sistema;
- Sistemas legados com os quais o sistema novo terá algum tipo de comunicação;
- Tipo de interface que se deseja utilizar.

A arquitetura deve garantir a confiabilidade, desempenho e escalabilidade do sistema. Cabe ao arquiteto de *software* realizar uma análise de riscos e garantir que o projeto a ser desenvolvido cumpra os requisitos não funcionais do sistema a ser desenvolvido.

Em alguns casos, a empresa corre o risco de não atender a todas as exigências do projeto. Por exemplo: o cliente pode querer que seu sistema fique 24 horas no ar. Essa solicitação não é problema para os dias atuais, mas a estrutura do sistema deverá prever esse requisito.

07

Enghotm Jr. Define que a etapa de arquitetura pode ser resumida nos seguintes passos:



08

d) Processo de design

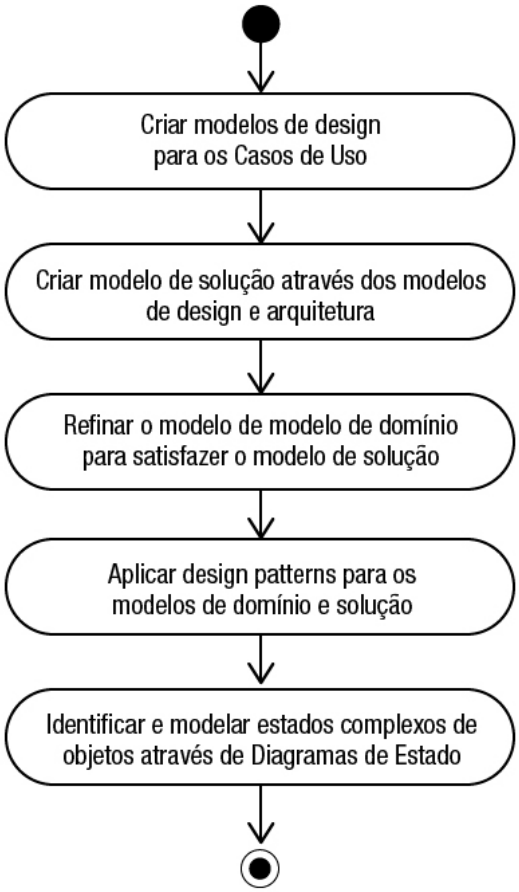
Nesta fase deve-se caracterizar o sistema em termos dos requisitos de tecnologia, escolhendo o que melhor atende as necessidades do projeto. Nesse momento definimos os detalhes da tecnologia, linguagens, bancos de dados, parâmetros, instâncias etc.

A fase de design deve atender os **objetivos** abaixo:

- Resolver o problema do cliente.
- Verificar elementos de suporte que farão o sistema funcionar;

- Definir uma estratégia para implementar o sistema;
- Produzir especificações detalhadas do sistema;
- Construir os casos de testes.

Segundo Enghotm Jr., a fase de design pode ser resumida nos seguintes passos:



09

e) Implementação

Nessa fase o processo de construção pode ser visto como um processo de desenvolvimento dos produtos de *software* a serem utilizados pelos clientes finais, os usuários.

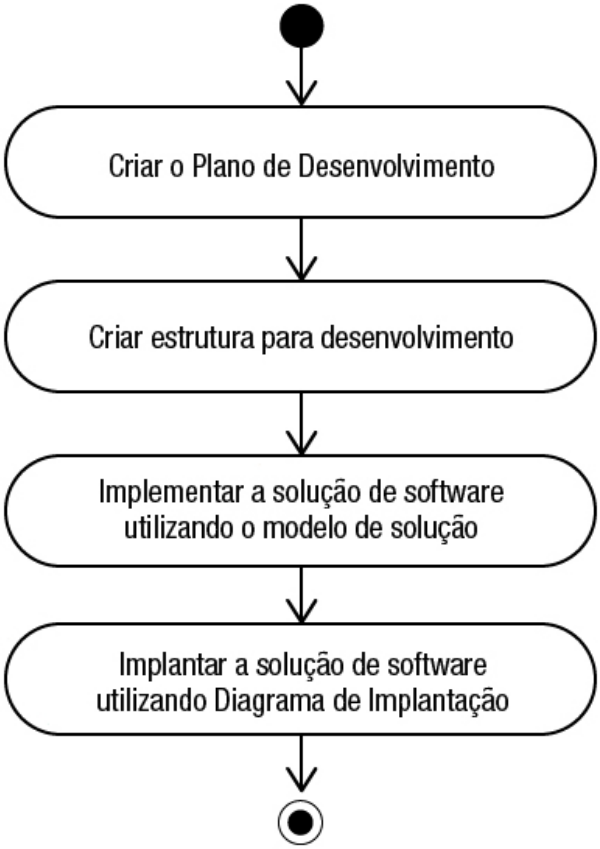
Temos os seguintes **objetivos** nessa fase:

- Desenvolver código com qualidade, no menor custo e no menor prazo. A Qualidade nada mais é que o esperado pelo cliente em sua solicitação;
- Desenvolver código de maneira eficaz e com rapidez, com a menor quantidade de erros possíveis. Os testes no desenvolvimento visam pegar os erros, antes de irem para o cliente

homologar a aplicação;
- Atender a todas as especificações contidas nas especificações e diagramas construídos;
- Desenvolver de modo interativo e incremental o produto completo a ser implantado em produção.

Para que a equipe de desenvolvimento comece os trabalhos de construção deve primeiramente montar um ambiente para isso, o **ambiente de desenvolvimento**, que é onde se criam os códigos e são realizados os primeiros testes da aplicação.

De acordo com Enghotm Jr., a fase de construção pode ser resumida nos seguintes passos:



10

f) Testes

Essa fase é importantíssima para o projeto, pois garante que o pacote de *software* gerado esteja de acordo com os requisitos especificados e ao uso solicitado. Em suma, o teste verifica se a aplicação está funcionando e fazendo o que o cliente solicitou.

Para que tudo ocorra corretamente, é importante a elaboração de um plano de testes a ser seguido pela equipe de testes. No planejamento, o gerente de projeto com o analista de testes deverá definir a missão e os motivadores para os testes, que serão realizados no projeto. Eles devem definir uma lista dos principais itens que serão sujeitos a teses no documento de estratégia de testes.

Com base no estágio do ciclo de vida do projeto, os testes serão realizados pelo analista de testes, que definirá o tipo e a técnica a ser aplicada a cada produto ou componente de produto que testará. As abordagens de testes podem ser registradas no documento de estratégias de testes pelo analista de testes.

11

Em grandes empresas de tecnologia e grandes bancos existem vários ambientes de testes. Além do desenvolvimento, quando ocorrem os primeiros testes, é interessante que a empresa tenha um ambiente similar ao do cliente, assim evitará o retrabalho ou idas e vindas ao cliente e à empresa para corrigir o código.

O ambiente de desenvolvimento é feito para que a aplicação seja construída e geralmente não há problemas ou regras de antivírus ou firewall. Assim, nem tudo o que funciona no desenvolvimento vai funcionar no cliente.

O analista de teste, então, deve executar os testes previstos nos planos de testes do projeto, sempre registrando os resultados no documento de evidência de testes. Esse documento comprova junto ao cliente que os testes foram realizados. Na atividade de teste o analista deve informar os eventuais erros na aplicação, de modo que o gerente de projetos possa planejar as correções e acompanhar a resolução dos erros encontrados.

Após os testes, é prudente analisar os resultados e verificar as ações que possam contribuir para a melhoria do processo de testes. A análise da execução dos testes pode levar a empresa a realizar mais treinamentos do processo ou até mesmo corrigir pontos desnecessários no processo.

Na imagem temos os vários pacotes que foram desenvolvidos e testados (testes unitários), formando o produto final. Depois deverão ser testados novamente todos os produtos entregues, mas de forma integral (teste integral) e o todo deverá funcionar conforme solicitação do cliente. Os erros encontrados devem ser corrigidos antes de o produto ser entregue ao cliente e antes de ser disponibilizado aos usuários. Após todo esse processo, o produto está homologado e implantado.

12

g) Implantação

Após todos os testes realizados, os primeiros pacotes de *softwares* vão sendo entregues, lembrando que estamos em um processo iterativo e incremental. O sistema vai ganhando corpo com as funcionalidades novas, até que tenhamos todo o produto disponível para o cliente, que disponibilizará para seus usuários.

A implantação de pacotes iniciais está de acordo com o planejamento, que define quais funcionalidades serão construídas primeiramente. Segundo o RUP, os primeiros pacotes devem contemplar a arquitetura do sistema, e os próximos pacotes deve contemplar as funcionalidades que o cliente tanto deseja.

13

2 - PAPÉIS ENVOLVIDOS NO PROJETO DE DESENVOLVIMENTO DE *SOFTWARE*

Nos processos de desenvolvimento de *software* há o envolvimento do dono do negócio, de alguns gerentes e dos usuários do *software*. Do lado da empresa que irá desenvolver o projeto de *software* ou que vai dar a manutenção no sistema, temos:

- o gerente de projetos,
- analistas de negócios,

- analistas de requisitos,
- arquitetos,
- desenvolvedores,
- analistas da qualidade,
- testadores etc.

Os papéis variam conforme a estrutura e o processo seguido pela empresa desenvolvedora.

Dependendo da empresa, alguns desses papéis atuam em algum momento junto ao cliente (gerentes e dono do negócio), podendo ser os analistas da qualidade, gerente de projetos ou até mesmo o desenvolvedor do produto. Como elencado acima, tudo depende de como a organização está aparelhada.

É importante frisar a importância de se trabalhar com o **analista de negócios em projetos**. Existem projetos em que a participação de um analista de negócios é extremamente e fundamental para o sucesso do projeto.

Projetos de grandes bancos, que possuem vários serviços e alguns complexos, como cobrança bancária, têm regras de negócio que devem ser tratadas com um analista dedicado e que passe a entender do negócio, além do cliente. Para esse papel é necessário um especialista que acompanhe desde o levantamento de requisitos até a fase de análise do projeto, podendo ser envolvido posteriormente para realizar o plano de teste da aplicação.

14

Linguagem Unificada de modelagem

Vimos nesse módulo as fases de um processo de desenvolvimento, que podem ser utilizadas como base para qualquer tipo de projeto de *software*, por qualquer empresa ou equipe de projeto. Poderemos utilizar, também, para apoiar os projetos a UML e o mais importante: vai facilitar o processo de manutenção.

A UML fornece diagramas de modelagem que podem ser utilizados em diversos processos de desenvolvimento de *software*. Você já deve ter estudado esses diagramas em outro momento, então iremos apenas listá-los abaixo, a título de lembrete. São eles:

Diagrama de caso de uso	Utilizado na fase de levantamento de requisitos, análise e design, onde é possível observar as funcionalidades previstas para o desenvolvimento do projeto, bem como os usuários (atores) que utilizarão as funcionalidades do sistema.

Diagrama de classes	Pode ser utilizado na fase de análise, onde se identificam as possíveis classes. O diagrama de classes registra o modelo de domínio da aplicação, o qual contém o relacionamento dos objetos de dados com o sistema. O diagrama mostra cada objeto em detalhe.
Diagrama de interação	Pode ser utilizado na fase de levantamento de requisitos e na fase de design, na qual apresenta o relacionamento entre os objetos do sistema. O diagrama auxilia a equipe de testes na construção de planos e execuções de testes, onde é possível perceber os objetos se interagindo.
Diagrama de atividades	Pode ser utilizado na fase de levantamento de requisitos e na fase de design, serve para identificar os fluxos do processo dentro do sistema. O diagrama apresenta os usuários e as atividades que executarão, permitindo descobrir os perfis de usuários que o sistema irá precisar.
Diagrama de implantação	Elaborado na fase de design para indicar como o sistema deverá ser implantado e como os recursos serão implantados no hardware.

15

RESUMO

Nos dias atuais possuímos uma variedade de roteiros para se desenvolver um *software*, a Orientação a Objetos tem evoluído e vem sendo usado em vários países. Assim o processo ao desenvolvimento de *software*, orientado a objetos, utiliza vários processos, que suportam todo o desenvolvimento, como Elicitação de requisitos, Análise de requisitos, Arquitetura, Design, Implementação, Testes e Implantação. Cada fase possui vários papéis envolvidos, ou poderíamos dizer, pessoas que executam uma determinada ação dependendo do ciclo de vida do projeto.

Importante frisar que o desenvolvimento de *software* somente existe porque o cliente solicitou, e a entrega do que ele solicitou é testada e homologada antes de colocar em produção.