MÓDULO DE:

ENGENHARIA de SOFTWARE

AUTORIA:

Ms. CARLOS VALENTE

SUMÁRIO

UNIDADE 1

O que é Engenharia de Software ? - Qual a diferença entre Engenharia de Software e Engenharia de Sistemas ? - O que é um método de Engenharia de Software ?

Objetivo: Conceituar a Engenharia de Software, apresentar diferenças e definir método.

A Engenharia de Software, conforme Sommerville, um dos papas dessa área, é uma disciplina da engenharia que se ocupa de todos os aspectos da produção de software. Isso vai desde os estágios iniciais de especificação de um Sistema, até propriamente a Manutenção para que esse mesmo Sistema sobreviva ao longo do tempo.

A construção de software é uma das atividades mais complexas e vitais para o pleno sucesso de um Sistema informatizado. A Engenharia de Software justamente tenta, através dos princípios básicos de outras engenharias, trazer um pouco mais de luz para essa atividade complexa.

A "cobrança" hoje das áreas de Informática e de T.I. (Tecnologia da Informação) é desenvolver Sistemas de forma rápida, com qualidade, e com custos cada vez menores. Somente através de tecnologias adequadas, e com as melhores práticas, podemos atender a esses novos desafios.

A Engenharia de Software é constituída de Metodologias, Métodos e Ferramentas que permitem ao profissional especificar, projetar, implementar e manter Sistemas, avaliando e garantindo as qualidades especificadas pelos usuários.

Engenharia de Sistemas

A Engenharia de Sistemas é mais genérica e mais abrangente do que a Engenharia de Software. Na verdade, a segunda faz parte da primeira. A Engenharia de Sistemas é mais antiga do que a de Software. Enquanto a primeira está mais envolvida com o Sistema como um todo e seus detalhes, a Engenharia de Software é mais específica no que tange aos componentes do sistema, em especial ao hardware e software.

Método de Engenharia de Software

Sommerville afirma que um método de Engenharia de Software é uma “abordagem estruturada” para o desenvolvimento de software. Podemos definir como “abordagem estruturada” a estratégia de desenvolver algo com uma estrutura previamente estudada, ou baseada nas melhores práticas. O objetivo maior de tudo isso é facilitar a produção, em curto espaço de tempo, de software de alta qualidade, apresentando uma relação custo-benefício interessante.

Um ponto importante a observar é que não existe, repito, não existe um método ideal. As possibilidades e os ambientes de desenvolvimento são tão complexos, que dependendo de cada situação e momento, existe um método que possa explorar mais alguns tópicos, mas deixará outros em aberto.

Outro ponto a ressaltar é que existem vários métodos na Engenharia de Software, mas poucas Metodologias. Podemos entender Metodologia tanto pelas palavras de Maddison, como sendo um conjunto recomendado de filosofias, fases, procedimentos, técnicas, regras, ferramentas, documentação, gerenciamento e treinamento para o desenvolvimento de um sistema de informação, como também o estudo de um ou mais métodos.

No início da computação poucos programadores seguiam algum tipo de metodologia baseando-se, em sua maioria, na própria experiência. Na Engenharia de Software existem basicamente duas grandes metodologias. Uma originária da década de 70, chamada de Metodologia Estruturada, e a mais recente intitulada de Metodologia Orientada a Objetos.

Diferenças das Metodologias

Tanto a abordagem estruturada quanto a orientada a objetos promovem soluções práticas. A diferença entre as duas metodologias é a vida útil e facilidade de manutenção de projetos. A possível reutilização de um código estruturado não é comum, enquanto que um código orientado a objetos por possuir embutido em sua própria filosofia as facilidades de reutilização e de descrição, utilizando UML (Unified Modeling Language), aumenta naturalmente a vida útil dos códigos.
Abordando o software sob um ponto de vista puramente estruturado, definem-se os dados do sistema em uma posterior sequência de eventos que acarretará na transformação do estado do sistema.

Por outro lado, numa abordagem focada em orientação a objetos, definem-se estruturas abstratas, denominadas classes, que serão responsáveis por partes da solução do problema. Cada classe incorporará dada (forma) e métodos (comportamentos). Projetos orientados a objetos utilizam da linguagem de modelagem UML (Unified Modeling Language). Esta linguagem é fruto dos esforços, em conjunto, dos autores Booch, Rumbaugh e Jacobson.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Engenharia_de_software
http://pt.wikipedia.org/wiki/Metodologia_(engenharia_de_software)

Ouça um PODCAST sobre METODOLOGIAS:
http://www.improveit.com.br/podcasts/quem-se-importa-com-metodologia.mp3

Atividades

Responda, por escrito, as seguintes perguntas:

- O que é Engenharia de Software?
- Qual a diferença entre Engenharia de Software e Engenharia de Sistemas?
- O que é um método de Engenharia de Software?

UNIDADE 2

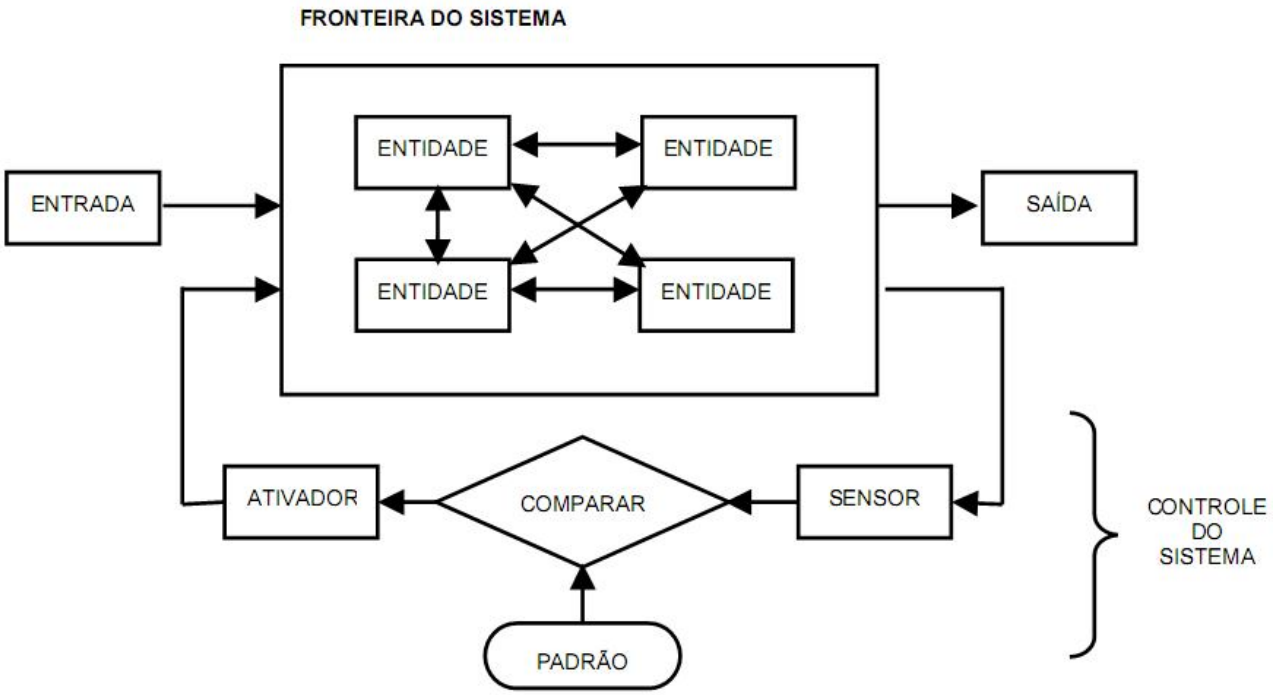
Teoria de Sistemas - Interdependência de Sistemas

Objetivo: Visualizar a interação que os sistemas possuem entre si.

Um Sistema é uma coleção significativa de componentes inter-relacionados, que trabalham em conjunto para atingir alguns objetivos (Sommerville). É organizado para executar certo método, procedimento ou controle ao processar informações. Automatiza ou apoia a realização de atividades humanas através do processamento de informações.

As complexas relações entre os componentes de um sistema significam que o sistema em si é maior do que simplesmente a soma de suas partes.

As Arquiteturas de Sistema são normalmente descritas com o uso de Diagramas de Blocos, mostrando os subsistemas principais e suas relações. Veja a figura abaixo como um exemplo desse conceito:



Nós encontramos, nessa imagem, os elementos de ENTRADA e de SAÍDA do SISTEMA. E na parte interna do SISTEMA a composição da inter-relação de várias ENTIDADES. Na parte

inferior do SISTEMA temos um importante conceito de feed-back chamado de CONTROLE do SISTEMA.

Inicia-se esse controle com um PADRÃO de comparação. Através de um SENSOR que mensura periodicamente as alterações do SISTEMA, compara-se com o PADRÃO. Uma vez o SISTEMA sofrendo alterações em relação ao PADRÃO, o ATIVADOR irá passar parâmetros para o SISTEMA se autocorrigir.

Um bom exemplo prático deste conceito é imaginarmos o ar condicionado. Parte-se inicialmente da base de uma temperatura estabelecida por nós (PADRÃO). O sensor mensura as variações de temperatura. E o ATIVADOR irá deixar o ar condicionado ativado até novamente o SENSOR verificar que a temperatura está no PADRÃO desejado.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Teoria_de_sistemas

Atividades

Acesse o documento no site abaixo e veja maiores detalhes da interessante "Teoria de Sistemas":

http://www.abdl.org.br/filemanager/download/4/teoria%20de%20sistema.pdf

UNIDADE 3

Quais são os atributos de um bom Software ? - Quais são os desafios enfrentados pela Engenharia de Software ?

Objetivo: Definir atributos de software e contextualizar a Engenharia de Software.

Os atributos de um bom software refletem seu comportamento quando em funcionamento, a estrutura e a organização do programa fonte, e também a documentação associada (Sommerville). Como exemplos, temos o tempo de resposta do software à consulta de um usuário e a facilidade de compreensão do código do programa. Esses mesmos exemplos também podem ser chamados de atributos não funcionais.

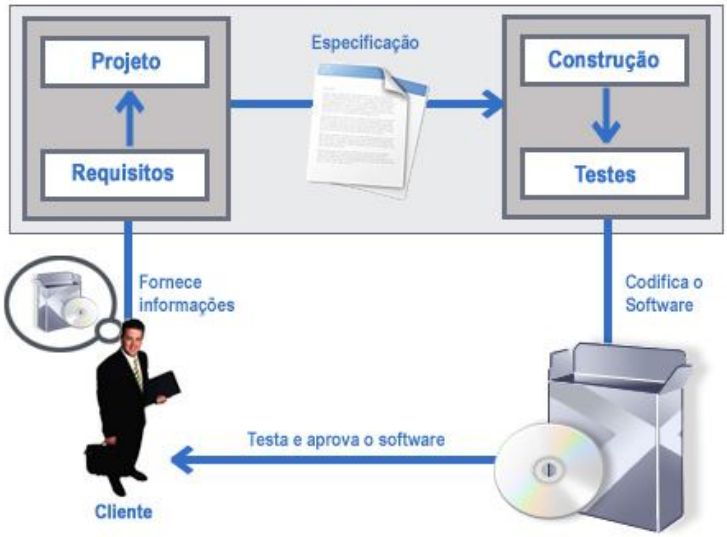
Resumidamente o software deve proporcionar ao usuário a funcionalidade e o desempenho requeridos e deve ser passível de manutenção, confiável, eficiente e de fácil uso (veja mais detalhes no quadro abaixo).

ATRIBUTOS	Descrição
Facilidade de Manutenção	O software deve ser escrito de modo que possa evoluir para atender às necessidades mutáveis dos clientes. Esse é um atributo crucial, porque as modificações em um software são uma consequência inevitável de um ambiente de negócios em constante mutação.
Nível de Confiança	O nível de confiança do software tem uma gama de características que incluem confiabilidade, proteção e segurança. O software confiável não deve ocasionar danos físicos ou econômicos, no caso de um defeito no sistema.
Eficiência	O software não deve desperdiçar os recursos do sistema, como memória e ciclos do processador. A eficiência, portanto, inclui a rapidez de resposta, o tempo de processamento, a utilização da memória, entre outros.
Facilidade de Uso	O software deve ser utilizável, sem esforços indevidos, pelo tipo de usuário para quem foi projetado. Isso significa que ele deve dispor de uma interface apropriada com o usuário e de documentação adequada.

Atributos essenciais de um bom software (adaptado de Sommerville)

Crise do Software e o início da Engenharia de Software

A crise do software, termo usado nos anos 70, referia-se às dificuldades do desenvolvimento de software da época. Por haver um rápido crescimento da demanda por software, imaginava-se que com toda a complexidade no desenvolvimento, haveria uma forte crise. Com a inexistência da Engenharia de Software nessa época, não havia técnicas estabelecidas para o desenvolvimento de sistemas que funcionassem adequadamente ou que pudessem ser validadas.



Já em 1988, AMBLER afirmava: “Desenvolver software não é somente modelar e escrever código. É criar aplicações que resolvam os problemas dos usuários. É fazer isto dentro do prazo, de forma precisa e com alta qualidade”. Logo, com a crise de software, os desafios para a criação da disciplina de Engenharia de Software eram muito grandes.

Alguns dos típicos problemas que essa nova disciplina enfrentou foram:

- Identificar adequadamente os requisitos do Sistema, ou seja, saber o que o software deve fazer;
- Que ferramentas, linguagem, sistema operacional usar;
- Como diminuir os tempos e custos de desenvolvimento;
- Prever falhas antes da entrega final;
- Como fazer manutenção e controlar versões;
- Dificuldades de prever o progresso durante o desenvolvimento;
- Inexistência de histórico, ou documentação, no desenvolvimento de Sistemas;
- Comunicação com os usuários precária;
- Conceitos quantitativos inexistentes tais como confiabilidade, qualidade e reusabilidade;
- Manutenção, no software existente, com difícil execução.

Esse início difícil da Engenharia de Software, com tantos desafios, gerou vários paradigmas e modelos de desenvolvimento. Iremos ver nas próximas unidades quais foram as formas que a engenharia clássica veio a ajudar nesse difícil início.

Desafios da Engenharia de Software

Atualmente os principais desafios da Engenharia de Software, conforme Sommerville são:

DESAFIOS	Descrição
O desafio do legado	Ainda os grandes sistemas de software existentes foram desenvolvidos no passado, e com importantes funções corporativas. O desafio é fazer a manutenção e atualização desses softwares a custos baixos e com qualidade.
O desafio da heterogeneidade	Os sistemas exigem em ambientes distribuídos por redes de diferentes tipos de computadores e sistemas de apoio. O desafio é desenvolver técnicas para construir softwares flexíveis para lidar com a heterogeneidade.
O desafio do fornecimento	Nos dias atuais existe uma demanda enorme de sistemas que sejam desenvolvidos no menor tempo possível e com facilidade de adaptação. O desafio é fornecer sistemas cada vez maiores e complexos com a qualidade desejada, e em curto espaço de tempo.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Crise_do_software

Atividades

Responda, por escrito, as seguintes perguntas com os seus próprios comentários a respeito:

- Quais são os atributos de um bom Software?
- Quais são os desafios enfrentados pela Engenharia de Software?

UNIDADE 4

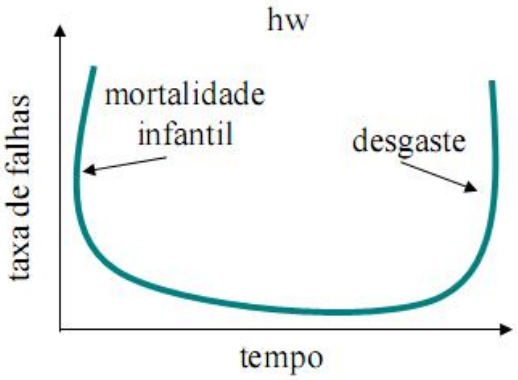
Conceitos sobre a Engenharia de Software - O que é Software ? A Importância do Software - SWEBOK

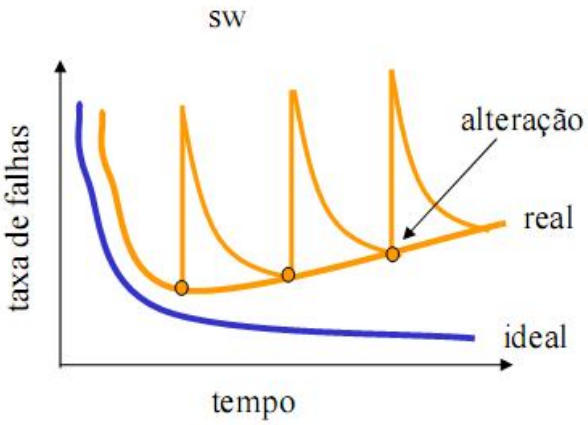
Objetivo: Abordar a Engenharia de Software e os seus principais tópicos (visão geral).

A Engenharia de Software basicamente tenta apresentar processos, ferramentas e métodos que permitam desenvolver de forma racional e controlável um Sistema Computacional. Todo o foco é a Qualidade, utilizando um método eficaz e o uso de ferramentas adequadas.

Características do software

- É desenvolvido/projetado por engenharia, não é fabricado
- Não se "desgasta", mas deteriora!! Veja a figura abaixo o comparativo entre a taxa de falhas do hardware com as de software





- Ainda hoje a maioria é feita sob encomenda em vez de ser montada a partir de componentes

Tipos de Software

- Tempo real
- Software básico
- Sistema de informação
- Embutido
- Técnicos
- Especialistas
- Apoio à decisão
- Jogos
- Apoio (processador de textos)

Mitos do Software

1 - “Já temos um manual repleto de padrões e procedimentos para a construção de software. Isso já é suficiente para o pessoal do desenvolvimento”.
2 - “Meu pessoal tem ferramentas de última geração, afinal de contas compramos os mais novos computadores”.
3 - “Se nós estamos atrasados nos prazos, podemos adicionar mais programadores e tirar o atraso”.
4 - “Uma declaração geral dos objetivos é suficiente para se começar a escrever programas, podemos preencher os detalhes mais tarde”.
5 - “Os requisitos de projeto modificam-se continuamente, mas as mudanças podem ser facilmente acomodadas, porque o software é flexível”.
6 - “Assim que escrevermos o programa e o colocarmos em funcionamento, nosso trabalho estará completo”.
7 - “Enquanto não tiver o programa funcionando, eu não terei realmente nenhuma maneira de avaliar sua qualidade”.
8 - “A única coisa a ser entregue em um projeto bem-sucedido é o programa funcionando”.

Importância do Software

Um dos pontos fundamentais da importância do software é pelo seu uso cotidiano, aonde praticamente no mundo moderno, inexiste a possibilidade de não utilizá-lo. E o outro ponto é pela manipulação da informação (dado - informação - conhecimento), e quem a tem possui poder.

SWEBOK

O SWEBOK (Guide to the Software Engineering Body of Knowledge) é o documento técnico desenvolvido com o apoio do IEEE (Instituto de Engenheiros Elétricos e Eletrônicos, também popularmente chamado de I3E). Esse documento estabelece uma classificação hierárquica

dos tópicos tratados pela Engenharia de Software, onde o nível mais alto são as Áreas do Conhecimento.

As dez Áreas do Conhecimento tratadas pelo SWEBOK são:

- Requisitos de Software
- Projeto de Software
- Construção de Software
- Teste de Software
- Manutenção de Software
- Gerência de Configuração de Software
- Gerência da Engenharia de Software
- Processo de Engenharia de Software
- Ferramentas e Métodos da Engenharia de Software
- Qualidade de Software

Importante ressaltar as diferenças entre o SWEBOK e o PMBOK. Estaremos detalhando melhor o PMBOK na Unidade 28. Mas, somente mostrando genericamente o diferencial dos dois, é que enquanto o SWEBOK é dirigido especificamente para a Engenharia de Software, o PMBOK é mais generalizado quanto a Gerenciamento de Projetos como um todo.

Estudo Complementar

Wikipédia sobre o SWEBOK
http://pt.wikipedia.org/wiki/Software_Engineering_Body_of_Knowledge

Site do IEEE, e no Brasil
http://www.ieee.org/
http://www.ieee.org.br/

Atividades

Navegue pelo site do SWEBOK (http://www.swebok.org/) e tente baixar gratuitamente o documento PDF que contém todas as especificações técnicas das dez áreas do Conhecimento abordadas por essa importante referência na Engenharia de Software.

UNIDADE 5

Modelos de Processo de Software - Paradigmas do desenvolvimento de Software - Modelo Balbúrdia - Modelo Cascata

Objetivo: Entender os principais modelos de processo de software.

Os Modelos de Processo de Software descrevem basicamente as principais etapas do desenvolvimento de software, desde a produção até a sua própria manutenção. Existem vários Modelos de Processo de Software, mas praticamente todos eles seguem o princípio das três principais macro-etapas:

Requisitos - o analista deve obter respostas a várias perguntas junto aos usuários: O que exatamente se espera que seja feito? Qual a abrangência do software? Quais os limites, ou o escopo do sistema? Por que se faz aquilo daquela forma? Quais as restrições que existem nos procedimentos e dados utilizados? E muitas outras.

Projeto/Desenvolvimento - o analista faz especificações técnicas detalhando a solução criada para atender ao usuário conforme os requisitos anteriores. Os programadores codificam os programas em alguma linguagem de programação. Devem-se testar os programas exaustivamente para atingir um alto nível de qualidade, e após isso liberá-los para o uso.

Implantação/Manutenção - na implantação do software podem ocorrer vários problemas não previstos nas fases anteriores. E a manutenção permanecerá durante toda sua vida útil e pode ocorrer motivada por 03 fatores: a correção de algum problema existente no software, sua adaptação decorrente de novas exigências (internas ou externas da empresa) e algum melhoramento funcional que seja incorporado ao software.

Cabe ressaltar que não existe um consenso sobre o nome mais adequado para **Modelos de Processo de Software**. Os principais autores se referem a esse mesmo conceito com os seguintes nomes:

- Modelos de Ciclo de Vida de Software;
- Modelos Prescritivos de Processo
- Paradigmas do Ciclo de Vida;
- Paradigmas do Desenvolvimento de Software;
- Modelagem do Ciclo de Vida.

Modelo Balbúrdia

Como falamos anteriormente, no início da computação, poucos programadores seguiam algum tipo de metodologia baseando-se, em sua maioria, na própria experiência. Era o que chamamos hoje de **Modelo Balbúrdia**, sistemas desenvolvidos na informalidade sem nenhum tipo de projeto ou documentação.

Nesse modelo, o software tende a entrar num ciclo de somente duas fases: o de implementação e de implantação. E os ajustes ao software para atender aos novos requisitos, sempre são em clima de urgência e de stress, motivados por vários fatores, e principalmente por pressão política.

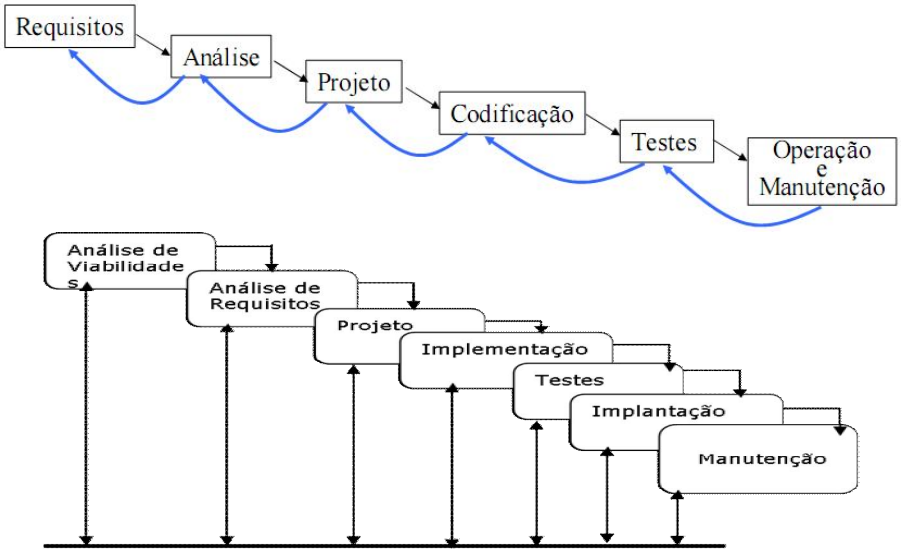
Portanto, havia a necessidade se criar um “Ciclo de Vida” mais inteligente para o desenvolvimento de Software. Ou seja, um “Ciclo de Vida” semelhante à própria natureza, com início, meio e fim bem definidos.

Modelo Cascata

Essa foi a proposta do **Modelo Cascata** (ou do inglês Waterfall), da década de 70. Onde cada etapa do ciclo de vida pressupõe atividades que devem ser completadas antes do início da próxima etapa. Ou ainda, um modelo basicamente de atividades sistemáticas e sequenciais onde para cada etapa cumprida, segue-se a etapa imediatamente posterior, como se fosse uma “cascata”.

O Modelo Cascata é extremamente clássico e antigo, por isso é também chamado de Ciclo de Vida Clássico. Originou-se dos velhos modelos de engenharia na elaboração de projetos. E na verdade, hoje em dia, é somente uma grande referência.

Vivemos num mundo de atividades paralelas, e esse modelo de atividades sequenciais, provocaria demoras excessivas, esperas indesejadas e problemas quando houvesse necessidade de voltar em etapas anteriores.
Repare nas duas figuras abaixo. Embora as duas refiram-se ao Modelo Cascata observe como a terminologia dessas imagens é distinta. Cada etapa praticamente tem um nome diferente em cada figura. Isso ocorre devido a não existir um padrão para esse modelo. Embora sendo clássico, para cada autor existe uma interpretação de cada etapa e é criado um nome distinto.



O próprio Pressman, outro papa da Engenharia de Software, na última edição do seu famoso livro de Engenharia de Software, alterou os nomes da quinta edição, colocando o nome dessas fases respectivamente de: Comunicação, Planejamento, Modelagem, Construção e Implantação.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Processo_de_desenvolvimento_de_software
http://pt.wikipedia.org/wiki/Modelo_em_cascata
http://www.macoratti.net/proc_sw1.htm

Atividades

Com base nas duas últimas imagens dessa unidade, faça uma possível relação entre os nomes das etapas e com a proposta citada pelo Pressman. Ou seja, a primeira etapa da primeira figura seria equivalente a que etapa da segunda imagem, e com a qual do Pressman??

UNIDADE 6

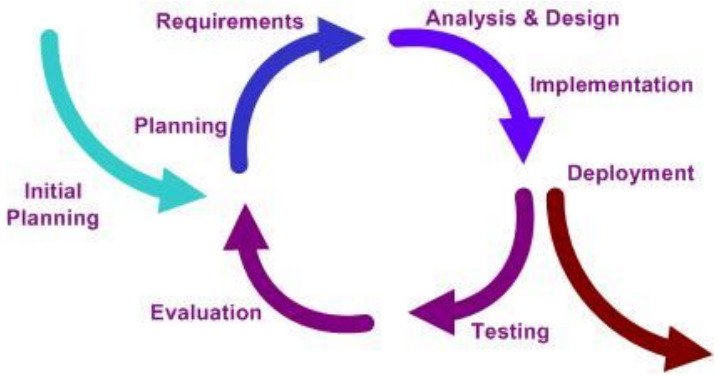
Paradigmas do Desenvolvimento de Software (continuação) - Modelo Incremental - Prototipação

Objetivo: Entender os principais modelos de desenvolvimento de software.

Modelo Incremental

Como vimos anteriormente o tradicional Modelo Cascata é mais um modelo teórico do que prático. Na prática o usuário quer sempre o Sistema para ontem, e com qualidade. Para tanto, o Modelo Incremental parte do pressuposto que é preferível o usuário receber o Sistema em partes, permitindo que esses recursos já sejam utilizados, enquanto os demais estão sendo desenvolvidos.

O Modelo Incremental, ou Interativo, é desenvolvido com o conceito de versões. Nesse modelo o sistema será especificado na documentação dos requisitos, e "quebrado" em subsistemas por funcionalidades. As versões são definidas, começando com um pequeno subsistema funcional e então adicionadas mais funcionalidades a cada versão. Pode-se então dizer que o Modelo Incremental chega lentamente à funcionalidade total, por meio dessas novas versões.



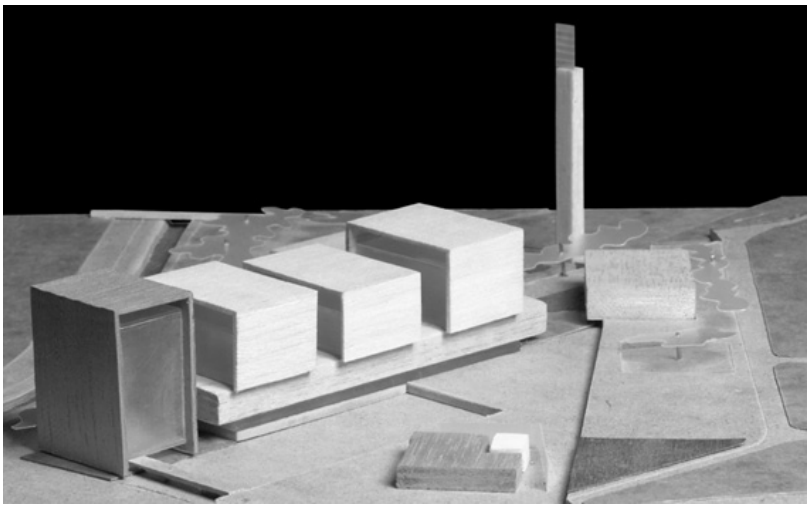
Importante observar a diferença entre INTERATIVO e ITERATIVO. As duas palavras embora com escrita extremamente parecidas, e muitas utilizadas em Informática, possuem significados distintos.

Quanto à palavra ITERATIVA que significa, pelo próprio Aurélio, "diz-se de procedimento (como algoritmo, programa, etc.) que se baseia no uso ou aplicação da iteração". Por sua vez

ITERAÇÃO possui o significado de: “Processo de resolução (de uma equação, de um problema) mediante uma seqüência finita de operações em que o objeto de cada uma é o resultado da que a precede”.
Ainda do próprio Aurélio podemos extrair a definição de INTERATIVO “de, ou relativo a sistemas ou procedimentos computacionais, programas, etc. em que o usuário pode (e, por vezes, necessita) continuamente intervir e controlar o curso das atividades do computador, fornecendo novas entradas (de dados ou comandos) à medida que observa os efeitos das anteriores”. Portanto, no nosso caso específico utilizamos o processo INTERATIVO, e não ITERATIVO.

Prototipação

A **Prototipação** tem o mesmo objetivo que uma maquete para um arquiteto (ver figura abaixo). Antes da entrega final do sistema desenvolve-se rapidamente um esboço para melhorar o entendimento de desenvolvedores e clientes sobre todas as problemáticas das questões.



Dentro dessa visão, o projeto passa por várias investigações para garantir que o desenvolvedor, usuário e cliente cheguem a um consenso sobre o que é necessário e o que deve ser proposto. Como muitos usuários não possuem uma visão ampla sobre a Tecnologia, esse método de desenvolvimento é bastante interessante, permitindo que o usuário interaja significativamente no Sistema.

A prototipação é um processo que possibilita desenvolvedor e usuários a examinarem antecipadamente os requisitos. Com isso se reduz os riscos e as incertezas do desenvolvimento.

Basicamente as etapas de desenvolvimento desse modelo são:

1. Começar com um conjunto bem simples de requisitos fornecidos pelos clientes e usuários;
2. Clientes e usuários fazem testes e experimentações, e assim que eles decidem o que querem, os requisitos são revisados, alterados, detalhados, documentados e o sistema passa a ser codificado;
3. Novamente as alternativas são apresentadas e discutidas com os usuários, e voltamos para a etapa dois, até a entrega definitiva do Sistema.

Logo, este modelo propicia duas grandes vantagens: velocidade de desenvolvimento no sentido de propiciar ao usuário uma visão mais real do software que se está projetando (o usuário poderá "enxergar" as telas e os relatórios resultantes do software) e o envolvimento direto do usuário na medida em que o desenvolvimento do software evolui, o usuário passa a ser um co-autor do desenvolvimento.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Processo_de_desenvolvimento_de_software
http://pt.wikipedia.org/wiki/Desenvolvimento_interativo_e_incremental
http://pt.wikipedia.org/wiki/Prototipac%C3%A3o

Atividades

Quais as diferenças básicas entre o Modelo Incremental e a Prototipação??
Qual a diferença entre ITERATIVO e INTERATIVO??
Quais dos dois modelos explicados nessa Unidade você escolheria para o desenvolvimento de um Sistema?!?

UNIDADE 7

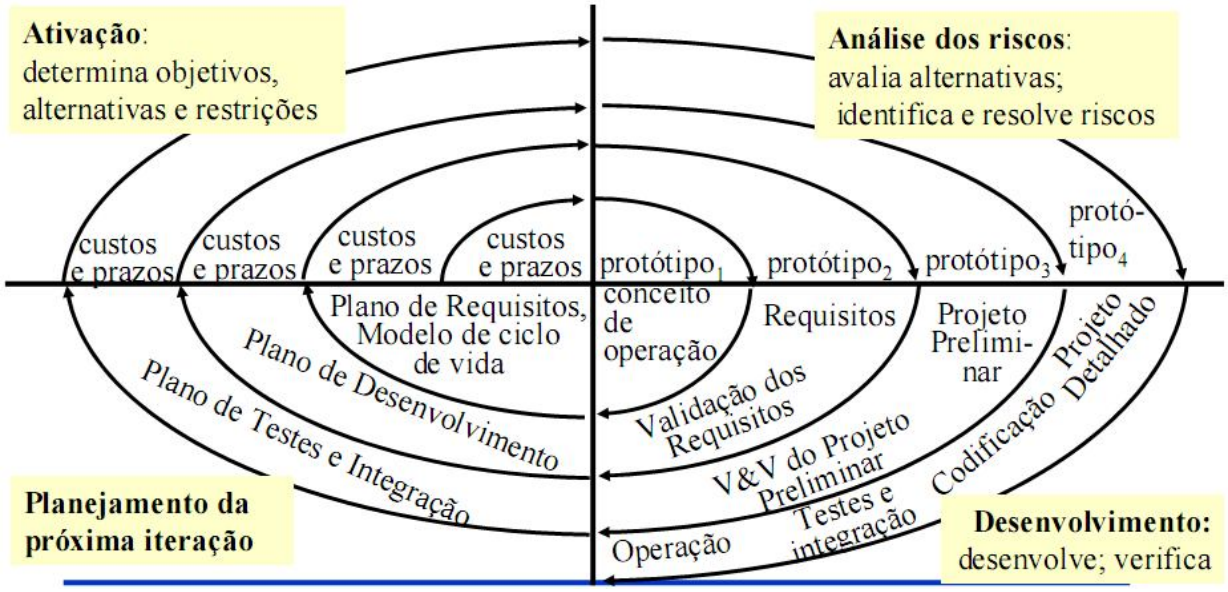
Paradigmas do desenvolvimento de Software (continuação) - Modelo Espiral - Modelos mistos e características genéricas

Objetivo: Relacionar os vários modelos de desenvolvimento de software.

Modelo Espiral

Este modelo se confunde com o de Prototipagem. Mas em princípio é mais adequado para sistemas mais complexos, e que exigem um alto nível de interações com os usuários para possibilitar a abordagem de todos os problemas desse Sistema.

Foi criado por Barry W. Boehm, ainda em 1988, e ao invés de representar o processo de software como uma sequência de atividades, a exemplo do Modelo Cascata, ele é representado através de uma espiral (veja figura abaixo).



Cada ciclo da espiral representa uma fase do processo de software. Na parte mais interna relaciona-se o início da visão da viabilidade do sistema. E a cada ciclo, passando por várias etapas, vai evoluindo a visibilidade do sistema como um todo.

O Modelo Espiral basicamente é dividido em quatro setores:

SETORES	Descrição
ATIVAÇÃO	Definem-se os objetivos específicos, identificam-se as restrições para o processo e é preparado um plano de gerenciamento detalhado. Identificam-se também os riscos sem analisá-los profundamente (foco da próxima fase).
ANÁLISE de RISCOS	Com base nos riscos identificados na fase anterior são realizadas análises detalhadas, e tomadas providências para amenizar esses riscos. Criam-se várias versões de protótipos para apoiar essa fase.
DESENVOLVIMENTO	Fundamentado pelas fases anteriores, escolhe-se o modelo mais adequado para o desenvolvimento do Sistema. A bagagem profissional e a vivência do desenvolvedor em outros sistemas são estratégicas para essa fase. Dependendo da complexidade do Sistema, às vezes, é necessária a presença de um consultor especialista.
PLANEJAMENTO	O projeto é revisto nessa fase, e é tomada uma decisão de realizar um novo ciclo na espiral ou não. Se continuar com o aperfeiçoamento do Sistema, é traçado um plano para a próxima fase do projeto.

Um diferencial nesse modelo comparado com outros, é a explícita consideração de riscos dentro do projeto como um todo. Para tanto, criou-se uma fase específica de Análise de Riscos nesse modelo.

Modelos Mistos e outros

Existem mais modelos fora os clássicos que nós vimos anteriormente. Alguns não deixam de ser um mix desses modelos. Misturam dois ou mais conceitos dos modelos estudados.

Mas gostaria de concentrar nos modelos mais atuais, e que são aplicados hoje em dia. Um deles é o Modelo RAD (Rapid Application Development). Em contraposto aos modelos clássicos que ficavam na tentativa de tentar abordar todos os principais tópicos, o RAD focou na variável tempo.

Ele é um processo de software incremental que enfatiza um ciclo de desenvolvimento curto (Pressman). A estratégia para isso é o uso da abordagem de construção baseada em componentes. Com isso o desenvolvimento completo de um Sistema, de relativa complexidade, chega a atingir 60 a 90 dias.

Os pontos a serem ressaltados nesse modelo é que se o sistema não puder ser adequadamente modularizado, a construção de componentes necessários ao RAD será problemática. E outro ponto é que o RAD pode não ser adequado quando os riscos técnicos são altos, por exemplo, se existir a necessidade de uma aplicação usufruir tecnologias novas não dominadas pela equipe.

Outro modelo é o Processo Unificado Racional, RUP em inglês, que utiliza maciçamente do UML (Unified Modeling Language). Utilizando métodos e linguagens de programação orientada a objetos, aprimora o modelo RAD. A ênfase desse modelo é na arquitetura de software. Veremos mais detalhes deste modelo na unidade 10.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Modelo_em_espiral

Atividades

Na figura apresentada existe um erro já discutido em unidades anteriores, com base nisso qual seria esse erro?!?

Para você fazer uma revisão geral do que vimos nessas últimas unidades, leia o texto sobre os Modelos de Ciclo de Vida:
http://pt.wikipedia.org/wiki/Modelos_ciclo_de_vida

UNIDADE 8

Paradigmas da Engenharia de Software: Processo, Métodos e Ferramentas

Objetivo: Entender os elementos dos paradigmas da Engenharia de Software.

A Engenharia de Software é uma tecnologia em camadas (Pressman). Conforme a figura a seguir, podemos observar que todo o foco desta disciplina é na qualidade, que é a base de todas as camadas. O alicerce da Engenharia de Software, para tal, fica sendo no PROCESSO, aonde a partir daí temos os MÉTODOS a serem aplicados, e as FERRAMENTAS como apoio a todo esse esquema. O arcabouço deste conjunto é conhecido paradigma de Engenharia de Software.

Processo

O Processo de Software é um conjunto de atividades, métodos, práticas e transformações ordenadas com a intenção de atingir a qualidade do software. Sua meta fundamental é entregar, de maneira eficiente e previsível um produto de software capaz de atender as necessidades de negócio, definidas pela análise de requisitos, dos usuários.

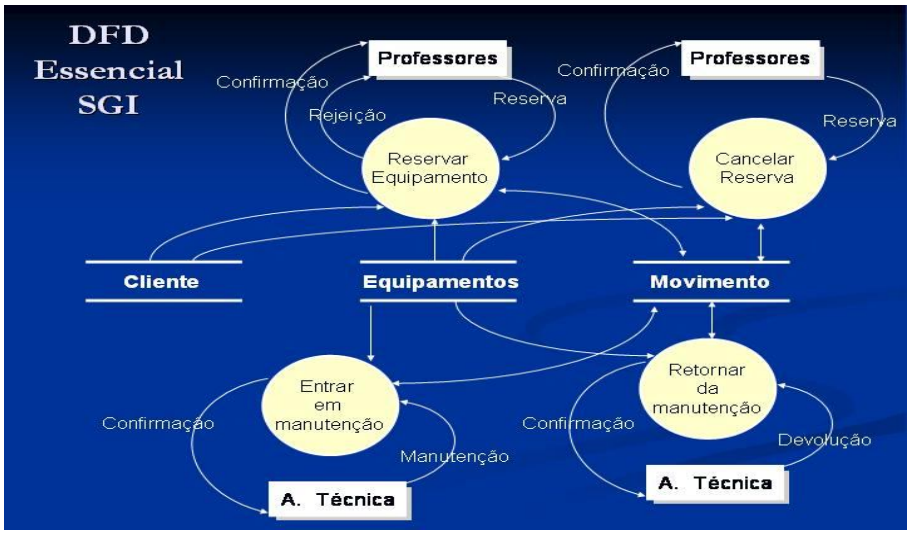
Pode-se também definir sucintamente como um conjunto completo de atividades necessárias para transformar os requisitos do usuário em um produto de qualidade de software. Um processo define QUEM está fazendo O QUE, QUANDO e COMO para atingir esse objetivo.

Métodos

Método é uma palavra que vem do grego méthodos, que significa “caminho para se chegar a um fim”. O termo metodologia é bastante controverso nas ciências em geral e na Engenharia de Software em particular. Muitos autores parecem tratar metodologia e método como sinônimos, porém seria mais adequado dizer que uma metodologia envolve princípios filosóficos que guiam uma gama de métodos que utilizam ferramentas e práticas diferenciadas para realizar algo.

As metodologias de Engenharia de Software objetivam ensinar “como fazer” para construir softwares. Esses métodos incluem atividades de modelagem, construção de programas, testes e manutenção. Na Engenharia de Software as principais abordagens de Metodologias são:

- Metodologia Estruturada: é a mais clássica das abordagens. Utiliza como ferramental Dicionário de Dados, Diagrama de Fluxo de Dados (DFD), e o Modelo Entidade Relacionamento (MER)



- Metodologia Orientada a Objetos: na Unidade 10 abordamos sobre RUP (veja maiores detalhes nessa Unidade).

- Metodologias de Desenvolvimento Ágil: Existem varias metodologias que podem ser consideradas como abordagens ágeis: XP, ASD, DSDM, Scrum, Crystal, FDD, AM entre outras. Veremos com maiores detalhes essas Metodologias nas Unidades 16 e 17.

Ferramentas

Ferramenta é uma palavra que vem do latim ferramentum significando “qualquer utensílio empregado nas artes e ofícios” (Aurélio). As ferramentas de Engenharia de Software fornecem apoio automatizado, ou semi-automatizado, para o processo e para os métodos.

Quando ferramentas são integradas de modo que a informação criada por uma ferramenta possa ser usada por outra, um sistema de apoio ao desenvolvimento de software, chamado de “engenharia de software apoiada por computador” (CASE), é estabelecido (Pressman).

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Engenharia_de_software#.C3.81reas_de_Conhecimento
http://pt.wikipedia.org/wiki/Ferramenta_CASE

Atividades

Após a leitura dessa unidade, e pelo material na Web, quais são as suas impressões quanto a divisão da Engenharia de Software em Processo, Métodos e Ferramentas ?!?

UNIDADE 9

Características de um bom processo - Características de um bom ambiente de desenvolvimento

Objetivo: Contextualizar um ambiente de desenvolvimento de software.

Processos de Engenharia de Software

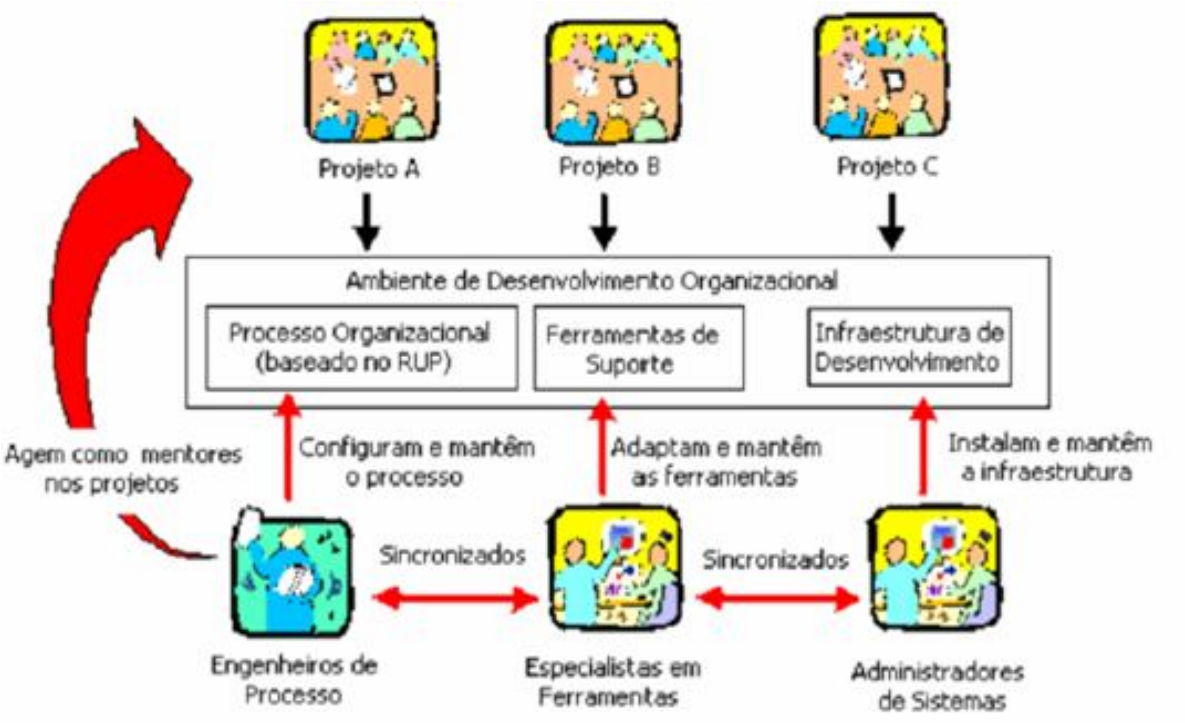
Processo de software, ou processo de Engenharia de Software, é uma sequência coerente de práticas, que objetiva o desenvolvimento ou evolução de sistemas de software. Estas práticas englobam as atividades de especificação, projeto, implementação, testes e caracterizam-se pela interação de ferramentas, pessoas e métodos.

As principais características de um bom processo são:

- Configurável para diferentes organizações.
- Adaptável para diferentes tamanhos e tipos de projetos.
- Bem definido, gerenciável e repetível.
- Com nomenclatura universal e métricas para planejamento e gerenciamento do projeto.
- Integrado com ferramentas que o suportem.

Características de um bom ambiente de desenvolvimento

- Processo de desenvolvimento definido.
- Integração entre processo e ferramentas.
- Integração entre ferramentas.
- Gerenciamento de configuração.
- Gerenciamento de mudanças.
- Gerenciamento de projetos.
- Automação de testes funcionais e de desempenho.
- Documentação consistente.
- E outros.



Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Processo_de_desenvolvimento_de_software

Atividades

Pesquise em sua empresa, ou na de um colega, como é constituído o ambiente de desenvolvimento da equipe de Sistemas. Como é a sua infraestrutura? Que ferramental possui para desenvolver?

Atividades

Antes de dar continuidades aos seus estudos é fundamental que você acesse sua SALA DE AULA e faça a Atividade 1 no “link” ATIVIDADES.

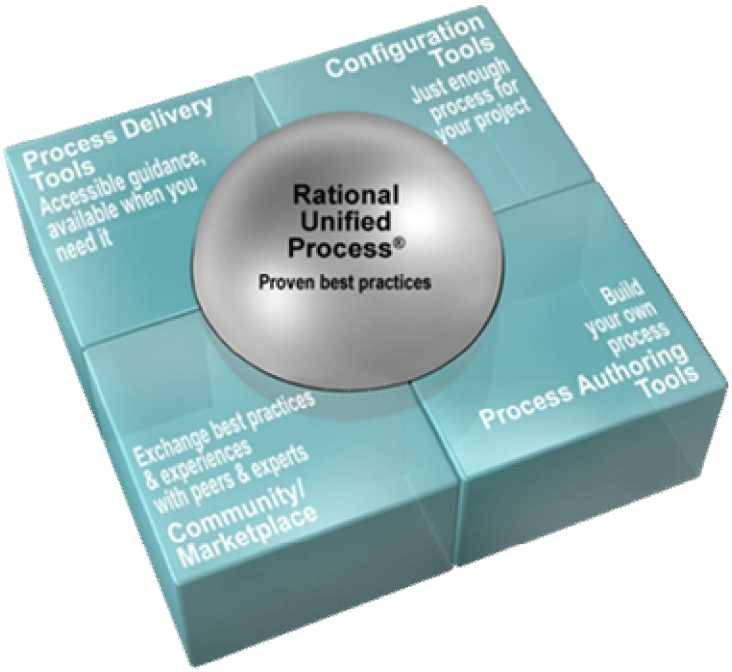
UNIDADE 10

Introdução ao RUP (Rational Unified Process) - Características - Fases e Workflows

Objetivo: Conceituar RUP e suas principais características.

RUP (Rational Unified Process) usa a abordagem da orientação a objetos em sua concepção e é projetado e documentado utilizando a notação UML (Unified Modeling Language) para ilustrar os processos em ação. Utiliza técnicas e práticas aprovadas pelo mercado.

Atualmente o RUP é um produto desenvolvido e mantido pela Rational Software (Divisão IBM). Sistemas concebidos por esse processo normalmente são desenvolvidos um uma linguagem de programação orientada a objetos, como Java ou C++.



As principais características do RUP são:

- Desenvolvimento Interativo
- Gerência de requisitos
- Uso de arquitetura baseada em componentes
- Modelagem visual

- Controle contínuo da qualidade
- Gerência de mudanças

A solução iterativa requer uma compreensão crescente do problema por meio de aperfeiçoamentos sucessivos e de desenvolvimento incremental em vários ciclos.

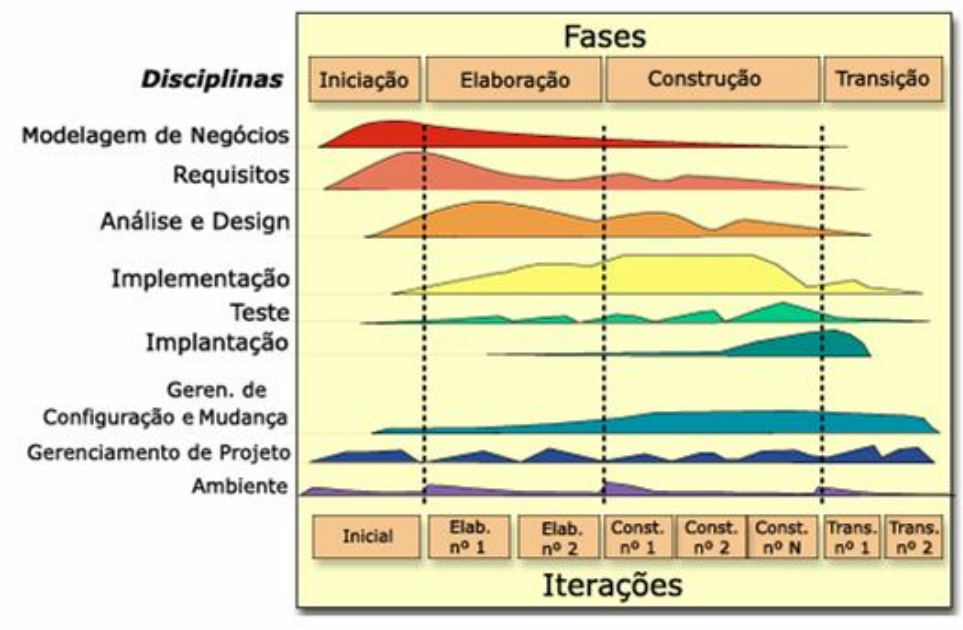
Modelagem

A abstração do sistema de software através de modelos que o descrevem é um poderoso instrumento para o entendimento e comunicação do produto final que será desenvolvido. A maior dificuldade nesta atividade está no equilíbrio entre simplicidade (favorecendo a comunicação junto ao usuário) e a complexidade (favorecendo a precisão com detalhes) do modelo. É comum a utilização de linguagens para modelagem como UML.

Fases

Estruturar um projeto junto à dimensão de tempo envolve a adoção das seguintes fases baseadas em tempo (veja maiores detalhes na tabela e figura abaixo):

FASES	Descrição
Iniciação (Inception)	Estabelece a visão, o escopo e o plano inicial para o projeto.
Elaboração (Elaboration)	Projeta, implementa e testa a arquitetura do sistema e completa o plano do projeto.
Construção (Construction)	Desenvolve a primeira versão do sistema.
Transição (Transition)	Implantar o produto no ambiente de produção.



Workflow de Processo

Estruturar um projeto junto à dimensão de componente de processo inclui as seguintes atividades:

ATIVIDADES	Descrição
Modelagem do negócio	Descreve o negócio através de casos de uso de negócio.
Requisitos	Narrativa da visão do sistema. Descrição das funções do sistema.
Análise e Projeto	Descrição de como o sistema será realizado na etapa de implementação.
Implementação	Produção do código que resultará em um sistema executável.
Testes	Verificar a integração entre todos os componentes de software, identificar e corrigir erros de implementação.
Distribuição	Gerar um release do produto. Entrega do produto e treinamento dos usuários.

Workflow de Suporte

ATIVIDADES	Descrição
Gestão de Projetos	Especifica um conjunto de princípios a aplicar na gestão de projetos no nível da alocação de recursos, planejamento, identificação e gestão de riscos, etc.
Gestão de Configuração e Mudança	Controla as mudanças e mantém a integridade dos artefatos do projeto.
Definição do Ambiente	Cobre a infraestrutura necessária para desenvolver um sistema (seleção de ferramentas, definição das regras de negócio, interface, testes, etc)

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Rational_Unified_Process
http://pt.wikipedia.org/wiki/UML

Atividades

Leia adicionalmente o interessante artigo sobre a importância do UML nos dias atuais através do seguinte link:
http://www.anacristinamelo.eti.br/artigos/Artigo_Buscando_Novos_Caminhos.pdf

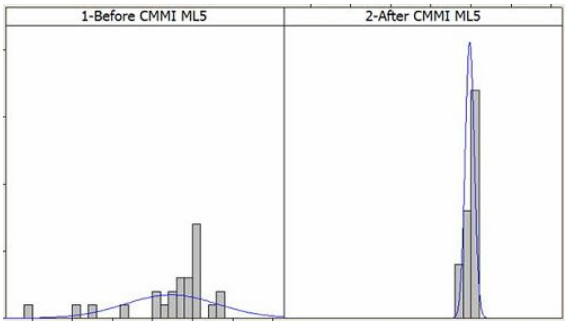
UNIDADE 11

Modelos de Maturidade – CMM (Capability Maturity Model)

Objetivo: Conceituar Modelos de Maturidade e a sua importância na Engenharia de Software.

Modelos de Maturidade

O conceito de Modelo de Maturidade de Capacitação para Software, que é um metamodelo de PROCESSO, foi desenvolvido pela Carnegie Mellon University através do seu órgão SEI (Software Engineering Institute). O SEI é um centro de pesquisa e desenvolvimento criado, em 1984, pelo Departamento de Defesa dos Estados Unidos. Podemos definir "Capacitação para Software" como sendo a habilitação que a organização tem em sistematicamente produzir software possuindo a qualidade esperada, dentro dos prazos concordados e com os recursos alocados.



Atente para o gráfico apresentado. Ele representa que quanto maior a capacitação, menor será a variação dos erros de estimativa (de custos, prazos, etc.) em torno da média. Ou seja, enquanto no gráfico da esquerda as estimativas "fogem" muito da média, o da direita as variações em relação à média foram aprimoradas após a implantação do CMM (nível 5).

O CMM (Capability Maturity Model) é o mais famoso representante desse conceito. Ele basicamente é uma metodologia de diagnóstico e avaliação da maturidade da capacitação em desenvolvimento de softwares numa organização (empresa ou instituição).

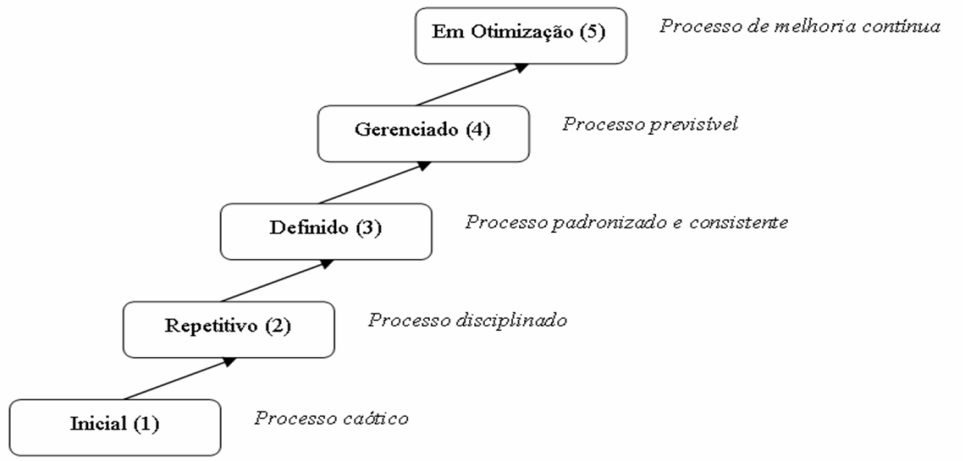
O objetivo maior do CMM é determinar em que estágio de maturidade uma empresa está em seu ciclo de desenvolvimento de software. Nasceu da necessidade do Departamento de

Defesa americano em como avaliar as empresas terceirizadas que desenvolviam softwares para eles.

O uso de estratégia de melhoria de processos através de avaliação contínua, identificação de problemas e suas devidas ações corretivas permite estabelecer cinco níveis de maturidade (veja a tabela em seguida).
CMMi (Capability Maturity Model Integration) é o modelo de maturidade surgido recentemente com o fim de unificar e agrupar as diferentes usabilidades do CMM e de outros modelos de processos de melhoria corporativo. Somente por curiosidade, raras são as empresas no mundo que conseguem atingir o nível 5, a grande maioria fica nos estágios iniciais. No Brasil, até o presente ano (2007), existiam somente 4 empresas que tinham alcançado o nível 5 do CMMi.

Estágios	Descrição
Nível 1 – Inicial	Caótico, estágio aonde que a maioria das empresas de software encontra-se.
Nível 2 – Repetitivo	Capacidade de repetir sucessos anteriores através do acompanhamento de custos, cronogramas e funcionalidades.
Nível 3 – Definido	O processo de software é bem definido, documentado e padronizado.
Nível 4 – Gerenciado	Realiza uma gerência quantitativa e qualitativa do processo de software e do produto.
Nível 5 – Em Otimização	Usa a informação quantitativa para melhorar continuamente e gerenciar o processo de software.

Para podermos visualizar melhor, de forma gráfica, todos os níveis de maturidade e a interação entre eles, podemos observar a figura a seguir:



Estudo Complementar

Software Engineering Institute (SEI) da Universidade Carnegie Mellon
http://www.sei.cmu.edu/cmm/

Atividades

Dentro do que foi visto nesta Unidade, como você visualiza a sua empresa dentro do conceito do CMM ? Em que estágio você acredita que ela esteja ?!?

UNIDADE 12

Requisitos de Software - Requisitos Funcionais e não Funcionais - Requisitos de Usuário e de Sistema

Objetivo: Identificar os vários tipos de requisitos e suas definições.

É muito comum que o cliente não saiba o que ele realmente deseja, que haja problemas na comunicação e ainda que haja mudança constante desses requisitos. O termo requisito pode ser utilizado na indústria de software tanto com o significado de algo abstrato, como matematicamente formal. Para aprimorar esse conceito A.M.Davis ilustra o seguinte case em seu livro Software requirements - objects, functions and states:

"Se uma empresa deseja estabelecer com contrato para o desenvolvimento de um grande projeto de software (para selecionar entre vários fornecedores), ela tem de definir suas necessidades de maneira suficientemente abstrata para que uma solução não seja predefinida. Os requisitos devem ser redigidos de modo que os diversos fornecedores (de software) possam apresentar propostas, oferecendo, talvez, diferentes maneiras de atender às necessidades organizacionais do cliente. Uma vez estabelecido um contrato (entre ambas as partes), o fornecedor (que ganhou) precisa preparar uma definição de sistema para o cliente, com mais detalhes, de modo que o cliente compreenda e possa validar o que o software fará. Esses dois documentos podem ser chamados de documentos de requisitos do sistema".

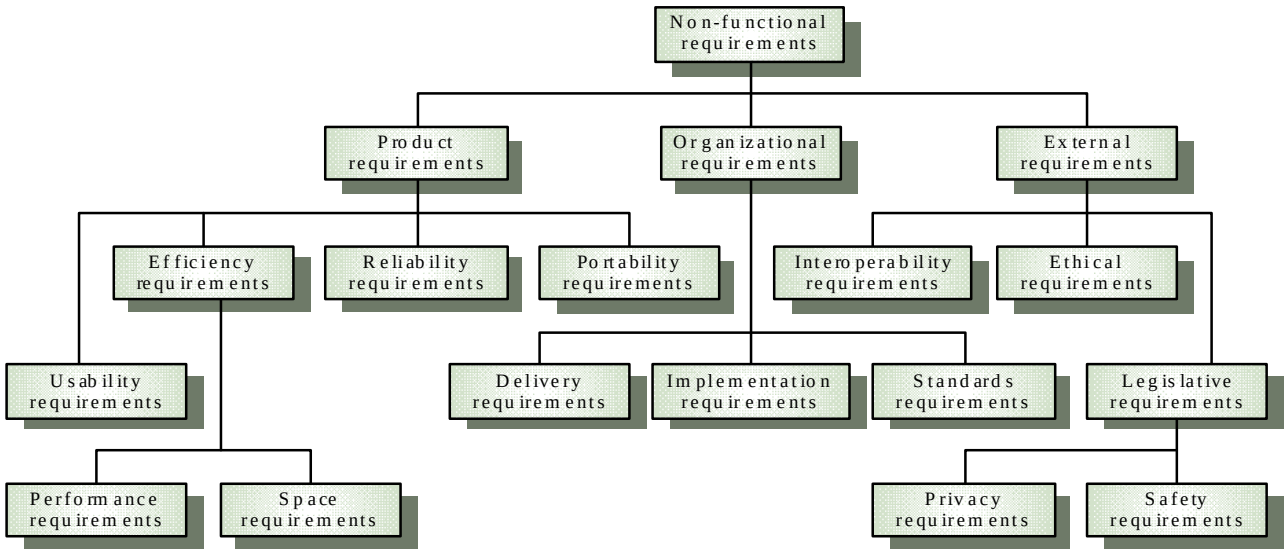
As atividades de Análise de Requisitos concentram-se na identificação, especificação e descrição dos requisitos do sistema de software. Em resumo, requisito é uma necessidade que o software deve cumprir. Há várias interpretações e classificações sobre requisitos tais como:

Tipos de Requisitos	Descrição
Requisitos do Usuário	São declarações, em linguagem natural e também em diagramas, sobre as funções que o sistema deve fornecer, e as restrições, sob as quais deve operar.

Requisitos de Sistema (ou do desenvolvedor)	Estabelecem detalhadamente as funções e as restrições de sistema. O documento de requisitos de sistema, algumas vezes chamado de especificação funcional, deve ser preciso. Ele pode servir como um contrato entre o comprador do sistema e o desenvolvedor do software.
Requisitos Funcionais	São declarações de funções que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem, de forma explícita, declarar o que o sistema não deve fazer.
Requisitos não Funcionais	São restrições sobre os serviços ou as funções oferecidos pelo sistema. Entre eles destacam-se restrições sobre o processo de desenvolvimento, padrões, entre outros.

Adaptado de Sommerville

Ressalta-se que essa classificação não é tão precisa, e até um pouco artificial. Ou seja, por exemplo, um requisito de usuário relacionado à proteção, aparenta ser um requisito não funcional. No entanto, ao detalharmos esse requisito ele pode assumir uma abrangência mais típica de um requisito funcional. Pois podemos necessitar de incluir recursos de autorização de usuários no sistema.



Pela figura anterior, podemos observar como os Requisitos não Funcionais são bastante abrangentes. Podem compreender desde requisitos éticos, como de desempenho ou mesmo de interoperabilidade.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Processo_de_Engenharia_de_Requisitos

Atividades

Visite o site http://www.ic.unicamp.br/~ariadne/inf301/modulo2-v.pdf e explore as informações das transparências com a temática “Extração de Requisitos”.

UNIDADE 13

Técnicas de Análise de Requisitos - O Documento de Requisitos de Software

Objetivo: Identificar um documento de requisito de software e suas técnicas.

Técnicas de Análise de Requisitos

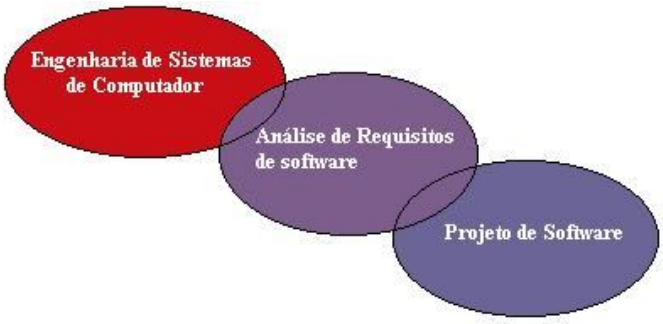
Existem 10 princípios básicos, e engraçados, sugeridos por Pressman, implementados por nós, no processo de levantamento de requisitos junto aos usuários numa reunião presencial:

Princípio nº 1: Escute, escute e escute.
Esta talvez seja a atitude mais importante na hora da captação dos requisitos de usuários.
Se associado ao princípio 5, transforma-se num ponto estratégico para que o usuário/cliente perceba que você está querendo entender todos os seus problemas.

Princípio nº 2: Prepare-se bem antes de se comunicar.
Gere bastantes perguntas fundamentais à resolução e visão do negócio do usuário/cliente. Além de ser importante para essa atividade, todos gostam de responder questionamentos sinceros sobre as suas atividades.

Princípio nº 3: Deve existir um facilitador na reunião.
Não é interessante que o próprio analista seja o condutor dessa reunião. Existindo um personagem como "facilitador" na reunião, ameniza problemas de discussões ou maus entendidos. Seria praticamente um "animador" das discussões.

Princípio nº 4: Foco da discussão num Desenho ou Documento, não nas pessoas.
Se a discussão por ventura ficar pessoal, deve-se sempre voltar o foco da reunião para um desenho, documento ou mesmo sobre o processo envolvido. Isso abranda possíveis conflitos pessoais.



Princípio nº 5: Faça sempre anotações e documente as decisões.
Por mais que não se queira a memória humana é fraca, muito fraca. Portanto, deve-se registrar o máximo das posições e informações dos usuários/clientes. Você irá se surpreender no futuro como anotou várias coisas que nem você não lembrava mais. E isso será muito importante nos conflitos que ocorrem ao longo do projeto.

Princípio nº 6: Buscar ao máximo a colaboração de todos.
Animosidades não ajudam a ninguém. O bom humor ajuda muito nessa fase de levantamento. Procurar ser agradável e simpático ameniza a grande maioria dos problemas pessoais. E por incrível que pareça, os problemas pessoais são os que mais atrapalham num projeto.

Princípio nº 7: Conserve-se focado, departamentalize sua discussão.
Discuta cada tema profundamente. Tente evitar questionar, ou discursar sobre vários temas simultaneamente. Vai eliminando a discussão tema a tema. A produtividade irá aumentar.

Princípio nº 8: Se algo não estiver claro, sempre desenhe.
Como o velho provérbio diz: "Uma imagem vale mil palavras". Não existe a necessidade de aplicar as técnicas de modelagem nessa hora, mas com desenhos simples, "mapas mentais", transparências do PowerPoint, quadros e imagens ajudam muito nessa fase do projeto.

DICA: visite o site www.mapasmentais.com.br para ver a técnica que a própria NASA utiliza em seus projetos.

Princípio nº 9: (a) Se você concorda com algo, prossiga;
(b) Se você não concordar com algo, prossiga;
(c) Se algo não estiver claro, e sem condições de esclarecer naquele momento, prossiga.
Há momentos que não adianta, como se diz no popular, "dar murro em ponta de faca". Prepare-se e aguarde o momento certo para voltar a tocar num tema polêmico. O uso da criatividade, na abordagem de um tema desse tipo, é super estratégico.

Princípio nº 10: Negociar sempre no ganha-ganha.
Existem várias posturas numa negociação entre dois personagens (perde-perde, ganha-perde, perde-ganha e ganha-ganha). A melhor relação é o ganha-ganha. Essa é a postura dos vencedores. Ou seja, é conduzida a solução de conflitos de tal forma criativa e rica em oportunidades, que os dois lados ganham na negociação.

DICA: o site http://www.golfinho.com.br/artigospnl/artigodomes1299.asp apresenta várias dicas pessoais sobre o processo de negociação ganha-ganha.

O Documento de Requisitos de Software

O Documento de Especificação de Requisitos de Software pode variar nos detalhes de empresa para empresa, mas normalmente possui os seguintes campos:

- Definição do Contexto
- Definição de Requisitos
  - Requisitos Funcionais
  - Requisitos de Interface
  - Requisitos não Funcionais
- Análise de Risco
- Anexos

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Requisitos_de_Software

Atividades

Veja o documento de Especificação de Requisitos de Software em http://www.ic.unicamp.br/~ariadne/inf301/doc-requisitos.pdf e tente você mesmo gerar um documento com base num Sistema genérico (um sistema hipotético, ou um sistema que você está trabalhando, ou ainda um sistema que precisaria ser desenvolvido, etc.).

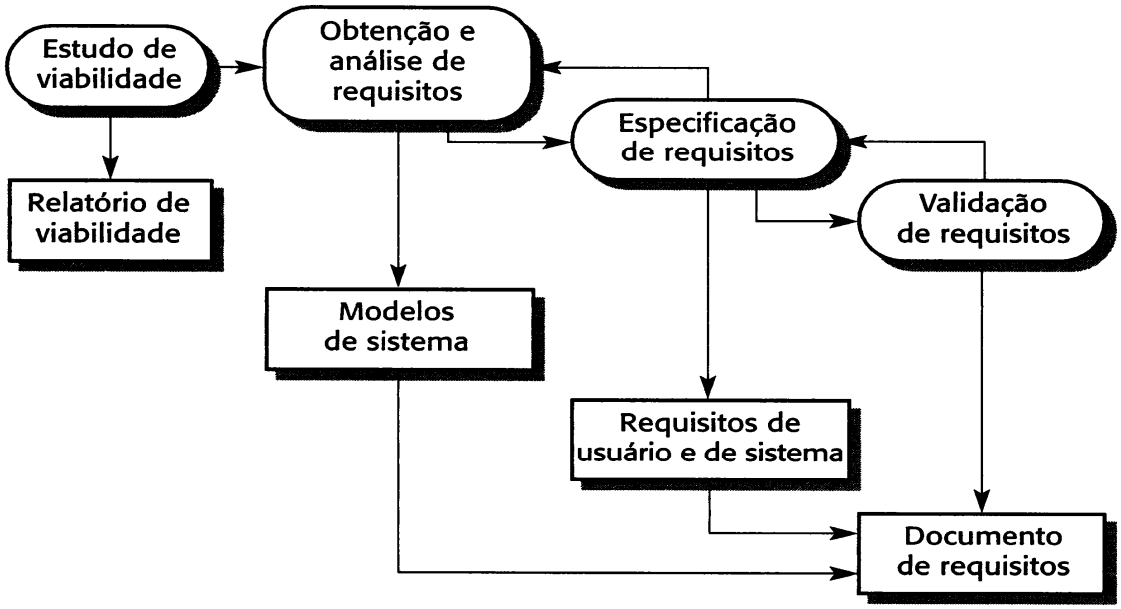
UNIDADE 14

Processos de Engenharia de Requisitos - Estudos de Viabilidade

Objetivo: Conceituar os processos de engenharia de requisitos e a viabilidade técnica.

A Engenharia de Requisitos é um processo que envolve todas as atividades exigidas para criar e manter o Documento de Requisitos de Sistema (Sommerville). Pela imagem logo abaixo podemos observar as quatro atividades genéricas de alto nível (caixas arredondadas): Estudo de Viabilidade, Obtenção e Análise de Requisitos, Especificação de Requisitos e Validação de Requisitos.

Segundo Rumbaugh, alguns analistas consideram a Engenharia de Requisitos como um processo de aplicação de um método estruturado, como a análise orientada a objetos. No entanto, a Engenharia de Requisitos possui muito mais aspectos do que os que são abordados por esses métodos.



Processo de Engenharia de Requisitos conforme Sommerville

Estudos de Viabilidade

O primeiro processo a ser realizado num Sistema novo é o Estudo de Viabilidade. Os resultados deste processo devem ser um relatório com as recomendações da viabilidade técnica ou não da continuidade no desenvolvimento do Sistema proposto.
Basicamente um estudo de viabilidade, embora seja normalmente rápido, deverá abordar fundamentalmente as seguintes questões:

- O Sistema proposto contribui para os objetivos gerais da organização?
- O Sistema poderá ser implementado com as tecnologias dominadas pela equipe dentro das restrições de custo e de prazo? Ou precisaria de treinamentos adicionais?
- O Sistema pode ser integrado, e é compatível com os outros sistemas já em operação?

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Processo_de_Engenharia_de_Requisitos

Atividades

Como estamos explorando constantemente o WIKIPÉDIA, queremos que você entre no site http://pt.wikipedia.org/wiki/Engenharia_de_software e veja o que você pode aprimorar e contribuir para melhorar cada vez mais essa fantástica enciclopédia virtual.

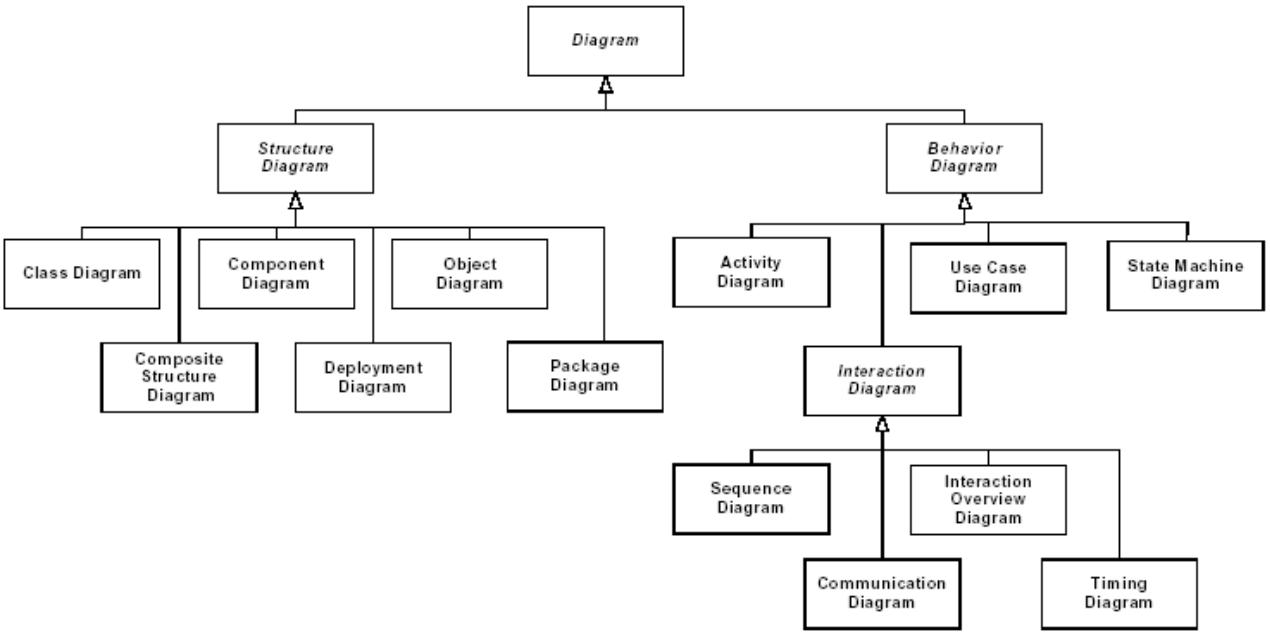
UNIDADE 15

Modelagem – UML: Unified Modeling Language – Linguagem de Modelagem Unificada

Objetivo: Explicar o processo de modelagem utilizando o UML.

O UML (Unified Modeling Language - Linguagem de Modelagem Unificada) é um padrão para a modelagem orientada a objetos. É uma linguagem de diagramação ou notação para especificar, visualizar e documentar modelos de sistemas de software Orientados a Objeto. O UML é controlado pela OMG (Object Management Group - OMG). Veja, na figura abaixo, a árvore de diagramas do UML.

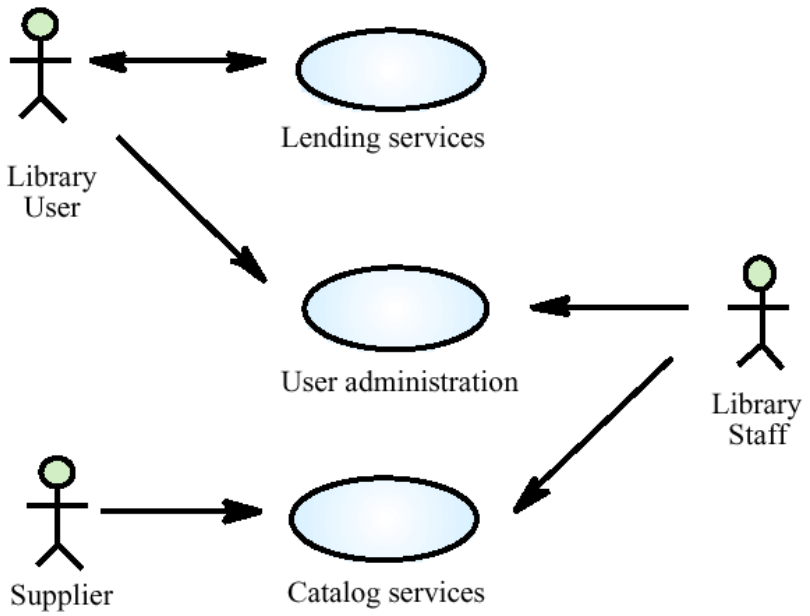
DICA: tenha a oportunidade de conhecer o site da OMG em www.uml.org



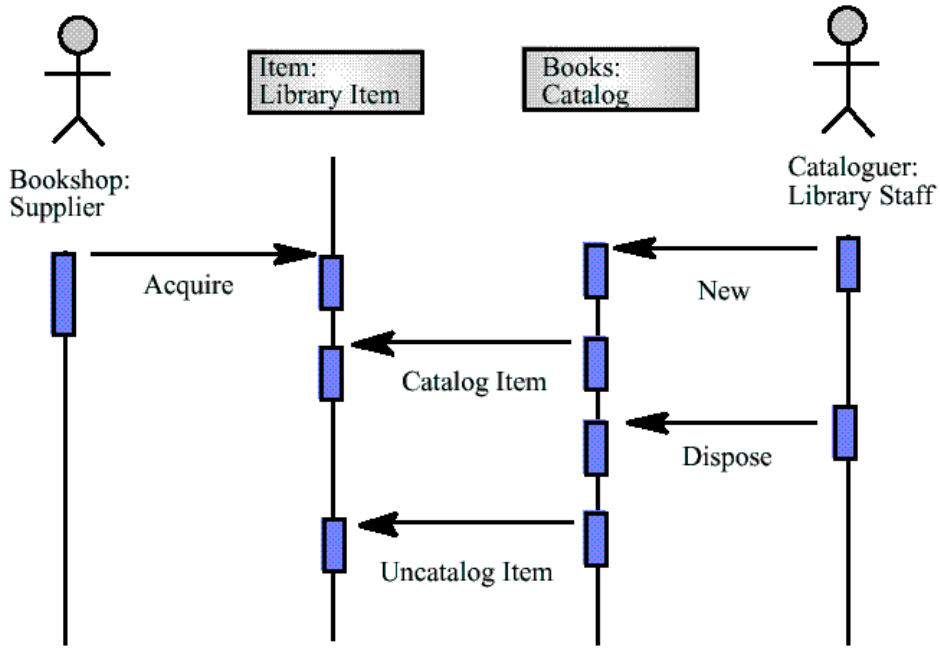
Principais **DIAGRAMAS**	Descrição
Diagrama de Caso de Uso	Mostra atores (pessoas ou outros usuários do sistema), casos de uso (os cenários onde eles usam o sistema), e seus relacionamentos.
Diagrama de Classe	Diagrama as classes e os relacionamentos entre elas.

Diagrama de Sequência	Mostra objetos e uma sequência das chamadas do método feitas para outros objetos.
Diagrama de Colaboração	Apresenta objetos e seus relacionamentos, colocando ênfase nos objetos que participam na troca de mensagens.
Diagrama de Estado	Exibe estados, mudanças de estado e eventos em um objeto ou em uma parte do sistema.
Diagrama de Atividade	Apresenta as atividades e as mudanças de uma atividade para outra com os eventos ocorridos em alguma parte do sistema.
Diagrama de Componente	Mostra os componentes de programação de alto nível (como KParts ou Java Beans).
Diagrama de Distribuição	Destaca as instâncias dos componentes e seus relacionamentos.

Os use-cases são cada vez mais utilizados para a obtenção de requisitos, e são uma característica fundamental na notação UML. São técnicas baseadas em cenários para a obtenção de requisitos. Os use-cases identificam os agentes envolvidos numa interação e o tipo dessa interação. Veja exemplo abaixo.



Diagramas de Sequência podem ser utilizados para acrescentar informações a um use-case. Esses diagramas mostram os agentes envolvidos na interação, os objetos dentro do sistema com os quais eles interagem, e as operações que estão associadas a esses objetos. A imagem abaixo ilustra isso.



Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/UML
http://pt.wikipedia.org/wiki/Caso_de_uso
http://pt.wikipedia.org/wiki/Casos_de_Uso

Atividades

Leia o excelente artigo que mostra um estudo de caso aplicado à modelagem UML:
http://www.cefetsp.br/edu/sinergia/6p10c.html

UNIDADE 16

Metodologias de Desenvolvimento Ágeis de Software: XP - FDD e DSDM

Objetivo: Abordar as várias metodologias ágeis e suas aplicações

Através do "Manifesto for Agile Software Development" (Manifesto para Desenvolvimento Ágil de Software) criado em 2001 por Kent Beck, e mais 16 notáveis desenvolvedores, se reuniram para defender as seguintes regras:

> "Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através desse trabalho, passamos a valorizar:
>
> - Indivíduos e interação entre eles mais que processos e ferramentas;
> - Software em funcionamento mais que documentação abrangente;
> - Colaboração com o cliente mais que negociação de contratos;
> - Responder a mudanças mais que seguir um plano.
>
> Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda".

DICA: visite o site do MANIFESTO ÁGIL em http://www.agilemanifesto.org/

Portanto, com base no Manifesto Ágil, chega-se aos seguintes princípios básicos:

- Simplicidade acima de tudo;
- Rápida adaptação incremental às mudanças;
- Desenvolvimento do software preza pela excelência técnica;
- Projetos de sucesso surgem através de indivíduos motivados, e com uma relação de confiança entre eles;
- Desenvolvedores cooperam constantemente e trabalham junto com os usuários/clientes;

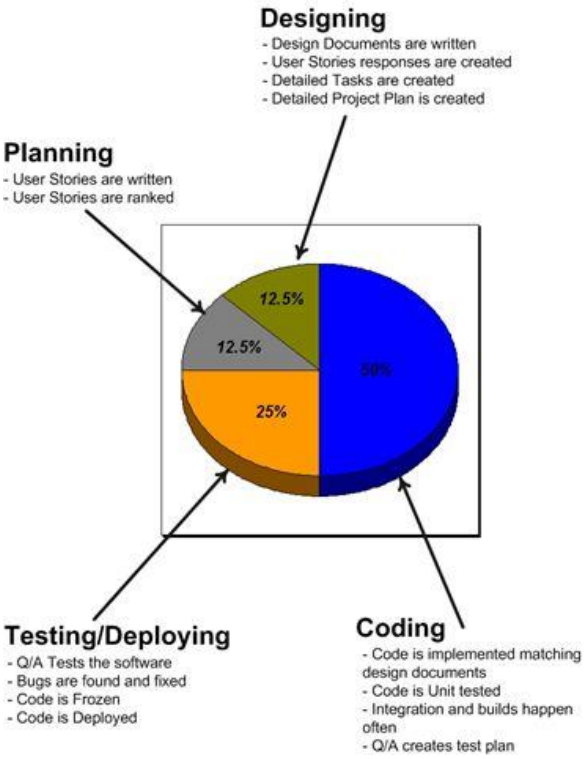
- Atender o usuário/cliente, entregando rapidamente e continuamente produtos funcionais em curto espaço de tempo (normalmente a cada 2 semanas);
- Software funcionando é a principal medida de progresso;
- Mudanças no escopo, ou nos requisitos, do projeto não é motivo de chateação;
- A equipe de desenvolvimento se auto-organiza, fazendo ajustes constantes em melhorias.

Esse Manifesto ocorreu para ser um contraponto as Metodologias de Desenvolvimento Prescritivas. Ou seja, enquanto o RUP (visto na Unidade 10) é extremamente rígido com altos níveis de controle, e forte documentação, as metodologias ágeis caminham ao contrário. Destacamos que, mesmo assim, ela não inflige a uma sólida prática da Engenharia de Software.

No gráfico anterior vemos num extremo o RUP enfatizando os controles, e uma política de trabalho rígida. Ele é mais interessante de ser utilizado com equipes grandes de desenvolvimento. Na outra ponta temos o XP, que veremos a seguir, sinalizando maior liberdade e mais adequada para equipes pequenas. E num ponto intermediário o FDD, que veremos no final desta Unidade, como um modelo conciliador dessas duas estratégias.

Um dos pontos de destaque na Metodologia Ágil é a liberdade dada para as equipes de desenvolvimento. A declaração de Ken Schwaber define isso da seguinte forma: “A equipe seleciona quanto trabalho acredita que pode realizar dentro da iteração, e a equipe se compromete com o trabalho. Nada desmotiva tanto uma equipe quanto alguém de fora assumir compromissos por ela. Nada motiva tanto uma equipe quanto a aceitação das responsabilidades de cumprir os compromissos que ela própria estabeleceu”.

XP (Extreme Programming)



O modelo ágil mais conhecido é o XP (Extreme Programming). Ele usa preferencialmente a abordagem orientada a objetos.

O XP inclui um conjunto de regras e práticas que ocorrem no contexto de quatro atividades (veja a figura ao lado):

- Planejamento
- Projeto
- Codificação
- Teste

Existe uma grande ênfase ao trabalho em duplas, no qual um analista mais experiente trabalha com um novato. Enquanto o mais jovem trabalha na programação o mais antigo vai revisando o código. Dessa forma ao mesmo tempo desenvolve-se a equipe, e melhora-se automaticamente a qualidade do código fonte gerado.

FDD – Feature Driven Development

O FDD (Desenvolvimento guiado por Características), concebido por Peter Coad, teve como premissa criar um modelo prático de processo para a Engenharia de Software orientado a objetos.

No entanto, Stephen Palmer e John Felsing aprimoraram o modelo descrevendo um processo ágil e adaptativo que pode ser aplicado a projetos de software tanto a projetos de médio como de grande porte.

Dentro do contexto do FDD, o significado de "**característica**" vem a ser uma função, relativamente pequena, acertada com o cliente que pode ser implementada em menos de duas semanas, com os seguintes benefícios:

- Sendo as "características" pequenos blocos de funcionalidade, os usuários e desenvolvedores têm melhor controle e entendimento de todo o processo.
- Organizam-se as "características" em um agrupamento hierárquico relacionado ao negócio, melhorando a visão para o usuário. E para os desenvolvedores facilitando o planejamento de todo o projeto.

- A equipe tem metas de desenvolvimento dessas “características” a cada duas semanas.

O FDD enfatiza mais as diretrizes e técnicas de gestão de projetos do que outros métodos ágeis. O projeto é muito bem acompanhado, ficando claro para todos os envolvidos os avanços e problemas que o Projeto vai sofrendo. Para tanto, o FDD define seis marcos de referência durante o projeto e implementação de uma “característica”:

- Travessia do projeto;
- Projeto;
- Inspeção do projeto;
- Código;
- Inspeção do código;
- Promoção para a construção.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Desenvolvimento_%C3%A1gil_de_software

Outros sites:
http://iscte.pt/~mms/events/agile_seminar/apresentacoes.htm

Atividades

Dentro da sua empresa, ou na de amigos, verifique qual das estratégias apresentadas nesta Unidade que melhor poderia ser utilizada. Se você, como empresário, criasse uma empresa, qual das estratégias discutidas você adotaria?!?

UNIDADE 17

Continuação das Metodologias de Desenvolvimento Ágil de Software: Scrum - Crystal - ASD e AM

Objetivo: Abordar as várias metodologias ágeis e suas aplicações.

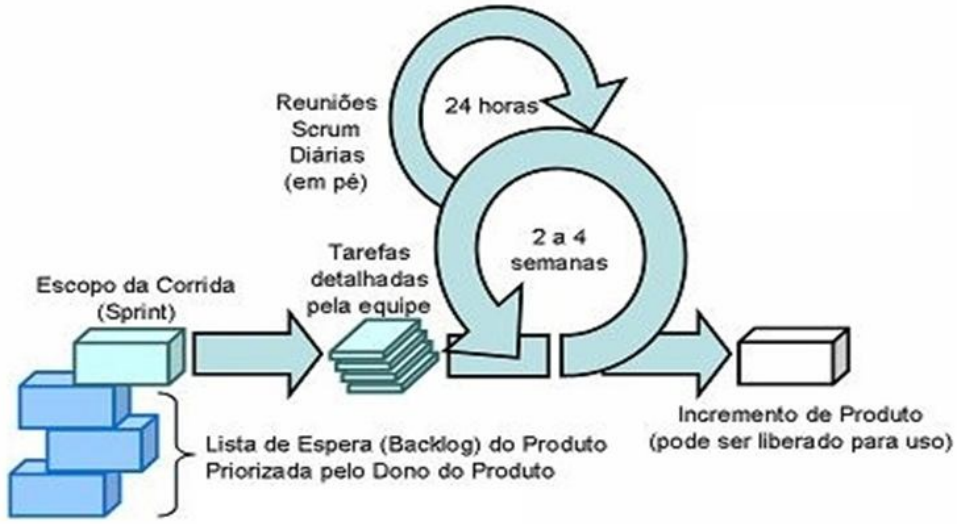
Scrum

O significado peculiar desse modelo de desenvolvimento ágil vem do nome da atividade de jogadores de rugby ao trabalharem "fortemente" juntos para deslocar a bola pelo campo. Foi desenvolvida por Jeff Sutherland, ainda na década de 90. Seus princípios básicos seguem o manifesto ágil.

Um ponto que se destaca nesse modelo são as **Reuniões Scrum**. Sugere-se que sejam realizadas diariamente por 15 minutos, mas com base em nossa realidade brasileira, acreditamos que o período ideal seria semanal, com uma duração de 1 hora (preferencialmente as sextas-feiras à tarde).

São somente três questões que são apresentadas para todos os envolvidos, e com excelentes resultados. Todos devem apresentar suas respostas com base nas seguintes perguntas:

- O que você fez desde a última Reunião Scrum?
- Que obstáculos você está encontrando que podemos ajudar?
- O que você planeja realizar até a próxima Reunião Scrum?



Crystal

Criado por Alistair Cockburn e Jim Highsmith com intuito de fazer uma analogia com os cristais geológicos que apresentam na natureza com a sua própria cor, forma e dureza. Destaca-se a "manobrabilidade" com o significado de um jogo cooperativo de invenção e comunicação de recursos limitados, com o principal objetivo de entregar softwares úteis funcionando e com o objetivo secundário de preparar-se para o jogo seguinte (Presman).

A família Crystal é, na verdade, um conjunto de processos ágeis que se mostraram efetivos para diferentes tipos de projeto. A intenção é permitir que equipes ágeis selecionem o membro da família Crystal mais apropriado para o seu projeto e ambiente.

ASD – Adaptative Software Development

O ASD (Desenvolvimento Adaptativo de Software) foi proposto por Jim Highsmith, com o intuito de ser uma técnica para construção de sistemas e softwares complexos. O foco desse modelo é a colaboração humana e na auto-organização da equipe de desenvolvimento. O ciclo de vida de um ASD incorpora três fases, detalhadas na tabela abaixo:

FASES	Descrição
Especulação	Planejamento do ciclo adaptativo usa informações de iniciação do projeto para definir o conjunto de ciclos de versão (incrementos de software) que serão necessários para o projeto.
Colaboração	Os analistas precisam confiar um no outro para: criticar sem animosidade, ajudar sem ressentimentos, trabalhar mais do que estão acostumados, potencializar suas habilidades, e comunicar problemas de um modo que conduza à ação efetiva.
Aprendizado	À medida que os membros de uma equipe ASD começam a desenvolver os componentes que fazem parte de um ciclo adaptativo, a ênfase está tanto no aprendizado quanto no progresso em direção a um ciclo completo.

AM – Agile Modeling

Conforme o site que se auto-intitula The Official Agile Modeling (veja maiores detalhes, e vale a pena visitar, em: http://www.agilemodeling.com/) Scott W. Ambler, seu criador, descreve a Modelagem Ágil (AM) como sendo (adaptado por nós):

"A Modelagem Ágil (AM) é uma metodologia baseada na prática, para modelagem e documentação efetiva de sistemas baseados em software. Modelagem Ágil é uma coleção de valores, princípios e práticas de modelagem de software que podem ser aplicados a um projeto de desenvolvimento de software de modo efetivo e leve. Os modelos ágeis são mais efetivos do que os modelos tradicionais, porque eles são suficientemente bons, não precisando ser perfeitos !"

Dos princípios mais importantes da Modelagem Ágil (AM), anunciados por Ambler, destacamos os dois mais significativos:

Modelos Múltiplos: há muitos modelos e notações diferentes que podem ser usados para descrever softwares. Importante: apenas um pequeno subconjunto é essencial para a maioria dos projetos. A AM sugere que, para fornecer a visão necessária, cada modelo apresente um aspecto diferente desse sistema e que apenas aqueles modelos que ofereçam valor à sua pretensa audiência sejam usados.

Viajar Leve: essa expressão se refere aos turistas que para não ficar carregando pesadas malas, adotam esse princípio. No caso, para a AM, ela dita que à medida que o trabalho de Engenharia de Software prossegue, conserve apenas aqueles modelos que fornecerão valor em longo prazo e livre-se do resto.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Desenvolvimento_%C3%A1gil_de_software

Outros sites:
http://www.heptagon.com.br/?q=scrum

Atividades

Leia adicionalmente o interessante artigo em: http://www.heptagon.com.br/?q=node/5 para ampliar o seu conhecimento sobre as Metodologias Ágeis. Termine essa atividade revendo as principais características, diferenças e semelhanças que existem entre os diversos modelos ágeis.

UNIDADE 18

Engenharia de Projeto - Projeto Modular - Projeto de interface com o usuário

Objetivo: Apresentar as principais diretrizes para projetos e das interfaces com o usuário

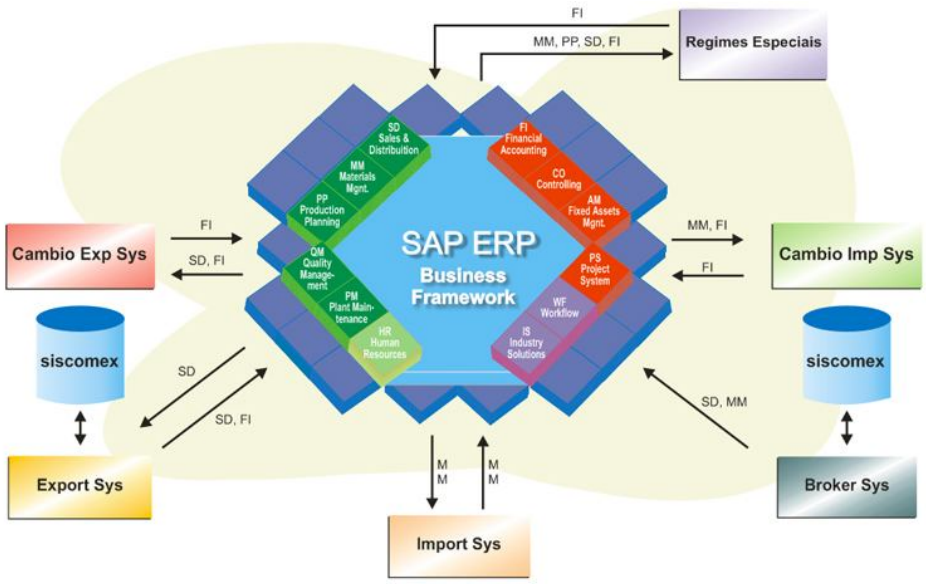
O que é **Projeto de Software**? Podem-se pegar as interessantes palavras de Mitch Kapor para definir bem essa ação:

> ***"É onde você se instala com um pé em dois mundos – o mundo da tecnologia e o mundo das pessoas e objetivos humanos – e você tenta juntar os dois..."***

Portanto, é um lugar de criatividade aonde os requisitos do usuário/cliente, as necessidades do negócio, e as considerações técnicas se juntam na formulação de um produto ou sistema. É o momento mágico aonde o engenheiro de software modela, cria e constroi a estrutura de todas as partes de um Sistema, antes dele mesmo existir. Veremos mais detalhes sobre Gestão de Projetos na Unidade 28.

Projeto Modular

A Modularidade consiste na divisão do software em componentes nomeados separadamente e endereçáveis, muitas vezes chamado de módulos. Os mesmos são integrados para satisfazer aos requisitos do Sistema (adaptado de Pressman). Veja a figura abaixo, aonde é apresentado os vários módulos do ERP da SAP.



Uma prática de Engenharia de Software condenável é a construção de softwares monolíticos. Ou seja, um software composto de um único e grande módulo. Isso gera uma complexidade global quanto ao número de caminhos de controle, intervalos de referencia, número de variáveis, que faz um programa ter uma baixa compreensão para todos.

Outro problema é a manutenabilidade do Sistema. Com poucas pessoas compreendendo o Sistema, mais difícil e custoso fica sendo a sua manutenção. Por outro lado, um software com excesso de módulos pode acarretar no mesmo erro. O bom senso novamente é a melhor resposta.

Projeto de interface com o usuário

Os computadores atuais fornecem uma interface chamada de GUI (Graphical User Interface - Interface Gráfica do Usuário), mas nem sempre foi assim. As primeiras versões eram 1D (uma única dimensão), aonde o usuário simplesmente alimentava um terminal que podia se deslocar para a direita e esquerda. Atualmente temos os de 2D (duas dimensões), graças ao mouse podemos deslocar o ponteiro por toda a tela. E como tendência temos já as interfaces 3D (três dimensões). Um bom exemplo seria o Second Life.

DICA: visite o SECOND LIFE no site americano www.secondlife.com .

Podemos ver na tabela abaixo, as diretrizes gerais para a elaboração de uma boa interface com o usuário:

DIRETRIZES	Descrição
Familiaridade com o usuário	Deve utilizar termos e conceitos que tenham como base a experiência das pessoas que vão utilizar o sistema.

Consistência	Sempre que possível, operações semelhantes devem ser ativadas da mesma maneira.
Mínimo de surpresa	Os usuários nunca devem ser surpreendidos com o comportamento do Sistema.
Facilidade de recuperação	A interface deve incluir mecanismos para permitir aos usuários a recuperação a partir de erros humanos.
Orientação do usuário	Na ocorrência de erros fornecer feedback significativo, e oferecer recursos sensíveis ao contexto de ajuda.
Diversidade de usuários	A interface deve fornecer recursos de interação apropriados a diferentes tipos de usuários do sistema.

Adaptado de Sommerville

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Interface_%28ci%C3%AAncia_da_computa%C3%A7%C3%A3o%29
http://pt.wikipedia.org/wiki/Interface_do_utilizador
http://pt.wikipedia.org/wiki/Interface_gr%C3%A1fica_do_utilizador

UNIDADE 19

Arquiteturas de Sistemas Distribuídos - Arquitetura de Multiprocessadores

Objetivo: Diferenciar as várias e principais arquiteturas de sistemas

Os sistemas com base em Mainframes, ou seja, computadores de grande porte, na prática são Sistemas Distribuídos. O conceito de Sistema Distribuído é a conexão de várias máquinas iguais, ou mesmo diferentes, para processar um ou mais sistemas.

Os três principais sistemas existentes atualmente são:

- Sistemas pessoais
- Sistemas embutidos
- Sistemas distribuídos

Nos primeiros temos como exemplo os editores de texto e as planilhas eletrônicas. Um exemplo de Sistema embutido seria a ignição eletrônica, aonde num processador existe toda uma lógica de controle.

As mais importantes características dos Sistemas Distribuídos são:

CARACTERÍSTICAS	Descrição
Compartilhamento de Recursos	Compartilha recursos de hardware e de software gerenciados por computadores centrais, ou servidores.
Abertura	Pode-se facilmente incluir hardware e software de diferentes fabricantes.
Concorrência	Vários processos podem operar ao mesmo tempo em diferentes computadores na rede.
Escabilidade	Em princípio, pode-se aumentar infinitamente a capacidade dos Sistemas distribuídos, somente limitado pela capacidade de sua rede.
Tolerância a Defeitos	Com a estrutura dos Sistemas Distribuídos, há o potencial da duplicação de informações, evitando algumas falhas de hardware e software.

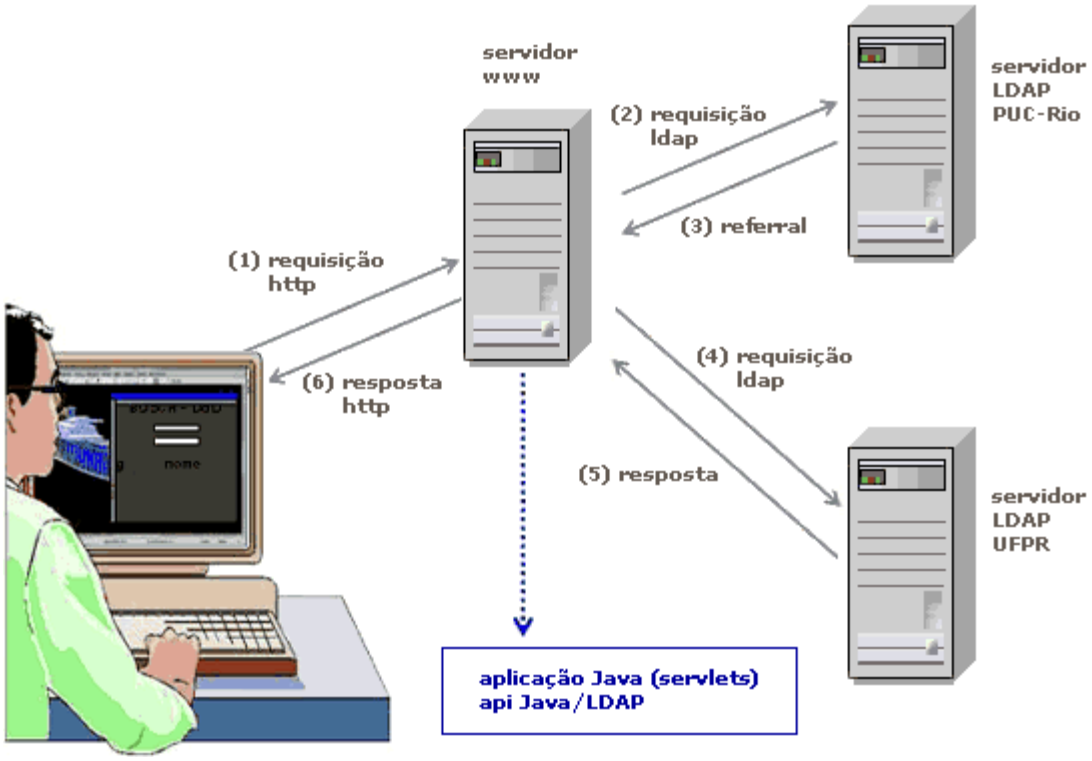
Transparência	Embora tendo complexidade alta, os usuários conseguem o que querem do Sistema, e sem a necessidade de se inteirar dessa complexidade.

Adaptado de Sommerville

Por outro lado, as principais desvantagens desse Sistema são:

- Alta Complexidade
- Segurança baixa
- Dificuldade de Gerenciamento
- Imprevisibilidade nos tempos de resposta

Veremos na próxima unidade os dois tipos de arquitetura de Sistemas Distribuídos mais importantes: a arquitetura cliente-servidor e a arquitetura de objetos distribuídos. De um modo geral os Sistemas Distribuídos são desenvolvidos com o uso da abordagem orientada a objetos.



Arquitetura de Multiprocessadores

O modelo mais simples de Sistema Distribuído é a Arquitetura de Multiprocessadores. Essa arquitetura, típica de sistemas em tempo real, consiste em uma série de diferentes processos que podem ser executados em processadores distintos.

Os Sistemas de software compostos de vários processos não são necessariamente sistemas distribuídos. Se mais de um processador estiver disponível, então a distribuição poderá ser implementada, mas os projetistas de sistema não precisam sempre considerar as questões de distribuição durante o processo de projeto. A abordagem de projeto para esse tipo de sistema é essencialmente aquela utilizada em sistemas de tempo real.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Computa%C3%A7%C3%A3o_distribu%C3%ADda

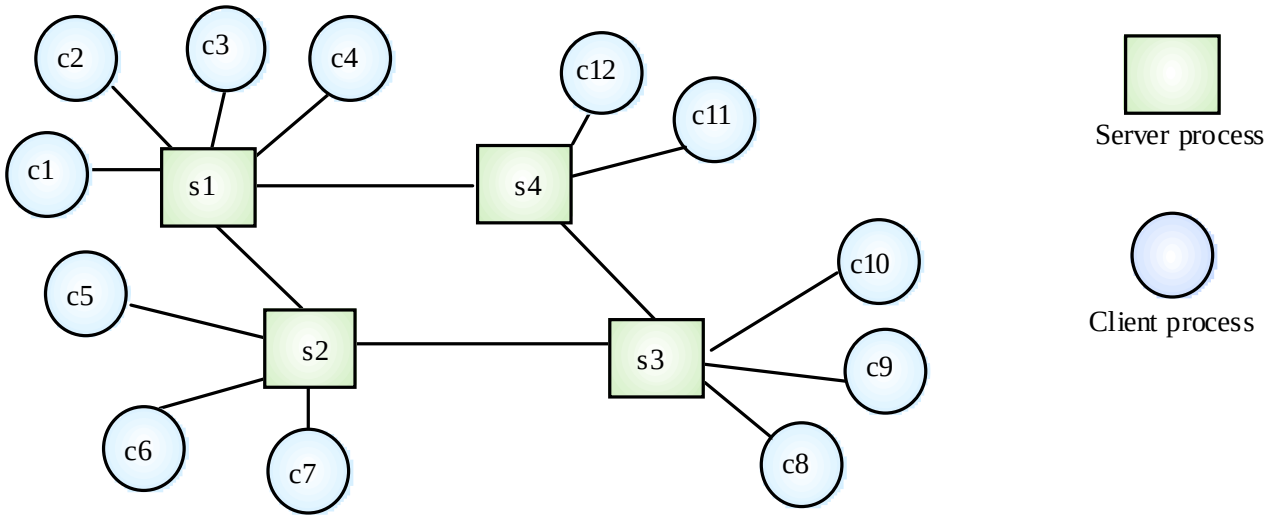
UNIDADE 20

Arquitetura cliente/servidor - Arquitetura de objetos distribuídos

Objetivo: Apresentar as características das principais arquiteturas

Arquitetura cliente-servidor

Pela definição de Orfali e Harkey, em uma arquitetura cliente-servidor (client/server), uma aplicação é modelada como um conjunto de serviços que são fornecidos por servidores e um conjunto de clientes que utilizam desses serviços. Veja o modelo lógico de uma arquitetura cliente-servidor distribuída na figura abaixo.



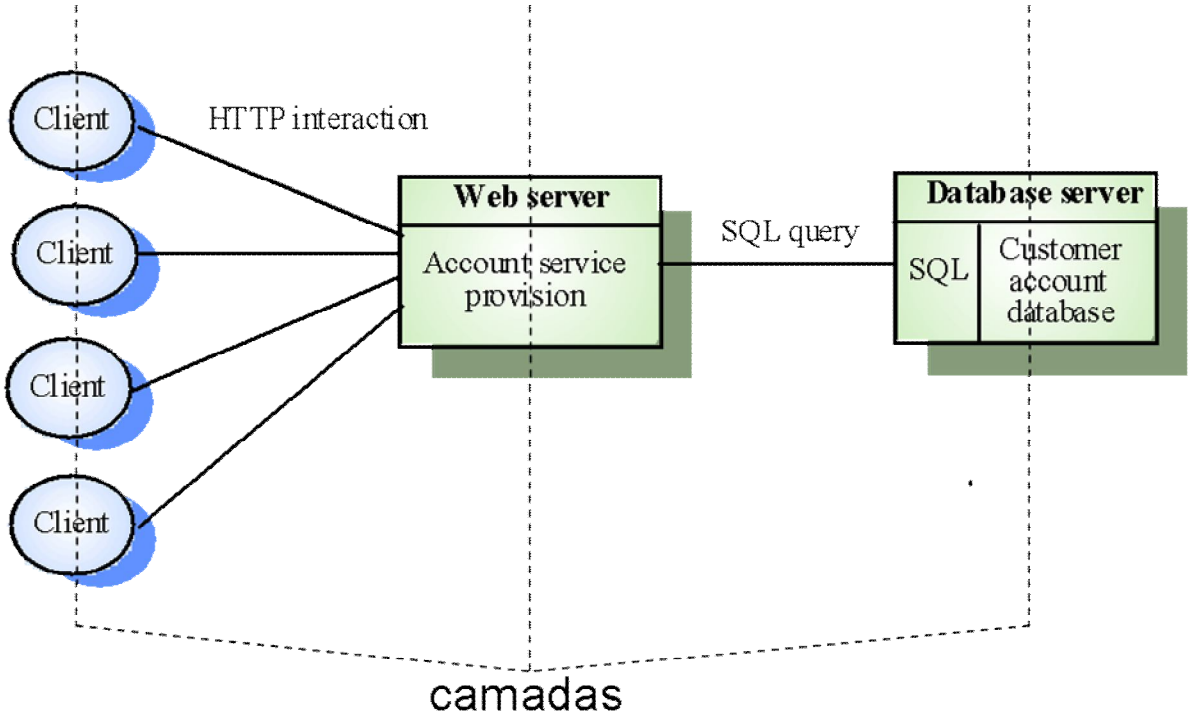
Um projeto de sistema cliente-servidor deve refletir a estrutura lógica da aplicação que está sendo desenvolvida (Sommerville). O tipo de arquitetura cliente-servidor mais utilizada em aplicações de baixa complexidade é a arquitetura cliente-servidor de duas camadas. Nessa situação a aplicação reside em um ou mais servidores, e um conjunto de clientes usufruindo desse serviço.

Existem basicamente dois tipos: “Thin client”, ou cliente magro, e “Fat client” (às vezes chamado de thick client), ou cliente gordo. No primeiro modelo todo o processamento é realizado no servidor. A segunda estrutura é mais complexa e mais comum. O servidor é

responsável somente pelo gerenciamento de dados. E nos clientes é implementada a lógica da aplicação e as interações com o usuário do sistema.

Arquitetura de 3 camadas

A arquitetura de três camadas não necessariamente representa que existam três tipos de computadores conecados numa rede. É possível implementar esse modelo simplesmente com um servidor assumindo a camada de dados e a de negócio simultaneamente. Para visualizarmos melhor todos esses relacionamentos vejamos a próxima figura.



Do lado direito temos um Database Server - Servidor de Banco de Dados fornecendo as solicitações do Web Server – Servidor Web (Camada de Dados). Do lado oposto, à esquerda, vemos os clients (clientes), na Camada de Apresentação, como interface com os usuários. E no meio, Camada de Negócio, o Web Server prove, através das regras de negócio, os serviços desejados ao conjunto de clientes.

Arquitetura de Objetos Distribuídos

A Arquitetura de Objetos Distribuídos é uma abordagem distinta da cliente/servidor onde elimina o conceito de distinguir quem é servidor ou mesmo cliente. Entretanto é criado um novo conceito chamado de middleware.

O middleware intermedeia os computadores a ele conectados. É também chamado de requisitor de objetos, e seu papel é fornecer uma interface contínua de comunicação entre esses objetos.

Os objetos no sistema podem ser implementados com o uso de diferentes linguagens de programação, podem ser executados em diferentes plataformas e seus nomes não precisam ser conhecidos por todos os outros objetos no sistema. Os dois padrões normais para o middleware são o CORBA e o DCOM.
Por todas as vantagens do padrão CORBA (flexibilidade, por ser genérico, sistemas operacionais adotados), organizado pelo OMG que é constituída por mais de 500 empresas, deva ser o padrão de fato que o mercado irá adotar.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Cliente-servidor
http://pt.wikipedia.org/wiki/Modelo_em_tr%C3%AAs_camadas
http://pt.wikipedia.org/wiki/Corba

Atividades

Pela importância do conceito do modelo de 3 camadas, e do padrão CORBA nos dias atuais, explore detalhadamente esses dois itens na Internet.

Atividades

Antes de dar continuidades aos seus estudos é fundamental que você acesse sua SALA DE AULA e faça a Atividade 2 no “link” ATIVIDADES.

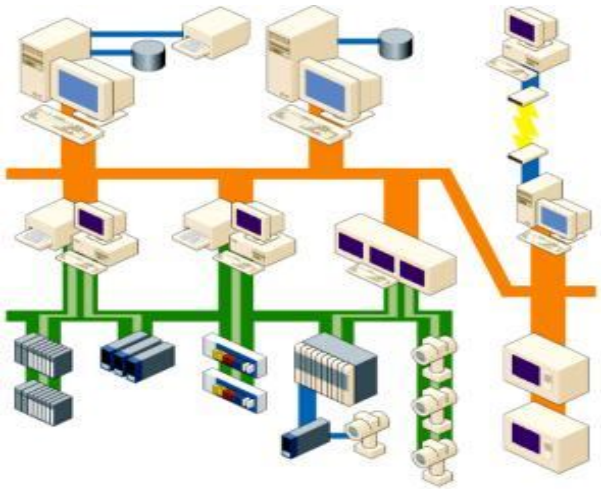
UNIDADE 21

Mudanças em Software - Dinâmica da Evolução de Programas - Evolução da Arquitetura

Objetivo: Contextualizar os impactos das mudanças de software.

Depois que os sistemas são entregues aos usuários/clientes, os softwares tem que sofrer mudanças para que possam responder às exigências das constantes mudanças impostas pelos mercados cada vez mais competitivos. Conforme Warren, existem diversas estratégias para essas mudanças:

ESTRATÉGIAS	Descrição
Evolução da Arquitetura	Os sistemas normalmente evoluem de uma arquitetura mais centralizada, para uma arquitetura cliente/servidor (veremos a seguir nesta mesma Unidade).
Manutenção	Quando a estrutura fica estável, mas sofre modificações para adaptar a novos requisitos dos usuários (veremos mais detalhes na Unidade 27).
Reengenharia	Ao contrário da Manutenção, sofre alterações na estrutura, para que o sistema torne-se mais fácil sua compreensão e também para alterações (veremos mais detalhes na Unidade 22).



Dinâmica da Evolução de Programas

Vamos exemplificar a Dinâmica da Evolução de Programas pegando como exemplo o Microsoft Word. Ele começou operando como um simples processador de texto, ocupando 256Kb de memória. Hoje é um gigante com tantos recursos disponíveis que a grande maioria dos usuários pouco os utiliza.

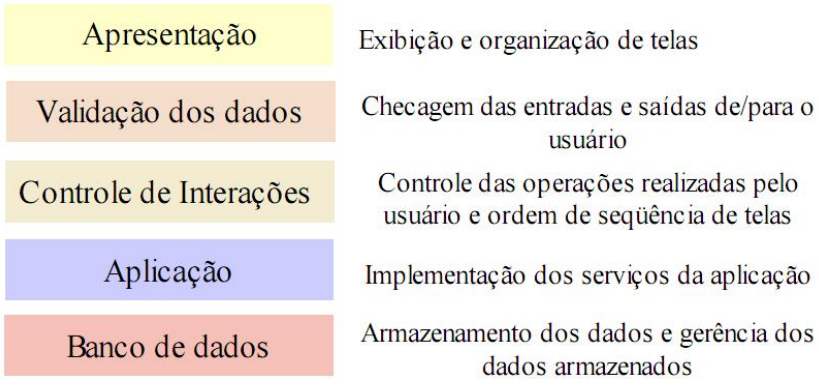
Necessita atualmente de muitos megabytes de memória, e mais um processador ágil para que possa ser executado. A evolução desse software, na verdade, passou por várias fases. Alguns podem achar que trabalham somente com o mais novo release desse editor de texto.

No entanto, ele não é uma simples sequência de revisões, mas sim um software que sofreu várias mudanças na sua estrutura. Em certos momentos, ele passou não só por manutenções, mas também por reengenharias. E atualmente até evoluindo na sua arquitetura para ficar mais condizendo com o mundo da Internet.

Evolução da Arquitetura

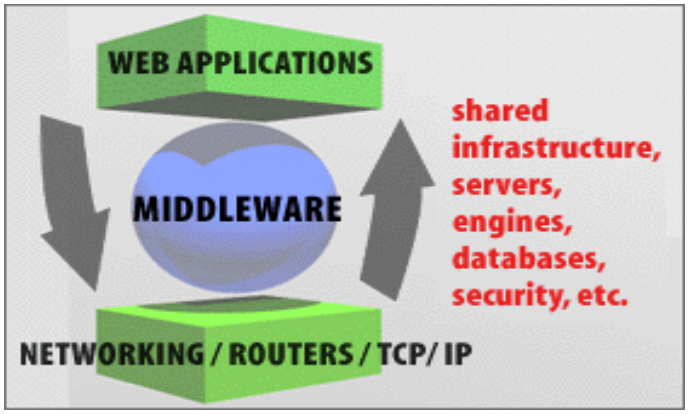
Os principais sistemas antigos, ou mesmo os legados, foram desenvolvidos na concepção de arquiteturas centralizadas. Hoje, a tendência geral é do desenvolvimento de sistemas com arquiteturas distribuídas cliente/servidor.

Quando estamos alterando a arquitetura de um sistema já existente é interessante que tenhamos um modelo de camadas lógicas para nos orientar. A imagem abaixo representa as estruturas de um sistema divididas por camadas, facilitando a modularização para uma arquitetura distribuída.



A Evolução da Arquitetura envolve modificar a arquitetura de um sistema, a partir de uma arquitetura centralizada, centrada em dados, para uma arquitetura distribuída. Tanto a interface com o usuário quanto a funcionalidade do sistema podem ser distribuídas.

Uma estratégia comum da Evolução da Arquitetura, para sistemas legados em especial, é encapsular o sistema legado como um servidor. E implementa-se uma interface com o usuário distribuída, que acesse a funcionalidade do sistema (por meio de um middleware de propósito especial).



Estudo Complementar

Wikipédia
www.twiki.dcc.ufba.br/pub/Residencia/MaterialModuloTI1/slides_aula04arquiteturadesoftware.pdf

Atividades

Para você fazer uma revisão geral dos principais tópicos vistos até aqui, visite o site sugerido em ESTUDO COMPLEMENTAR, e faça um resumo por escrito dos principais tópicos de seu interesse.

UNIDADE 22

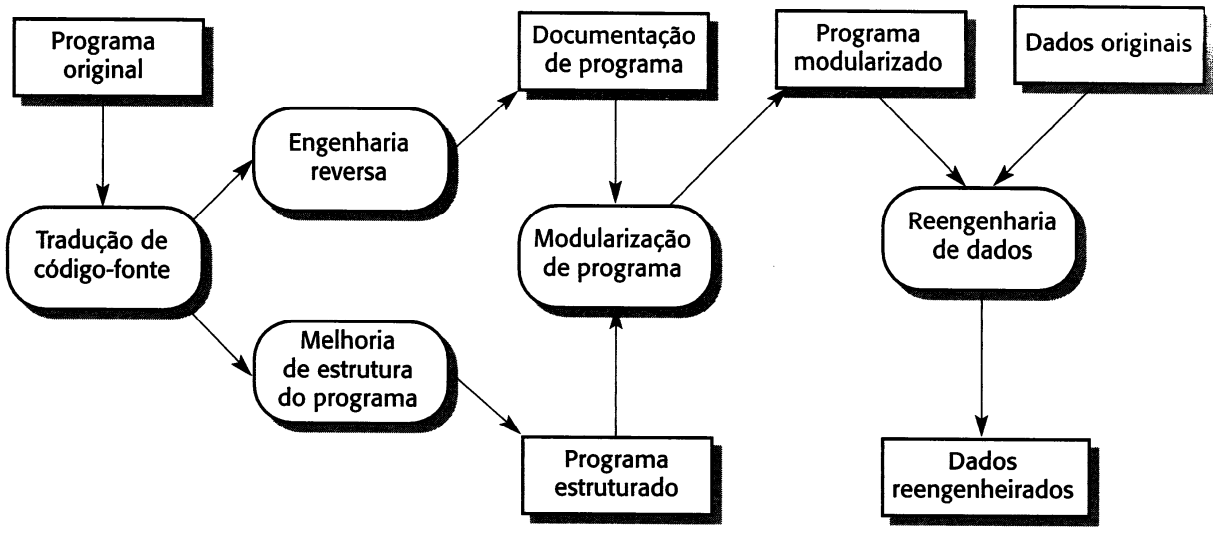
Reengenharia de Software - Tradução de código fonte - Engenharia Reversa - Melhoria de estrutura de programa

Objetivo: Visualizar a importância da reengenharia na Engenharia de Software.

A principal diferença entre Reengenharia de Software e o desenvolvimento de um novo Sistema é da onde que se parte esse próprio desenvolvimento. Num Sistema novo inicia-se com uma especificação escrita (os requisitos) dos usuários/clientes. Enquanto que, numa reengenharia, o sistema existente (normalmente um Sistema Legado) é que é a base para esse início.

Chikofsky e Cross chegam até definir o desenvolvimento tradicional como "Engenharia Direta", para distinguir da Reengenharia de Software.

Pode-se perceber, pela figura abaixo, a complexidade do processo de Reengenharia de Software. Embora, nem toda reengenharia passe por todos esses processos, essencialmente o programa é reestruturado. Vejamos mais detalhadamente o que cada processo realiza (caixas arredondadas).



Processo de Reengenharia conforme a visão de Sommerville

PROCESSOS	Descrição
Tradução de código-fonte	O programa é convertido da linguagem de programação original para uma versão mais moderna ou mesmo para uma nova linguagem mais adequada.
Engenharia Reversa	O programa é analisado conforme essas técnicas e as informações são extraídas dele, a fim de ajudar a documentar sua organização e funcionalidade.
Melhoria de estrutura do programa	A estrutura de controle do programa é analisada e modificada, a fim de torná-la mais fácil de ser lida e compreendida. Visa-se a manutenabilidade do sistema.
Modularização de programa	Partes em comum do programa são agrupadas e, quando apropriado, a redundância é removida. Em alguns casos, esse estágio pode envolver a transformação da arquitetura.
Reengenharia de dados	Os dados processados pelo programa são modificados, a fim de refletir as mudanças feitas nele. Ou mesmo adota-se uma nova estrutura de Banco de Dados.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Reengenharia
http://pt.wikipedia.org/wiki/Reengenharia_de_Processos

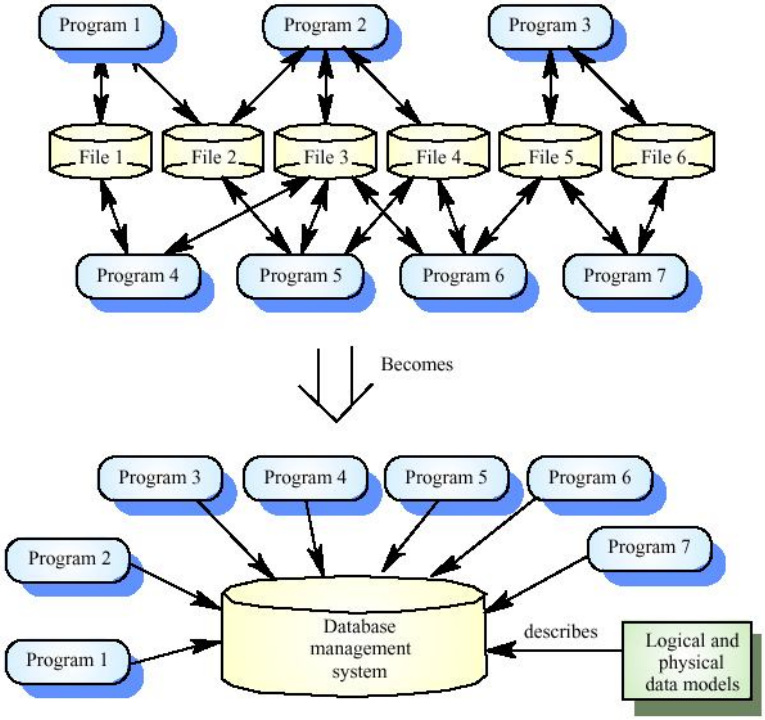
UNIDADE 23

Reengenharia de Dados e suas abordagens

Objetivo: Abordar os vários aspectos da reengenharia de dados.

A necessidade de analisar, reorganizar a estrutura dos dados, e mesmo os valores contidos num Sistema é chamada de Reengenharia de Dados. Vejamos, a seguir, as possíveis abordagens visualizadas por Sommerville:

ABORDAGENS	Descrição
Limpeza de dados	Os registros e valores de dados são analisados, a fim de melhorar sua qualidade. As duplicações são removidas, as informações redundantes são excluídas e um formato consistente é aplicado a todos os registros. Normalmente, isso não deve requerer quaisquer mudanças nos programas associados.
Extensão de dados	Nesse caso, os dados e programas associados passam pelo processo de reengenharia, a fim de eliminar os limites no processamento de dados. Isso pode exigir mudanças no programas para aumentar a extensão de campos, modificar limites superiores na tabelas e assim por diante. Os dados em si podem precisar ser reescritos e limpos, para que reflitam as mudanças no programa.
Migração de dados	Nesse caso, ocorre a migração dos dados para o controle de um Sistema de Gerenciamento de Banco de Dados. Os dados podem ser armazenados em arquivos separados ou serem gerenciados por um tipo de sistema de gerenciamento de Banco de Dados antigo. Essa situação é ilustrada na figura abaixo.



Estudo Complementar

Strategies for Data Reengineering
http://www.info.fundp.ac.be/~dbm/publication/2003/FNRS-ReEngineering.pdf

Atividades

Realize uma pesquisa na Internet sobre esse importante tópico. Procure em inglês ("Data reengineering") para encontrar mais material a respeito.

UNIDADE 24

Gerenciamento de Configuração - Gerenciamento de Mudanças - Gerenciamento de Versões e Releases

Objetivo: Abordar os principais aspectos do gerenciamento de mudanças.

Gerenciamento de Configuração

É o desenvolvimento e a aplicação de padrões e procedimentos para gerenciar um Sistema em desenvolvimento. Esses procedimentos definem como registrar e processar as mudanças do Sistema, como relacioná-los aos seus componentes e os métodos utilizados para identificar as diferentes versões desse Sistema (adaptado de Sommerville).

As quatro atividades principais do Gerenciamento de Configuração são:

ATIVIDADES	Descrição
Planejamento do Gerenciamento de Configuração	Descreve os padrões e os procedimentos que devem ser utilizados para o Gerenciamento de Configuração.
Gerenciamento de Mudanças	Com as constantes mudanças exercidas em cima dos softwares, as mesmas devem ser registradas e aplicadas ao sistema de forma prática e econômica.
Gerenciamento de Versões e Releases	Consiste em acompanhar e identificar o desenvolvimento das diferentes versões e releases de um Sistema.
Construção de Sistemas	Processo de compilar e ligar componentes de software em um programa que é executado em uma configuração-alvo específica.

Gerenciamento de Mudanças

Durante os testes de sistemas, ou ainda depois da entrega do software ao cliente, devem sofrer os procedimentos de Gerenciamento de Mudanças. O primeiro passo desse processo é

a utilização de um formulário intitulado de “Requisição de Mudança” (CRF – change request form).

O formulário CRF deverá conter informações do tipo: registro da solicitação da mudança, recomendações, custos, datas de solicitação, aprovação, implementação e validação da mudança. É aconselhável também existir um espaço para um esboço especificando como a mudança deverá ser implementada. Um exemplo do formulário CRF é mostrado a seguir:

Change Request Form

Project: Proteus/PCL-Tools **Number:** 23/94
Change requester: I. Sommerville **Date:** 1/12/98
Requested change: When a component is selected from the structure, display the name of the file where it is stored.

Change analyser: G. Dean **Analysis date:** 10/12/98
Components affected: Display-Icon.Select, Display-Icon.Display

Associated components: FileTable

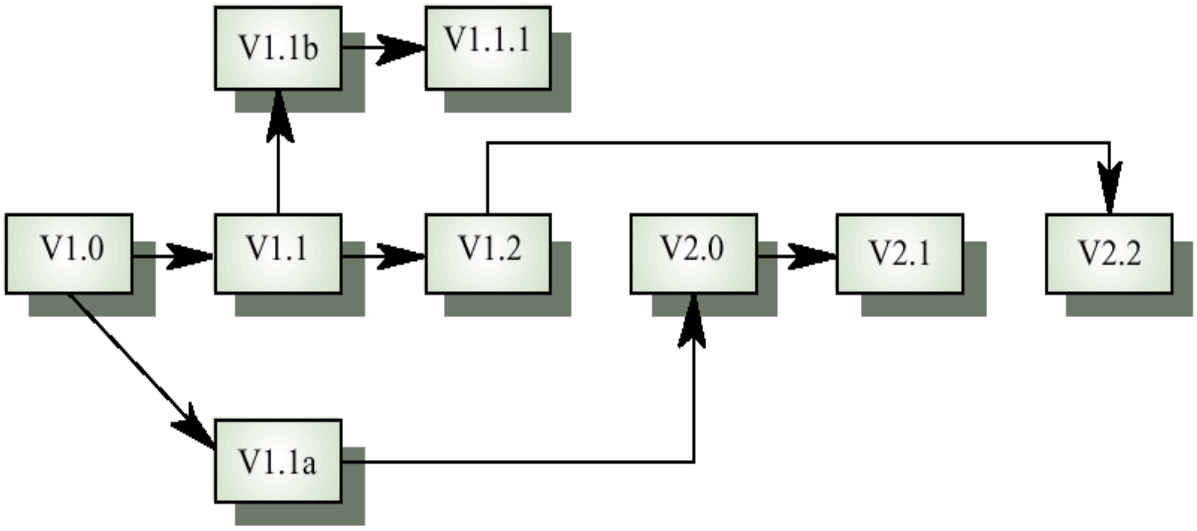
Change assessment: Relatively simple to implement as a file name table is available. Requires the design and implementation of a display field. No changes to associated components are required.

Change priority: Low
Change implementation:
Estimated effort: 0.5 days
Date to CCB: 15/12/98 **CCB decision date:** 1/2/99
CCB decision: Accept change. Change to be implemented in Release 2.1.
Change implementor: **Date of change:**
Date submitted to QA: **QA decision:**
Date submitted to CM:
Comments

Gerenciamento de Versões e Releases

Objetivo: acompanhar e identificar o desenvolvimento das diferentes versões e releases de um Sistema. Também chamado de versionamento.

O ***release*** de um sistema é uma versão que é distribuída para os clientes (Sommerville). Logo, sempre existem muita mais versões de um sistema do que releases, pois existem muitas versões criadas para testes, ou desenvolvimento interno e não são liberadas para os clientes.



Para a devida identificação de componentes existem três técnicas básicas:

TÉCNICAS BÁSICAS	Descrição
Numeração de versões	Esse é o esquema de identificação mais comum. Atribuí-se um número, explícito e único, de versão ao componente (ver figura)
Identificação baseada em atributos	Cada componente recebe um nome e um conjunto de atributos (que não é único em todas as versões).
Identificação orientada a mudanças	Além do anterior é associado uma ou mais solicitações de mudança.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Ger%C3%AAncia_de_Configura%C3%A7%C3%A3o_de_Software
http://pt.wikipedia.org/wiki/Sistema_de_controle_de_vers%C3%A3o

Atividades

Devido à excelente qualidade dos artigos colocados no Wikipédia especificamente neste tema, e para você explorar mais detalhadamente os assuntos dessa unidade, visite e leia todos os links mencionados no item anterior chamado ESTUDO COMPLEMENTAR.

UNIDADE 25

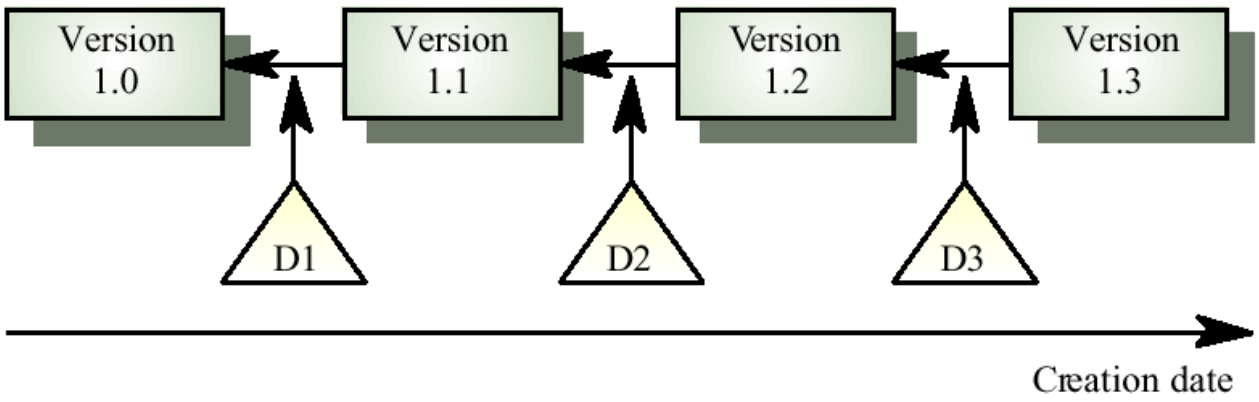
(continuação) Construção de Sistemas - Ferramenta CASE

Objetivo: Explorar o conceito da ferramenta CASE.

Construção de Sistemas

Na construção de um Sistema, a partir dos seus componentes, devem-se questionar os seguintes pontos:

1. Todos os componentes foram incluídos nas instruções de construção?
2. A versão de cada componente requerido foi incluído nas instruções de construção?
3. Todos componentes requeridos estão disponíveis?
4. Os arquivos de dados do componente utilizado são iguais aos da máquina-alvo?
5. A versão do compilador e outras ferramentas estão disponíveis?



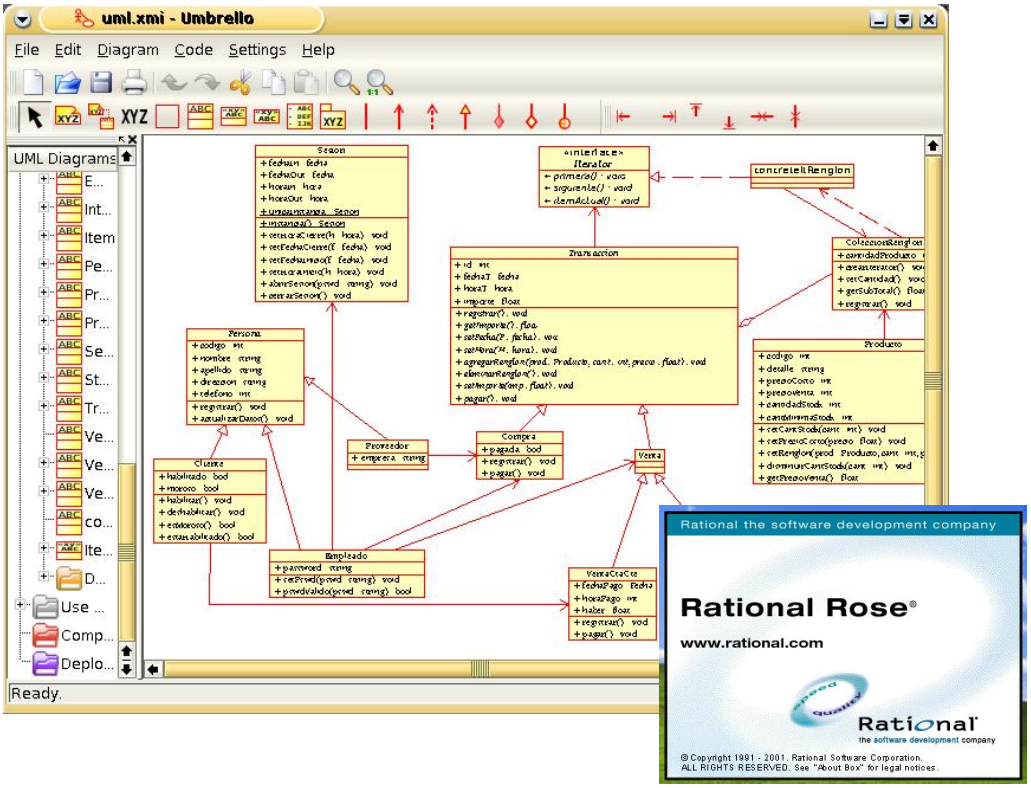
Ferramenta CASE

Uma ferramenta CASE (Computer-Aided Software Engineering) significa Engenharia de Software com o auxílio de computador. Ela possibilita apoiar as atividades de processo do software, como: a análise de requisitos, a modelagem de sistema, a depuração e os testes. Ferramentas CASE são constituídas com uma ampla gama de diferentes tipos de programas.

As ferramentas CASE podem também incluir um gerador de códigos que, automaticamente, origina o código-fonte a partir da modelagem do Sistema. Adicionalmente pode ter alguma orientação de processo, ao fornecer conselhos ao engenheiro de software sobre o que fazer em seguida.

Uma diferenciação que pode existir nas ferramentas CASE é a denominação Upper-CASE e Lower-CASE. As primeiras têm como utilidade de dar apoio à análise e ao projeto, ou seja, apoiar as fases iniciais do processo de software. As ferramentas Lower-CASE, por outro lado, são projetadas para dar apoio à implementação e aos testes, como depuradores, sistemas de análise de programa, geradores de casos de testes e editores de programas.

Na imagem abaixo damos um destaque para telas da ferramenta CASE como o Rational Rose da IBM:



Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Ferramenta_CASE

Atividades

Leia o site http://www2.dem.inpe.br/ijar/case.htm e faça um comparativo com o que você aprendeu até agora.

UNIDADE 26

Sistemas Legados - Estruturas dos Sistemas Legados - Avaliação dos Sistemas Legados

Objetivo: Valorar a importância dos sistemas legados para a Engenharia de Software.

As empresas continuamente evoluem em seus Sistemas, adaptando-os a sua realidade, e as constantes mudanças do mercado. No entanto, descartar Sistemas mais antigos, os legados, e puramente substituí-los por softwares mais modernos envolve riscos empresariais significativos.

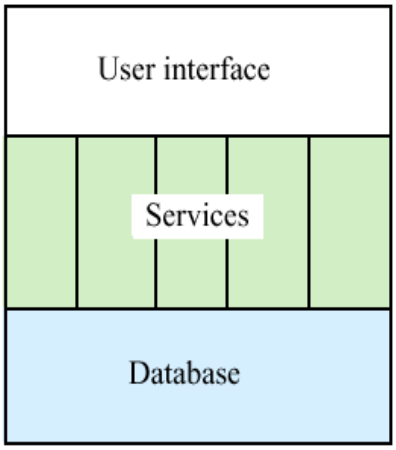
Podemos dar um simples exemplo atual. Imagine, de repente, mudarmos todos os Sistemas Operacionais Windows XP, de uma grande empresa, para o novo Windows Vista, ou mesmo para um Linux. A quantidade de problemas e adaptações necessárias será tão grande, que poderia chegar a paralisar essa empresa.

Empresas com grande número de Sistemas Legados enfrentam dilemas fundamentais. Se continuarem com os sistemas velhos, os custos de adaptação aumentam. E se substituírem por novos, terão um custo inicial alto, e com a possibilidade de não atenderem as expectativas. Exige-se estar ciente das técnicas de Engenharia de Software para resolver esses problemas.

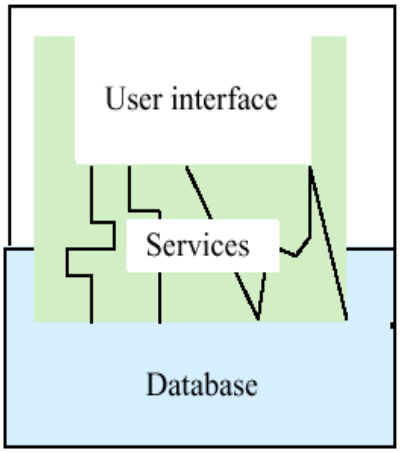
Estruturas dos Sistemas Legados

Pode-se dividir um Sistema Legado, do ponto vista didático, em seis partes lógicas: Hardware do Sistema, Software de Apoio, Software de Aplicação, Dados de Aplicação, Processos de Negócios e Políticas e Regras de Negócios. Mas, veremos a seguir, que na prática, pode-se dividir um Sistema Legado de forma mais simplificada.

Ao observarmos atentamente as imagens a seguir, veremos as estruturas ideais e as reais de um Sistema Legado. O ideal seria termos a estrutura da esquerda, para numa possível migração termos cada componente claramente separado. No entanto, na grande maioria das vezes, encontramos a estrutura da direita.



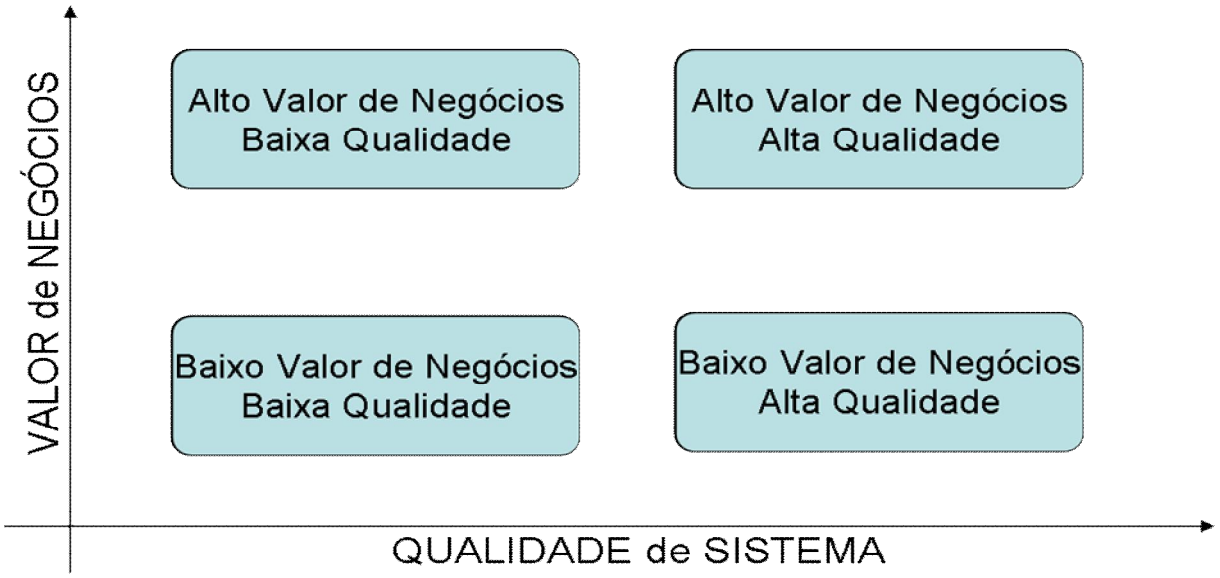
Ideal model for distribution



Real legacy systems

Os serviços sobrepõem interagindo com outros componentes do Sistema. A interface com o usuário e o código do serviço está integrada nos mesmos componentes, e pode não haver uma nítida distinção entre os serviços e o Banco de Dados do Sistema. Nesses casos, pode não ser possível identificar as partes do Sistema que podem ser distribuídas (Sommerville).

Avaliação dos Sistemas Legados



Podemos através da figura acima caracterizar tipicamente as ações que devemos tomar quanto aos Sistemas Legados. Enquanto num eixo mensuramos a importância que um Sistema Legado tem para o negócio da empresa, no outro quantificamos a sua respectiva qualidade. Vamos, a seguir, tabular essas ações a serem tomadas.

Valor X Qualidade	Descrição
Alto Valor de Negócios X Baixa Qualidade	São sistemas com importante contribuição à empresa e não devem ser descartados. Contudo, pela sua baixa qualidade, os custos operacionais são altos, de modo que são fortes candidatos à reengenharia ou à substituição total do sistema.
Baixo Valor de Negócios X Baixa Qualidade	Manter sistemas desse tipo em operação é dispendioso, e a taxa de retorno de investimento para os negócios é bastante pequena. Esses sistemas são fortes candidatos a receberem nenhum investimento, e mesmo a serem descartados.
Alto Valor de Negócios X Alta Qualidade	Sistemas com essas características devem ser mantidos em operação pela sua importância. E pela sua alta qualidade significa que não é necessário investir na sua transformação ou substituição. Portanto, devem continuar a manutenção normal no sistema.
Baixo Valor de Negócios X Alta Qualidade	São sistemas que não contribuem muito para os negócios, mas por outro lado, a manutenção não é muito dispendiosa. Não vale o risco de substituir esses sistemas, de modo que a manutenção normal pode ser continuada ou eles podem ser descartados.

Adaptado de Sommerville

Estudo Complementar

Artigo “Uma Proposta de Evolução em Sistemas Legados”: http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER04/Luciana_Paiva.pdf

Atividades

Levante na sua empresa, ou na de colegas, quantos Sistemas Legados existem. Quais são as características deles? Há quanto tempo eles foram desenvolvidos? Qual é o processo de mantê-los no ar?!?

UNIDADE 27

Manutenção: fundamentos da fase de Manutenção de Software, tipos de Manutenção, procedimentos, técnicas e ferramentas

Objetivo: Identificar as principais características da manutenção de software.

As leis de Lehman (1985) foram produzidas com base no estudo da mudança em Sistemas. Foram examinados o crescimento e a evolução de uma série de grandes sistemas de software para chegar nessas leis. Duas delas que destacamos são a da:

- **Mudança Contínua**: afirma que um programa utilizado em um ambiente do mundo real necessariamente tem de ser modificado ou se tornará de maneira progressiva menos útil nesse ambiente;

- **Aumento da Complexidade**: à medida que um programa em evolução se modifica, sua estrutura tende a se tornar mais complexa. Recursos extras precisam ser dedicados para preservar e simplificar a estrutura.

Tipos de Manutenção

A manutenção será necessária durante todo o Ciclo de Vida útil, e pode ocorrer motivada por três tipos fundamentais:

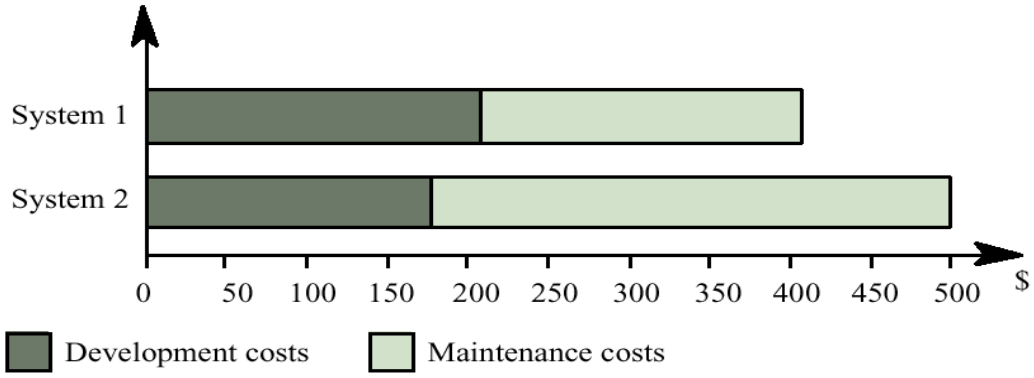
Tipos de Manutenção	Descrição
Manutenção para reparar os defeitos no software	A correção de erros de codificação é um processo relativamente barato comparado com os erros de projeto. Os maiores custos estão nos erros de requisitos, pois irá implicar num reprojeto.
Manutenção para adaptar o software a um ambiente operacional diferente	É a típica manutenção de adaptação sofrida por alguma alteração no software de apoio tal como o Sistema Operacional, Banco de Dados ou mesmo o próprio hardware.

Manutenção para fazer acréscimos à funcionalidade do sistema ou modificá-la	Na alteração dos requisitos, devido a mudanças organizacionais, ou nos negócios, que são bastante constantes, ocorre a manutenção mais comum entre todas as outras.

Adaptado de Sommerville

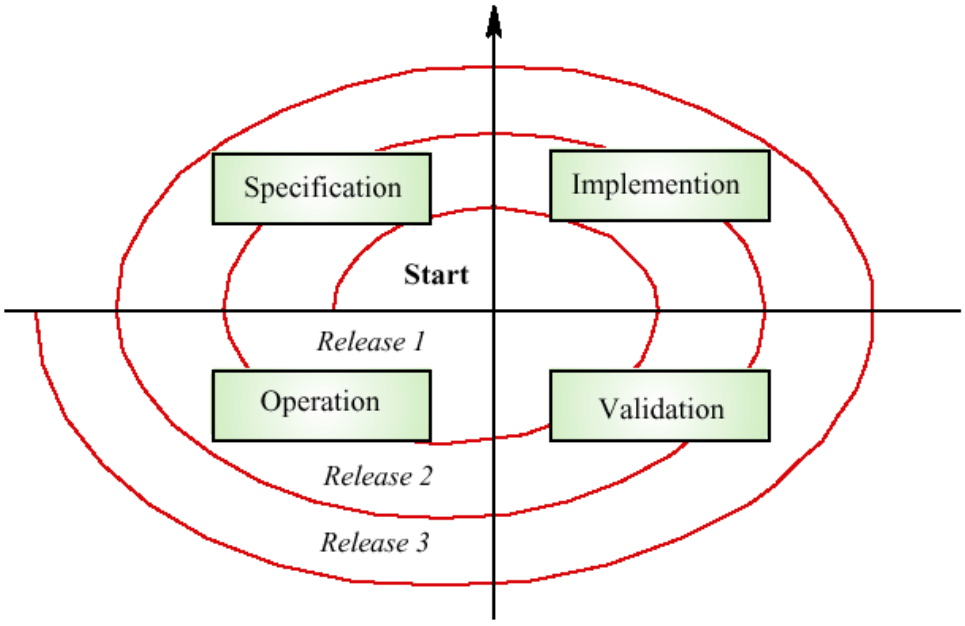
Procedimentos de Manutenção

O Processo de Manutenção é normalmente iniciado pelos pedidos de mudança por parte dos vários usuários que utilizam o Sistema. Isso pode ser de maneira informal, ou preferencialmente formalizado, com uma documentação estruturada. Em seguida é verificado o custo e o impacto das mudanças sugeridas. Com as mudanças aprovadas, um novo release do sistema é planejado.



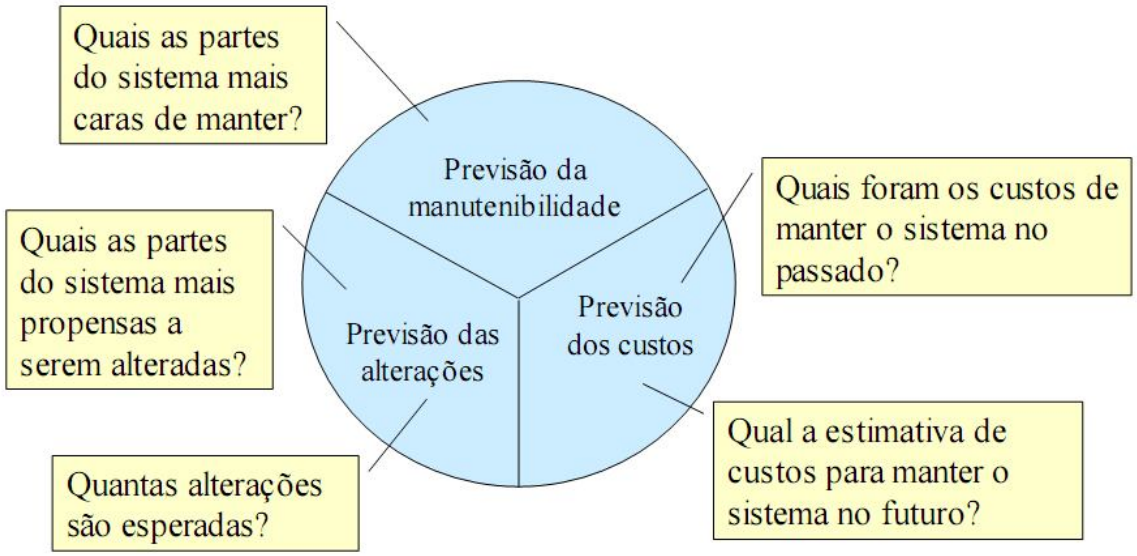
Repare atentamente na figura acima. Veja que uma vez bem estruturado um sistema, no caso o System 1, que embora tenha despendido maiores custos de desenvolvimento, exigiu no período de manutenção menos tempo e recursos. Ou seja, o System 2 foi desenvolvido mais rapidamente, mas por não investir, ou visualizar, nos processos de manutenção, ao chegar nessa fase, despende maior tempo e custos.

Interessante observar que a manutenção segue o mesmo modelo do processo de desenvolvimento de sistema. Na figura abaixo vemos que a representação das etapas que a manutenção que está sendo realizada, segue o mesmo Modelo Espiral que estudamos na Unidade 7.



Existem equipes de manutenção que atuam somente em corretivas, ou seja, somente quando existir um pedido dos usuários é que se atua no problema. No entanto, a melhor estratégia é a Manutenção Preventiva na qual se detecta previamente onde estão ocorrendo um maior número de corretivas, e destaca-se uma força-tarefa para realizar uma reengenharia nesses processos.

No quadro a seguir, veja as principais perguntas a serem feitas no Processo de Manutenção:



Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Manuten%C3%A7%C3%A3o_de_software

Atividades

Na sua empresa como é realizada a atividade de Manutenção? As equipes de Informática estão sempre realizando corretivas, ou estão mais focadas em preventivas?!?
Quanto porcentualmente no mês você imagina é dedicado para essa função? Essa atividade é específica de uma equipe, ou é a mesma de desenvolvimento?!?

UNIDADE 28

Gestão de Projetos de Software e o PMBOK

Objetivo: Apresentar os princípios da gestão de projetos e a base do PMBOK.

Gestão de Projetos de Software

A Gerência de Projetos se preocupa em entregar o sistema de software no prazo e de acordo com os requisitos estabelecidos, levando em conta sempre as limitações de orçamento e tempo.

A Gestão de Projetos de Software se caracteriza por tratar sobre um produto intangível, muito flexível e com processo de desenvolvimento com baixa padronização. Ou seja, não trata de processos rotineiros ou de prévio conhecimento.

A gestão efetiva de projetos de software focaliza os quatros P's: (P)essoal, (P)roduto, (P)rocesso e (P)rojeto (Pressman). Quanto ao PESSOAL existe até um padrão equivalente ao CMM, estudado anteriormente, intitulado de PM-CMM. Um dos pontos importantes do PRODUTO é a determinação adequada dos objetivos e o escopo do projeto. No PROCESSO é estabelecido o ferramental para apoiar um plano abrangente de desenvolvimento de software. E finalmente no PROJETO as diretrizes do PMBOK (que veremos a seguir) auxiliam na construção de um projeto de sucesso.

O planejamento de um projeto de desenvolvimento de software inclui:

- Organização do Projeto (incluindo equipes e responsabilidades)
- Estruturação das Tarefas (WBS - Work Breakdown Structure)
- Cronograma do Projeto (normalmente um Diagrama de Barras)
- Análise de Risco

Essas atividades sofrem com dificuldades típicas de desenvolvimento de software. A produtividade não é linear em relação ao tamanho da equipe e o aumento de produtividade não é imediato devido os custos de aprendizado dos novos membros. A diminuição de qualidade para acelerar o desenvolvimento constantemente prejudica a produtividade.

A estimativa de dificuldades e custos de desenvolvimentos são muito difíceis, além do surgimento de problemas técnicos. Esses fatores requerem uma Análise de Riscos cuidadosa.

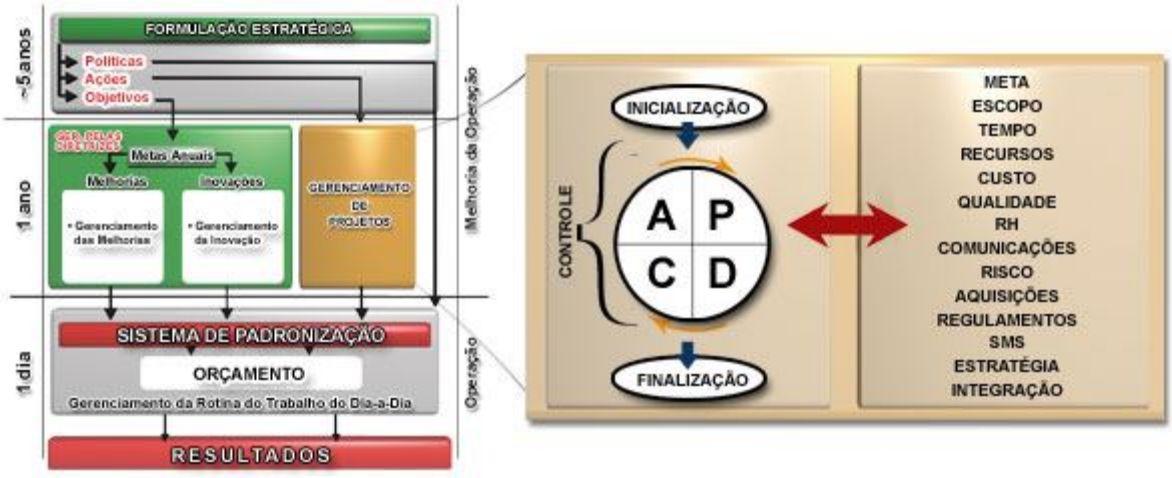
PMBOK

O PMBOK é uma importante referência em Gerenciamento de Projetos. Desenvolvido pelo PMI (Project Management Institute) possibilitou utilizar termos em comum para se discutir, escrever e aplicar o Gerenciamento de Projetos.

O guia atualmente é base para uma certificação específica e bem remunerada no mercado. Como os profissionais de Engenharia de Software praticamente são gerentes de projetos, existe a necessidade do entendimento desse conjunto de práticas para o bom desenvolvimento de um projeto de software.

A estrutura do PMBOK Guide (veja a imagem a seguir - os números entre parênteses representam respectivamente os blocos da imagem) contempla nove áreas de conhecimento específicas, que são:

- Gerenciamento da Integração do Projeto (4)
- Gerenciamento do Escopo do Projeto (5)
- Gerenciamento do Prazo do Projeto (6)
- Gerenciamento do Custo do Projeto (7)
- Gerenciamento da Qualidade do Projeto (8)
- Gerenciamento dos Recursos Humanos do Projeto (9)
- Gerenciamento da Comunicação do Projeto (10)
- Gerenciamento dos Riscos do Projeto (11)
- Gerenciamento das Aquisições do Projeto (12)



Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/PMBOK
http://pt.wikipedia.org/wiki/Gerenciamento_de_Projetos

Site do PMBOK, e no Brasil
http://www.pmi.org
http://www.pmisp.org.br/exe/educacao/pmbok.asp

Atividades

Para você explorar mais adequadamente os objetivos do PMBOK, baixe o arquivo do site:
http://www.prodepa.psi.br/sqp/pdf/Capítulo 01 - Introdução.pdf
e leia o primeiro capítulo, em português, desse importante livro de Gerenciamento de Projetos.

UNIDADE 29

Gerenciamento de Qualidade e Estratégias de Teste de Software

Objetivo: Visualizar os elementos da qualidade e de teste de software.

Existe uma relação direta entre a qualidade do produto de software desenvolvido, e qualidade do processo de software utilizado para criar esse produto. Ou seja, qualquer melhoria no processo de software irá resultar diretamente um impacto na qualidade do produto final.

Portanto, os principais itens do PROCESSO que deverão receber atenção especial do desenvolvedor para a melhoria da qualidade, e as perguntas mais significativas a serem questionadas são:

ITENS	Perguntas
Facilidade de compreensão	Até que ponto o processo está definido e com que facilidade se compreende a definição do processo?
Visibilidade	As atividades culminam em resultados nítidos, de forma que o progresso do processo seja visível?
Facilidade de suporte	Até que ponto as atividades do processo podem ser apoiadas por ferramentas CASE?
Aceitabilidade	O processo é aceitável e utilizável pelos desenvolvedores?
Confiabilidade	Os erros podem ser evitados ou identificados antes que o produto seja entregue aos usuários?
Robustez	Existe continuidade no processo mesmo que surjam problemas inesperados?
Facilidade de manutenção	Existe evolução no processo para refletir os requisitos mutáveis da organização ou para receber melhorias?
Rapidez	A partir de uma determinada especificação com que rapidez pode ser alterado o processo?

(adaptado de Sommerville)

Os principais fatores da qualidade de produtos de software, ou mesmo para quaisquer outros produtos intelectuais (livros, filmes, etc.), são:

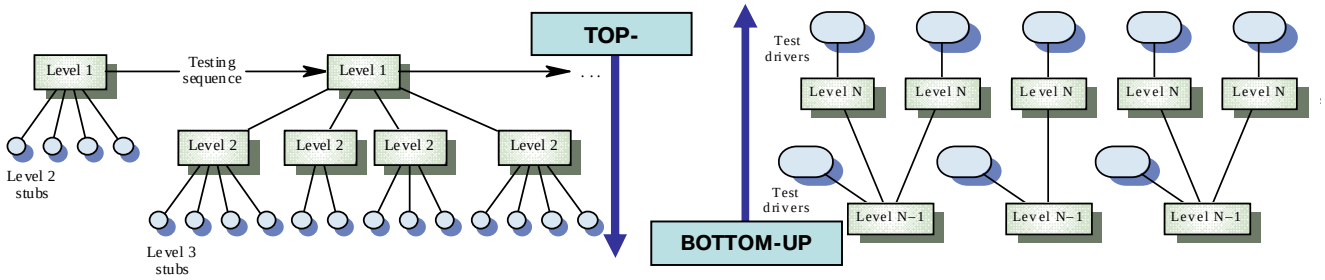
- A tecnologia de desenvolvimento
- Qualidade do pessoal
- Qualidade do processo (como vimos anteriormente !)
- Custo, tempo e cronograma

E de todos esses elementos o mais significativo é o último. Pois se um projeto tiver um orçamento baixo, ou ainda pior, um cronograma de entrega fora da realidade, a qualidade será diretamente afetada.

Estratégias de Teste de Software

Um princípio básico na realização de Testes de Software (principalmente em Sistemas de media complexidade para cima) é diferenciar a equipe puramente de desenvolvimento, da equipe especificamente de testes. Ou seja, quem desenvolve não testa, e quem testa não desenvolve.

Uma das estratégias de Teste de Software é a abordagem Top-down e a Bottom-up (veja a figura abaixo). Enquanto que a primeira (lado esquerdo da figura) tenta ver a integração de todos os componentes de um Sistema começando pelos níveis superiores, a segunda a Bottom-up (lado direito da figura), começa pelos níveis inferiores. As duas estratégias têm pontos positivos e negativos. O mais comum é utilizar a abordagem Top-down, por ser mais natural.



Testes Alfa e Beta

Quando um software é construído especificamente para um cliente, é normal ele passar por um Teste de Aceitação. Esse teste por ser conduzido pelo próprio usuário, pode passar por uma bateria de testes levando às vezes semanas, ou mesmo meses, para ser finalizado.

No entanto, se o software é feito para vários clientes, o Teste de Aceitação não é viável de ser realizado por cada usuário principal. Por isso, a estratégia melhor a ser aplicada é a dos Testes Alfa e Beta.

Para a realização dos Testes Alfa existe a necessidade de um ambiente controlado. Ou seja, os usuários são levados a testar o software desde os seus estágios iniciais de instalação, até a sua operação completa. Tudo isso é realizado num ambiente especial, onde fiquem registradas todas as impressões dos usuários, suas reações às interfaces homem-máquina, e assim por diante.

Os Testes Beta são realizados exclusivamente no *habitat* do usuário. E é realizado tipicamente sem a presença dos desenvolvedores, ao contrário do Alfa. Normalmente é selecionado um público especial de usuários, com um perfil crítico e colaborador. É importante a escolha adequada de usuários nesse tipo de teste. Pois existe a necessidade do próprio usuário deixar todas suas observações, questionamentos e sugestões, registrados de forma minuciosa e com riqueza de detalhes.

Testes Caixa-Branca e Caixa-Preta

O Teste Caixa-Branca, também chamado de Teste Estrutural, foca-se mais nos possíveis erros internos ao Sistema. E o Teste Caixa-Preta visa identificar as falhas em seu comportamento externo.

Enquanto o Teste Caixa-Branca realiza testes na estrutura dos componentes de um Sistema, o Caixa-Preta refere-se aos testes que são conduzidos na interface do software.

Para realizar os Testes da Caixa-Branca são utilizadas técnicas tais como:

- Testes de Caminho Básico
  - Notação de Grafo de Fluxo
  - Caminhos Independentes de Programa
  - Derivação de Casos de Teste
  - Matrizes de Grafos
- Testes de Estrutura de Controle
  - Teste de Condição
  - Teste de Fluxo de Dados
  - Teste de Ciclo

No caso dos Testes de Caixa-Preta que focalizam nos requisitos funcionais do software são os mais utilizados no mundo prático. Os Caixa-Branca demandam muito tempo, e praticamente não conseguem realizar todas as possibilidades de resposta que um software fornece. As principais técnicas utilizadas nos Testes de Caixa-Preta são:

- Métodos de Teste baseados em Grafo
  - Modelagem de fluxo de transação
  - Modelagem de estado finito
  - Modelagem do fluxo de dados
- Particionamento de Equivalência
- Análise de Valor-limite
- Teste de Matriz Ortogonal

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/Qualidade_de_Software
http://pt.wikipedia.org/wiki/Teste_de_software

Atividades

Responda, por escrito, aos questionamentos abaixo:

Que itens você levaria em consideração para a melhoria da qualidade de um software?
Quais são as diferenças das estratégias aplicadas para testar um software?

UNIDADE 30

Engenharia de Software na WEB – Sistemas e Aplicações baseadas na WEB

Objetivo: Apresentar as diferenciações quanto ao desenvolvimento na WEB.

A Engenharia de Software na Web, também utilizada pela sigla WebE, é o processo usado para criar WebApps (aplicações baseadas na Web) de alta qualidade. Embora os princípios básicos da WebE sejam muito próximos da Engenharia de Software clássica, existem peculiaridades especificas e próprias.

Com o advento do B2B (e-business) e do B2C (e-commerce), e ainda mais com aplicações para a Web 2.0, maior importância ficou sendo esse tipo de engenharia. Como as WebApps evoluem continuamente, devem ser estabelecidos mecanismos para controle de configuração, garantia de qualidade e suporte continuado.

Tipicamente as WebApps são desenvolvidas incrementalmente, sofrendo modificações frequentemente, e possuindo cronogramas extremamente curtos. Por tudo isso, normalmente, o modelo de processo utilizado na WebE é o da filosofia do desenvolvimento ágil, por ter uma abordagem de desenvolvimento simples e com ciclos rápidos de desenvolvimento.

Os métodos adotados na WebE são os mesmos conceitos e princípios da Engenharia de Software. No entanto, os mecanismos de análise, projeto e teste devem ser adaptados para acomodar as características próprias das WebApps.

Quanto às ferramentas e tecnologias aplicadas na WebE englobam várias linguagens de modelagem (HTML, VRML,XML, etc.), recursos baseados em componentes (CORBA,COM, ActiveX, .NET, AJAX, etc.), navegadores, ferramentas multimídia, ferramentas de autoria de sites, ferramentas de conectividade de Banco de Dados, ferramentas de segurança, servidores e utilitários de servidor, e ferramentas de gestão e análise de sites.

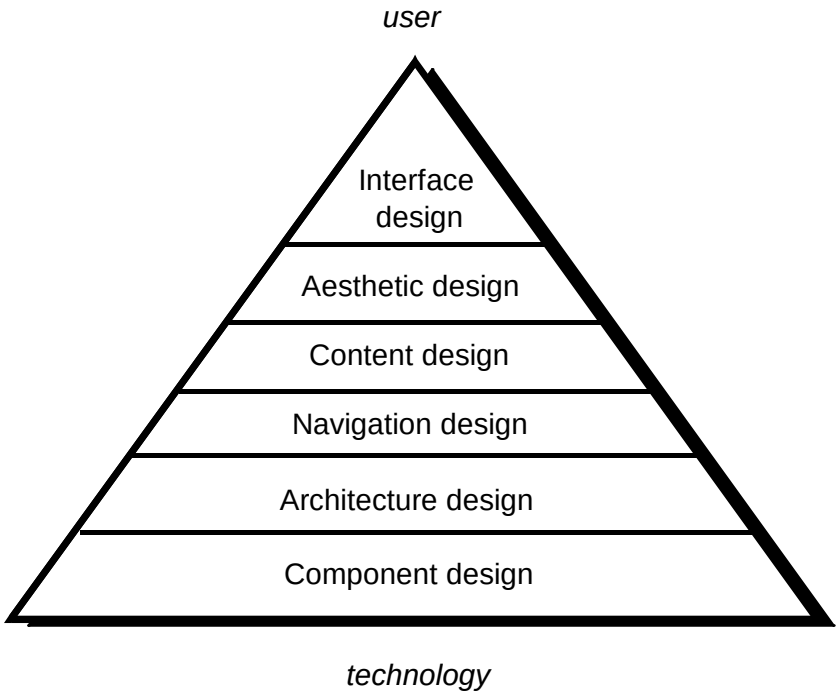
Para quem desenvolve aplicações na Web deve observar os seguintes requisitos de qualidade:

- Usabilidade
- Funcionalidade
- Confiabilidade
- Eficiência
- Manutenibilidade
- Segurança

- Disponibilidade
- Escabilidade
- Prazo de colocação no mercado

Pirâmide de Projeto da WebE

Um projeto no contexto de Engenharia da Web leva a um modelo que contém a combinação adequada de estética, conteúdo e tecnologia. Repare na figura a seguir. Enquanto a base da pirâmide é a tecnologia (technology), todos os seus itens são direcionados para atender o usuário (user).



Cada nível da pirâmide representa uma atividade de projeto. Veja maiores detalhes de cada fase no quadro abaixo, vendo a pirâmide de cima para baixo:

Nível da Pirâmide	Descrição
Projeto de Interface	Descreve a estrutura e organização da interface com o usuário.
Projeto Estético	Atenta para os esquemas de cor, leiaute, fonte, uso de gráficos, etc.
Projeto de Conteúdo	Define a estrutura e o esboço de todo o conteúdo, relacionando os objetos de conteúdo.

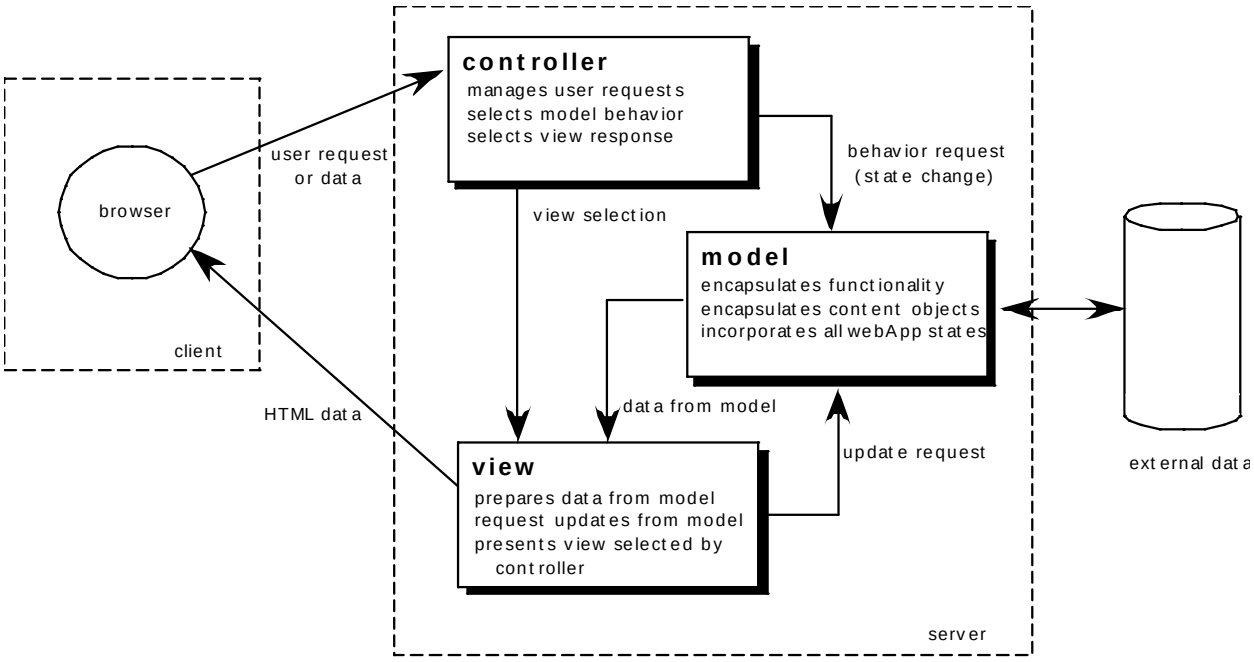
Projeto de Navegação	Representa o fluxo de navegação entre objetos de conteúdo e todas as funções da WebApp.
Projeto Arquitetural	Identifica a estrutura de hipermídia para a WebApp.
Projeto de Componente	Desenvolve a lógica de processamento detalhada necessária para implementar os componentes funcionais.

Adaptado de Pressman

Arquitetura da WebApp

Conforme Jacyntho, as aplicações devem ser construídas usando camadas nas quais diferentes preocupações são levadas em conta. Em particular, dados da aplicação devem ser separados dos conteúdos da página da Web. E esses conteúdos, por sua vez, devem ser claramente separados dos aspectos da interface.

Os autores sugerem um projeto de arquitetura em três camadas (veja a figura abaixo) que desaclopa a interface da navegação, e do comportamento da aplicação. Os mesmos argumentam que manter a interface, aplicação e navegação separadas simplifica a implementação e aumenta o reuso.



A arquitetura mais utilizada nesse caso é a Modelo-Visão-Controlador (MVC – Model-View-Controller). Embora seja um padrão de projeto arquitetural desenvolvido para o ambiente Smalltalk (linguagem de programação orientada a objeto), ele pode ser utilizado para qualquer aplicação interativa. Veja os detalhes de cada item da arquitetura MVC na tabela abaixo:

ITEM do MVC	Descrição
MODELO	Encapsula funcionalidade, objetos de conteúdo e incorpora todos os estados da WebApp. É o conteúdo em si, normalmente armazenado num Banco de Dados externo.
VISÃO	Prepara dados do Modelo, requisita atualizações dele, apresenta visão selecionada pelo Controlador. Geralmente é a própria página HTML.
CONTROLADOR	Gera requisições do usuário, seleciona comportamento do Modelo e seleciona resposta de visão. É o código que gera os dados dinâmicos para dentro da página HTML.

Estudo Complementar

Wikipédia
http://pt.wikipedia.org/wiki/MVC
http://pt.wikipedia.org/wiki/Web_2.0
http://pt.wikipedia.org/wiki/Web_3.0

Atividades

Veja os links colocados no ESTUDO COMPLEMENTAR, e escreva quais são as características e diferenças da Web 2.0 e da Web 3.0. Aproveite e veja com maior riqueza de detalhes a arquitetura MVC no link
http://pt.wikipedia.org/wiki/MVC

MÓDULO: ENGENHARIA de SOFTWARE

Apresentação

- O estudo da Engenharia de Software permite entender os principais aspectos da produção e manutenção de programas e Sistemas. Para tanto, abordam-se desde os estágios iniciais da construção de um Sistema, até mesmo a manutenção de Sistemas legados.

Objetivo

- Apresentar conceitos básicos da Engenharia de Software. Detalhar os principais métodos, ferramentas e procedimentos ligados à disciplina da Engenharia de Software. Discutir os principais aspectos que levam as organizações a utilizar as melhores práticas da Engenharia de Software.
- Capacitar os alunos a identificar quais os métodos, ferramentas e procedimentos mais adequados ao processo de desenvolvimento ou manutenção de softwares.

Carga horária

- 40 horas

Ementa

- Apresentação dos métodos, ferramentas e procedimentos da Engenharia de Software, através das fases do Ciclo de Vida do Desenvolvimento de Software. E como podem ajudar as organizações a desenvolver Sistemas de acordo com os custos, prazos, recursos e qualidades planejadas.

Requisitos

- Ter realizado e sido aprovado no módulo anterior.

Bibliografia do módulo

- PRESSMAN, Roger. *Engenharia de Software*. São Paulo: McGraw-Hill Brasil, 2006
- SOMMERVILLE, Ian. *Engenharia de Software*. São Paulo: Pearson Addison Wesley, 2005
- REZENDE, Denis Alcides. *Engenharia de Software e Sistemas de Informação*. Rio de Janeiro: Brasport, 2005.

Sobre o autor

- Professor e Consultor de Tecnologia de Informação
- Doutorando (ITA) e Mestre (IPT) em Engenharia de Computação, Pós-Graduado em Análise de Sistemas (Mackenzie), Administração (Luzwell-SP), e Reengenharia (FGV-SP). Graduado/Licenciado em Matemática.
- Professor e Pesquisador da Universidade Anhembi Morumbi, UNIBAN, e ESAB (Ensino a Distância). Autor de 3 livros em Conectividade Empresarial. Prêmio em E-Learning no Ensino Superior (ABED/Blackboard).
- Consultor de T.I. em grandes empresas como Sebrae, Senac, Granero, Transvalor, etc. Viagens internacionais: EUA, França, Inglaterra, Itália, Portugal, Espanha, etc.

Atividades

Antes de dar continuidades aos seus estudos é fundamental que você acesse sua SALA DE AULA e faça a Atividade 3 no “link” ATIVIDADES.

Atividades

Atividade Dissertativa

Desenvolva uma pesquisa gerando um texto, de 2 a 3 folhas adicionando imagens, de uma das unidades da nossa apostila, de sua livre escolha, permitindo a expansão da temática selecionada.

Atenção: Qualquer bloco de texto igual ou existente na internet será devolvido para que o aluno realize a atividade novamente.