WEB E

WEB APP

WEBDESIGN

CMS JOOMLA!

http://www.juliocoutinho.com.br
@cout45

Webdesign

Design de Página

- Designer gráfico / Webdesigner
- Wireframe
- Criatividade
- A porta de entrada do "website"
- Credibilidade
- **TEMPLATE**

Design de Site

- Webdesigner
- Usabilidade
- Acessibilidade
- Persuasão
- Segurança
- **TEMPLATE**

Design de Conteúdo

- Webdesigner/Webwriter
- Alinhamento
- Correção ortográfica
- Fontes
- ENCONTRABILIDADE
  CMS

Design de Navegação WEB

- ISP
- Processo de busca pela informação

  DESENVOLVEDOR

http://www.juliocoutinho.com.br
@cout45

DESIGN DE NAVEGAÇÃO WEB

O processo de busca por informação

Uma abordagem holística para explicar a experiência do usuário na busca por informação. O processo de busca por informação (Information Search Process – ISP) é um modelo de busca por informação, com uma diferença: ele leva emoções em conta. Desenvolvido por Carol Kuhlthau, professora da Universidade Rutgers. O ISP tem seis estágios:

1) Iniciação

O usuário torna-se consciente de uma lacuna no conhecimento. Sentimentos de incertezas e apreensão são comuns; a principal tarefa é reconhecer a necessidade de informação.

2) Seleção

A incerteza frequentemente dá lugar a sentimentos de otimismo e uma prontidão para iniciar as buscas. A tarefa é identificar e selecionar o tópico a ser investigado. Os pensamentos são direcionados para frente e tentam prever um resultado.

3) Exploração

Sentimentos de incerteza, confusão e dúvida retornam. A inabilidade geral de expressar precisamente uma necessidade de informação comumente resulta em uma interação desconfortável com o sistema.

4) Formulação

Confiança aumentada e incerteza descrente marcam um ponto de virada no processo. Formar um foco torna-se uma tarefa chave á medida que os pensamentos tornam-se mais claros.

5) Coleção

A interação com o sistema de informação é mais efetiva e eficiente. Decisões sobre o escopo e o foco do tópico foram feitas e um senso de direção instala-se. A confiança continua a aumentar.

6) Apresentação

O objetivo agora é completar a busca e satisfazer a necessidade de informação. Uma sensação de alívio comum, bem como sensações de satisfação ou descontentamento (em caso de um resultado negativo). Os pensamentos centram-se em sintetizar e internalizar o que foi aprendido.

Kuhlthau também observou uma queda na confiança normalmente vista após um usuário

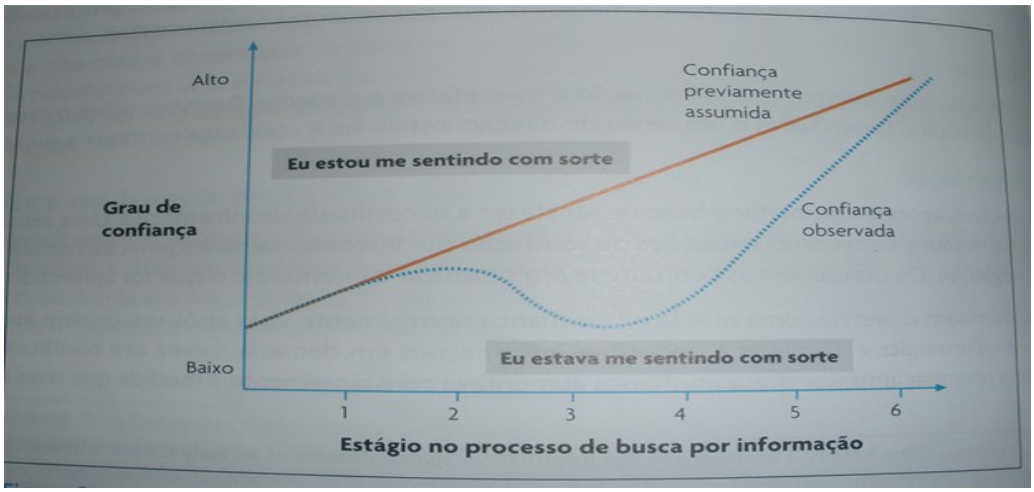
iniciar uma busca por informação e começar e encontrar informações em demasia, talvez até conflitantes. Isso contradiz a premissa anterior que a confiança aumentava constantemente à medida que mais informação era encontrada. Um usuário buscando informações pode, durante tal queda, experimentar incerteza, confusão e até ansiedade e até que um foco seja formado ou a busca seja encerrada.
A existência dessa queda sugere uma diferença entre o uso natural das informações por parte do usuário e o projeto do sistema de informação. Adquirir mais informação em estágio inicial (particularmente no estágio de exploração) aumenta, ao invés de diminuir, a incerteza. Em termos emocionais, a busca por informação é um empreendimento descontínuo com altos e baixos na confiança e na certeza.

Adaptando o ISP

Em uma tentativa de evitar tal queda, você pode usar o modelo teórico de Kuhlthau como o framework para o design de navegação, adaptando um ISP para refletir as ações, pensamentos e sentimentos para os visitantes de seu site. Os passos são:

1. Segmente os usuários e crie perfis. Um ISP aplica-se apenas a um grupo-alvo em particular.
2. Identifique os estágios de busca por informação e os objetivos dos usuários para cada um. As fases estabelecidas servirão como um ponto inicial, mas podem ser adaptadas.
3. Registre os sentimentos, pensamentos e ações típicos em cada estágio.
4. Mapeie os objetivos dos interessados no processo para cada estágio. O que sua organização está tentando atingir e como isso se encaixa com o processo natural de navegação dos usuários?
5. Derive para o site as características e os requisitos mapeados para cada fase no processo de busca.

Esses itens são mais bem resumidos em uma grande tabela. Nomeie as colunas de “Ações”, “Pensamentos”, “Sentimentos”, “Características” e “Objetivos de Negócios”. Faça com que as linhas sejam os estágios em seu ISP adaptado, então mapeie as ações típicas, pensamentos e sentimentos a possíveis características do site e aos objetivos de negócios.



(Modelo de Análise de ISP para um Portal de Empregos)

Fase	Ações	Pensamentos	Sentimentos	Características	Negócio
Iniciação: Reconhecer a necessidade de procurar um emprego	Identificar o problema e as estratégias de resolução	Vagos, sem certeza	Incerteza, apreensão	Campanha online e offline para aumentar o conhecimento sobre o site e melhorar a imagem da empresa	Aumentar o conhecimento sobre a empresa por parte das pessoas que buscam empregos
Seleção: Escolher os recursos apropriados	Localizar um ponto de início; Identificar os critérios do emprego	Gerais, orientados a tarefas abertos a novas idéias	Curiosidade, impaciência; ceticismo	Design de alta qualidade; ferramenta de auto-avaliação	Atrair pessoas altamente qualificadas que estejam buscando por um emprego
Busca: Localizar vagas relevantes	Informar a consulta ou navegar nas listagens de emprego	Positivos, pensando à frente para encontrar um emprego	Antecipação, otimismo	Busca, navegação com diferentes facetas; Filtros por grupos	Tornar as vagas de emprego publicamente disponíveis na web
Diferenciação: Priorizar os resultados da busca	Efetuar varredura nos resultados; Priorizar; refinar a busca	Mistos, sem clareza	Incerteza, confusão, se sentindo sobrecarregado	Número de itens próximos ao nome da categoria; Empregos relacionados: “carrinho de compras”	
Decisão: Determinar quais vagas são mais relevantes	Tomar uma decisão	Restritos/ focados, interesse e entendimento	Sentimentos de clareza, satisfação ou descontent	Habilidade de ordenar; fatos sobre a empresa, sua localização e outras	Obter confiança de potenciais candidatos

		aumentados	amento	informações	
Monitoramento: Verificar o status/disponibilidade no decorrer do tempo	Visitar o site novamente	Lembrança de detalhes	Esperança, sentimentos de apego	Página de “favoritos”; notícias; login e perfil; características “push”	Relacionamento com futuros empregados em potencial
Ação: Submeter a um emprego	Preencher formulários online ou offline; coletar os dados pessoais necessários	Claros, focados em completar as tarefas corretamente	Alívio, nervosismo	Aplicação online e offline; informações sobre a entrevista	Obter candidatos altamente qualificados

A tabela tenta descrever o processo de busca por informação para quem está procurando um emprego: neste caso, profissionais que já possuem um emprego e que estão procurando por uma nova posição. Usar a abordagem dessa tabela garante o fluxo da navegação e sua arquitetura básica é compatível com os padrões naturais de busca por informação dos visitantes do site.

ENGENHARIA DE SOFTWARE NA WEB – SISTEMAS & APLICAÇÕES BASEADAS NA WEB

A Engenharia de Software na Web, também utilizado a 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 peculiariedades 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 freqüentemente, 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.

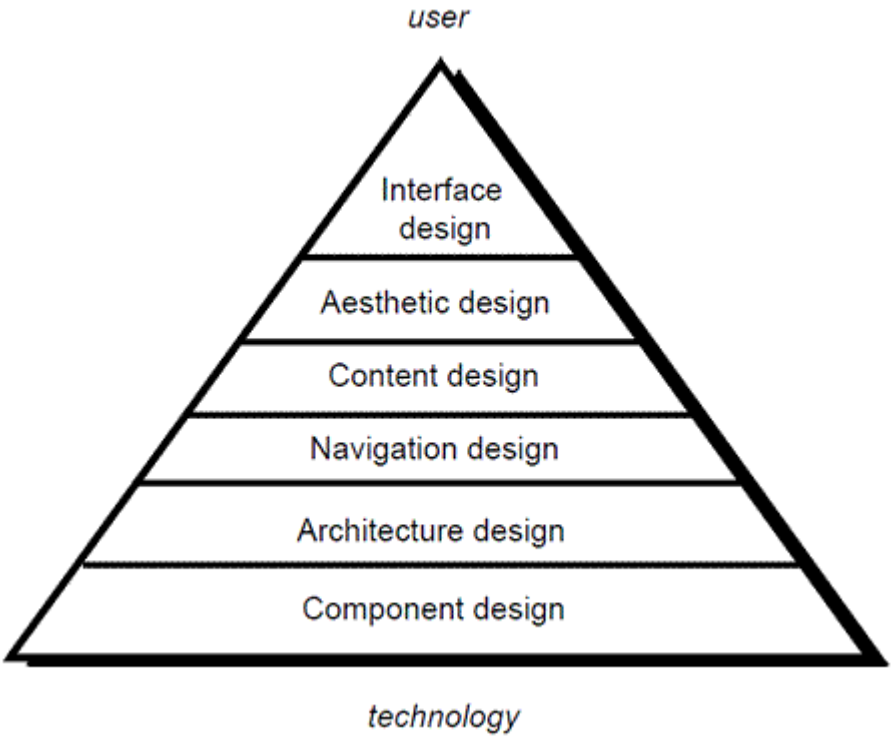
Quanto as ferramentas e tecnologias aplicadas na WebE englobam várias linguagens de modelagem (HTML, VRML,XML, etc.), recursos baseados em componentes (CORBA,COM, ActiveX, .NET, AJAX, etc.), navegadores, ferramentas multimídia, ferramentas de autoria de sites, ferramentas de conectividade de Banco de 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 WEB E

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:

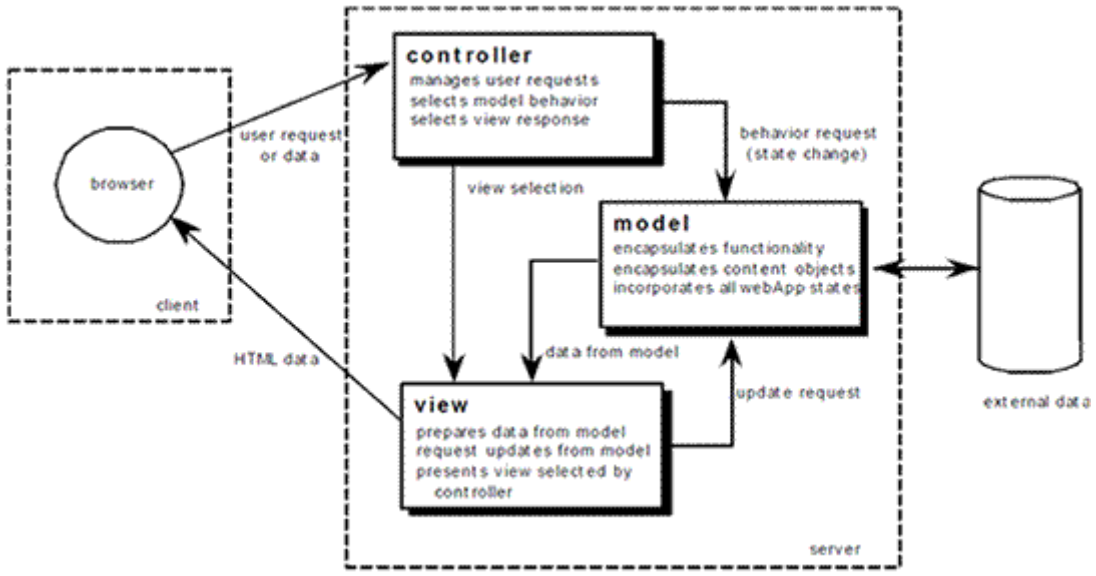
Nível da Pirâmide	Descrição
Projeto de Interface	Descreve a estrutura e organização da interface com o usuário.
Projeto Estético	Atenta para os esquemas de cor, leiaute, fonte, uso de gráficos, etc.
Projeto de Conteúdo	Define a estrutura e o esboço de todo o conteúdo, relacionando os objetos de conteúdo.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

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.

Técnicas de Análise de Requisitos

Existem 10 princípios básicos, e engraçados, sugeridos por Pressman, implementada 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.

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.

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

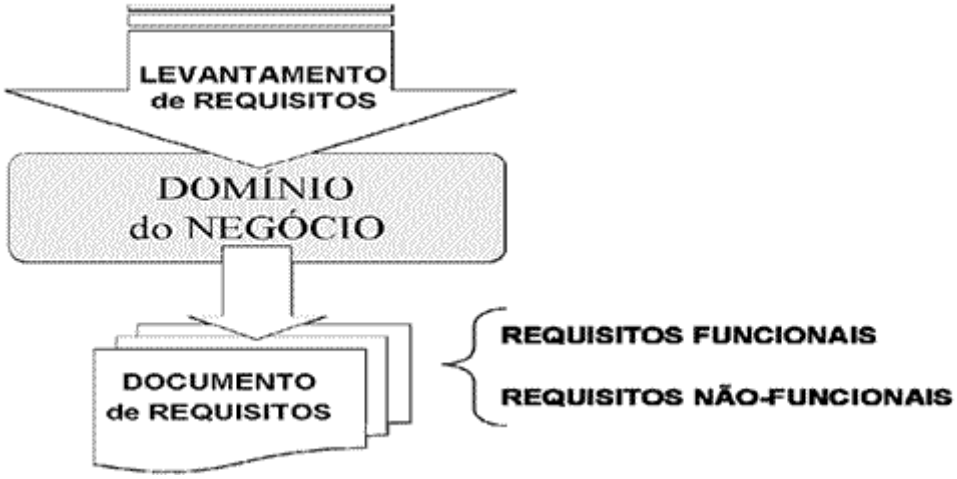
LEVANTAMENTO DE REQUISITOS

Pela definição de Maciaszek (2000), temos que requisito é uma condição ou capacidade que deve ser alcançada ou possuída por um sistema ou componente deste para satisfazer um contrato, padrão, especificação ou outros documentos formalmente impostos.

Normalmente os requisitos de um sistema são identificados a partir do domínio do negócio.

Denomina-se domínio de negócio a área de conhecimento específica na qual um determinado sistema de software será desenvolvido. Ou seja, domínio de negócio corresponde à parte do mundo real que é relevante ao desenvolvimento de um sistema. O domínio de negócio também é chamado de domínio do problema ou domínio da aplicação.

Veja a figura a seguir:



Durante o levantamento de requisitos, a equipe de desenvolvimento tenta entender o domínio do negócio que deve ser automatizado pelo sistema de software. O levantamento de requisitos compreende um estudo exploratório das necessidades dos usuários e da situação do sistema atual (se esse existir). Alguns autores aconselham ao analista de não perder tempo com o sistema atual, e partir diretamente para a concepção do novo. Este conselho se deve devido o analista não ficar “amarrado” aos conceitos da estrutura antiga, e poder ser mais inovador possível em sua nova proposta.

O produto final do levantamento de requisitos é o Documento de Requisitos, que declara os diversos tipos de requisitos do sistema. Normalmente esse documento é escrito em linguagem natural (notação informal). As principais seções de um documento de requisitos são:

REQUISITOS FUNCIONAIS

Definem as funcionalidades do sistema. Veremos mais adiante nesta apostila que tipicamente os Requisitos Funcionais serão alvo na UML, dos modelos de Caso de Uso.

Alguns exemplos práticos de requisitos funcionais são:

“o sistema deve permitir que cada professor realize o lançamento de notas das turmas nas quais lecionou”
“o sistema deve permitir que o aluno realize a sua matrícula nas disciplinas oferecidas em um semestre”
“os coordenadores de escola devem poder obter o número de aprovações, reprovações e trancamentos em todas as turmas em um determinado período”

REQUISITOS NÃO-FUNCIONAIS

Declaram as características de qualidade que o sistema deve possuir e que estão relacionadas às suas funcionalidades. Para você visualizar melhor esse conceito, alguns tipos de requisitos não-funcionais são relacionados a seguir:

CONFIABILIDADE
Corresponde a medidas quantitativas da confiabilidade do sistema, tais como tempo médio entre falhas, recuperação de falhas ou quantidade de erros por milhares de linhas de código fonte.

DESEMPENHO
Requisitos que definem tempos de respostas esperados para as funcionalidades do sistema.

PORTABILIDADE
restrições sobre as plataformas de hardware e de software nas quais o sistema será implantado e sobre o grau de facilidade para transportar o sistema para outras plataformas.

SEGURANÇA
Limitações sobre a segurança do sistema em relação a acessos não-autorizados.

USABILIDADE
Requisitos que se relacionam ou afetam a usabilidade do sistema. Exemplos incluem requisitos sobre a facilidade de uso e a necessidade ou não de treinamento dos usuários. Uma das formas de se medir a qualidade de um sistema de software é através de sua utilidade. E um sistema será útil para seus usuários se atender aos requisitos definidos. O enfoque prioritário do levantamento de requisitos é responder claramente a questão: “O que o usuário necessita do novo sistema ?”. Requisitos definem o problema a ser resolvido pelo sistema de software; eles não descrevem o software que resolve o problema.

Lembre-se sempre: novos sistemas serão avaliados pelo seu grau de conformidade aos requisitos, não quão complexa a solução tecnológica tenha sido aplicada.

O levantamento de requisitos é a etapa mais importante em termo de retorno de investimento feitos para o projeto de desenvolvimento. Muitos sistemas foram abandonados ou nem chegaram a serem usados porque os membros da equipe não dispensaram tempo suficiente para compreender as necessidades do cliente. Em um estudo baseado em 6.700 sistemas feitos em 1997, Carper Jones mostrou que os custos resultantes da má realização desta etapa de entendimento podem ser 200 vezes maiores que o realmente necessário.

O Documento de Requisitos serve como um termo de consenso entre a equipe técnica de desenvolvedores e o cliente. Esse documento constitui a base para as atividades subsequentes do desenvolvimento do sistema e fornece um ponto de referência para qualquer validação futura do software construído. O envolvimento do cliente desde o início do processo de desenvolvimento ajuda a assegurar que o produto desenvolvido realmente atenda às necessidades identificadas.

Além disso, o Documento de Requisitos estabelece o escopo do sistema, isto é, o que faz parte e o que não faz parte do sistema. O escopo de um sistema muitas vezes muda durante o seu desenvolvimento. Desta forma, se o escopo muda, tanto clientes quando desenvolvedores têm um parâmetro para decidirem o quanto de recursos de tempo e financeiros devem mudar.

Um outro ponto importante sobre requisitos é a sua característica de volatilidade. Um requisito volátil é aquele que pode sofrer modificações durante o desenvolvimento do sistema.

A menos que o sistema a ser desenvolvido seja bastante simples e estático, características cada vez mais raras nos sistemas atuais, é humanamente impossível pensar em todos os detalhes a princípio. Além disso, quando o sistema entrar em produção e os usuários começarem a utilizá-lo, eles próprios descobrirão requisitos nos quais nem sequer tinham pensado anteriormente.

Em resumo, os requisitos de um sistema complexo inevitavelmente mudarão durante todo o seu desenvolvimento. No desenvolvimento de sistemas de software, a existência de requisitos voláteis corresponde mais a regra do que à exceção.

· Porcentagem de projetos que terminam dentro do prazo estimado: 10%
· Porcentagem de projetos que são descontinuados antes de chegarem ao fim: 25%
· Porcentagem de projetos acima do custo esperado: 60%
· Atraso médio nos projetos: 1 ano

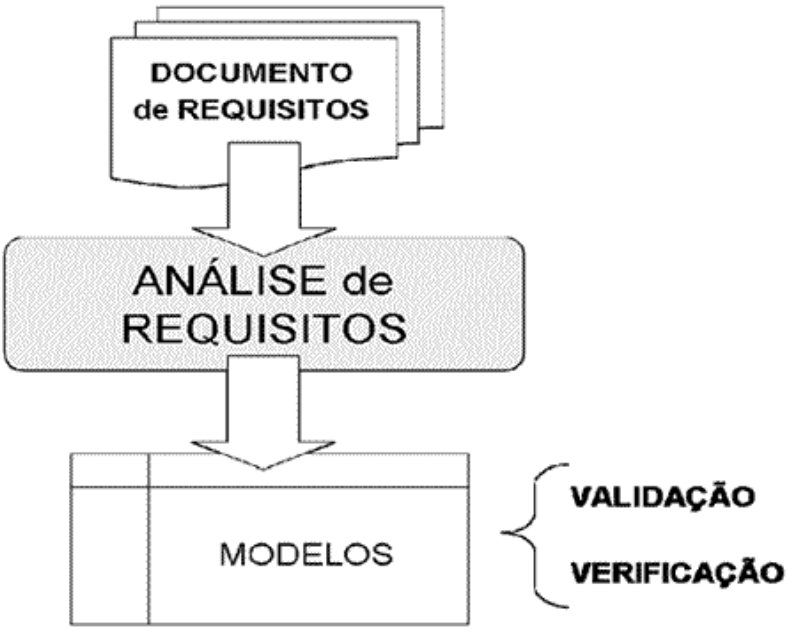
E há vários processos de desenvolvimento propostos, no entanto não existe o melhor processo de desenvolvimento. Cada processo tem suas particularidades em relação ao modo de arranjar e encadear as atividades de desenvolvimento. Veremos a seguir as mais comuns e tradicionais atividades de um processo de desenvolvimento.

ANÁLISE DE REQUISITOS

Formalmente, o termo análise corresponde a quebrar um sistema em seus componentes, e estudar como tais componentes interagem com o objetivo de entender como esse mesmo sistema funciona.

No contexto dos sistemas de software, esta é a etapa na qual os analistas realizam um estudo detalhado no Documento de Requisitos levantado na atividade anterior.

A partir desse estudo, são construídos modelos para representar o sistema a ser construído (veja mais detalhes na figura a seguir). A Análise de Requisitos é também chamada por alguns autores como Especificação de Requisitos.



Assim como no levantamento de requisitos, a Análise de Requisitos não leva em conta o ambiente tecnológico a ser utilizado. Nessa atividade, o foco de interesse é tentar construir uma estratégia de solução, sem se preocupar com a maneira como essa estratégia será realizada.

A razão dessa prática é tentar obter a melhor solução para o problema sem se preocupar com os detalhes da tecnologia a ser utilizada. É necessário saber o que o sistema proposto precisa fazer para então, definir como esse sistema irá fazê-lo.

Ocorre, frequentemente na prática, equipes de desenvolvimento que passam para a construção da solução sem antes terem definido completamente o problema. Portanto, os

modelos construídos nesta fase devem ser cuidadosamente validados e verificados, através da validação e verificação dos modelos respectivamente.

O objetivo da validação é assegurar que as necessidades do cliente estão sendo atendidas pelo sistema: “será que o software correto está sendo construído ?”.
Já a verificação tem o objetivo de verificar se os modelos construídos estão em conformidade com os requisitos definidos: “será que o software está sendo construído corretamente ?”.

Na verificação dos modelos, são analisadas a exatidão de cada modelo em separado e a consistência entre os modelos. Em um processo de desenvolvimento orientado a objetos, o resultado da análise são modelos que representam as estruturas das classes de objetos que serão componentes do sistema. Além disso, a análise também resulta em modelos que especificam as funcionalidades do sistema de software.

PROTOTIPAGEM

A construção de protótipos é uma técnica que serve de complemento à análise de requisitos.

No contexto do desenvolvimento de software, um protótipo é um esboço de alguma parte do sistema.

Protótipos são construídos para telas de entrada, telas de saída, subsistemas, ou mesmo para os sistema como um todo. A construção de protótipos utiliza as denominadas linguagens de programação visual. Exemplos são o Delphi, o PowerBuilder e o Visual Basic que, na verdade, são ambientes com facilidades para a construção da interface gráfica (telas, formulários, etc.). Além disso, muitos Sistemas de Gerência de Bancos de Dados também fornecem ferramentas para a construção de telas de entrada e saída de dados.

Na prototipagem, após o levantamento de requisitos, um protótipo do sistema é construído para ser usado na validação. O protótipo é revisto por um ou mais usuários, que fazem suas avaliações e críticas acerca das características apresentadas. O protótipo é então corrigido ou refinado de acordo com as intervenções dos usuários.

Esse processo de revisão e refinamento continua até que o protótipo seja aceito pelos usuários. Portanto, a técnica de prototipagem é muito útil e tem o objetivo de assegurar que os requisitos do sistema foram realmente bem entendidos.

O resultado da validação através do protótipo pode ser usado para refinar os modelos do sistema. Após a aceitação, o protótipo (ou parte dele) pode ser descartado ou utilizado como uma versão inicial do sistema. Embora a técnica de prototipagem seja opcional, ela é freqüentemente aplicada em projetos de desenvolvimento de software, especialmente quando há dificuldades no entendimento dos requisitos do sistema, ou há requisitos arriscados que precisam ser mais bem entendidos.

A idéia é que um protótipo é mais concreto para fins de validação do que modelos representados por diagramas bidimensionais (tipo DFD, ou mesmo o UML). Isso incentiva a participação ativa do usuário na validação. Consequentemente, a tarefa de validação se torna menos suscetível a erros. No entanto, destacamos que alguns desenvolvedores usam essa técnica como um substituto à construção de modelos de sistema.

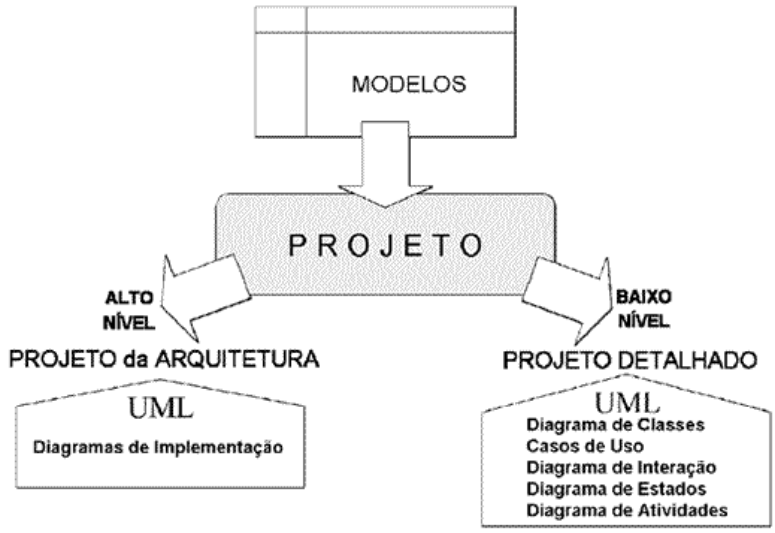
Tenha em mente que a prototipagem é uma técnica complementar à construção dos modelos do sistema. Os modelos do sistema devem ser construídos, pois são eles que guiam as demais fases do projeto de desenvolvimento de software. Idealmente, os erros detectados na validação do protótipo devem ser utilizados para modificar e refinar os modelos do sistema. Portanto, devemos utilizar a prototipagem como complemento ao Modelo de Ciclo de Vida Iterativo e Incremental, e não para substituí-la.

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- **Projeto**
- **Implementação**
- **Testes**
- **Implantação**

O foco principal da análise são os aspectos lógicos e independentes de implementação de um sistema, os requisitos. Na fase de Projeto, determina-se “como” o sistema funcionará para atender aos requisitos, de acordo com os recursos tecnológicos existentes – a fase de projeto considera os aspectos físicos e dependentes de implementação.

Aos modelos construídos na fase de análise são adicionadas as determinadas “restrições de tecnologia”. Exemplos de aspectos a serem considerados na fase de projeto: arquitetura do sistema, padrão de interface gráfica, a linguagem de programação, o Gerenciador de Banco de Dados, etc.



Esta fase produz uma descrição computacional do que o software deve fazer, e deve ser coerente com a descrição feita na análise. Em alguns casos, algumas restrições da tecnologia a ser utilizada já foram amarradas no Levantamento de Requisitos. Em outros casos, essas restrições devem ser especificadas. Mas, em todos os casos, a fase de projeto do sistema é direcionada pelos modelos construídos na fase de análise e pelo planejamento do sistema.

O projeto consiste de duas atividades principais: Projeto da Arquitetura, também conhecido como projeto de alto nível, e Projeto Detalhado denominado também como projeto de baixo nível.
Em um processo de desenvolvimento orientado a objetos, o Projeto da Arquitetura consiste em distribuir as classes de objetos relacionados do sistema em subsistemas e seus componentes. Consiste também em distribuir esses componentes fisicamente pelos recursos de hardware disponíveis. Os diagramas da UML normalmente utilizados nesta fase do projeto são os Diagramas de Implementação.

No Projeto Detalhado, são modeladas as colaborações entre os objetos de cada módulo com o objetivo de realizar as funcionalidades do módulo. Também são realizados o projeto da interface com o usuário e o projeto de Banco de Dados. Os principais diagramas da UML utilizados nesta fase do projeto são: Diagrama de Classe, Diagrama de Casos de Uso, Diagrama de Interação, Diagrama de Estados e Diagrama de Atividades. Embora a Análise e o Projeto sejam descritos didaticamente em seções separadas, é importante notar que na prática não há uma distinção tão clara entre essas duas fases. Principalmente no desenvolvimento de sistemas orientados a objetos, as atividades dessas duas fases frequentemente se misturam.

IMPLEMENTAÇÃO
Na Implementação, o sistema é codificado, ou seja, ocorre a tradução da descrição computacional da fase de projeto em código executável através do uso de uma ou mais linguagens de programação.

TESTES
Diversas atividades de teste são realizadas para verificação do sistema construído, levando-se em conta a especificação feita na fase de Projeto. O principal produto dessa fase é o Relatório de Testes, contendo informações sobre erros detectados no software. Após a atividade de testes, os diversos módulos do sistema são integrados, resultando finalmente no produto de software.

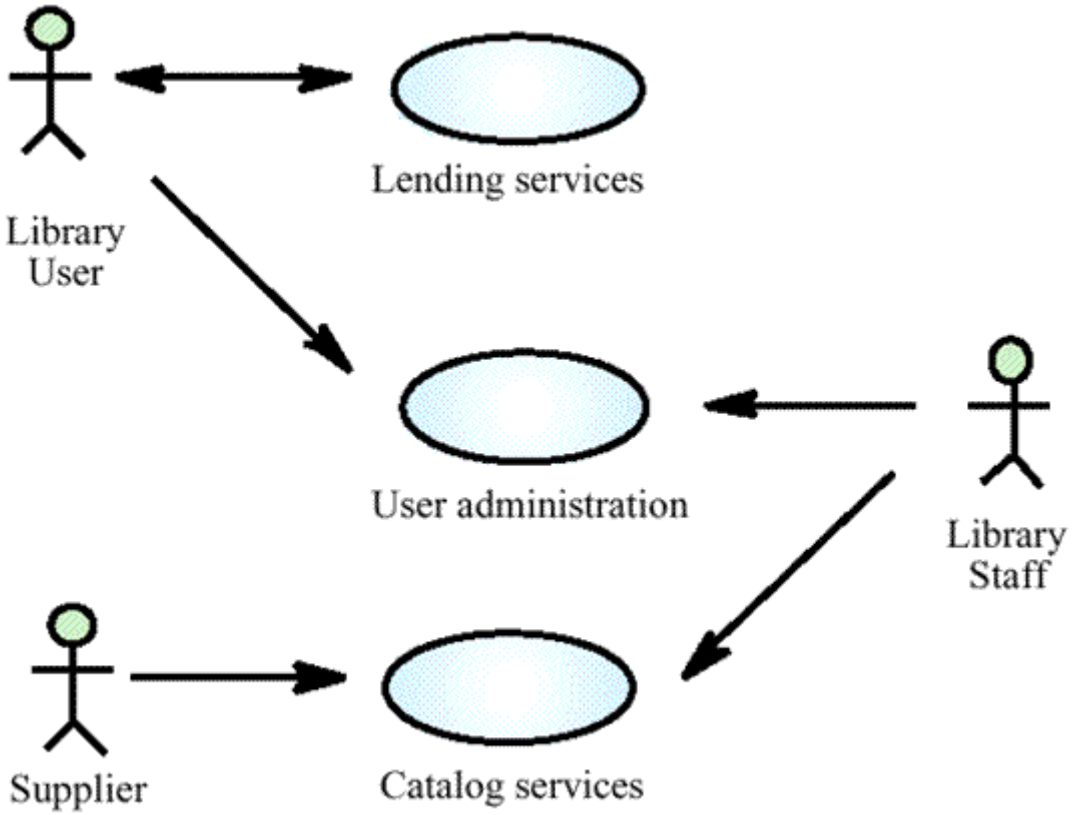
IMPLANTAÇÃO
O sistema é empacotado, distribuído e instalado no ambiente do usuário. Os manuais do sistema são escritos, os arquivos são carregados, os dados são importados para o sistema e os usuários são treinados para utilizar o sistema corretamente. Em alguns casos, principalmente em sistemas legados, aqui também ocorre a migração de sistemas de software e de dados preexistentes.

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).

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.

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.



METODOLOGIAS DE DESENVOLVIMENTO ÁGIL DE SOFTWARE: XP – FDD & DSM

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".

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 constante 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.

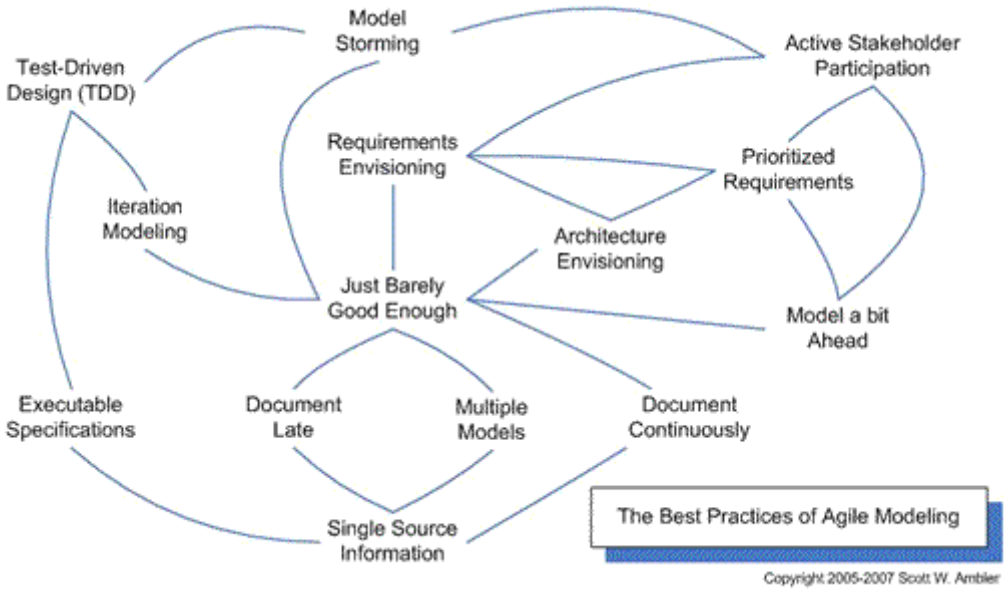
No gráfico acima vemos num extremo o RUP enfatizando os controles, e uma política de trabalho rígida. Ele é mais interessante de ser utilizado com equipes grandes de desenvolvimento. Na outra ponta temos o XP, que veremos a seguir, sinalizando maior liberdade e mais adequada para equipes pequenas. E 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".

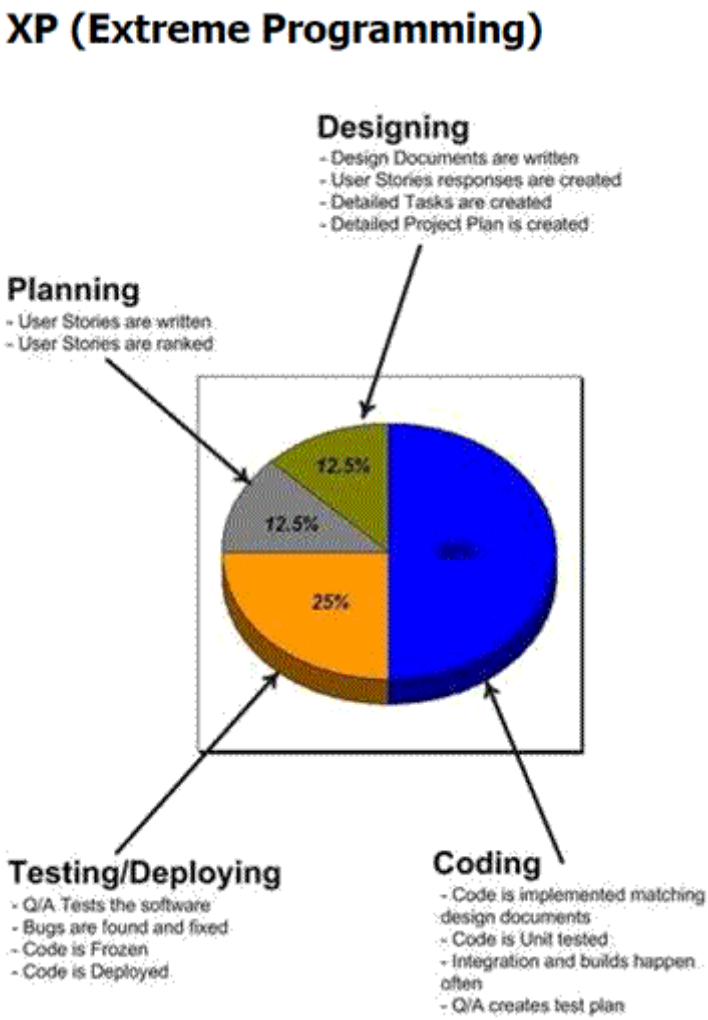
1. Critical Points

1. The fundamental issue is communication, not documentation.
2. Agilists write documentation if that's the best way to achieve the relevant goals, but there often proves to be better ways to achieve those goals than writing static documentation.
3. Document stable things, not speculative things.
4. Take an evolutionary approach to documentation development, seeking and then acting on feedback on a regular basis.
5. Prefer executable work products such as customer tests and developer tests over static work products such as plain old documentation (POD).
6. You should understand the total cost of ownership (TCO) for a document, and someone must explicitly choose to make that investment.
7. Well-written documentation supports organizational memory effectively, but is a poor way to communicate during a project.
8. Documentation should be concise: overviews/roadmaps are generally preferred over detailed documentation.

9. With high quality source code and a test suite to back it up you need a lot less system documentation.
10. Travel as light as you possibly can.
11. Documentation should be just barely good enough.
12. Comprehensive documentation does not ensure project success, in fact, it increases your chance of failure.
13. Models are not necessarily documents, and documents are not necessarily models.
14. Documentation is as much a part of the system as the source code.
15. Your team's primary goal is to develop software, its secondary goal is to enable your next effort.
16. The benefit of having documentation must be greater than the cost of creating and maintaining it.
17. Developers rarely trust the documentation, particularly detailed documentation because it's usually out of sync with the code.
18. Each system has its own unique documentation needs, one size does not fit all.
19. Ask whether you NEED the documentation, not whether you want it.
20. The investment in system documentation is a business decision, not a technical one.
21. Create documentation only when you need it at the appropriate point in the life cycle.
22. Update documentation only when it hurts.



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 acima):
· Planejamento
· Projeto
· Codificação
· Teste

Existe um grande ênfase ao trabalho em duplas, aonde 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:

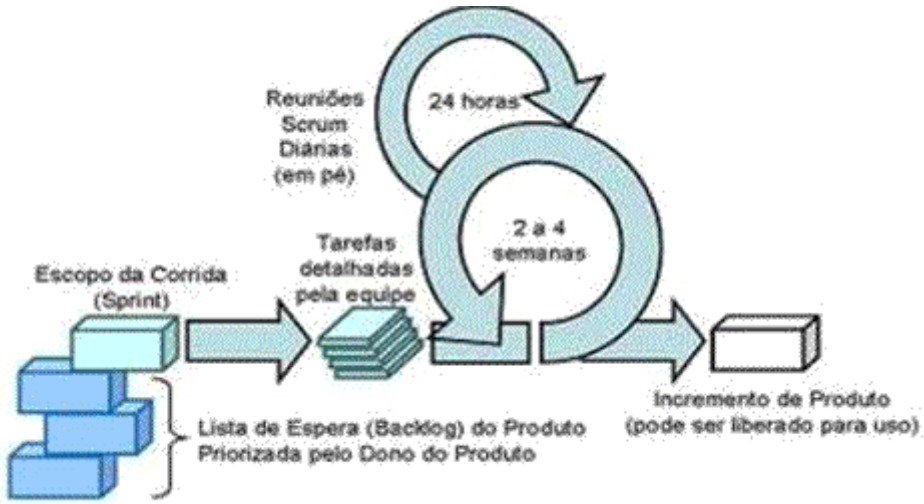
• Sendo as "características" pequenos blocos de funcionalidade, os usuários e desenvolvedores têm melhor controle e entendimento de todo o processo.
• Organizam-se as "características" em um agrupamento hierárquico relacionado ao negócio, melhorando a visão para o usuário. E para os desenvolvedores facilitando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

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 ?



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.

"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 a longo prazo e livre-se do resto.

MODELO ITERATIVO & INCREMENTAL

O Modelo de ciclo de vida Iterativo e Incremental foi proposto justamente para ser a resposta aos problemas encontrados no Modelo em Cascata. Um processo de desenvolvimento segundo essa abordagem divide o desenvolvimento de um produto de software em ciclos.

Em cada ciclo de desenvolvimento, podem ser identificadas as fases de análise, projeto, implementação e testes. Essa característica contrasta com a abordagem clássica, na qual as fases de análise, projeto, implementação e testes são realizadas uma única vez.
No Modelo de ciclo de vida Iterativo e Incremental, um sistema de software é desenvolvido em vários passos similares (iterativo). Em cada passo, o sistema é estendido com mais funcionalidades (incremental).

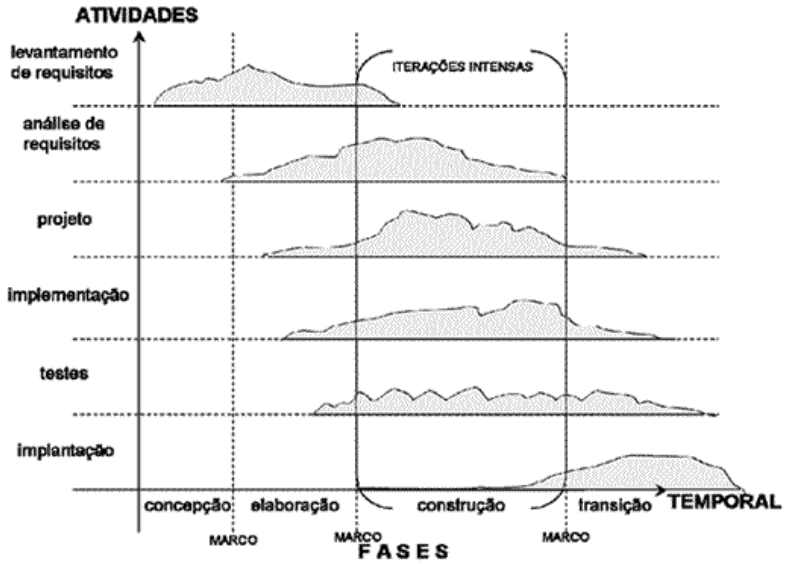
A abordagem incremental incentiva a participação do usuário nas atividades de desenvolvimento do sistema, o que diminui em muito a probabilidade de interpretações erradas em relação aos requisitos levantados.

Uma outra vantagem dessa abordagem é que os riscos do projeto podem ser mais bem gerenciados. Um risco de desenvolvimento é a possibilidade de ocorrência de algum evento que cause prejuízo ao processo de desenvolvimento, juntamente com as consequências desse prejuízo. Os requisitos a serem considerados primeiramente devem ser selecionados com base nos riscos que eles fornecem. Os requisitos mais arriscados devem ser considerados tão logo possível.

Para entender o motivo de que o conjunto de requisitos deve ser considerado o mais cedo possível, vamos lembrar de uma famosa frase do consultor Tom Gilb (1988): “Se você não atacar os riscos ativamente, então estes irão ativamente atacar você”. Ou seja, quanto mais cedo a equipe de desenvolvimento considerar os requisitos mais arriscados, menor é a probabilidade de ocorrerem prejuízos devido a esses requisitos.

Estrutura Geral de um Desenvolvimento Incremental e Iterativo

Basicamente o ciclo de vida de processo Incremental e Iterativo pode ser estudado segundo duas dimensões: a temporal e a de atividades. Veja mais atentamente a figura abaixo:



A dimensão temporal, na horizontal (na parte mais inferior do gráfico), os processos estão estruturados em FASES (concepção, elaboração, construção e transição). Em cada uma dessas fases, há uma ou mais iterações. Cada iteração tem uma duração preestabelecida (de duas a seis semanas). Ao final de cada iteração, é produzido um incremento, ou seja, uma parte do sistema final. Um incremento pode ser liberado para os usuários, ou pode ser somente um incremento interno.

A dimensão de atividades (ou de fluxos de trabalho) são apresentadas verticalmente na figura anterior. Essa dimensão compreende as atividades realizadas durante a iteração de uma dessas fases: levantamento de requisitos, análise de requisitos, projeto, implementação, testes e implantação (as mesmas atividades que veremos mais detalhadamente adiante).

Em cada uma dessas fases, diferentes artefatos de software são produzidos, ou artefatos começados em uma fase anterior são estendidos com novos detalhes. Cada fase é concluída com um MARCO. Um marco é um ponto do desenvolvimento no qual decisões sobre o projeto são tomadas e importantes objetivos são alcançados. Os marcos são úteis para o gerente de projeto estimar os gastos e o andamento do cronograma de desenvolvimento.

Como vimos anteriormente as fases que são delimitadas pelos marcos são: concepção, elaboração, construção e transição. Vejamos agora com mais detalhes cada uma dessas fases:

CONCEPÇÃO
Nessa fase a idéia geral, e o escopo do desenvolvimento, são desenvolvidos. Um planejamento de alto nível do desenvolvimento é realizado. São determinados os marcos que separam as fases.
ELABORAÇÃO
É alcançado um entendimento inicial sobre como o sistema será construído. O planejamento do projeto de desenvolvimento é completado. Nessa fase, o domínio do negócio é analisado.
Os requisitos do sistema são ordenados considerando-se prioridade e risco. Também são planejadas as iterações da próxima fase, a de construção. Isso envolve definir a duração de cada iteração e o que será desenvolvido em cada iteração.

CONSTRUÇÃO
As atividades de análise e projeto aumentam em comparação às demais. Esta é a fase na qual ocorrem mais iterações incrementais. No final dessa fase, é decidido se o produto de software pode ser entregue aos usuários sem que o projeto seja exposto a altos riscos. Se este for o caso, tem início a construção do manual do usuário e da descrição dos incrementos realizados no sistema.

TRANSIÇÃO
Os usuários são treinados para utilizar o sistema. Questões de instalação e configuração do sistema também são tratadas. Ao final desta fase, a aceitação do usuário e os gastos são avaliados. Uma vez que o sistema é entregue aos usuários, provavelmente surgem novas questões que demandam a construção de novas versões do mesmo. Neste caso, um novo ciclo de desenvolvimento é iniciado.

Em cada iteração, uma proporção maior ou menor de cada dessas atividades é realizada, dependendo da fase em que se encontra o desenvolvimento. Na figura, com o intuito de mostrar um modelo amplo da estrutura geral de um Desenvolvimento Incremental e Iterativo, permite-se perceber, por exemplo, que na fase de transição, a atividade de implantação é a predominante. Por outro lado, na fase de construção, as atividades de análise, projeto e implementação são as de destaque. Normalmente, a fase de construção é a que possui mais interações. No entanto, as demais fases também podem conter iterações, dependendo da complexidade do sistema.