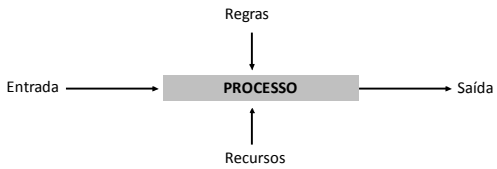
Processos de Desenvolvimento de Software

Marcos Monteiro

Processo

- Mensurável
- Resultados específicos
- Cliente
- Responder a um evento especifico

Processo



Matriz de RACI

- Responsável
- Accountable (quem é cobrado)
- Consultado
- Informado

	Diretor	Gerente	Supervisor	Operário
Processo1	I	AC		R
Processo2	I	A	R	

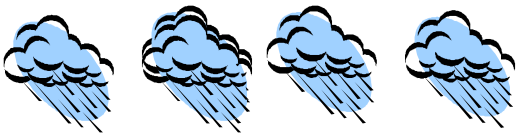
Extreme Programming (XP)

9

Utopia do Software?

Desenvolver o produto que atenda as **necessidades** do cliente e seja entregue no **prazo**, com o **custo** e o nível de qualidade desejado.

10



- Não há nada mais trabalhoso do que fazer software!
- Essa frase vem sendo usada na indústria desde "crise do software" em 1968.

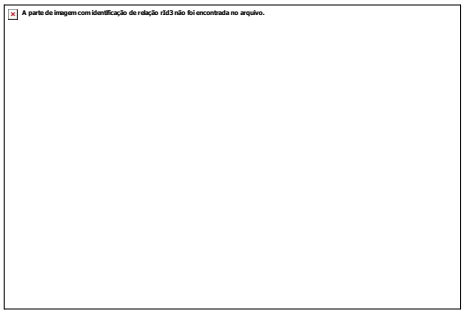
11

Principais Problemas

- Atrasos no cronograma.
- Falta de planejamento.
- Inúmeros bugs.
- Incompreensão dos requisitos de negócio e sistema.
- Mudanças constantes nos requisitos.

12

Uso das funcionalidades?



13

O surgimento do XP

- Em meados de 1990, ***Kent Beck*** procurou formas mais simples e eficientes de desenvolver software
  - Identificou o que tornava simples e o que dificultava o desenvolvimento de software
- Em Março de 1996, ele iniciou um projeto com novos conceitos que resultaram na metodologia XP - *eXtreme Programming*

14

Motivações do XP

- Tornar o desenvolvimento do software melhor
- Produzir software de qualidade em um ritmo sustentável
- Trazer transparência, responsabilidade e capacidade de justificativa (accountability)
- Abraçar as mudanças
- Reconciliar humanidade e produtividade

15

- Interação entre pessoas MAIS QUE processos e ferramentas;
- Software em funcionamento MAIS QUE documentação abrangente;
- Colaboração com o cliente MAIS QUE negociação de contratos;
- Responder as mudanças MAIS QUE seguir um plano.

16

Equipe X Clientes X usuários do software

17

Mudanças em um projeto de software são boas ou ruins?

18

Valores, Princípios e Práticas

- Valores: são nossas crenças fundamentais, aquilo em que acreditamos no desenvolvimento de software.
- Práticas: são atitudes, atividades concretas que fazemos no dia-a-dia. Garantem a aplicação dos valores.
- Princípios: servem como guias, específicos do contexto de desenvolvimento, que ajudam a adaptar as práticas à realidade particular de cada equipe ou projeto.

19

Valores de XP

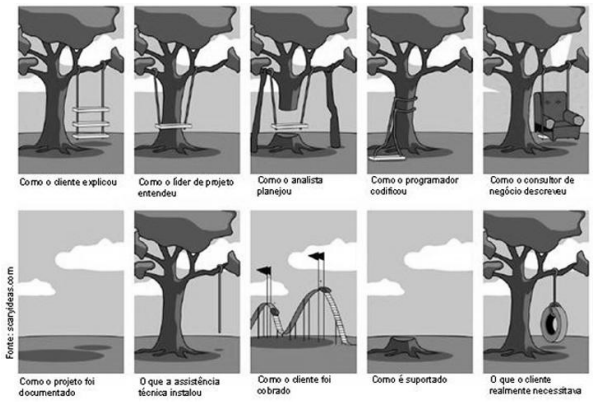
- Comunicação
- Simplicidade
- Feedback
- Coragem
- Respeito

20

Comunicação

- Falhas na comunicação é uma das principais causas de atrasos e de fracassos em projetos de software.
- A pergunta: o que é a coisa certa a fazer?
- O XP prioriza o dialogo presencial com o objetivo de garantir que todas as partes envolvidas em um projeto tenham a chance de se compreender da melhor maneira possível.

21



31

Telefone sem fio

32

Feedback

- Equipes XP acreditam que projetos de software são iniciativas freqüentemente caras, arriscadas e com um histórico repleto de falhas
- Isso pode ser tratado de forma mais econômica e eficaz se as falhas forem detectadas rapidamente.
- Em projetos XP procura-se entregar novas funcionalidades no menor prazo possível, para que o cliente compreenda rapidamente as conseqüências daquilo que pediu.

33

Coragem

- A única constante em um projeto de software é a mudança
- Os clientes mudam porque aprendem durante o projeto e descobrem problemas mais prioritários a serem solucionados ou formas mais apropriadas de resolvê-los.
- São uma grande oportunidade de descobrirmos o que é a coisa certa a se fazer

34

Coragem

- XP não tem uma solução mágica para eliminar esse risco. Ele existe em um projeto XP, como existe em qualquer outro.
- Equipes XP consideram as mudanças inevitáveis!
- Em projetos com XP procuramos nos adaptar a elas com confiança em seus mecanismos de proteção, tais como desenvolvimento orientado a testes, programação em par e integração contínua.

35

Respeito

36

Respeito

- Respeito é um valor que dá sustentação a todos os demais pilares de um projeto.
- Respeito é o mais básico de todos os valores. Se ele não existir em um projeto, não há nada que possa salvá-lo.
- Saber ouvir, saber compreender e respeitar o ponto de vista do outro é essencial para que um projeto de software seja bem sucedido.

37

Princípios da XP

Auto-semelhança	Benefício Mútuo
Diversidade	Economia
Falha	Fluidez
Humanismo	Melhoria
Oportunidade	Passos de Bebê
Qualidade	Redundância
Reflexão	Responsabilidade Aceita

38

Economia

- Software é um investimento.
- Desenvolver consome dinheiro e tempo.
- Deve gerar retorno para os negócios.
- O XP antecipa as receitas e adia as despesas.
- Implementar o mais cedo possível as funcionalidades que gerem retorno financeiro.
- Em XP os investimentos na arquitetura do software são os mínimos possíveis.

39

Qualidade

XP gera valor rapidamente e evita desperdícios.

- Perdas para o negócio
  - Insatisfação do cliente
- Conflito entre cliente e desenvolvedores
  - Desconfiança
  - Ansiedade
  - Relacionamentos desgastados
  - Perda de tempo corrigindo defeitos
- Dificuldade para adaptar o software a novas necessidades de maneira segura

40

Humanismo

- Pessoas desenvolvem software.
- É importante compreender a natureza humana e potencializar o que ele tem de melhor.
- Programar é uma atividade complexa.
- Programadores gostam de programar.
- O XP coloca as pessoas no centro.

41

Diversidade

- Desenvolver software é uma atividade complexa.
- Indivíduos iguais ficam limitados a pontos de vistas semelhantes.
- Explorar idéias diferentes e inovadoras.
- Opiniões diferentes, soluções ricas.

42

Práticas da XP

Inicialmente o XP definia um conjunto de 12 práticas. Estas práticas continuam válidas até hoje, mas foram remodeladas nos últimos anos.

Todas as práticas do XP focam que o maior valor possível seja gerado para o cliente.

43

12 Práticas originais

- Jogo do Planejamento
- Releases Curtos
- Cliente Presente
- Desenvolvimento orientado a testes
- Design Simples
- Refatoração
- Programação em Pares
- Metáfora
- Stand up Meeting
- Padrões de Codificação
- Código Coletivo
- Integração Contínua
- Ritmo Sustentável

Nível do Cliente
Nível da Equipe
Nível do Par

44

Jogo de planejamento

- Pessoal de Negócio
  - ...toma as decisões de negócio
    - Definir as datas
    - Escrever as histórias (escopo)
    - Priorizar o que deve ser feito
- Pessoal Técnico
  - ...toma as decisões técnicas
    - Estimar as histórias
    - Informar os riscos técnicos
    - Informar sua velocidade

45

Histórias

- Pequenos **pedaços de funcionalidades** que produzem valor para os usuários do sistema.
- Por simplicidade, são **escritas em cartões**.
- **Escritas pelo cliente** ou por algum tipo de ***user proxy*** *(gerentes, equipes de marketing, especialistas do domínio, analistas de negócio, etc).*
- Promessas de **conversa** futura, não especificações detalhadas

46

Histórias

Um usuário pode reservar um quarto com um cartão de crédito.

Nota: Serão aceitos VISA, MasterCard e American Express. Verificar Diner's.

47

. Cliente Presente

- Visão diferente do modelo tradicional.
- Cliente presente no dia-a-dia do projeto.
- Dialogar para entender.
- Feedback necessário.

Releases Curtos

- Pequenos investimento efetuados de forma gradual.
- Seleciona histórias de maior valor para o cliente.
- O release deve ter sentido isoladamente.
- Promove desenvolvimento iterativo e incremental.
- Promove retorno rápido para o cliente em curto prazo de tempo.
- Maior valor de negócio possível entregue ao final de cada release.

49

Desenvolvimento orientado a testes

- Técnica preventiva utilizada durante todo o projeto.
- Todo código implementado deve coexistir de um teste automatizado.
- Teste deve ser codificado antes de implementar a cada história.
- Teste de Unidade:
  - Verifica os resultados de cada classe separadamente.
- Teste de aceitação:
  - Verifica a implementação de uma história.

50

. Design Simples

- No desenvolvimento tradicional, algumas funcionalidades que não terão utilidade para o cliente são implementadas.
- O XP não se preocupa com necessidades futuras.
- Prega um design do sistema simples para que atenda as necessidades do cliente.
- Implementar apenas funcionalidades pedidas pelo cliente.

51

Refatoração

- Desenvolvedor pode alterar o código de outros, deixando-os mais legíveis e simples.
- Anda de mãos dadas com o código coletivo e com os padrões de codificação.
- Apoiada pelos testes automatizados.
- Reestruturação do código não pode afetar as funcionalidades do sistema.
- Alterar *como* ele faz, mas não *o que* ele faz.
- Aprimorar a estrutura interna.

52

Metáfora

- Criar comparações e analogias com o assunto em questão para um melhor entendimento do assunto.
- Utilizado com intensidade no projeto.
- Facilita comunicação e fixação dos assuntos entre as partes.
- Vista como aspecto crucial na disseminação e compreensão de novas idéias.
- Exige criatividade e desenvolvimento de idéias.

53

Uma metáfora: Aprendendo a dirigir

- Dirigir um carro, mesmo em uma estrada reta, implica em pequenos ajustes ao longo do tempo
- Trânsitos, imprevistos, mudanças de prioridades podem mudar seu caminho

54

Stand Up Meeting

- Cada dia de trabalho "começa" com uma reunião, onde todos conversam sobre os problemas encontrados no dia anterior e definem as prioridades do dia que se inicia.
- A reunião é curta, mais ou menos uns 15 minutos. O fato de ser em pé ajuda a alcançar este objetivo.
- Registrar o progresso da iteração (quadro de acompanhamento)

55

Programação em Par

- Duas cabeças realmente pensam melhor que uma.
- Todo código de produção é escrito por pares de desenvolvedores.
- Dois papéis:
- **- Piloto**: quem está com o teclado e com o mouse. Pensa focado no código a sua frente.
- **- Navegador**: o outro. Pensa mais estrategicamente.

56

Programação em Par (Continuação)

- Pressão do par favorece a produtividade e simplicidade.
- Maior qualidade do código, mais correto, menos bugs.
- Disseminação do conhecimento e padrões.

57

Padrões de Codificação

- O código também é uma forma da equipe se comunicar.
- Manter um padrão para o entendimento rápido e fácil de qualquer membro da equipe.
- Gera maior consistência dentro do código.
- Ajuda na programação em par e tornar o código coletivo.

58

Código Coletivo

- No modelo tradicional, o desenvolvedor fica responsável por apenas uma parte do projeto. Este tipo de divisão traz sérios problemas.
- Cada desenvolvedor tem acesso a qualquer parte do sistema.
- Alterar qualquer parte do código a qualquer momento e sem aviso.
- Consequência de um código revisado por diversas pessoas.

59

Integração Contínua

- No desenvolvimento tradicional, os períodos de integração e de testes são extremamente longos o que propicia a sérios erros.
- Se possível, integração da produção diversas vezes ao dia.
- Sincronização das atividades concluídas.
- Facilidade para encontrar e depurar erros.
- Descoberta de eventuais conflitos o mais cedo possível.

60

Ritmo Sustentável

- Grande problema em projetos de software é a falta de tempo, submetendo os programadores a horas extras.
- Em projetos XP é proibido trabalhar até mais tarde.
- Ritmo sustentável de 40 horas semanais.
- Evitar sempre que possível fazer horas-extras.
- Mais concentração e menos erros de implementação.

61

Comentários finais sobre as práticas

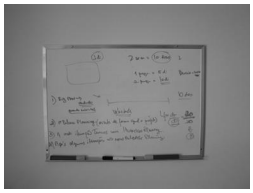
- Participação do **cliente** é fundamental para o sucesso de um projeto.
- XP estabelece um **ritmo de trabalho** sustentável.

A parte da imagem com identificação de relação rId3 não foi encontrada no arquivo.

62

Preparando-se para o XP

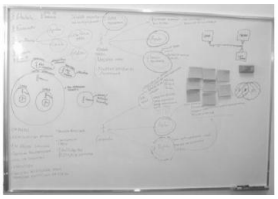
- A primeira coisa que deve ser feita para se começar um projeto XP é reorganizar o ambiente.
- White Boards.
- Mobília preparada para o trabalho em duplas.
- Mural para colocar os cartões.



Uma das mais importantes ferramentas do XP

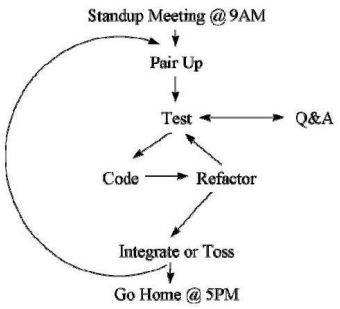
63

Área de trabalho



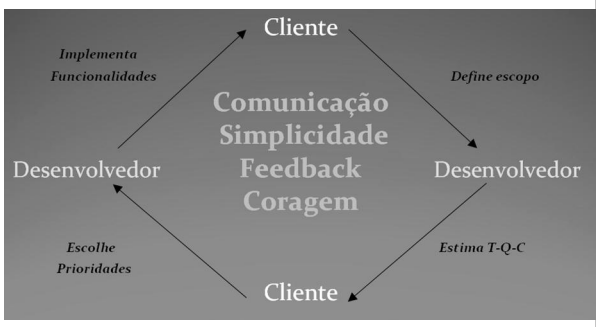
64

O ritmo XP



65

Ciclo de Vida



66

Equipe XP

- Os papéis em uma equipe XP não são fixos e rígidos. O objetivo é fazer com que cada um contribua com o melhor que tem a oferecer para que a equipe tenha sucesso.

Analista de teste	Arquitetos
Designers de Interação	Executivos
Gerentes de Projeto	Usuários
Programadores	Redatores
Técnicos	

67

• Analistas de Testes

- Responsável em garantir a qualidade do sistema através dos testes escritos. Ajudam clientes e desenvolvedores a escrever os testes para as histórias.

• Designers de Interação

- Designers de interação trabalham próximo aos clientes em um projeto XP. Eles os ajudam a escrever histórias e escolher metáforas consistentes para o projeto.

68

• Gerentes de Projeto

- Asseguram que as pessoas certas dialoguem dentro da equipe e fora dela. Nesse sentido, age como um facilitador no fluxo de comunicação do projeto.

• Programadores

- Programadores em uma equipe XP trabalham em pares implementando histórias.

69

• Redatores Técnicos

- Redatores técnicos ajudam a equipe a criar e manter a documentação do projeto. Redatores técnicos asseguram que a documentação evolua de forma iterativa.

• Arquitetos

- Ajudam os desenvolvedores no dia-a-dia através da programação em par. Ajudam a equipe a criar testes mais amplos.

70

• Executivos

- Executivos ajudam na definição do escopo do projeto. Administram as pressões dos usuários e removendo obstáculos.

• Usuários

- Ajudam a escrever e selecionar histórias e tomam decisões relativas ao domínio do negócio durante o desenvolvimento. Sua participação é tão valiosa quanto for seu conhecimento sobre o domínio do negócio.

71

Mercado

- HP
- Ford
- Objective Solutions
- Improvelt
- Symantec
- Motorola
- Brasil Telecom
- Embrapa
- Chrysler
- BMW
- Borland
- Qualiti
- APOENA Software Livre
- Argonavis
- IBM
- First National Bank
- Thought Works
- CETIP
- Secretaria da Fazenda SP
- Smart Tech Consulting
- CC Pace Systems
- Industrial Logic
- Moore
- iQualy
- IME-USP
- Xerox
- Siemens
- UFRJ

72

Livros

- Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade
  Vinicius Manhães Teles, editora Novatec, 2004.

- Extreme Programming: Embrace Change, 2nd edition
  Kent Beck e Cynthia Andres, Addison-Wesley, 2004

- Extreme Programming: Embrace Change, 1st edition
  - Kent Beck, Addison-Wesley, 200
  Com versão em Português com o título:
  •Programação Extrema, Acolha as mudanças.

RUP – *Rational Unified Process*

INTRODUÇÃO

O processo de engenharia de software define quem faz o quê, quando e como para atingir um determinado objetivo. Neste momento, iremos dissertar sobre o *Rational Unified Process*, ou RUP, que é uma plataforma de processo de desenvolvimento de software configurável que oferece melhores práticas comprovadas e uma arquitetura configurável, abordando os seguintes tópicos:

- LINHAS MESTRAS;
- FASES;
- DISCIPLINAS;
- PRINCÍPIOS E MELHORES PRÁTICAS.

CONCEITOS

RUP - *Rational Unified Process* (ou Processo Unificado da Rational)

Três Princípios básicos do RUP:

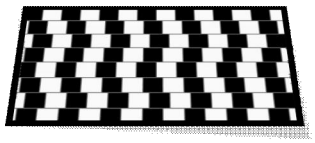
***Orientação a Casos de Uso*:** pois orientam todo o processo de desenvolvimento;

***Arquitetura**:* pois defende a definição de uma base (modelo) para a aplicação, o qual gradualmente vai ganhando forma ao longo do desenvolvimento.

Iteração: é iterativo e incremental, oferecendo uma abordagem para particionar o trabalho em porções menores

LINHAS MESTRAS

O RUP define as seguintes linhas-mestras e esqueletos (templates) para os membros da equipe de um ciclo de produção: parte do cliente, e uma avaliação do progresso do projeto pela sua gerência. Além disso, ajuda os programadores a manterem-se concentrados no projeto.



GESTÃO DE REQUISITOS

•Uma documentação apropriada é essencial

•Descreve como documentar a funcionalidade

• Restrições de sistema, projeto e requisitos de negócio

•Os casos de uso e os cenários são exemplos de artefatos dependentes do processo

USO DE ARQUITETURA BASEADA EM COMPONENTES

Cria um sistema que pode ser facilmente extensível, promovendo a reutilização de software e um entendimento intuitivo.

O RUP oferece uma forma sistemática para construir este tipo de sistema, focando-se em produzir uma arquitetura executável nas fases iniciais do projeto.

USO DE DE SOFTWARE DE MODELOS VISUAIS

Ao abstrair a programação do seu código e representá-la utilizando blocos de construção gráfica, o RUP consegue uma maneira efetiva de se ter uma visão geral de uma solução, facilitando assim o entendimento de todos os *Stakeholderes* (em português: parte interessada) como o próprio diz: toda a equipe que está envolvida, desde o Gerente do projeto até os desenvolvedores. A linguagem da UML (Linguagem de Modelagem Unificada) tornou-se um padrão industrial para representar projetos, e é amplamente utilizada pelo RUP.

VERIFICAÇÃO DA QUALIDADE DO SOFTWARE

Não assegurar a qualidade do software é a falha mais comum em muito dos projetos de sistemas computacionais, certo que também temos que considerar a regra do negócio, na maioria das vezes os sistemas são reflexo do investimento do cliente. O RUP visa auxiliar no controle do planejamento da qualidade, verificando-a na construção de todo o processo e envolvendo todos os membros da equipe de desenvolvimento.

GESTÃO E CONTROLE DE MUDANÇAS DO SOFTWARE

O RUP define métodos para controlar e monitorar mudanças e define áreas de trabalho seguras, garantindo a um programador que as mudanças efetuadas em outro sistema não afetarão o sistema.

FASES

As fases indicam a ênfase que é dada no projeto em um dado instante. Para capturar a dimensão do tempo de um projeto, o RUP divide o projeto em quatro fases diferentes.

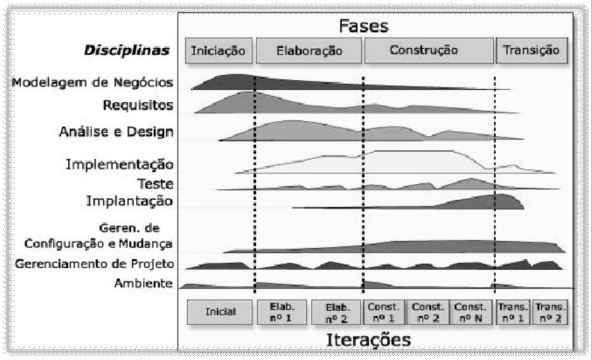
Iniciação - entendimento da necessidade e visão do projeto.

Elaboração - especificação e abordagem dos pontos de maior risco.

Construção – ênfase no desenvolvimento principal do sistema.

Transição – ênfase nos ajustes, implantação e transferência de propriedade do sistema.

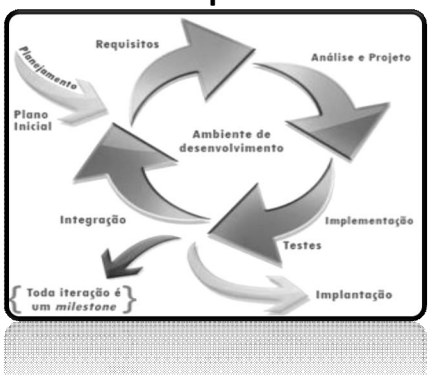
Estrutura básica do RUP



DISCIPLINAS DE ENGENHARIA

- **Modelagem de Negócios**
- **Requisitos**
- **Análise e Projeto**
- **Implementação**
- **Teste**
- **Implantação**

Disciplinas



Disciplina de Modelagem de Negócios

Modelagem de negócios, explica como descrever uma visão da organização na qual o sistema será implantado e como usar esta visão como uma base para descrever o processo, papéis e responsabilidades.

Disciplina de Requisitos

Esta disciplina explica como levantar pedidos dos stakeholders e transformá-los em um conjunto de requisitos

Disciplina de Análise e Projeto

O objetivo da análise e projeto é mostrar como o sistema vai ser realizado.

Disciplina de Implementação

Os efeitos da implementação são:

•Definir a organização do código, em termos de subsistemas de implementação organizadas em camadas;

•Implementar classes e objetos em termos de componentes;

•Testar os componentes desenvolvidos como unidades;

•Integrar os resultados produzidos por desenvolvedores individuais (ou equipes), em um sistema executável.

Disciplina de Teste

As finalidades da disciplina de teste são:

•Verificar a interação entre objetos;

•Verificar a integração adequada de todos os componentes do softwares;

•Verificar se todos os requisitos foram corretamente implementados;

•Identificar e garantir que os defeitos são indentificados antes da implantação do software;

•Garantir que todos os defeitos são corrigidos e reanalisados.

Disciplina de Implantação

O objetivo da implantação é o de produzir com sucesso lançamentos de produtos e entregar o software para seus usuários finais. Os processos workflows de "Implantação e Ambiente" do RUP contêm menos detalhes do que outros workflows.

TRÊS DISCIPLINAS DE APOIO/SUPORTE

- **Ambiente**
- **Configuração e Gerência de Mudança**
- **Gerência de projeto**

Disciplina de Ambiente

A proposta das atividades de ambiente é prover à organização de desenvolvimento de software.

Disciplina de Configuração e Gerência de Mudança

A disciplina de Gestão e mudança em negócios com RUP abrange três gerenciamentos específicos:

- **Gerenciamento de configuração**
- **Gerenciamento de solicitações de mudança**
- **Gerenciamento de status e medição**

Disciplina de Gerência de projeto

Esta disciplina concentra-se principalmente sobre os aspectos importantes de um processo de desenvolvimento iterativo: Gestão de riscos; Planejamento um projeto iterativo através do ciclo de vida e para uma iteração particular;

Não abrange questões como:

- Gestão de Pessoas: contratação, treinamento, etc
- Orçamento Geral: definição, alocação, etc
- Gestão de Contratos: com fornecedores, clientes, etc.

PRINCÍPIOS E MELHORES PRÁTICAS

RUP é baseado em um conjunto de principios de desenvolvimento de software e melhores práticas, por exemplo:

1.Desenvolvimento de software iterativo
2.Gerenciamento de requisitos
3.Uso de arquitetura baseada em componentes
4.Modelagem visual de software
5.Verificação da qualidade do software
6.Controle de alteração no software

1. Desenvolvimento interativo

O RUP usa desenvolvimento interativo e incremental pelas seguintes razões:

- A integração é feita passo a passo durante o processo de desenvolvimento;
- Mudanças de requisitos são registradas e podem ser acomodadas
- Os riscos são abordados no inicio do desenvolvimento ;

2. Gerenciamento de requisitos

O Gerenciamento de requisitos no RUP está concentrado em encontrar as necessidades do usuário final pela identificação e especificação do que ele necessita e identificando aquilo que deve ser mudado.

3. Uso de arquitetura baseada em componentes

Arquitetura baseada em componentes cria um sistema que é facilmente extensível, intuitivo e de fácil compreensão e promove a reusabilidade de software. Um componente freqüentemente se relaciona com um conjunto de objetos na programação orientada a objetos.

4. Modelagem visual de software

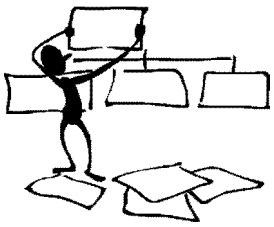
Usando esta representação, recursos técnicos podem determinar a melhor forma para implementar a dado conjunto de interdependências lógicas. Isto também constrói uma camada intermediária entre o processo de negócio e o código necessário através da tecnologia da informação.

5. Verificação da qualidade do software

RUP assume que cada membro da equipe é responsável pela qualidade durante todo o processo. O processo foca na descoberta do nível de qualidade esperado e provê testes nos processos para medir este nível.

6. Controle de alteração no software

RUP também define espaços de trabalho seguros (do inglês secure workspaces), garantindo que um sistema de engenharia de software não será afetado por mudanças em outros sistemas. Este conceito é bem aderente com arquiteturas de software baseados em componentização.

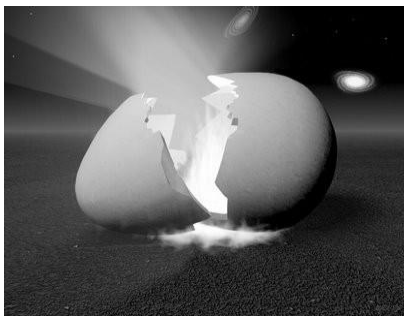


REFERÊNCIAS BIBLIOGRÁFICAS

- Rational Software - Best Practices for Software Development Teams, www.rational.com.
- Rational Software - Rational Unified Process, www.rational.com. The Rational Edge - RUP and XP, Part I - Finding Common Ground, http://therationaledge.com.
- IBM Rational Unified Process - http://pt.wikipedia.org/wiki/IBM_Rational_Unified_Process

OPENUP

(INICIO)



IBM

Tudo começou na IBM.

- ❑Ideia de uma versão ágil do IBM Rational Unified Process, ou simplesmente RUP.
- ❑Boas praticas contidas no RUP
- ❑Processo enxuto, completo e extensível
- ❑Um sub-sete do RUP.

IBM


Per Kroll

IBM com projeto EPF (Eclipse Process Framework), Liderado por Per Kroll (Gerente de Métodos IBM e líder no pensamento OPENUP), disponibiliza a versão inicial desse novo processo, até então considerada uma versão "light" e "ágil" do RUP.

IBM

A disponibilidade de um novo processo faz com que comunidades e empresas contribuam muito com novas implementações e práticas exemplos:

- SCRUM - O uso do Sprints ou interação no OPENUP
- XP (extreme programming) – Utilização testes.

ARTEFATOS

- Documento de caso de uso
- Definição de escopo

Princípios

O OPENUP possui quatro princípios.

1.Equilibrar as prioridades concorrentes para aumentar os benefícios aos *Stakeholders*.

2.Colaborar para alinhar os interesses e compartilhar o entendimento.

3.Focar na arquitetura, o mais cedo possível, para reduzir o risco e organizar o desenvolvimento

4.Evoluir para continuamente obter feedback e promover melhorias

1_Equilibrar as prioridades concorrentes para aumentar os benefícios aos Stakeholders

É preciso alinhar as necessidades dos Stakeholders com as restrições do projeto.

•Problema ser resolvido.

•Restrições impostas á equipe de desenvolvimento(custo, cronograma, recursos, regulamentos).

•Restrições impostas á solução.

2_Colaborar para alinhar os interesse e compartilhar o entendimento.

Num projeto cada membro tem seus próprios conhecimentos, habilidades e maneiras de se fazer algo.

Ex:

•Um problema pode ter varias soluções diferentes.

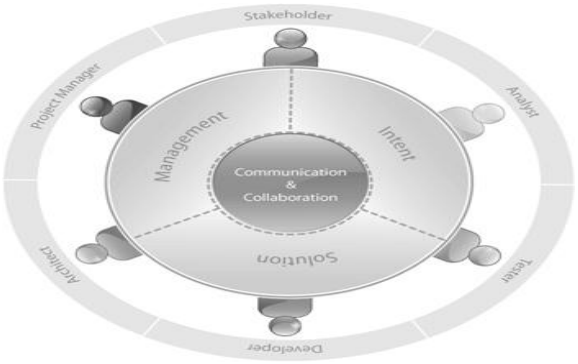
3_Focar na arquitetura, o mais cedo possível, para reduzir o risco e organizar o desenvolvimento

O desenvolvimento do projeto depende de sua arquitetura.

4_Evoluir para continuamente obter feedbacks e obter melhorias.

Este principio tem por objetivo trazer melhorias ao sistema por meio de feedback, podendo assim reparar falhas.

PAPÉIS



Papéis

Steakholder

•Representa o cliente.

•Fundamental durante todo o ciclo do projeto. para a coleta de requisitos, definição de prioridades, avaliação das versões entregues, feedback das versões apresentadas.

Analyst
(analista)

•O papel do Analista no Open UP é chave no processo de captação das necessidades do cliente, fazendo a função de coletar, organizar, filtrar e apresentar à equipe os requisitos do cliente.

Papéis

Architect
(arquiteto)

•responsável por fornecer o raciocínio para estas decisões, balancear os interesses dos Stakeholders, reduzir os riscos técnicos e assegurar que as decisões sejam eficazmente comunicadas, validadas e seguidas.

Project Manager
(gerente de projeto)

•Suas atividades são relacionadas ao cotidiano do projeto, objetivando manter a comunicação com o StakeHolder e focando em ter a equipe determinada a cumprir os prazos, cumprir requisitos. Tem a tarefa de gerar e manter o plano do projeto, participar do planejamento e do gerenciamento das iterações, de mostrar valor continuamente ao cliente

PAPÉIS

Developer
(desenvolverdor)

•Responsável pelo desenvolvimento do projeto, pela codificação, participando diretamente também da fase de testes das iterações.

Tester
(testador)

•Este papel é responsável principalmente pelas seguintes tarefas:
•Identificar os testes que necessitam ser executados
•Identificar a abordagem de implementação mais apropriada para um determinado teste
•Implementar testes individuais
•Preparar e executar os testes
•Registrar os resultados e verificar se os testes executaram
•Analisar e recuperar dos erros de execução
•Comunicar os resultados do teste à equipe

Disciplinas

- Uma disciplina é uma coleção de tarefas que se relacionam a uma "área de interesse" maior em todo o projeto. O agrupamento de tarefas em disciplinas serve principalmente para ajudar a compreender o projeto dentro de uma visão tradicional em cascata. Embora seja comum executar simultaneamente tarefas que pertençam a várias disciplinas (por exemplo, determinadas tarefas de requisitos são executadas sob a mesma coordenação de tarefas de análise e design), separar estas tarefas em disciplinas distintas é uma forma eficaz de organizar o conteúdo, tornando mais fácil a compreensão.

disciplina

O OPENUP contém seis disciplinas:

- Requisitos
- Gestão de Projeto
- Arquitetura
- Desenvolvimento
- Teste
- Gestão de mundanças

Requisitos

Esta disciplina é responsável por definir as tarefas mínimas necessárias para eliciar, analisar, especificar,validar, e gerenciar as necessidades do sistema a ser desenvolvido.

TAREFAS DA DISCIPLINA:

- Entender o problema a ser resolvido.
- Definir o escopo do sistema(limites).

Gestão de Projeto

O objetivo desta disciplina é instruir e ajuda a equipe a ter um alto desempenho.

TAREFAS DA DISCIPLINAS:

- Planejar projeto.
- Planejar interação.
- Avaliar os resultados.

Arquitetura

A arquitetura tem importante papel na compreensão e extensão do sistema, ela mostra como arquiteta o sistema a partir dos requisitos.

TAREFAS DA DISCIPLINA:

- Definir a arquitetura.
- Refinar a arquitetura.

Desenvolvimento

A disciplina de desenvolvimento explica como projetar e implementar uma solução técnica que seja aderente à arquitetura e atenda aos requisitos.

Tarefas da disciplina:

- Projetar a solução
- Implementar a solução
- Implementar os testes de desenvolvedor
- Executar os testes de desenvolvedor

Teste

Esta disciplina define um conjunto mínimo de tarefas requeridas para planejar, implementar, executar e avaliar o teste do sistema.

Testar desafia as suposições, riscos e incertezas inerente no trabalho de outras disciplinas, e trata dessas preocupações que usam demonstração concreta e avaliação imparcial.

Teste

- O propósito desta disciplina é:
- Encontrar e documentar defeitos.
- Validar e provar as suposições feitas no projeto e requisitos especificados através de demonstrações concretas.
- Validar que o produto de software foi feito como projetado.
- Validar que os requisitos estão apropriadamente implementados.

Tipos de teste

- **Funcionalidade**
- Teste de função.
- Teste de segurança.
- Teste de volume.
- **Usabilidade**
- Testes de Usabilidade.
- Teste de integridade.
- Teste de estrutura.

Teste

- **Confiança**
- Teste de stress.
- Teste de avaliação de desempenho.
- Teste de contenção.
- **Performance**
- Teste de carga .
- Perfil de desempenho.
- Teste de configuração.
- **Suportabilidade**
- Teste de instalação.

Gestão de configuração e mudança

A disciplina de gestão de configuração e mudança explica como controlar as mudanças nos artefatos, assegurando uma evolução sincronizada do conjunto de Produtos de Trabalho que compõem um sistema de software.

Gestão de configuração e mudança

O propósito desta disciplina é:

- Manter um conjunto de produtos de trabalho consistente a medida que evolue.
- Manter construções de software consistentes.
- Fornecer meios eficientes para se adaptar às mudanças, replanejando o trabalho adequadamente.
- Fornecer dados para a medição do progresso.

Gestão de configuração e mudança

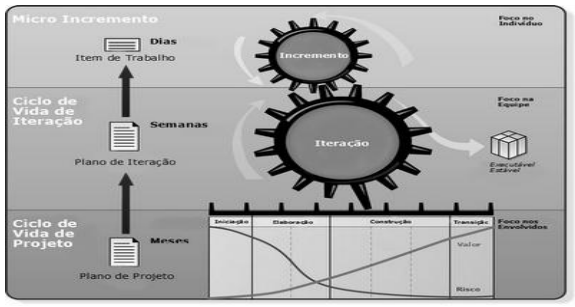
- Esta disciplina expande-se por todo o ciclo de vida. Todas as outras disciplinas contam com a disciplina de Gerência de Configuração e Mudança para manter um conjunto de produtos de trabalho consistente e atualizado, e para priorizar e rastrear as mudanças nesses produtos de trabalho durante todo o ciclo de vida.

Gestão de configuração e mudança

- A Gerência de Configuração e Mudança é feita por todos na equipe de desenvolvimento. Por causa da importância e da penetração desta disciplina, a orientação da Gerência de Configuração e Mudança é associada as tarefas e aos produtos de trabalho de todas as outras disciplinas.

CAMADAS

Open Up é um processo interativo incremental de desenvolvimento de software estruturado por 3 camadas



CAMADAS

1ª - Ciclo de Vida de Projeto

Iniciação:
- processo de análise de negócios
- análise de requisitos
- menor arquitetura e implementação

Elaboração:
- desenvolvimento da análise arquitetural da solução proposta

Construção:
- implementação da solução

Transição:
- implementação do releases
- testes são importantes

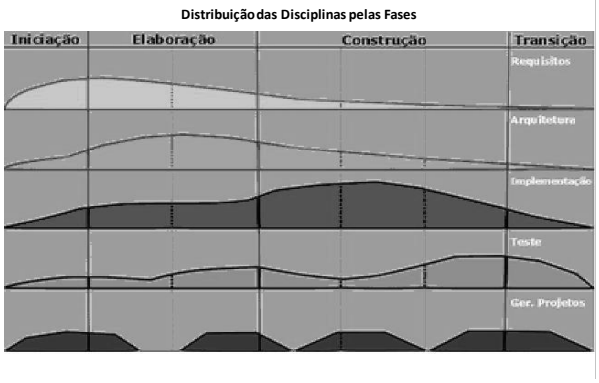
CAMADAS

CAMADAS

2ª - Ciclo de Vida de Iteração :

- Divisão de atividades principais em subatividades
- Disciplinas tratadas pelo Open Up: Requisitos, Arquitetura, Implementação, Teste e Gerência de Projeto
- Focado no desenvolvimento de um build, ou seja, um executável com duração de algumas semanas
- Testes para garantia de que a aplicação atende os requisitos especificados pelos Stakeholders
- Micro incrementos nada mais são do que sub-divisões de uma interação

CAMADAS



CAMADAS

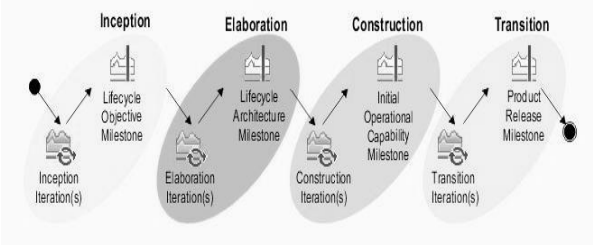
3ª - Ciclo de Vida de Micro Incremento

- Auxilio aos indivíduos do desenvolvimento
- Divisão de atividades em pequenas unidades
- Esforço de algumas horas ou dias
- Quem desenvolve são membros formados em grupos de 1 à 3 pessoas da equipe
- Encerra com um artefato
- Feedbacks rápidos

FASES

Ciclo de Vida do Projeto :

- Cada fase tem o foco nas necessidades dos Stakeholders.
- O OpenUP apresenta a mesma distribuição de fases já conhecidas no RUP.



FASES

Iniciação

- Todos os stakeholders concordam com o escopo e objetivos do projeto.
- Determinam se o projeto deve continuar.

FASES

Elaboração

- Todos concordam com a arquitetura proposta
  cenários críticos, interfaces, etc.
- É produzido para o cliente um cronograma e uma estimativa de custo preciso.

FASES

Construção

- A aplicação que está quase pronta bem próxima a ser finalizada.
- É detalhado todos os requisitos, design e implementação do projeto.
- Teste da maior parte do software.

FASES

Transição

- O desenvolvimento do produto está completo.
- A aplicação está finalizada e o cliente satisfeito.

BIBLIOGRAFIA

- http://epf.eclipse.org/wikis/openuppt/
- http://www.ibm.com/developerworks/br/rational/local/open_up/index.html
- http://pt.wikipedia.org/wiki/OpenUP
- http://www.ibm.com/developerworks/br/rational/local/open_up/index.html