Engenharia de software

3 Processo de software

Definição

Um processo de Engenharia de Software é formado por um conjunto de passos de processo parcialmente ordenados, relacionados com artefatos, pessoas, recursos, estruturas organizacionais e restrições, tendo como objetivo produzir e manter os produtos de software finais requeridos.

Conceitos

Os processos são divididos em atividades ou tarefas. Uma atividade é um passo de processo que produz mudanças de estado visíveis externamente no produto de software. Atividades incorporam e implementam procedimentos, regras e políticas, e têm como objetivo gerar ou modificar um dado conjunto de artefatos.

Um artefato é um produto criado ou modificado durante um processo. Tal produto é resultado de uma atividade e pode ser utilizado posteriormente como matéria prima para a mesma ou para outra atividade a fim de gerar novos produtos.

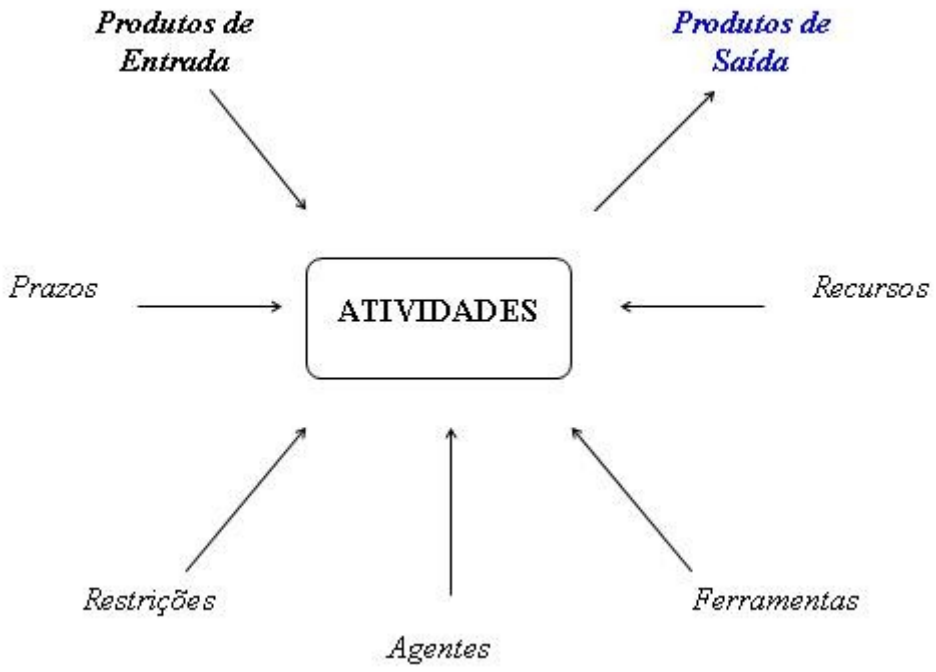
Uma atividade aloca recursos (por exemplo, computadores, impressoras e material de expediente), é escalonada, monitorada e atribuída a desenvolvedores (agentes), que podem utilizar ferramentas para executá-la.

Toda atividade possui uma descrição, a qual pode especificar os artefatos necessários, as relações de dependência com outras atividades, as datas de início e fim planejadas, os recursos a serem alocados e os agentes responsáveis pela mesma.

O agente pode ser uma pessoa ou uma ferramenta automatizada (quando a atividade é automática) e relaciona-se com as atividades de um processo. Os agentes podem estar organizados em cargos, aos quais podem ser definidas diferentes responsabilidades.

A realização do processo é afetada pelas restrições, que podem atingir atividades, agentes, recursos, artefatos, papéis e seus relacionamentos. Uma restrição é uma condição definida que um passo de processo deve satisfazer antes ou depois de ser executado.

Atividade de Processos de Engenharia de Software e seus inter-relacionamentos



Ou seja, 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.

Devido ao uso da palavra projeto em muitos contextos, por questões de clareza, há vezes em que se prefira usar o original em inglês design.

Modelos de processo de software

Um modelo de processo de desenvolvimento de software, ou simplesmente modelo de processo, pode ser visto como uma representação, ou abstração dos objetos e atividades envolvidas no processo de software. Além disso, oferece uma forma mais abrangente e fácil de representar o gerenciamento de processo de software e consequentemente o progresso do projeto.

Exemplos de alguns modelos de processo de software;

Modelos ciclo de vida

1. Sequencial ou Cascata (do inglês *waterfall*) - com fases distintas de especificação, projeto e desenvolvimento.
2. Desenvolvimento iterativo e incremental - desenvolvimento é iniciado com um subconjunto simples de Requisitos de Software e iterativamente alcança evoluções subsequentes das versões até o sistema todo estar implementado
3. Evolucional ou Prototipação - especificação, projeto e desenvolvimento de protótipos.
4. V-Model - Parecido com o modelo cascata, mas com uma organização melhor, que permite que se compare com outros modelos mais modernos.
5. Espiral - evolução através de vários ciclos completos de especificação, projeto e desenvolvimento.
6. Componentizado - reuso através de montagem de componentes já existentes.
7. Formal - implementação a partir de modelo matemático formal.

8. Ágil
9. RAD
10. Quarta geração.

Modelos de Ciclos de Vida

O Desenvolvimento de um Sistema de Informação, independentemente do modelo de ciclo de vida utilizado, abrange basicamente 4 estágios :

1) Problema inicial, que pode ser um erro em um sistema/software existente ou a necessidade da criação de um software para automatizar um processo;

2) Definição e Análise do problema que deverá ser resolvido;

3) Desenvolvimento Técnico ou Codificação que resolverá o problema através da aplicação de alguma tecnologia.

4) Implantação da solução, ou seja, o sistema é entregue ao usuário final.

3.1. Modelo em Cascata

O modelo de ciclo de vida em cascata foi o primeiro modelo a ser conhecido em engenharia de software e está na base de muitos ciclos de vida utilizados hoje em dia. Este consiste basicamente num modelo linear em que cada passo deve ser completado antes que o próximo passo possa ser iniciado.

Por exemplo, a análise de requisitos deve ser completada antes que o desenho do sistema possa ser iniciado.

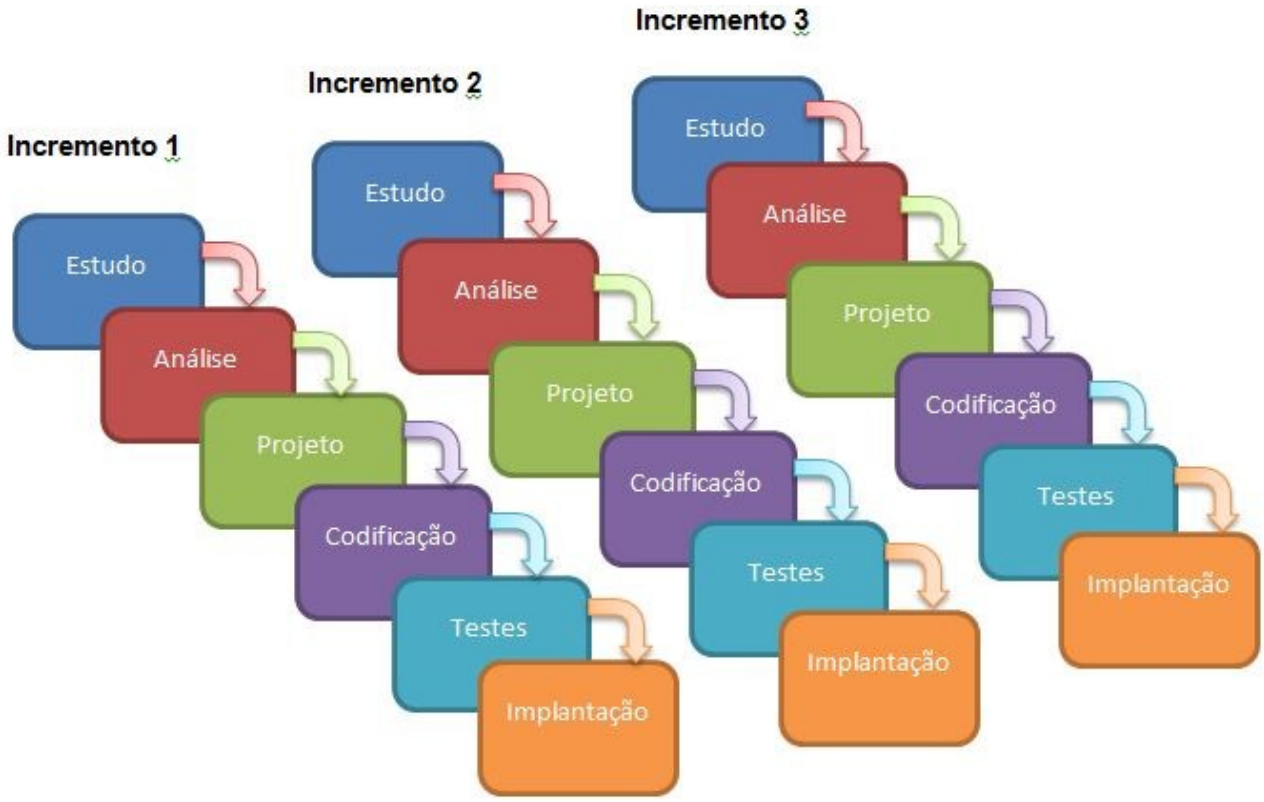
Os nomes dados a cada passo variam, assim como varia a definição exata de cada um deles, mas basicamente o ciclo de vida começa com a análise de requisitos movendo-se de seguida para a fase de desenho, codificação,

implementação, teste e finalmente manutenção do sistema.

Uma das grandes falhas deste modelo é o fato de os requisitos estarem constantemente a mudar já que os negócios e ambiente em que se inserem mudam rapidamente. Isto significa que não faz sentido parar os requisitos durante muito tempo, enquanto o desenho e implementação do sistema são completados.

Foi então reconhecido que seria necessário dar *feedback* às atividades iniciais a partir do momento em que este modelo começou a ser usado em grande escala. A ideia de interacção não foi incorporada na filosofia do modelo de cascata. Neste momento, é incluído algum nível de interação na maior parte das versões deste modelo e são comuns sessões de revisão entre os elementos responsáveis pelo desenvolvimento do sistema. No entanto, a possibilidade de revisão e avaliação com os utilizadores do sistema não está contemplada neste modelo.

3.2 Modelo Incremental e Iterativo



O Desenvolvimento Incremental é uma estratégia de planejamento estagiado em que várias partes do sistema são desenvolvidas em paralelo, e integradas quando completas. Não implica, requer ou pressupõe desenvolvimento iterativo ou em cascata – ambos são estrategias de retrabalho. A alternativa ao desenvolvimento incremental é desenvolver todo o sistema com uma integração única.

Desenvolvimento iterativo é uma estrategia de planejamento de retrabalho em que o tempo de revisão e melhorias de partes do sistema é pré-definido. Isto não pressupõe desenvolvimento incremental, mas funciona muito bem com ele. Uma diferença típica é que a saída de um incremento não é necessariamente assunto de um refinamento futuro, e seu teste ou retorno do usuário não é utilizado como entrada para planos de revisão ou especificações para incrementos sucessivos. Ao contrario, a saída de uma iteração é examinada para modificação, e especialmente

para revisão dos objetivos das iterações sucessivas.

A idéia básica por trás da abordagem iterativa é desenvolver um sistema de software incremental, permitindo ao desenvolvedor tirar vantagem daquilo que foi aprendido durante a fase inicial de desenvolvimento de uma versão do sistema. O aprendizado ocorre simultaneamente tanto para o desenvolvedor, quanto para o usuário do sistema.

Os passos fundamentais do processo estão em iniciar o desenvolvimento com um subconjunto simples de Requisitos de Software e iterativamente alcançar evoluções subseqüentes das versões até o sistema todo estar implementado. A cada iteração, as modificações de projeto são feitas e novas funcionalidades são adicionadas.

O projeto em si consiste da etapa de inicialização, iteração e da lista de controle do projeto.

A etapa de inicialização cria uma versão base do sistema. O objetivo desta implementação inicial é criar um produto para que o usuário possa avaliar. Ele deve oferecer um exemplo dos aspectos chave do problema e prover uma solução que seja simples o bastante para que possa ser compreendida e implementada facilmente. Para guiar o processo iterativo, uma lista de controle de projeto é criada.

Ela conterá um registro de todas as tarefas que necessitam ser realizadas. Isto inclui itens tais como novas características a serem implementadas e áreas para serem projeto na solução atual. A lista de controle deve ser continuamente revisada como um resultado da fase de análise.

A etapa iterativa envolve o re-projeto e implementação das tarefas da lista de controle do projeto e a análise da versão corrente do sistema. O objetivo para o projeto de implementação de qualquer iteração é ser simples, direto e modular, preparado para suportar re-projeto neste estágio ou como uma tarefa a ser

adicionada na lista de controle do projeto. O código pode, em alguns casos, representar uma fonte maior da documentação do sistema. A análise de uma interação é baseada no feedback do usuário, e facilidades da análise do programa disponíveis. As estruturas de análise envolvidas são a modularidade, usabilidade, reusabilidade, eficiência e obtenção dos objetivos. A lista de controle do projeto é modificada à luz dos resultados da análise.

Linhas básicas para direcionar a implementação e análise incluem:

- Qualquer dificuldade no projeto, codificação e teste de uma modificação deve ser sinalizada para que possa ser re-projetada ou recodificada.
- Modificações devem ser ajustadas facilmente em módulos isolados e fáceis de encontrar. Se não atendem a isso, algum re-projeto deverá ser necessário.
- Modificações de tabelas devem ser especialmente fáceis de fazer. Se qualquer modificação não é rápida e fácil de ser feita, indica-se a realização de um re-projeto.
- Modificações devem ser fáceis para serem feitas na forma de iterações. Se elas não são, haverá um problema básico tal como um projeto falho ou uma proliferação de correções.
- Correções devem normalmente ser permitidas por somente uma ou duas iterações. Correções devem ser necessariamente para evitar re-projeto durante uma fase de implementação.
- A implementação existente deve ser analisada freqüentemente para determinar quão bem estão sendo atingidos os objetivos do projeto.
- As ferramentas de análise de programa devem ser usadas sempre que necessário para ajudar na análise de implementações parciais.

- Reclamações do usuário devem ser solicitadas e analisadas para registrar as deficiências da implementação atual.

Características

O uso de análise e medições como guia do processo de aprimoramento é uma das maiores diferenças entre o desenvolvimento iterativo e o atual Desenvolvimento ágil de software. Isto provê suporte determinante para a efetividade do processo de qualidade do produto, permitindo estudar o processo para um ambiente em particular. As atividades de medição e análise podem ser adicionadas a métodos de desenvolvimento ágil existentes.

De fato, o contexto das interações múltiplas provê vantagens no uso de medições. Medições são algumas vezes difíceis de serem compreendidas no valor absoluto, mas mudanças relativas nas medições ao longo da evolução de um sistema podem ser muito instrutivas como base para uma análise. Por exemplo, um vetor de medições, m1, m2, ... mn, pode ser definido para caracterizar vários aspectos do produto em algum ponto no tempo, por exemplo, esforço para dados, mudanças, defeitos, atributos lógicos, físicos e dinâmicos, considerações do ambiente, entre outros. Portanto, um observador pode dizer como o produto se caracteriza quanto ao tamanho, complexidade, acoplamento e coesão, se estão aumentando ou diminuindo em relação ao tempo. Tais atributos podem ser monitorados em relação a mudanças de vários aspectos do produto ou podem prover parâmetros para a medição de sinais de potenciais problemas e anomalias.

Equívocos

O nome correto é **iterativo** (repete várias vezes) e não **interativo** (capaz de interagir, comunicar) como muitos pensam.

3.3 Prototipação

Prototipação é uma abordagem baseada numa visão evolutiva do desenvolvimento de software, afetando o processo como um todo. Esta abordagem envolve a produção de versões iniciais - protótipos (análogo a maquetes para a arquitetura) - de um sistema futuro com o qual pode-se realizar verificações e experimentos,com intuito de avaliar algumas de suas caracteristicas antes que o sistema venha realmente a ser construído, de forma definitiva.

Quando usar?

- Em muitos casos o cliente define somente um conjunto de objetivos gerais para o Sistema (Software), mas não foi capaz de gerar requisitos definidos, de entrada , processamento e saída, para o Sistema (Software).
- Desenvolvedor não tem certeza da eficiência de um algoritmo, ou como ele pode se comportar em um determinado Sistema Operacional,ou durante a comunicação com alguma interface,perifericos/componentes;
- Interação homem-máquina pode não ser aceita pelo cliente, ou seja a interface de comunicação com o aplicação (Software) pode ser confusa ou não usual.

O que gerar como protótipo?

Para gerar o protótipo existem varias formas e ferramentas , sendo as mais usuais

- Modelo de papel: que ilustra como o sistema(Software), ira se comportar e interagir como o Usuário de forma a capacitar a todos entender como ocorrerão os processos de interação.
- Modelo de trabalho: que implemente algumas características do software , em sua maioria a interface de comunicação com usuário como a navegação em telas, entre outros subconjunto de funcionalidade existente no sistema; Toda a funcionalidade existente que será melhorada em um novo esforço de desenvolvimento,gerando um novo protótipo mais completo

Desvantagens

O cliente vê a versão em funcionamento e exige alguns acertos para colocar logo em uso; A codificação utilizada para apresentar o protótipo pode no final ser a definitiva; O descartamento do protótipo pode ser visto com perda de tempo para o cliente.

3.4 Modelo em V

The V-model of the Systems Engineering Process.

O Modelo V é um modelo conceitual de gestão de projeto visto como melhoria ao problema de reatividade do modelo em cascata. Ele permite, em caso de anomalia, delimitar um retorno às etapas precedentes. As fases das partes acima devem devolver as informações sobre elas, camada a camada, quando os padrões são detectados, a fim de melhorar o programa.

O Modelo V virou um padrão da indústria de software depois de 1980 e, após o surgimento da Engenharia de Sistemas, tornou-se um conceito padrão em todos os domínios da indústria. O mundo do software tinha feito poucos avanços em termos de maturidade, em achar na bibliografia corrente as referências que poderiam se aplicar ao sistema.

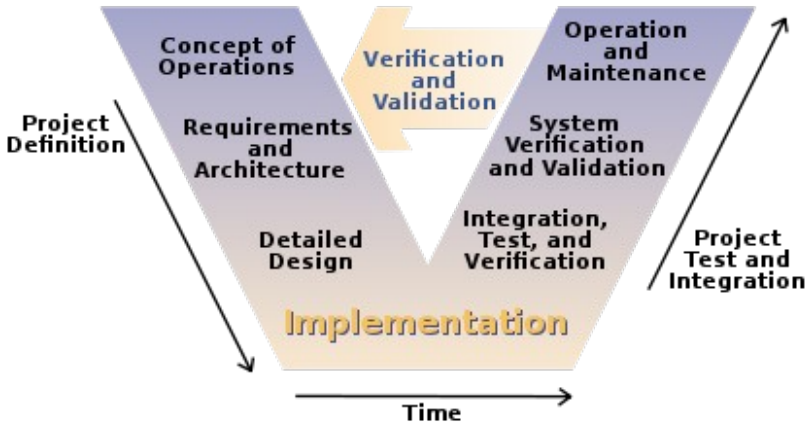
- Enfatiza a importância de considerar as atividades de testes durante o processo, ao invés de um teste posterior após o término do processo;
- Pode-se obter a retroalimentação mais rapidamente;
- Ajuda a desenvolver novos requisitos;
- Melhora a qualidade do produto resultante.

Desvantagens

- Dificuldade em seguir o fluxo seqüencial do modelo;
- Dificuldade para o cliente poder especificar os requisitos explicitamente.

Etapas

- Análise das necessidades e viabilidade;
- Especificação do software;
- Concepção: arquitetura;
- Concepção: detalhamento;
- Codificação;
- Teste individual;
- Teste de integração;
- Teste de validação;
- Receita.



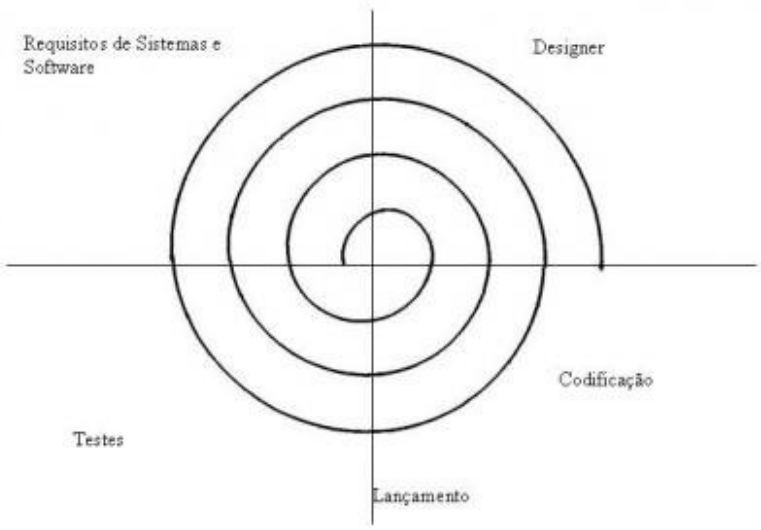
Modelo em Espiral

Durante muitos anos, o modelo cascata foi a base da maior parte do desenvolvimento de projetos de software, mas em 1988 Barry Boehm sugeriu o modelo em espiral. Do modelo em espiral para desenvolvimento de software saltam a vista dois aspectos: a análise de risco e prototipagem.

O modelo espiral incorpora-os de uma forma interativa permitindo que as idéias e o progresso sejam verificados e avaliados constantemente.

Cada interação à volta da espiral pode ser baseada num modelo diferente e pode ter diferentes atividades. No caso da espiral, não foi a necessidade do envolvimento dos utilizadores que inspirou a introdução de interação mas sim a necessidade de identificar e controlar riscos.

No modelo espiral para engenharia de requisitos mostra-se que as diferentes atividades são repetidas até uma decisão ser tomada e o documento de especificação de requisitos ser aceito. Se forem encontrados problemas numa versão inicial do documento, reentra-se nas fases de levantamento, análise, documentação e validação. Isto repete-se até que seja produzido um documento aceitável ou até que fatores externos, tais como prazos e falta de recursos ditem o final do processo de engenharia de requisitos.



3.6 Modelo Componentizado – utiliza o reuso através de montagem de componentes já existentes.

3.7 Modelo Formal - implementação a partir de modelo matemático formal.

Características de Vários Modelos

Na tabela seguinte estão sumariadas algumas vantagens e desvantagens de vários modelos de ciclo de vida utilizados em engenharia de requisitos para projetos de software:

Modelo	Vantagens	Desvantagens
Cascata	Minimiza o tempo de planejamento. Funciona bem para equipes tecnicamente	Inflexível. Apenas a fase final produz uma entrega que não é um documento. Torna-se

	mais fracas.	difícil voltar atrás para corrigir erros.
Espiral	As interações iniciais do projeto são as mais baratas, permitindo que as tarefas de maior risco sejam levadas com o mínimo de custos. Cada iteração da espiral pode ser customizada para as necessidades específicas de cada projeto.	É complexo e requer atenção e conhecimento especiais para o levar até o final.
Prototipagem Evolucionária	Os clientes conseguem ver os progressos. É útil quando os requisitos mudam rapidamente e o cliente está relutante em aceitar um conjunto de requisitos.	É impossível determinar com exatidão o tempo que o projecto vai demorar. Não há forma de sabero número de iterações que serão necessárias.
Codificação e Correcção	Não há tempo gasto em planejamento, documentação, gestão de qualidade e cumprimento de standards. Requer pouca experiência.	Perigoso. Não há forma de assegurar qualidade e identificar riscos. Falhas fundamentais não percebidas imediatamente resultando em trabalho jogado fora.

3.8 Desenvolvimento ágil de software

Desenvolvimento ágil de software (do inglês *Agile software development*) ou **Método ágil** é um conjunto de metodologias de desenvolvimento de *software*. O desenvolvimento ágil, tal como qualquer metodologia de *software*, providencia uma

estrutura conceitual para reger projetos de engenharia de software.

Introdução

Existem inúmeros frameworks de processos para desenvolvimento de software. A maioria dos métodos ágeis tenta minimizar o risco pelo desenvolvimento do software em curtos períodos, chamados de iteração, os quais gastam tipicamente menos de uma semana a até quatro.

Cada iteração é como um projeto de software em miniatura de seu próprio, e inclui todas as tarefas necessárias para implantar o mini-incremento da nova funcionalidade: planejamento, análise de requisitos, projeto, codificação, teste e documentação.

Enquanto em um processo convencional, cada iteração não está necessariamente focada em adicionar um novo conjunto significativo de funcionalidades, um projeto de software ágil busca a capacidade de implantar uma nova versão do software ao fim de cada iteração, etapa a qual a equipe responsável reavalia as prioridades do projecto.

Métodos ágeis enfatizam comunicações em tempo real, preferencialmente face a face, a documentos escritos. A maioria dos componentes de um grupo ágil deve estar agrupada em uma sala. Isso inclui todas as pessoas necessárias para terminar o software: no mínimo, os programadores e seus *clientes* (clientes são as pessoas que definem o produto, eles podem ser os gerentes, analistas de negócio, ou realmente os clientes). Nesta sala devem também se encontrar os testadores, projetistas de iteração, redatores técnicos e gerentes.

Métodos ágeis também enfatizam trabalho no software como uma medida primária de progresso. Combinado com a comunicação face-a-face, métodos ágeis produzem pouca documentação em relação a outros métodos, sendo este um dos pontos que podem ser considerados negativos. É recomendada a produção de documentação que realmente será útil.

Princípios

Os princípios do desenvolvimento ágil valorizam:

- Garantir a satisfação do consumidor entregando rapidamente e continuamente softwares funcionais;
- Softwares funcionais são entregues frequentemente (semanas, ao invés de meses);
- Softwares funcionais são a principal medida de progresso do projeto;
- Até mesmo mudanças tardias de escopo no projeto são bem-vindas.
- Cooperação constante entre pessoas que entendem do 'negócio' e desenvolvedores;
- Projetos surgem através de indivíduos motivados, entre os quais existe relação de confiança.
- Design do software deve prezar pela excelência técnica;
- Simplicidade;
- Rápida adaptação às mudanças;
- Indivíduos e interações mais do que processos e ferramentas;
- *Software* funcional mais do que documentação extensa;
- Colaboração com clientes mais do que negociação de contratos;
- Responder a mudanças mais do que seguir um plano.

História

As definições modernas de desenvolvimento de software ágil evoluíram a partir da metade de 1990 como parte de uma reação contra métodos "pesados", caracterizados por uma pesada regulamentação, regimentação e micro gerenciamento usado o modelo em cascata para desenvolvimento. O processo originou-se da visão de que o modelo em cascata era burocrático, lento e contraditório a forma usual com que os engenheiros de software sempre realizaram trabalho com eficiência.

Uma visão que levou ao desenvolvimento de métodos ágeis e iterativos era retorno a prática de desenvolvimento vistas nos primórdios da história do desenvolvimento de software

Inicialmente, métodos ágeis eram conhecidos como *métodos leves*. Em 2001, membros proeminentes da comunidade se reuniram em Snowbird e adotaram o nome *métodos ágeis*, tendo publicado o Manifesto ágil, documento que reúne os princípios e práticas desta metodologia de desenvolvimento. Mais tarde, algumas pessoas formaram a *Agile Alliance*, uma organização não lucrativa que promove o desenvolvimento ágil.

Os métodos ágeis iniciais—criado a priore em 2000— incluíam Scrum (1986), Crystal Clear, Programação extrema (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (1995).

Comparações com outros métodos

Métodos Ágeis são algumas vezes caracterizados como o oposto de metodologias *guiadas pelo planejamento* ou *disciplinadas*. Uma distinção mais acurada é dizer que os métodos existem em um contínuo do *adaptativo* até o *preditivo*.

Métodos ágeis existem do lado adaptativo deste contínuo. Métodos adaptativos buscam a adaptação rápida a mudanças da realidade. Quando uma necessidade de um projeto muda, uma equipe adaptativa mudará também. Um time adaptativo terá dificuldade em descrever o que irá acontecer no futuro. O que acontecerá em uma data futura é um item de difícil predição para um método adaptativo. Uma equipe adaptativa pode relatar quais tarefas se iniciarão na próxima semana. Quando perguntado acerca de uma implantação que ocorrerá daqui a seis meses, uma equipe adaptativa deve ser capaz somente de relatar a instrução de missão para a implantação, ou uma expectativa de valor versus custo.

Métodos preditivos, em contraste, colocam o planejamento do futuro em detalhe. Uma equipe preditiva pode reportar exatamente quais aspectos e tarefas estão planejados para toda a linha do processo de desenvolvimento. Elas porém tem dificuldades de mudar de direção. O plano é tipicamente otimizado para o objetivo original e mudanças de direção podem causar a perda de todo o trabalho e determinar que seja feito tudo novamente. Equipes preditivas freqüentemente instituem um comitê de controle de mudança para assegurar que somente as

mudanças mais importantes sejam consideradas.

Métodos ágeis têm muito em comum com técnicas de *Desenvolvimento rápido de aplicação* de 1980 como exposto por James Martin e outros.

Comparação com o desenvolvimento iterativo

A maioria dos métodos ágeis compartilha a ênfase no Desenvolvimento iterativo e incremental para a construção de versões implantadas do software em curtos períodos de tempo. Métodos ágeis diferem dos métodos iterativos porque seus períodos de tempo são medidos em semanas, ao invés de meses, e a realização é efetuada de uma maneira altamente colaborativa. estendendo-se a tudo.

Comparação com o modelo em cascata

O desenvolvimento ágil tem pouco em comum com o modelo em cascata. Na visão de alguns este modelo é desacreditado, apesar de ser um modelo de uso comum. O modelo em cascata é uma das metodologias com maior ênfase no planejamento, seguindo seus passos através da captura dos requisitos, análise, projeto, codificação e testes em uma seqüência pré-planejada e restrita.

O progresso é geralmente medido em termos de entrega de artefatos— especificação de requisitos, documentos de projeto, planos de teste, revisão do código, e outros. O modelo em cascata resulta em uma substancial integração e esforço de teste para alcançar o fim do ciclo de vida, um período que tipicamente se estende por vários meses ou anos.

O tamanho e dificuldade deste esforço de integração e teste é uma das causas das falhas do projeto em cascata. Métodos ágeis, pelo contrário, produzem um desenvolvimento completo e teste de aspectos (mas um pequeno subconjunto do todo) num período de poucas semanas ou meses. Enfatiza a obtenção de pequenos pedaços de funcionalidades executáveis para agregar valor ao negócio cedo, e continuamente agregar novas funcionalidades através do ciclo de vida do

projeto.

Algumas equipes ágeis usam o modelo em cascata em pequena escala, repetindo o ciclo de cascata inteiro em cada iteração. Outras equipes, mais especificamente as equipes de Programação extrema, trabalham com atividades simultaneamente.

Comparação com a "codificação cowboy" (code-fix)

A codificação cowboy, também chamada de Modelo Balbúrdia, é a ausência de metodologias de desenvolvimento de Software: os membros da equipe fazem o que eles sentem que é correto. Como os desenvolvedores que utilizam métodos ágeis freqüentemente reavaliam os planos, enfatizam a comunicação face a face e fazem o uso relativamente esparso de documentos, ocasionalmente levam as pessoas a confundirem isto com codificação *cowboy*. Equipes ágeis, contudo, seguem o processo definido (e freqüentemente de forma disciplinada e rigorosa).

Como em todas as metodologias, o conhecimento e a experiência dos usuários definem o grau de sucesso e/ou fracasso de cada atividade. Os controles mais rígidos e sistematizados aplicados em um processo implicam altos níveis de responsabilidade para os usuários. A degradação de procedimentos bem-intencionados e organizados pode levar as atividades a serem caracterizadas como codificação *cowboy*.

Aplicabilidade dos métodos ágeis

Embora os métodos ágeis apresentem diferenças entre suas práticas, eles compartilham inúmeras características em comum, incluindo o desenvolvimento iterativo, e um foco na comunicação interativa e na redução do esforço empregado em artefatos intermediários. (Cohen et al., 2004).

A aplicabilidade dos métodos ágeis em geral pode ser examinada de múltiplas perspectivas. Da perspectiva do produto, métodos ágeis são mais adequados quando os requisitos estão emergindo e mudando rapidamente, embora não exista um consenso completo neste ponto (Cohen et al., 2004).

De uma perspectiva organizacional, a aplicabilidade pode ser expressa examinando três dimensões chaves da organização: cultura, pessoal e comunicação. Em relação a estas áreas inúmeros fatores chave do sucesso podem ser identificados (Cohen et al., 2004):

- A cultura da organização deve apoiar a negociação.
- As pessoas devem ser confiantes.
- Poucas pessoas, mas competentes.
- A organização deve promover as decisões que os desenvolvedores tomam.
- A Organização necessita ter um ambiente que facilite a rápida comunicação entre os membros.

O fator mais importante é provavelmente o tamanho do projeto (Cohen et al., 2004). Com o aumento do tamanho, a comunicação face a face se torna mais difícil. Portanto, métodos ágeis são mais adequados para projetos com pequenos times, com no máximo de 20 a 40 pessoas.

De forma a determinar a aplicabilidade de métodos ágeis específicos, uma análise mais sofisticada é necessária. O método dinâmico para o desenvolvimento de sistemas, por exemplo, provê o denominado 'filtro de aplicabilidade' para este propósito. Também, a família de métodos Crystal provê uma caracterização de quando selecionar o método para um projeto. A seleção é baseada no tamanho do projeto, criticidade e prioridade. Contudo, outros métodos ágeis não fornecem um instrumento explícito para definir sua aplicabilidade a um projeto.

Alguns métodos ágeis, como DSDM e Feature Driven Development, afirmam se aplicar a qualquer projeto de desenvolvimento ágil, sem importar suas características (Abrahamsonn et al., 2003).

A comparação dos métodos ágeis irá revelar que eles suportam diferentes fases de um ciclo de vida do software em diferentes níveis. Estas características individuais dos métodos ágeis podem ser usadas como um critério de seleção de sua aplicabilidade.

Desenvolvimentos ágeis vêm sendo amplamente documentados (ver Experiências relatadas, abaixo, como também em Beck,[4] e Boehm & Turner[5]) como funcionando bem para equipes pequenas (< 10 desenvolvedores). O desenvolvimento ágil é particularmente adequado para equipes que têm que lidar com mudanças rápidas ou imprevisíveis nos requisitos.

A aplicabilidade do desenvolvimento ágil para os seguintes cenários é ainda uma questão aberta:

- esforços de desenvolvimento em larga escala (> 20 desenvolvedores), embora estratégias para maiores escalas tenham sido descritas.[6]
- esforços de desenvolvimento distribuído (equipes não co-alocadas). Estas estratégias tem sido descritas em *Bridging the Distance*[7] e *Using an Agile Software Process with Offshore Development*[8]
- esforços críticos de missão e vida.
- *Companhias* com uma cultura de comando e controle.

Barry Boehm e Richard Turner sugeriram que análise de risco pode ser usada para escolher entre métodos adaptativos ("ágeis") e preditivos ("dirigidos pelo planejamento"). Os autores sugerem que cada lado deste contínuo possui seu *ambiente ideal".*

Ambiente ideal para o desenvolvimento ágil:

- Baixa criticidade
- Desenvolvedores senior
- Mudanças freqüente de requisitos
- Pequeno número de desenvolvedores
- Cultura que tem sucesso no caos.
- Ambiente ideal para o desenvolvimento direcionado ao planejamento:
- Alta criticidade
- Desenvolvedores Junior
- Baixa mudança nos requisitos
- Grande número de desenvolvedores
- Cultura que procura a ordem.

Adaptabilidade dos métodos ágeis

Um método deve ser bastante flexível para permitir ajustes durante a execução do projeto. Há três problemas chaves relacionados ao tópico de adaptação dos métodos ágeis: **a aplicabilidade dos métodos ágeis** (no geral e no particular), e finalmente, o **suporte ao gerenciamento de projeto**.

Métodos ágeis e o gerenciamento de projeto

Os métodos ágeis diferem largamente no que diz respeito a forma de serem gerenciados. Alguns métodos são suplementados com guias para direcionar o gerenciamento do projeto, mas nem todos são aplicáveis.

PRINCE2™ tem sido considerado como um sistema de gerenciamento de projeto complementar e adequado.

Uma característica comum dos processos ágeis é a capacidade de funcionar em ambientes muito exigentes que tem um grande número de incertezas e flutuações (mudanças) que podem vir de várias fontes como: equipe em processo de formação que ainda não trabalhou junto em outros projetos, requisitos voláteis, baixo conhecimento do domínio de negócio pela equipe, adoção de novas tecnologias, novas ferramentas, mudanças muito bruscas e rápidas no ambiente de negócios das empresas: novos concorrentes, novos produtos, novos modelos de negócio.

Sistemas de gerenciamento de projetos lineares e prescritivos, neste tipo de ambiente, falham em oferecer as características necessárias para responder de forma ágil as mudanças requeridas. Sua adoção pode incrementar desnecessariamente os riscos, o custo, o prazo e baixar a qualidade do produto gerado, desgastando a equipe e todos os envolvidos no processo.

A abordagem Scrum, para gestão de projetos ágeis, leva em consideração planejamento não linear, porém de maneira mais exaustiva e está focada em agregar valor para o cliente e em gerenciar os riscos, fornecendo um ambiente seguro. Pode ser utilizada na gestão do projeto aliada a uma metodologia de desenvolvimento como Programação Extrema, FDD, OpenUP, DSDM, Crystal ou outras.

Metodologias

Programação extrema

Scrum

Feature Driven Development

DSDM

Adaptive Software Development

Crystal

Pragmatic Programming

Test Driven Development

Críticas

O método de desenvolvimento ágil é algumas vezes criticado como codificação cowboy. O início da Programação extrema soava como controverso e dogmático, tal como a programação por pares e o projeto contínuo, tem sido alvo particular de críticos, tais como McBreen e Boehm e Turner. Contudo, muitas destas críticas têm sido vistas pelos defensores dos métodos ágeis como mal entendidos a respeito do desenvolvimento ágil.

Em particular, a Programação extrema é revista e criticada por Matt Stephens' Extreme Programming Refactored.

As críticas incluem:

- falta de estrutura e documentação necessárias
- somente trabalhar com desenvolvedores de nível sênior
- incorpora de forma insuficiente o projeto de software
- requer a adoção de muita mudança cultural
- poder levar a maiores dificuldades nas negociações contratuais

3.9 Rapid Application Development

Rapid Application Development (RAD) ou Desenvolvimento Rápido de Aplicação

(em português), é um modelo de processo de desenvolvimento de software iterativo e incremental que enfatiza um ciclo de desenvolvimento extremamente curto (entre 60 e 90 dias).

O termo foi registrado por James Martin em 1991 e tem substituído gradativamente o termo de prototipação rápida que já foi muito utilizada no passado.

Histórico

Os modelos de processo de software apresentados durante a década de 70, cujo o modelo em cascata é um bom representante, possuíam longos períodos de desenvolvimento e muitas vezes os requisitos do sistema se alteravam antes do fim do processo. Os desenvolvedores de software necessitavam de um modelo mais ágil que permitisse um tempo de desenvolvimento mais curto e a mudança dos requisitos durante o processo.

Nos anos 80 os trabalhos de Barry Boehm (modelo de processo em espiral) e Tom Gilb (modelo de processo evolucionário) serviram de base para uma metodologia chamada de Rapid Iterative Production Prototyping (RIPP) criada por Scott Shultz. James Martin estendeu o RIPP agregando valores de outros processos tornando-o maior e mais formal sendo assim denominado de **RAD**. O RAD foi finalmente formalizado em 1991 com a publicação de um livro.

O Processo

O número de fases do processo varia de acordo com os autores.

Segundo Kerr, o processo se divide em 5 fases:

Modelagem de Negócio

O fluxo de informações entre as funções de negócio é modelado de modo a responder às seguintes questões: - Que informação direciona o processo de negócio? - Que informação é gerada? - Quem a gera? - Para onde vai à informação? - Quem a processa? Na modelagem de negócio são levantados os processos suportados pelo sistema.

Modelagem dos dados

A modelagem de dados responde a um conjunto de questões específicas que são relevantes a qualquer aplicação. O fluxo de informação definido na fase de modelagem de negócio refinado e de forma a extrair os principais objetos de dados a serem processados pelo sistema, qual a composição de cada um dos objetos de dados, onde costumam ficar, qual a relação entre eles e quais as relações entre os objetos e os processos que os transformam.

Modelagem do Processo

Os objetos de dados definidos na modelagem de dados são transformados para conseguir o fluxo necessário para implementar uma função do negócio. Descrições do processamento são criadas para adicionar, modificar, descartar ou recuperar um objeto de dados.

Geração da Aplicação

O RAD considera o uso de técnicas de quarta geração, trabalha com a reutilização de componentes de programa existentes quando possível, ou cria componentes reusáveis. São usadas ferramentas automatizadas para facilitar a construção do software.

Ex: Clarion, Delphi, Visual Basic, Asp.net, etc.

Teste e Modificação

Como o processo do RAD enfatiza o reuso, muitos componentes já estão testados, isso reduz o tempo total de teste. Todavia os novos componentes devem ser testados e todas as interfaces devem ser exaustivamente exercitadas.

Esta divisão do processo é compartilhada por diversos autores inclusive Roger S. Pressman, cuja obra é utilizada em diversas faculdades como livro guia para os estudantes.Porém existem outras abordagens utilizadas.

Segundo Stephen E. Cross Diretor do SEI - Software Engineering Institute da Carneggie Mellon, uma maneira de abordar o RAD de forma mais eficiente é dividi-lo em 6 passos:

- Projeto e análise baseado no cenário
- Projeto e análise de Arquitetura
- Especificação de Componentes com o máximo de reuso
- Desenvolvimento rápido dos módulos remanescentes
- Testes freqüentes com o usuário final
- Campo com ferramentas de suporte para permitir a evolução

A proposta de Stephen é disciplinar o RAD, que é muitas vezes criticado por sua suposta informalidade, de forma a conseguir até mesmo níveis de CMM - Capability Maturity Model para melhorar e formalizar ainda mais o processo.

Vantagens

- Permite o desenvolvimento rápido e/ou a prototipagem de aplicações;
- Enfatiza um ciclo de desenvolvimento extremamente curto (entre 60 e 90 dias);
- Cada função principal pode ser direcionada para a uma equipe RAD separada e então integrada a formar um todo;
- Criação e reutilização de componentes;
- Usado principalmente para aplicações de sistemas de informações;
- Comprar pode economizar recursos se comparado a desenvolver;
- Desenvolvimento é conduzido em um nível mais alto de abstração;
- Visibilidade mais cedo (protótipos);
- Maior flexibilidade (desenvolvedores podem reprojetar praticamente a vontade);
- Grande redução de codificação manual (wizards…);

- Envolvimento maior do usuário;
- Provável custo reduzido(tempo é dinheiro e também devido ao reuso);

- Aparência padronizada (As APIs ae outros componentes reutilizáveis permitem uma aparência consistente).

O RAD é apropriado quando

- A aplicação é do tipo "stand alone";
- Pode-se fazer uso de classes pré-existentes (APIs);
- A performance não é o mais importante;
- A distribuição do produto é pequena;
- O escopo do projeto é restrito;
- O sistema pode ser dividido em vários módulos independentes;
- A tecnologia necessária tem mais de um ano de existência.

Desvantagens

- Se uma aplicação não puder ser modularizada de modo que cada função principal seja completada em menos de 3 meses, não é aconselhável o uso do RAD;
- Para projetos grandes (mas escaláveis) o RAD exige recursos humanos suficientes para criar o número correto de equipes, isso implica um alto custo com a equipe;
- O envolvimento com o usuário tem que ser ativo;
- Comprometimento da equipe do projeto;
- O RAD não é aconselhável quando os riscos técnicos são altos e não é indicada quando se está testando novas tecnologias ou quando o novo software exige alto grau de interoperabilidade com programas de computador existentes. Falta de prazo pode implicar qualidade reduzida, e

há necessidade de habilidade maior dos desenvolvedores, e suporte maior da gerência e dos clientes.

- Desenvolver pode economizar recursos se comparado a comprar;
- Custo do conjunto de ferramentas e hardware para rodar a aplicação;
- Mais difícil de acompanhar o projeto(pois não existe os marcos clássicos);
- Menos eficientes;

- Perda de precisão científica (falta de métodos formais);
- Pode acidentalmente levar ao retorno das práticas caóticas no desenvolvimento;
- Funções reduzidas (reuso, "timeboxing");
- Funções desnecessárias (reuso de componentes);
- Problemas legais;
- Requisitos podem não se encaixar (conflitos entre desenvolvedores e clientes)
- Padronização (aparência diferente entre os módulos e componentes)
- Sucessos anteriores são difíceis de se reproduzir
-

O RAD deve ser evitado quando

- A aplicação precisa interagir com outros programas;
- Existem poucos plugins e componentes disponíveis;
- Performance é essencial;
- O desenvolvimento não pode tirar vantagem de ferramentas de alto nível;
- A distribuição do produto será em grande escala;
- Para se construir sistemas operacionais (confiabilidade exigida alta demais)
- Jogos de computador (performance exigida muito alta)
- Riscos tecnológicos muito altos devido a tecnologia ter sido recém lançada;
- O sistema não pode ser modularizado

Modelos de maturidade

Os modelos de maturidade são um metamodelo de processo. Eles surgiram para avaliar a qualidade dos processos de *software* aplicados em uma organização (empresa ou instituição). O mais conhecido é o *Capability Maturity Model Integration* (**CMMi**), do Software Engineering Institute – SEI.

O CMMi pode ser organizado através de duas formas: Contínua e estagiada. Pelo modelo estagiado, mais tradicional e mantendo compatibilidade com o CMM, uma organização pode ter sua maturidade medida em 5 níveis:

- Nível 1 - Caótico;
- Nível 2 - Capacidade de repetir sucessos anteriores pelo acompanhamento de custos, cronogramas e funcionalidades;
- Nível 3 - O processo de software é bem definido, documentado e padronizado;
- Nível 4 - Realiza uma gerência quantitativa do processo de software e do produto;
- Nível 5 - Usa a informação quantitativa para melhorar continuamente e gerenciar o processo de software.

O CMMi é um modelo de maturidade recentemente criado com o fim de agrupar as diferentes formas de utilização que foram dadas ao seu predecessor, o CMM.

O (**MPS.BR**), ou Melhoria de Processos do Software Brasileiro, é simultaneamente um movimento para a melhoria e um modelo de qualidade de processo voltada para a realidade do mercado de pequenas e médias empresas de desenvolvimento de software no Brasil.

CMMI

O **CMMI** (**Capability Maturity Model Integration**) é um modelo de referência que contém práticas (*Genéricas ou Específicas*) necessárias à maturidade em disciplinas específicas: (*Systems Engineering (SE), Software Engineering (SW), Integrated Product and Process Development (IPPD), Supplier Sourcing (SS*)). Desenvolvido pelo SEI (*Software Engineering Institute*) da Universidade Carnegie Mellon, o CMMI é uma evolução do CMM e procura estabelecer um modelo único para o processo de melhoria corporativo, integrando diferentes modelos e disciplinas.

O CMMI foi baseado nas melhores práticas para desenvolvimento e manutenção de produtos. Há uma ênfase tanto em engenharia de sistemas quanto em engenharia de software, e há uma integração necessária para o desenvolvimento e a manutenção.

A versão atual do CMMI (versão 1.3) foi publicada em 27 de outubro de 2010 e apresenta três modelos:

- *CMMI for Development* (CMMI-DEV), voltado ao processo de desenvolvimento de produtos e serviços.
- *CMMI for Acquisition* (CMMI-ACQ), voltado aos processos de aquisição e terceirização de bens e serviços.
- *CMMI for Services* (CMMI-SVC), voltado aos processos de empresas *prestadoras* de serviços.

Uma das premissas do modelo é "A qualidade é influenciada pelo processo", e seu foco é "Melhorar processo de uma empresa".

Histórico

Os processos de melhoria nasceram de estudos realizados por Deming (Out of the Crisis), Crosby (Quality is Free: The Art of Making Quality Certain) e Juran, cujo objetivo principal era a melhoria da **capacidade** dos processos. Entende-se por **capacidade** de um processo a habilidade com que este alcança o resultado desejado.

Um **modelo** tem como objetivo estabelecer - com base em estudos, históricos e conhecimento operacional - um conjunto de "melhores práticas" que devem ser utilizadas para um fim específico.

O CMMI tem como origens em três outros modelos de maturidade - SW-CMM (*SEI Software CMM*), EIA SECM (*Electronic Industries Alliances's Systems Engineer Capability Model*) e IPD-CMM (*Integrated Product Development CMM*).

Dimensões

O CMMI foi construído considerando três dimensões principais: pessoas, ferramentas e procedimentos. O processo serve para unir essas dimensões.

Disciplinas

O processo inclui três disciplinas ou corpos de conhecimento (*body of knowledges*), sendo elas:

- Engenharia de sistemas
- Engenharia de software
- Engenharia de hardware

A engenharia de software é similar à engenharia de sistemas em relação às áreas de processo, apenas com enfoque diferente nos processos. As áreas de processo requeridas para engenharia de sistemas são as mesmas para engenharia de software, mas o nível de maturidade que é diferente.

Representações

O CMMI possui duas representações: "contínua" ou "por estágios". Estas representações permitem à organização utilizar diferentes caminhos para a melhoria de acordo com seu interesse.

Representação Continua

Possibilita à organização utilizar a ordem de melhoria que melhor atende os objetivos de negócio da empresa. É caracterizado por Níveis de Capacidade (*Capability Levels*):

Nível 0: Incompleto (Ad-hoc)

Nível 1: Executado

Nível 2: Gerenciado / Gerido

Nível 3: Definido

Nesta representação a capacidade é medida por processos separadamente, onde é possível ter um processo com nível um e outro processo com nível cinco, variando de acordo com os interesses da empresa.

- No nível 1(um) o processo é executado de modo a completar o trabalho necessário para a execução de um processo. - No nível 2(dois) é sobre planejar a execução e confrontar o executado contra o que foi planejado. - No nível 3(três) o processo é construído sobre as diretrizes do processo existente, e é mantido uma descrição do processo. - No nível 4(quatro) é quando o processo é gerenciado quantitativamente através de estatísticas e outras técnicas. - No nível 5(cinco) o processo gerido quantitativamente é alterado e adaptado para atender às necessidades negociais/estratégicas da empresa.

A representação contínua é indicada quando a empresa deseja tornar apenas alguns processos mais maduros, quando já utiliza algum modelo de maturidade contínua ou quando não pretende usar a maturidade alcançada como modelo de comparação com outras empresas.

Representação Por Estágios

Disponibiliza uma seqüência pré-determinada para melhoria baseada em estágios que não deve ser desconsiderada, pois cada estágio serve de base para o próximo. É caracterizado por Níveis de Maturidade (*Maturity Levels*):

Nível 1: Inicial (Ad-hoc)

Nível 2: Gerenciado / Gerido

Nível 3: Definido

Nível 4: Quantitativamente gerenciado / Gerido quantitativamente

Nível 5: Em otimização

Nesta representação a maturidade é medida por um conjunto de processos. Assim é necessário que todos os processos atinjam nível de maturidade dois para que a empresa seja certificada com nível dois. Se quase todos os processos forem nível três, mas apenas um deles estiver no nível dois a empresa não irá conseguir obter o nível de maturidade três.

Esta representação é indicada quando a empresa já utiliza algum modelo de maturidade por estágios, quando deseja utilizar o nível de maturidade alcançado para comparação com outras empresas ou quando pretende usar o nível de conhecimento obtido por outros para sua área de atuação.

Áreas de Processo

O modelo CMMI v1.2 (CMMI-DEV) contém 22 áreas de processo. Em sua representação por estágios, as áreas são divididas da seguinte forma:

Nível 1: Inicial (Ad-hoc)

Não possui áreas de processo.

Nível 2: Gerenciado / Gerido

Gerenciamento de Requisitos - REQM (Requirements Management)

Planejamento de Projeto - PP (Project Planning)

Acompanhamento e Controle de Projeto - PMC (Project Monitoring and Control)

Gerenciamento de Acordo com Fornecedor - SAM (Supplier Agreement Management)

Medição e Análise - MA (Measurement and Analysis)

Garantia da Qualidade de Processo e Produto - PPQA (Process and Product Quality Assurance)

Gerência de Configuração - CM (Configuration Management)

Nível 3: Definido

Desenvolvimento de Requisitos - RD (Requirements Development)

Solução Técnica - TS (Technical Solution)

Integração de Produto - PI (Product Integration)

Verificação - VER (Verification)

Validação - VAL (Validation)

Foco de Processo Organizacional - OPF (Organizational Process Focus)

Definição de Processo Organizacional - OPD (Organizational Process Definition)

Treinamento Organizacional - OT (Organizational Training)

Gerenciamento Integrado de Projeto - IPM (Integrated Project Management)

Gerenciamento de Riscos - RSKM (Risk Management)

Análise de Decisão e Resolução - DAR (Decision Analysis and Resolution)

Nível 4: Quantitativamente gerenciado / Gerido quantitativamente

Desempenho de Processo Organizacional - OPP (Organizational Process Performance)

Gerenciamento Quantitativo de Projeto - QPM (Quantitative Project Management)

Nível 5: Em otimização

Gestão de Processo Organizacional - OPM (Organizational Process Management)

Análise Causal e Resolução - CAR (Causal Analysis and Resolution)

Modelos e áreas de processo

As áreas de processo variam com base no modelo escolhido, não sendo as mesmas áreas para todos os modelos (CMMI-DEV, CMMI-ACQ ou CMMI-SVC).

ISO/IEC 15504

A ISO/IEC_15504, também conhecida como SPICE, define um "processo para relatórios técnicos no assessoramento em desenvolvimento de software", e similarmente ao CMMI possui níveis de maturidade para cada processo. O CMMI não é baseado nesta norma, mas sim compatível.

Histórico de Avaliações

Até o ano de 2002, os EUA tinham realizado 1,5 mil avaliações de CMMI, a Índia feito 153, o Reino Unido 103 e o Brasil apenas 15. Em 2004 a TATA Consultancy Services (empresa indiana) alcançou o nível 5 em todas as unidades da empresa, tendo sido avaliada inclusive a unidade brasileira (a primeira empresa presente no Brasil a receber o nível máximo na avaliação).

Entre abril de 2002 e junho de 2006 foram conduzidas 1581 avaliações em 1377 organizações. Segue abaixo o resultado obtido pelas empresas na avaliação (resultados encaminhados para o SEI até 30 de junho de 2006):

18,2%: nível 5 (*Optimizing*);

4,4%: nível 4 (*Quantitatively Managed*);

33,8%: nível 3 (*Defined*);

33,3%: nível 2 (*Managed*);

1,9%: nível 1 (*Initial*);

8,4%: sem qualificação (*Not Given*).

Melhoria de Processos do Software Brasileiro

(Redirecionado de **MPS.BR**)

O custo de uma certificação para uma empresa pode ser de até US$ 400 mil, o que se torna inviável para empresas de micro, pequeno e médio porte. Então, em uma parceria entre a Softex, Governo e Universidades, surgiu o projeto MPS.Br (melhoria de processo de software brasileiro), que é a solução brasileira compatível com o modelo CMMI, está em conformidade com as normas ISO/IEC 12207 e 15504, além de ser adequado à realidade brasileira.

O modelo

O MPS.Br é dividido em 3 partes: MR-MPS, MA-MPS, MN-MPS

MR-MPS (Modelo de referência para melhoria do processo de software)

O **MR-MPS** apresenta 7 níveis de maturidade (o que é um diferencial em relação aos outros padrões de processo) que são:

A - Em Otimização;

B - Gerenciado Quantitativamente;

C - Definido;

D - Largamente Definido;

E - Parcialmente Definido;

F - Gerenciado;

G - Parcialmente Gerenciado.

Cada nível de maturidade possui suas áreas de processo, onde são analisados:

Os processos fundamentais:

- aquisição
- gerência de requisitos
- desenvolvimento de requisitos
- solução técnica

- integração do produto
- instalação do produto
- liberação do produto

Os processos organizacionais:

- gerência de projeto
- adaptação do processo para gerência de projeto
- análise de decisão e resolução
- gerência de riscos
- avaliação e melhoria do processo organizacional
- definição do processo organizacional,
- desempenho do processo organizacional
- gerência quantitativa do projeto
- análise e resolução de causas
- inovação e implantação na organização

Os processos de apoio:

- garantia de qualidade
- gerência de configuração
- validação
- medição
- verificação

– treinamento

Em seguida vem a Capacidade, onde são obtidos os resultados dos processos analisados, onde cada nível de maturação possui um número definido de capacidades a serem vistos.

AP 1.1 - O processo é executado;

AP 2.1 - O processo é gerenciado;

AP 2.2 - Os produtos de trabalho do processo são gerenciados;

AP 3.1 - O processo é definido;

AP 3.2 - O processo está implementado;

AP 4.1 - O processo é medido;

AP 4.2 - O processo é controlado;

AP 5.1 - O processo é objeto de inovações;

AP 5.2 - O processo é otimizado continuamente.

MA-MPS (Método de avaliação para melhoria do processo de software)

Tem como objetivo orientar a realização de avaliações, em conformidade com a norma ISO/IEC 15504, em empresa e organizações que implementaram o MR-MPS. Avaliação MA-MPS:

Equipe de avaliação: 3 a 8 pessoas, sendo:

- 1 avaliador líder
- no mínimo 1 avaliador adjunto
- no mínimo 1 técnico da empresa

Duração: 2 a 4 dias;

Validade: 3 anos;

Estruturação da Avaliação:

- Planejar e preparar avaliação
- Plano de Avaliação / Descrição dos indicadores de processo;
- Conduzir Avaliação
- Resultado da avaliação;
- Relatar resultados
- Relatório da avaliação;
- Registrar e publicar resultados

Banco de dados Softex (Ver portal MPS.BR nas 'Ligações Externas')

MPS.BR

MN-MPS (Modelo de negócio para melhoria do processo de software)

Instituições que se propõem a implantar os processos MPS.Br (Instituições Implementadoras) podem se credenciar através de um documento onde é apresentada a instituição proponente, contendo seus dados com ênfase na experiência em processos de software, estratégia de implementação do modelo, estratégia para seleção e treinamento de consultores para implementação do MR.MPS, estratégia para seleção e treinamento de avaliadores, lista de consultores de implementação treinados no modelo e aprovados em prova específica, lista de avaliadores treinados no modelo e aprovados em prova específica.

Cursos e certificação

A Softex realiza cursos para formação de consultores, compradores e avaliadores MPS.BR. São ao todo 4 cursos:

- Curso de Introdução - C1
- Curso de Implementação - C2
- Curso de Avaliaçao - C3
- Curso de Aquisição - C4

Periodicamente, são realizadas provas em nível nacional para certificar profissionais em cada um dos cursos descritos acima. Tanto os cursos e as provas são realizadas nos Agentes SOFTEX em cada estado.

Próximos passos

O modelo MPS.Br tem como objetivo implementar o Modelo de Referência para melhoria de processo de software em 120 empresas. E como objetivos secundários, a disseminação em diversos locais do país, capacitação no uso do modelo e o credenciamento de instituições implementadoras e avaliadoras do modelo, especialmente instituições de ensino e centros tecnológicos e também a implementação e avaliação do modelo com foco em grupos de empresas. A avaliação conjunta de grupos empresariais, objetiva a redução dos custos, porém há uma perda de foco, pois não há uma especificidade para cada empresa e sim um mesmo modelo de referência para todas elas. O MPS.Br já é uma realidade, e dentro de alguns anos, existe um projeto de implantação em seis países da América Latina, são eles: Chile, Argentina, Costa Rica, Peru, Uruguai e Cuba.

Fernanda Fernandes Ministério

Diretora de Produtos e Treinamentos

BlueStar Ensino e Tecnologia

www.bluestar.inf.br

61-3347-9255 e 8153-8888.

Acesse nosso Facebook e acompanhe on line as melhores vagas de tecnologia do mercado!!! Bluestar – Ensino e Tecnologia.