TEXTO-BASE

Desenvolvimento de sistemas de informação

PROJETOS

Pode ser definido como um esforço para se atingir um objetivo específico por meio de um conjunto único de tarefas inter-relacionadas e da utilização eficaz dos recursos. É um empreendimento finito, que tem objetivo claramente definido em função de um problema, oportunidade ou interesse de uma pessoa ou organização.

O resultado do projeto é o desenvolvimento da solução ou o atendimento do interesse, dentro de restrições de tempo e recursos. Para definir o grau de sucesso do resultado do projeto, é preciso verificar se esses critérios ou requisitos foram atendidos. Não alcançar o objetivo, não realizá-lo dentro do prazo previsto ou consumir recursos além do orçamento significa comprometer dimensões importantes do desempenho esperado.

A função do gerente de projetos (Links para um site externo)Links para um site externo, que pode ser qualquer profissional qualificado e habilitado, é agir como agente de integração entre os processos e as pessoas envolvidas em determinado projeto, atuando na definição de alocação de recursos e esforços e na gestão de conflitos, visando sempre a excelência do projeto em que está alocado.

A integração inclui características de consolidação, comunicação, unificação e inter-relacionamentos, incluindo tomadas de decisão sobre alocação de recursos, balanceamento de demandas concorrentes, exame das abordagens alternativas, adaptação dos processos à realidade do projeto e gerenciamento da interdependência entre as diversas áreas do conhecimento, citados a seguir:

- Integração;
- Escopo;
- Custos;
- Qualidade;
- Aquisições;
- Recursos;
- Comunicações;
- Risco;
- Cronograma;
- Partes interessadas.

IMPLEMENTAÇÃO DOS SISTEMAS DE INFORMAÇÃO

A implementação de um sistema de informação consiste em todas as etapas necessárias para a entrega de um produto de gestão da informação. Essas etapas são divididas em três grupos, identificados como: entradas de projeto, atividades de projeto e saídas de projetos.

- A entrada de projeto se refere ao levantamento de informações e requisitos importantes para o desenvolvimento dos sistemas de informação. As principais atividades são informação de plataforma, especificação de requisitos (funcionais e não funcionais) e a descrição de dados que serão armazenados para posteriormente serem processados.
- As atividades de projeto envolvem os projetos de arquitetura do sistema, interface de comunicação entre o usuário e o sistema, assim como entre os próprios componentes do sistema, os quais o integrarão para dar suporte aos processos tecnológicos e banco de dados (em que serão armazenados os dados).

Como saída de projeto, temos a arquitetura do sistema pronta ou parcialmente pronta, dependendo da fase, a especificação lógica e física do banco de dados, de interface para validação e dos componentes integrados e em produção (já gerando informação).

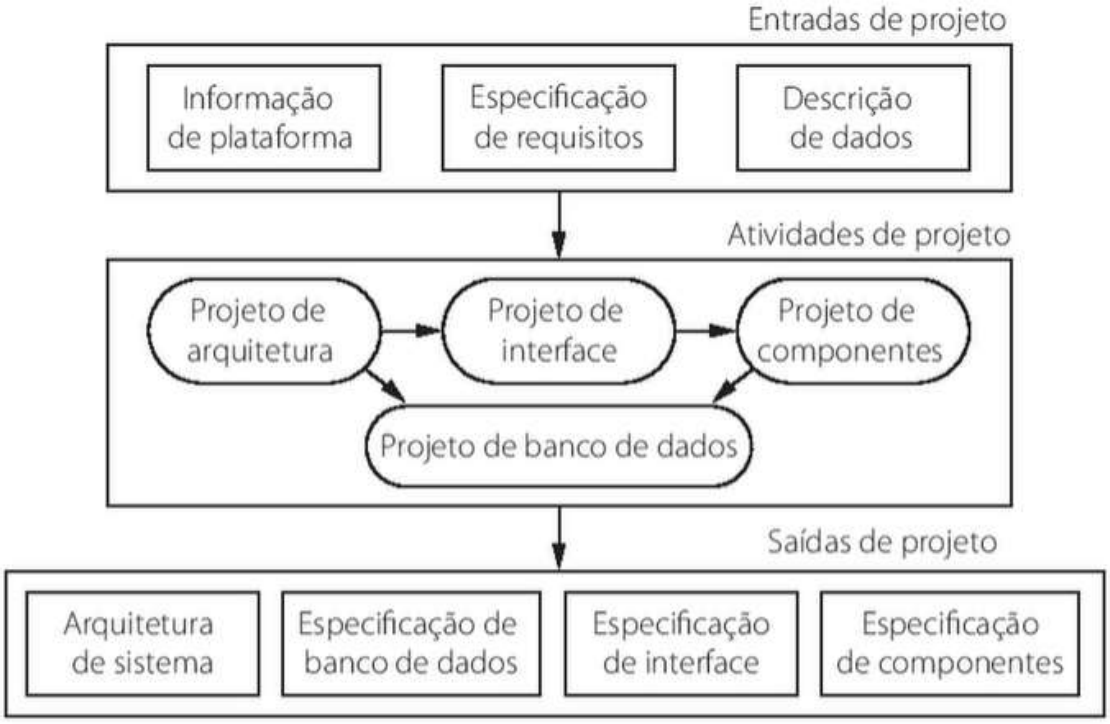

Figura 1 – Atividades de implementação do sistema de informação

MODELAGEM CONCEITUAL DOS SISTEMAS DE INFORMAÇÃO

Trata-se da tarefa mais importante de um processo de desenvolvimento do sistema de informação. Realiza-se a análise do domínio da aplicação e a modelagem das entidades e fenômenos desse domínio que o projetista considera importante, independentemente da implementação. A tarefa de modelagem conceitual envolve dois mecanismos: abstração e representação. Quando se pensa através de modelos, estamos em condições de colocar em prática a abordagem sistêmica.

VISÃO DOS SISTEMAS DE INFORMAÇÃO

Gerar uma visão daquilo que se vai construir é essencial para o sucesso do projeto de sistema de informação. Utilizar uma linguagem que forneça meios para auxiliar no levantamento dos requisitos que irão constituir um sistema, além de recursos para a modelagem de estruturas que farão parte do projeto, é uma alternativa indispensável para atingir os objetivos de forma eficaz.

O conhecimento do ambiente organizacional é um elemento importante para a compreensão dos problemas e das decisões organizacionais. É a partir dessa visão sistêmica que se podem entender e estudar o funcionamento das empresas.

Cada vez mais, a aplicação de novas tecnologias, um recurso estratégico, precisa ser cuidadosamente planejada, organizada e controlada pelas empresas. O planejamento tecnológico é uma das atividades mais importantes para a criação, sustentação e maximização da vantagem competitiva.

ORIENTAÇÃO A OBJETO (OO)

Orientação a objeto é um conceito que está relacionado com a ideia de classificar, organizar e abstrair coisas. Significa identificar grupos e organizar o mundo real, através de sua visão, como uma coleção de objetos que incorporam estrutura de dados e um conjunto de operações e processos que manipulam esses dados. É uma forma de desenvolvimento de sistemas de informação que trata atividades ou processos organizacionais como um conjunto de componentes que interagem entre si para resolver um problema, por isso chama-se orientação a objeto.

LINGUAGEM UNIFICADA DE MODELAGEM (UML)

UML é um acrônimo para a expressão *unified modeling language*. Pela definição de seu nome, vemos que a UML é uma linguagem que define uma série de artefatos que nos ajudam na tarefa de modelar e documentar os sistemas orientados a objetos que desenvolvemos. Ela possuiu diversos tipos de diagramas (no momento 14),

divididos em duas grandes categorias: estruturais (7 diagramas) e comportamentais (7 diagramas). Sete tipos de diagramas representam informações estruturais, e os outros sete representam tipos gerais de comportamento, incluindo quatro em uma subcategoria que representam diferentes aspectos de interação.

A maioria dos problemas encontrados em sistemas orientados a objetos tem sua origem na construção do modelo, no desenho ou projeto do sistema. Muitas vezes as empresas e profissionais não dão muita ênfase a essa fase do projeto, e acabam cometendo diversos erros de análise e modelagem. Isso quando há modelagem, pois, na maioria das vezes, o projeto começa já na fase de codificação, baseado num pequeno e frágil levantamento de requisitos iniciais.

Esses diagramas propostos documentam o que o sistema faz do ponto de vista do usuário. Ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema. Nesses diagramas, não nos aprofundamos em detalhes técnicos que dizem como o sistema faz. Esses artefatos gerados são comumente derivados das especificações de requisitos.

Um exemplo desses diagramas – o mais comum entre eles e, geralmente, o primeiro a ser gerado em função da sua aplicação no momento da validação – é o *diagrama de caso de uso*. Esse diagrama é composto basicamente por quatro partes:

- *Cenário* – sequência de eventos que acontecem quando um usuário interage com o sistema;
- *Ator* – usuário do sistema, ou melhor, um tipo de usuário;
- *Use case* – é uma tarefa ou uma funcionalidade realizada pelo ator (usuário);
- *Comunicação* – é o que liga um ator com um caso de uso.

Vamos criar um cenário de exemplo para vermos a notação de um diagrama de caso de uso:

- “Um cliente necessita efetuar algumas operações bancárias. Ele escolhe alguma agência na rede bancária em que tem conta, entra e se dirige a um terminal de consulta e operações. O usuário entra com seus dados, incluindo a senha, e o banco (sistema) lhe dispõe de algumas operações que podem ser: efetuar saque, emitir extrato ou informar saldo”.

Com esse cenário simples podemos começar a criar nosso diagrama. Inicialmente vamos definir nossos atores:

- Cliente;
- Banco.

Agora vamos definir algumas ações de cada usuário:

- Cliente
    - Efetuar saque;
    - Emitir extrato.
- Banco
    - Informar saldo.

Bom, agora já temos uma relação de atores e ações relacionadas a esses atores. Poderíamos criar um documento textual (como foi feito anteriormente), para registrar nossos atores e funcionalidades. Entretanto podemos melhorar nossa visão e expressar tudo o que definimos em um desenho simples utilizando os padrões da UML para documentação de casos de uso.

Veja no quadro a seguir a definição de algumas figuras do diagrama:

Figura 2 – Componentes do diagrama do caso de uso

No mercado, existem diversos tipos de ferramentas *case* que auxiliam na construção de diagramas. Fique à vontade para utilizar a ferramenta de sua preferência. Podemos agora construir o diagrama:

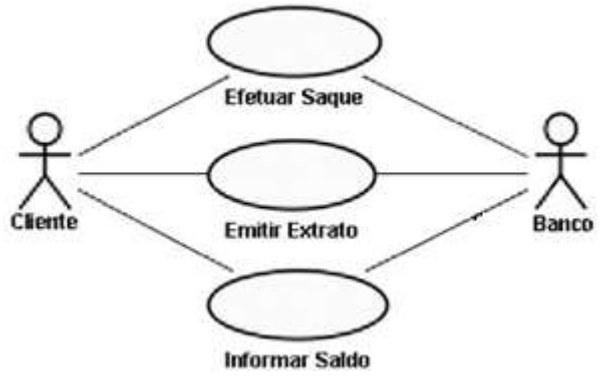

Figura 3 – Exemplo de diagrama de caso de uso da UML

Como podemos observar, esse diagrama composto por desenhos simples descreve, de maneira bem objetiva, o que textualmente poderia ficar extenso. Nele vemos as funcionalidades do sistema e as interações dos usuários com elas. Nesse caso de uso, cliente e banco interagem com todas ações.

O exemplo dado é de um diagrama estático, ou seja, não traz a funcionalidade com a narração ou sequência do processo, mas apenas uma função encapsulada. Já na figura a seguir, temos outro diagrama, só que dinâmico, em que podemos observar o processo como realmente ocorre. Esse diagrama, que apresenta uma criação de conta, é chamado de diagrama de atividades.

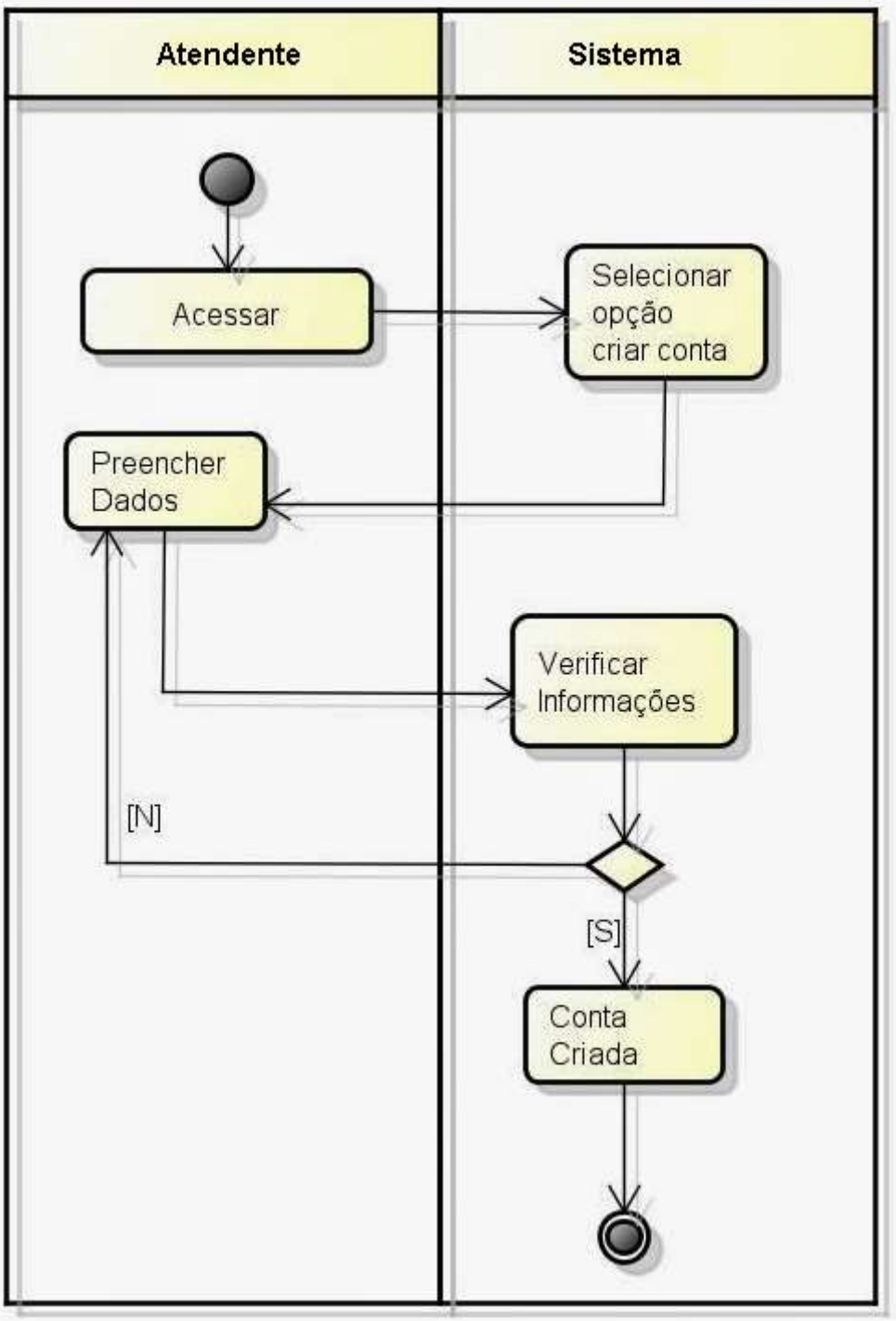

Figura 4 – Exemplo de diagrama de atividades da UML

PROJETO DE SISTEMAS ORIENTADO A OBJETO (OOP)

Os projetos lógicos e físicos de um sistema de informação podem ser executados com a abordagem tradicional de desenvolvimento de sistemas ou através do desenvolvimento de sistemas orientados a objetos (OO). Ambas as abordagens usam uma variedade de modelos de projeto para documentar os recursos do

sistema e a compreensão e concordância da equipe de desenvolvimento. Atualmente, muitas organizações utilizam o desenvolvimento orientado a objetos por causa do aumento de sua flexibilidade.

Usando a abordagem orientada a objetos, pode-se projetar objetos chaves e classes de objetos nos sistemas. Esse processo inclui considerar o domínio do problema, o ambiente operacional e a interface com o usuário como pontos fundamentais para se construir um sistema de informação que retrate fielmente a realidade dos processos. O domínio do problema envolve as classes de objetos relacionadas com a resolução de um problema ou na percepção de uma oportunidade ou expectativa.

MODELOS DE PROCESSOS DE SOFTWARE

Um processo de software é um conjunto de atividades, métodos, práticas e transformações que orientam profissionais na produção de software. Um processo eficaz deve, claramente, considerar as relações entre as atividades, os artefatos produzidos no desenvolvimento, as ferramentas e os procedimentos necessários e a habilidade, o treinamento e a motivação do pessoal envolvido. Os processos de software são, geralmente, decompostos em diversos processos, tais como: processo de desenvolvimento, processo de garantia da qualidade, processo de gerência de projetos etc. Esses processos, por sua vez, são compostos de atividades, que também podem ser decompostas. Para cada atividade de um processo, é importante saber quais as suas subatividades, as atividades que devem precedê-las (pré-atividades), os artefatos de entrada (insumos) e de saída (produtos) da atividade, os recursos necessários (humanos, hardware, software etc.) e os procedimentos (métodos, técnicas, modelos de documento etc.) a serem utilizados na sua realização.

MÉTODOS TRADICIONAIS

O termo *desenvolvimento tradicional* é utilizado para identificar as metodologias que, de alguma forma, adotam o desenvolvimento em cascata, conforme a figura 5. No desenvolvimento em cascata, o software é construído obedecendo uma sequência de fases, sendo que cada fase, com exceção da primeira (análise), depende da conclusão da fase anterior para ser iniciada.

O desenvolvimento em cascata, há tempos, é amplamente aplicado nos processos de desenvolvimento de sistemas de informação, e ainda é o mais empregado. Por esse motivo, é chamado de desenvolvimento tradicional.

Mesmo alguns processos de desenvolvimento de software que, em essência, não são em cascata, como o RUP (*rational unified process*), representado na figura 6, são adotados em grande parte dos projetos, de uma forma que se assemelha muito ao modelo em cascata. Nesses projetos, o processo é empregado de forma iterativa e incremental, mas ainda existe uma forte linearidade no desenvolvimento, caracterizada por cascatas menores dentro de cada iteração.

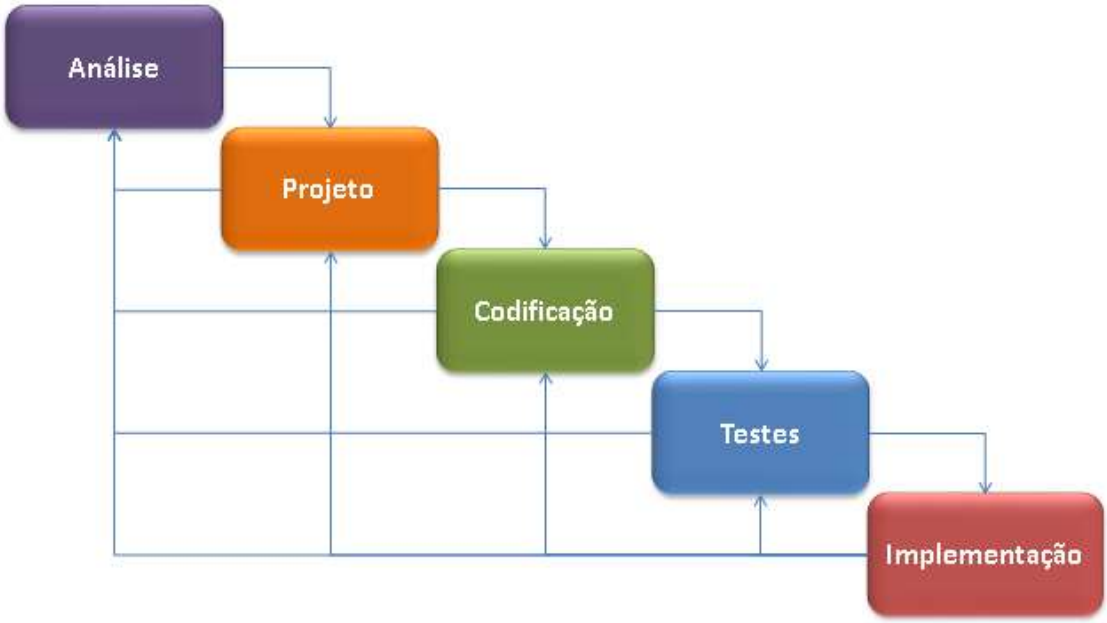

Figura 5 – Processo em cascata

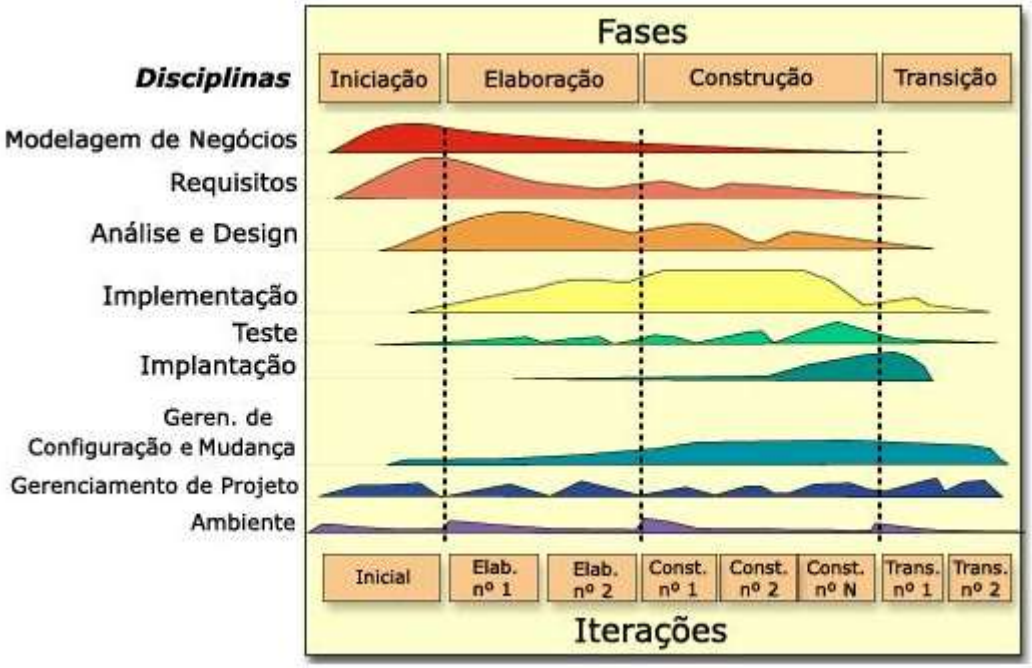

Figura 6 – Processo RUP

PREMISSAS DO DESENVOLVIMENTO TRADICIONAL

Existem algumas premissas básicas, derivadas do modelo de desenvolvimento industrial, que são muito comuns no desenvolvimento tradicional e o influenciam profundamente. Seguem essas premissas e seus significados dentro do contexto do desenvolvimento tradicional de software:

- *Linearidade* – fica evidente pela própria adoção de um modelo em cascata que, como já foi explicado, é linear por definição.
- *Determinismo* – um processo de desenvolvimento, previamente conhecido por gerar determinados resultados com base nas especificações, gerará resultados semelhantes sempre que suas regras forem rigidamente seguidas. O desenvolvimento tradicional, assim como a indústria, persegue o determinismo por acreditar que esta é uma forma de reduzir erros e perdas de tempo.
- *Especialização* – o processo de desenvolvimento pode ser desmembrado em atividades especializadas, que serão executadas de forma independente e depois terão seus resultados integrados para formar o produto final. A especialização torna as tarefas mais simples e, consequentemente, facilita o determinismo. No desenvolvimento de software tradicional, a especialização aparece claramente na rígida divisão de papéis entre membros de uma equipe, como analistas, projetistas, programadores, testadores e implantadores. Cada um deles realiza tarefas bem diferentes e especializadas e terão seus resultados integrados para compor o software finalizado.
- *Foco na execução* – quando existe especialização suficiente para que as tarefas sejam simples e determinísticas, não existe a necessidade, por parte de quem executa o trabalho, de pensar sobre o que se está fazendo: basta fazer. Ou seja, basta que cada pessoa envolvida no processo de desenvolvimento execute corretamente a tarefa de sua responsabilidade, conforme estipulado numa metodologia que presumivelmente torna o processo determinístico, para que a especificação seja transformada corretamente em software.
- *Crescimento exponencial do custo de alteração* – no modelo de desenvolvimento tradicional, em cascata, o custo de uma alteração cresce exponencialmente, à medida que o processo de desenvolvimento avança. A aceitação dessa premissa tem como consequência natural uma busca por processos determinísticos, já que estes prometem menos alterações e maior previsibilidade.

MÉTODOS ÁGEIS

O desenvolvimento ágil é um fruto da constatação feita, de forma independente, por diversos profissionais renomados na área de Engenharia de Software, de que, apesar de terem aprendido segundo os conceitos tradicionais, só conseguiriam minimizar os riscos associados ao desenvolvimento de software, pensando e agindo de forma muito diferente da tradicional. Esses profissionais, a maioria veteranos consultores para projetos de softwares, decidiram reunir-se no início de 2001 para discutir sobre suas experiências.

Embora cada envolvido tivesse suas próprias práticas e teorias preferidas, todos concordavam que, em suas experiências prévias, os projetos de sucesso tinham em comum um pequeno conjunto de princípios. Com base nisso, eles criaram o *Manifesto para o desenvolvimento ágil de software*, frequentemente chamado apenas de *Manifesto ágil*. O termo *desenvolvimento ágil* identifica metodologias de desenvolvimento que adotam os princípios do *Manifesto ágil*. Esses princípios são os seguintes:

- *Indivíduos e interação entre eles* mais que processos e ferramentas;
- *Software em funcionamento* mais que documentação abrangente;
- *Colaboração com o cliente* mais que negociação de contratos;
- *Respostas às mudanças* mais que seguir um plano.

Temos várias metodologias e processos para *desenvolvimento ágil*. Podemos citar: *feature driven development* (FDD), *eXtreme programming* (XP), *scrum* e *test driven development* (TDD).

FEATURE DRIVEN DEVELOPMENT (FDD)

Essa metodologia permite desenvolver um sistema de forma rápida e possibilita facilmente a introdução de novas *features* (funcionalidades) nesse sistema. Uma nova *feature* demora entre duas horas a duas semanas a ser concluída. O fato de, em FDD, usarem-se processos simples proporciona uma rápida exposição do projeto a novos elementos que venham a fazer parte da equipe, além de tornar o trabalho menos monótono, uma vez que se está sempre a iniciar/acabar um processo.

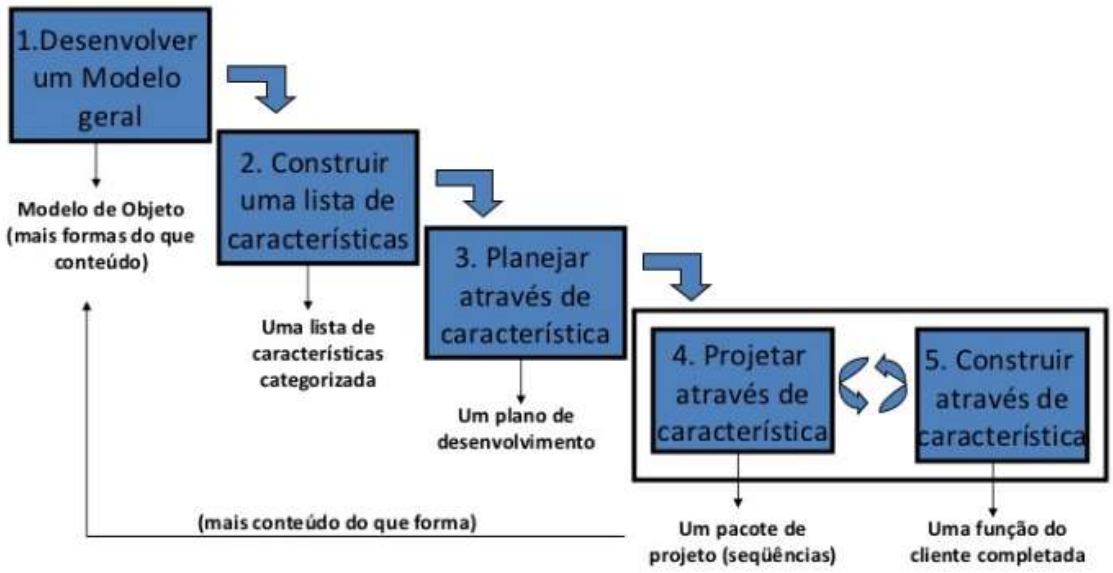

Figura 7 – Processo de *feature driven development* (FDD)

EXTREME PROGRAMMING (XP)

Programação extrema (do inglês *extreme programming*), ou simplesmente XP, é uma metodologia ágil (Links para um site externo)Links para um site externo para equipes pequenas e médias e que irão desenvolver softwares (Links para um site externo)Links para um site externo com requisitos vagos e em constante mudança. Para isso, adota-se a estratégia de constante acompanhamento e realização de vários pequenos ajustes durante o desenvolvimento de software.

Os cinco valores fundamentais da metodologia XP são: comunicação, simplicidade, feedback, coragem e respeito. A partir desses valores, estabelecem-se como princípios básicos: feedback rápido, presumir simplicidade, mudanças incrementais, abraçar mudanças e trabalho de qualidade.

Dentre as variáveis de controle em projetos (Links para um site externo)Links para um site externo (custo, tempo, qualidade e escopo), há um foco explícito em escopo. Para isso, recomenda-se a priorização de funcionalidades que representem maior valor possível para o negócio. Dessa forma, caso seja necessária a diminuição de escopo, as funcionalidades menos valiosas serão adiadas ou canceladas.

A XP incentiva o controle da qualidade como variável do projeto, pois o pequeno ganho de curto prazo na produtividade, ao diminuir qualidade, não é compensado por perdas (ou até impedimentos) a médio e longo prazo.

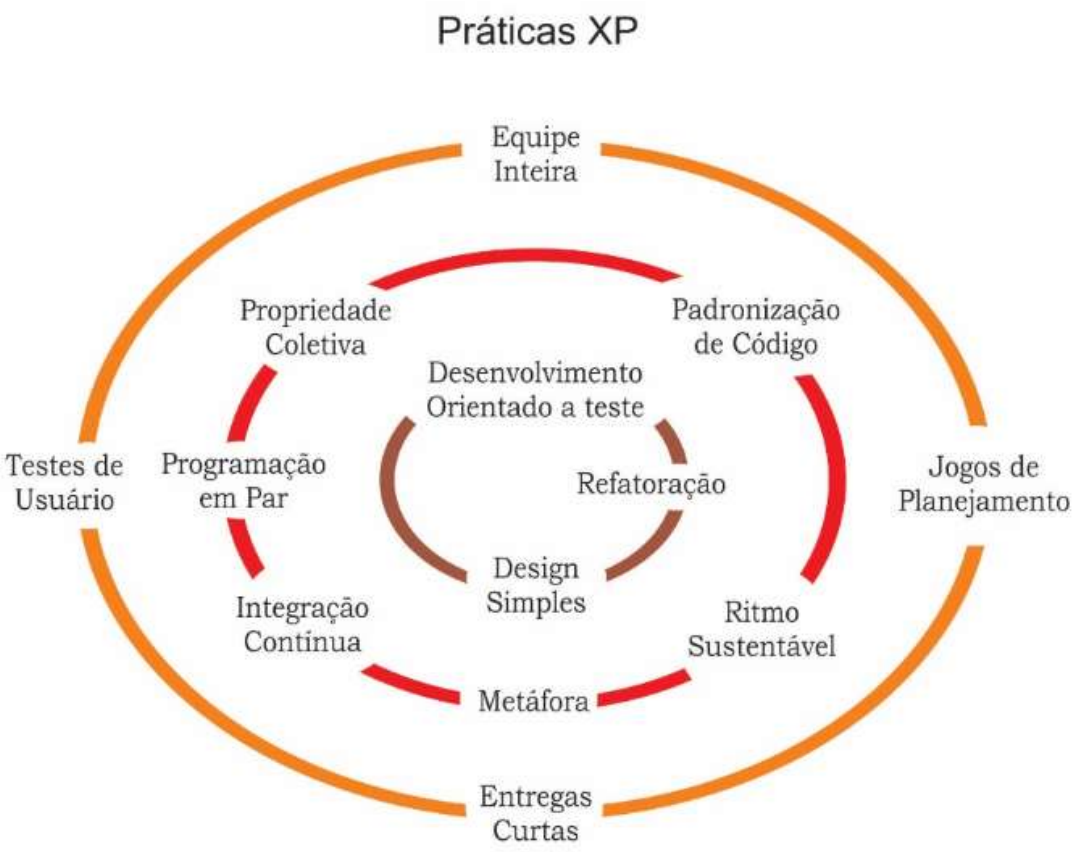

Figura 8 – Práticas do *eXtreme programming* (XP)

SCRUM

Scrum é uma metodologia ágil para gestão e planejamento de projetos de software. No *scrum*, os projetos são divididos em ciclos (tipicamente mensais) chamados de *sprints*. O *sprint* representa um *time box* dentro do qual um conjunto de atividades deve ser executado. Metodologias ágeis de desenvolvimento de software são iterativas, ou seja, o trabalho é dividido em iterações, que são chamadas de *sprints* no caso do *scrum*.

As funcionalidades a serem implementadas em um projeto são mantidas em uma lista que é conhecida como *product backlog*. No início de cada *sprint*, faz-se um *sprint planning meeting*, ou seja, uma reunião de planejamento na qual o *product owner* prioriza os itens do *product backlog* e a equipe seleciona as atividades que ela será capaz de implementar durante o *sprint* que se inicia. As tarefas alocadas em um *sprint* são transferidas do *product backlog* para o *sprint backlog*.

A cada dia de um *sprint*, a equipe faz uma breve reunião (normalmente de manhã), chamada *daily scrum*. O objetivo é disseminar conhecimento sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho do dia que se inicia.

Ao final de um *sprint*, a equipe apresenta as funcionalidades implementadas em um *sprint review meeting*. Finalmente, faz-se um *sprint retrospective*, e a equipe parte para o planejamento do próximo *sprint*. Assim, reinicia-se o ciclo, conforme a figura 9.

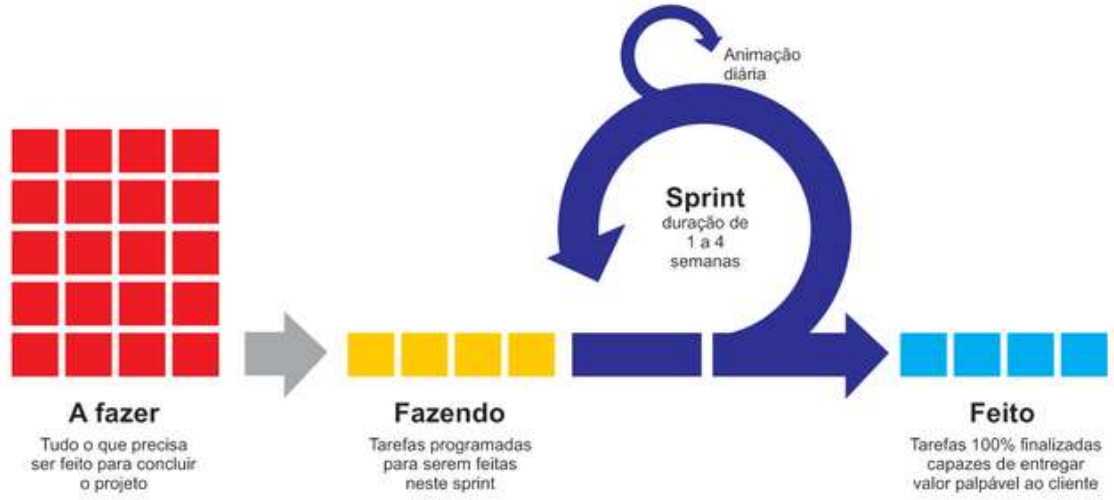

Figura 9 – Processo do *scrum*

***TEST DRIVEN DEVELOPMENT* (TDD)**

O TDD é também uma prática ágil de desenvolvimento (Links para um site externo)Links para um site externo, que resulta em uma suíte automatizada de testes unitários e permite praticar um desenvolvimento orientado a testes em que, antes de se desenvolver efetivamente, definem-se expectativas do software, buscando atender a necessidade do usuário.

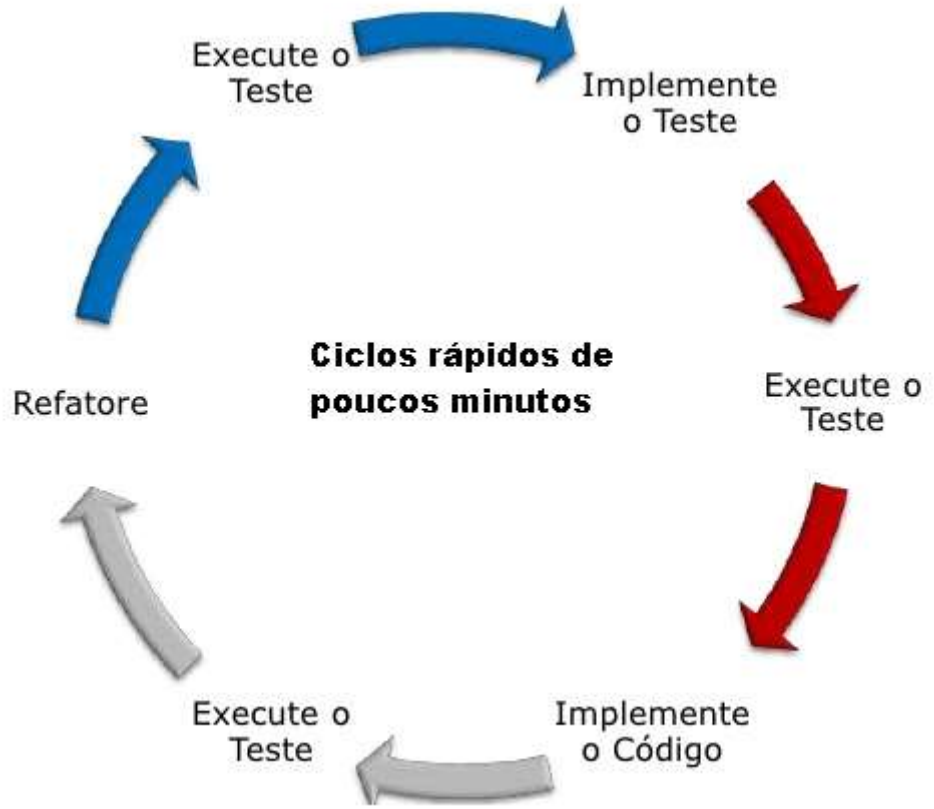

Figura 10 – Ciclos do test driven development (TDD)

IMPLANTAÇÃO DE PACOTES

A implementação de sistemas de informação através da aquisição de pacotes de software é uma prática comum nas organizações. A premissa básica dessa opção é a de que já existem no mercado soluções de software para uma boa parcela dos processos de negócio das empresas. Essas soluções trazem embutidas as melhores práticas de negócio de uma determinada área e representam um conhecimento acumulado que já foi operacionalizado na forma de produtos e serviços previamente testados e aceitos pelo mercado.

As razões para se optar por um pacote de software incluem a redução do trabalho de análise, projeto, construção, instalação e manutenção do sistema. Além disso, a aquisição de um pacote pode reduzir custo e tempo no desenvolvimento de aplicações de negócio comuns. Por fim, essa opção reduz a necessidade de recursos internos de sistema de informação. Também nesse caso a seleção do pacote deve estar pautada em uma lista de requisitos derivada dos problemas e das oportunidades que se quer atingir com a implementação do novo sistema.

Esses requisitos abrangem funcionalidades que o pacote oferece, como manutenção de cadastros e tabelas específicos, emissão de relatórios determinados

e recursos de integração com sistemas já existentes. Além disso, o pacote deve ser analisado do ponto de vista de requisitos tecnológicos como a plataforma de hardware, o sistema gerenciador de banco de dados e o suporte a tempos de resposta e formatos de dados específicos. De forma semelhante ao caso da terceirização, é preciso analisar as características oferecidas pelo fornecedor do pacote. Tais características podem incluir o tipo de suporte oferecido e a modalidade de contrato de manutenção e atualização de versões.

Entretanto, assim como nas demais abordagens, a implementação através de pacotes de software pode apresentar algumas desvantagens. O pacote pode não contemplar os requisitos da organização ou não realizar os processos de negócio da forma com que a organização atua. Em geral, essas situações fazem com que haja a necessidade de adequações do pacote e da própria organização. Além disso, pode ser necessário o desenvolvimento de interfaces com os sistemas legados. Sistemas legados são aqueles sistemas que já existem e permanecerão em funcionamento após a implementação do novo pacote.

A seguir um esquema do processo de implementação de um sistema pacote, conforme já apresentado e especificado em aulas anteriores.

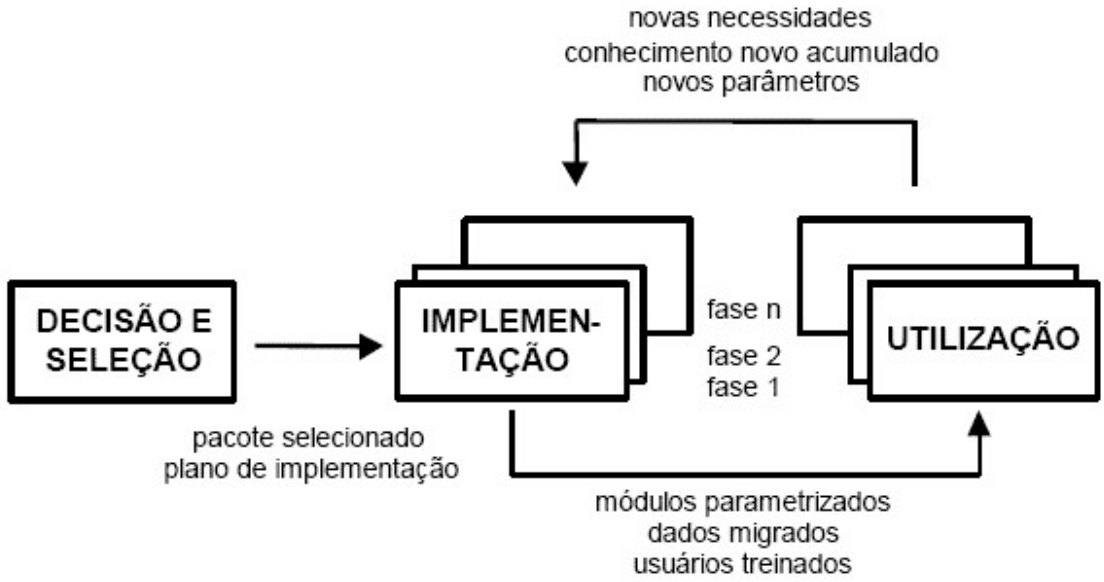

Figura 11 – Fases da implantação de pacote de software de gestão (ERP)

ETAPAS DA IMPLEMENTAÇÃO

Após a escolha do sistema ERP a ser implantado na empresa, dá-se início ao estudo do sistema, análise das necessidades de adaptações e customizações, preparação da infraestrutura tecnológica e início da implantação de módulos do ERP. Mesmo

estando apenas em fase de homologação pelo proprietário, o processo de instalação desses módulos gerou novos conhecimentos das funcionalidades do sistema e de necessidades de adaptação. Uma das adaptações realizadas foi a tradução de alguns termos que não receberam tradução mesmo com a instalação parcial e essencial para o uso pleno do ERP, já que tanto o proprietário quanto seus funcionários não têm, na maioria dos casos, conhecimento de inglês ou de termos técnicos existentes nesses pacotes. Após a instalação dos módulos, realiza-se a importação dos dados já existentes. Caso a empresa não possua partes de seus dados informatizados, não haverá a possibilidade de criação de rotinas programadas para realizar a inserção desses dados no sistema.

Basicamente todo processo se resume em desenvolver o plano de implementação, adaptação dos módulos, treinamento dos sistemas e dos usuários, conversão dos dados e operação dos módulos, até a conversão total.

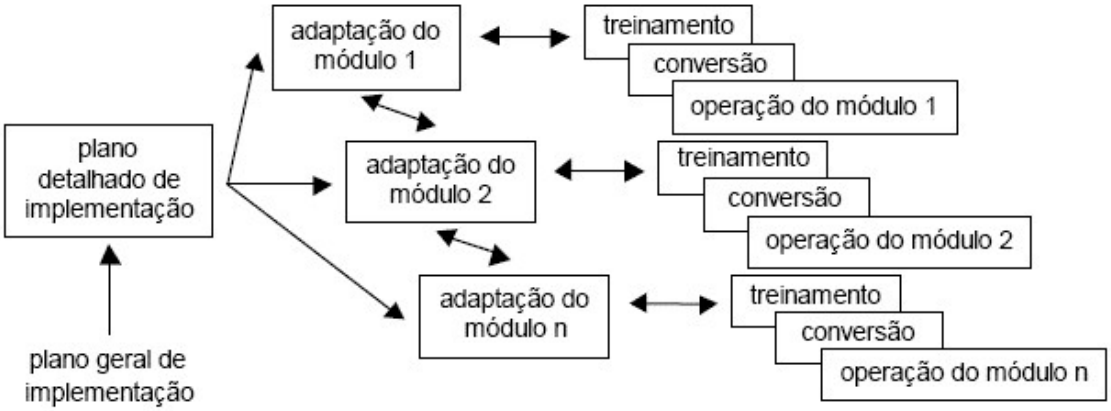

Figura 12 – Fases do plano de implementação dos módulos de software de gestão (ERP)

Cada módulo passa por um processo que envolve as várias atividades, resumido na figura a seguir:

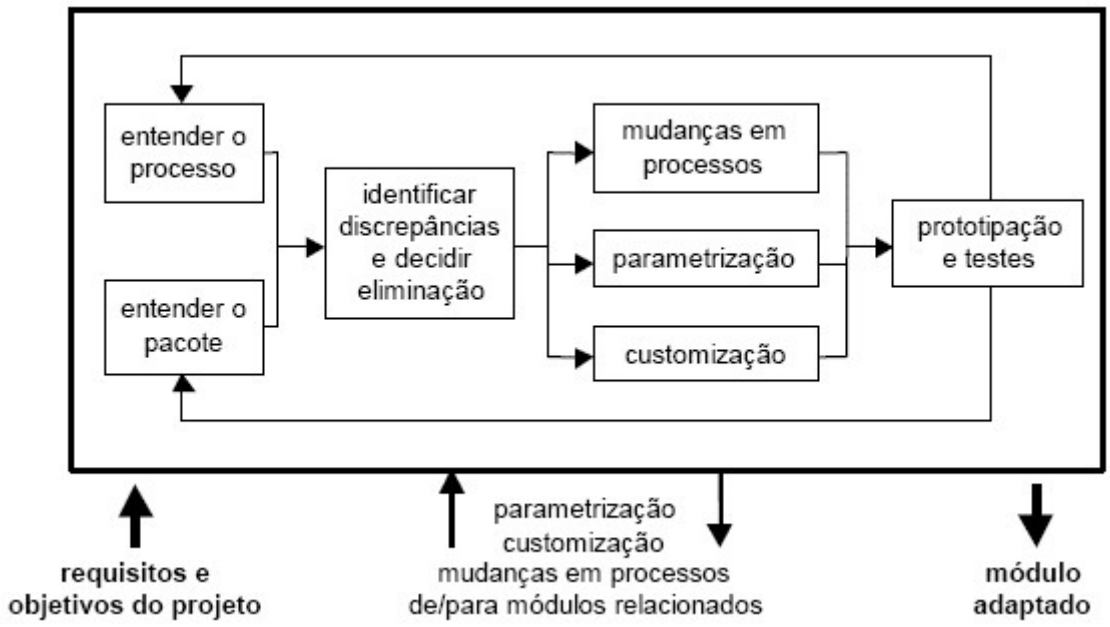

Figura 13 – Atividade de adaptação dos módulos de software de gestão (ERP)

CONCEITO DE QUALIDADE

Qualidade é um conceito subjetivo que está relacionado diretamente às percepções de cada indivíduo. Diversos fatores – como cultura, modelos mentais, tipos de produtos ou serviços prestados, necessidades e expectativas – influenciam diretamente nessa definição.

QUALIDADE DE SOFTWARE

Área de conhecimento da Engenharia de Software que visa garantir a *qualidade do software* através da definição e normatização de processos de desenvolvimento, com o objetivo de garantir um produto final que satisfaça às expectativas do cliente, dentro daquilo que foi acordado inicialmente.

CUSTO DA QUALIDADE

São considerados custos da qualidade o valor total desprendido para alcançar a qualidade total do software. O custo de qualidade inclui todos os gastos financeiros relacionados às atividades de qualidade, os quais podem ser divididos em custos de prevenção e custos de avaliação.

Os custos de prevenção incluem:

- Planejamento da qualidade;
- Revisões técnicas formais;
- Teste de equipamentos;

Os custos de avaliação incluem:

- Inspeções dos processos e relações entre eles;
- Manutenção dos equipamentos;
- Testes.

CUSTO DA FALHA

Os custos de falhas, que podem ser divididos em internas e externas, poderiam desaparecer se nenhum defeito ocorresse antes da entrega do produto para o cliente.

Os custos de falhas internas incluem:

- Retrabalho;
- Conserto de bugs;
- Análise de falhas.

Os custos de falhas externas incluem:

- Resolução de queixas;
- Revisão do produto;
- Aspectos de segurança.

GARANTIA DA QUALIDADE

A garantia de qualidade de software não é algo com o qual se começa a pensar depois que o código é gerado. É uma atividade que é aplicada ao longo de todo o processo de Engenharia de Software. Abrange:

- Métodos e ferramentas de análise, projeto, codificação e teste;
- Controle da documentação do software e das suas mudanças;
- Procedimento para garantir a adequação aos padrões;
- Revisões técnicas formais;

- Uma estratégia de teste de múltiplas fases;
- Mecanismos de medição e divulgação.

ATRIBUTOS DA QUALIDADE

Existem várias formas de classificar os fatores de qualidade. Uma delas é classificá-los como fatores externos e fatores internos. Fatores externos são aqueles cuja presença ou falta num produto de software pode ser detectada pelos usuários do produto (velocidade, usabilidade, portabilidade etc.). Fatores internos são aqueles que são perceptíveis apenas por profissionais de computação.

MÉTRICAS DA QUALIDADE DE SOFTWARE

Um elemento chave de qualquer processo de Engenharia é a medição. Usamos medidas para melhor entendermos os atributos dos modelos que criamos e para avaliarmos a qualidade dos produtos de Engenharia ou sistemas que nós construímos.

A forma de medir o sucesso da implementação de sistemas de informação é uma das preocupações das pesquisas em sistemas de informação. Desde a década de 1960, vêm sendo feitos inúmeros estudos com o intuito de oferecer um conjunto de medidas que permita avaliar de maneira confiável a efetividade dos processos de implementação e dos próprios sistemas de informação. De maneira geral, constatamos que medir o sucesso da implementação é uma tarefa complexa que exige múltiplos indicadores, já que os sistemas de informação apresentam dimensões organizacionais, humanas e tecnológicas.

FUNÇÕES DA MEDIÇÃO

A medição é algo comum no mundo da Engenharia. As métricas de sistemas possibilitam realizar uma das atividades mais fundamentais do processo de gerenciamento de projetos em sistemas de informação: o planejamento. A partir deste, pode-se identificar a quantidade de esforço, de custo e das atividades que serão necessárias para a realização do projeto.

Do ponto de vista da medição, podem ser divididas em duas categorias: medidas diretas e indiretas. Podemos considerar como medidas diretas do processo de Engenharia o custo e o esforço aplicados ao desenvolvimento e manutenção do produto ou serviço. Porém a qualidade e a funcionalidade dos sistemas de

informação, ou a sua capacidade de manutenção, são mais difíceis de serem avaliadas e só podem ser medidas de forma indireta.

Também podemos dividir as métricas de sistemas de informação, sob o ponto de vista de aplicação, em duas categorias: métricas de produtividade e de qualidade. As métricas de produtividade concentram-se na saída do processo de Engenharia. As métricas de qualidade indicam o quanto os sistemas de informação atendem aos requisitos definidos pelo usuário. O elemento principal de uma métrica ou de um indicador é o seu objetivo. Os processos de medição e de gestão de indicadores e métricas, principalmente quando não apoiados por ferramentas e softwares de automação, requerem especialistas e trabalho contínuo e podem se tornar caros, tediosos e pouco recompensadores.

Portando utilizam-se de processos e métodos específicos para medir produtos e serviços com o intuito de entender sua aplicação e funcionalidade, assim podendo controlar seus recursos e operações. Dessa forma, é possível prever as principais ocorrências e avaliar seus resultados.

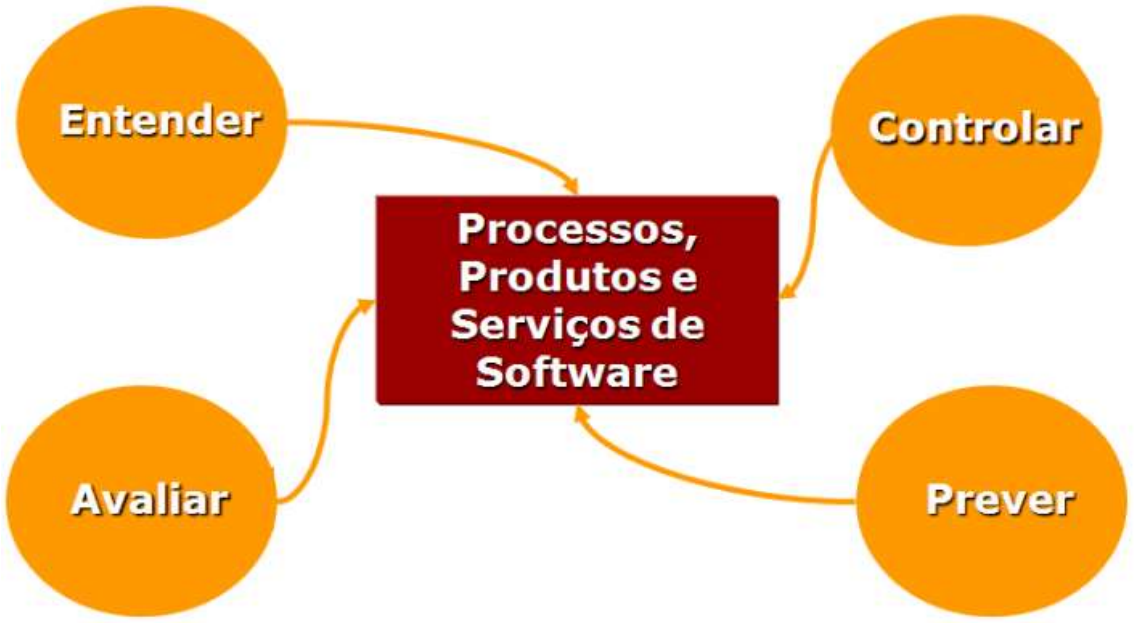

Figura 14 – Ciclo da medição dos processos de implementação do sistema de informação

ORGANISMOS NORMATIVOS

São instituições colaborativas responsáveis pela criação, edição, monitoramento e publicação, além de várias atividades que verificam e validam as normas e procedimentos. Tem como objetivo principal aprovar normas nacionais e

internacionais em todos os campos técnicos, como normas técnicas, classificações de países, normas de procedimentos e processos etc.

Normas e padrões em sistemas de informação

Padrões e normativas servem para medir diversos aspectos da qualidade de software ou do sistema de informação, como qualidade de produto de software ou sistema, qualidade do processo de desenvolvimento e o nível de maturidade da organização que desenvolve, adapta, implanta. Segue uma lista de padrões e normativas utilizadas em sistemas de informação:

- ISO 15504 (SPICE) – avaliação de processo de software;
- Br – melhoria do processo de software brasileiro;
- ISO 12207 – qualidade do processo de software;
- CMMi – maturidade do processo de software;
- ISO 9126 – qualidade de produtos de software;
- ISO 12119 – qualidade de pacotes de software.

REFERÊNCIAS BIBLIOGRÁFICAS:

1. AUDY, Jorge Luis Nicolas; BRODBECK, Ângela Freitag. **Sistemas de informação**: planejamento e alinhamento estratégico nas organizações. Porto Alegre: Bookman, 2008. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788577803972/cfi/0!/4/2@100:0.00 (Links para um site externo)Links para um site externo>. Acesso em: 19 dez. 2018.
2. BORGES, Carlos; ROLLIM, Fabiano. **Gerenciamento de projetos aplicado**: conceitos e guia prático. Rio de Janeiro: Brasport, 2015. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788574527604 (Links para um site externo)Links para um site externo>. Acesso em: 15 mar. 2019.
3. CARVALHO, Fábio Câmara Araújo de. **Gestão de projetos**. 2. ed. São Paulo: Pearson, 2018. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788543025674/pages/-14 (Links para um site externo)Links para um site externo>. Acesso em: 15 mar. 2019.
4. CRUZ, Tadeu. **Manual de técnicas administrativas**: métodos e procedimentos com formulários. São Paulo: Atlas, 2018. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788597018653/cfi/6/2!/4

/2@0:0 (Links para um site externo)Links para um site externo>. Acesso em: 15 mar. 2019.

5. GALLOTTI, Giocondo Marino Antonio (Org.). **Qualidade de software**. São Paulo: Pearson Education do Brasil, 2016. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788543020358/pages/-12 (Links para um site externo)Links para um site externo>. Acesso em: 18 mar. 2019.
6. LEE, Richard C.; TEPFENHART, William M. **UML e C++**: guia prático de desenvolvimento orientado a objeto. São Paulo: Makron Books, 2001. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788534613644/pages/_1 (Links para um site externo)Links para um site externo>. Acesso em: 18 mar. 2019.
7. MASSARI, Vitor L. **Agile Scrum Master no gerenciamento avançado de projetos**: base para certificação EXIN Agile Scrum Master. Rio de Janeiro: Brasport, 2016. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788574527857 (Links para um site externo)Links para um site externo>. Acesso em: 18 mar. 2019.
8. MEDEIROS, Ernani Sales de. **Desenvolvendo software com UML 2.0**: definitivo. São Paulo: Pearson Makron Books, 2004. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788534615297/pages/-23 (Links para um site externo)Links para um site externo>. Acesso em: 18 mar. 2019.
9. NEWTON, Richard. **O gestor de projetos**. ed. São Paulo: Pearson Prentice Hall, 2011. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788576058113/pages/_1 (Links para um site externo)Links para um site externo>. Acesso em: 15 mar. 2019.
10. OLIVEIRA, Bruno Sousa de. **Métodos ágeis de gestão de serviços de TI**: saiba como e por que montar equipes ágeis para gerenciar desenvolvimento, implantação e suporte de seus sistemas como serviço. Rio de Janeiro: Brasport, 2018. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788574528717 (Links para um site externo)Links para um site externo>. Acesso em: 19 dez. 2018.
11. PAGE-JONES, Meilir. **Fundamentos do desenho orientado a objeto com UML**. São Paulo: Makron Books, 2001. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788534612432/pages/_1 (Links para um site externo)Links para um site externo>. Acesso em: 15 mar. 2019.

12. PFLEEGER, Shari Lawrence. **Engenharia de software**: teoria e prática. 2. ed. São Paulo: Prentice Hall, 2004. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788587918314/pages/_1 (Links para um site externo)Links para um site externo>. Acesso em: 15 mar. 2019.
13. PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de software**. 8. ed. Porto Alegre: AMGH, 2016. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788580555349/cfi/0!/4/2@100:0.00 (Links para um site externo)Links para um site externo>. Acesso em: 15 mar. 2019.
14. ROSINI, Alessandro Marco; PALMISANO, Angelo. **Administração de sistemas de informação e a gestão do conhecimento**. 2. ed., São Paulo: Cengage Learning, 2012. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788522114672/cfi/0!/4/2@100:0.00 (Links para um site externo)Links para um site externo>. Acesso em: 19 dez. 2018.
15. SOMMERVILLE, Ian. **Engenharia de software**. 10. ed. São Paulo: Pearson Education do Brasil, 2018. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788543024974/pages/-14 (Links para um site externo)Links para um site externo>. Acesso em: 18 mar. 2019.
16. SOMMERVILLE, Ian. **Engenharia de software**. 6. ed. São Paulo: Pearson Addison-Wesley, 2003. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788588639072/pages/_1 (Links para um site externo)Links para um site externo>. Acesso em: 18 mar. 2019.
17. SOMMERVILLE, Ian. **Engenharia de software**. 8. ed. São Paulo: Pearson Addison-Wesley, 2007. Disponível em: <https://univesp.bv3.digitalpages.com.br/users/publications/9788588639287/pages/_1 (Links para um site externo)Links para um site externo>. Acesso em: 15 mar. 2019.
18. STAIR, Ralph M.; REYNOLDS, George W. **Princípios de sistemas de informação**. São Paulo: Cengage Learning, 2015. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788522124107/cfi/0!/4/2@100:0.00 (Links para um site externo)Links para um site externo>. Acesso em: 19 dez. 2018.