UNIVERSO

UNIVERSOEAD

ENSINO A DISTÂNCIA

1ª edição

Engenharia de Software

Luciane de Fátima Silva

FICHA TÉCNICA

Direção Editorial: Diogo Pereira da Silva e Patrícia Figueiredo Pereira Salgado
Texto: Luciane de Fátima Silva
Revisão Ortográfica: Rafael Dias de Carvalho Moraes
Projeto Gráfico e Editoração: Antonia Machado, Eduardo Bordoni, Fabrício Ramos e Victor Narciso
Supervisão de Materiais Instrucionais: Antonia Machado
Ilustração: Eduardo Bordoni e Fabrício Ramos
Capa: Eduardo Bordoni e Fabrício Ramos

COORDENAÇÃO GERAL:

Departamento de Ensino a Distância
Rua Marechal Deodoro 217, Centro, Niterói, RJ, CEP 24020-420 www.universo.edu.br

Palavra da Reitora

Acompanhando as necessidades de um mundo cada vez mais complexo, exigente e necessitado de aprendizagem contínua, a Universidade Salgado de Oliveira (UNIVERSO) apresenta a UNIVERSO EAD, que reúne os diferentes segmentos do ensino a distância na universidade. Nosso programa foi desenvolvido segundo as diretrizes do MEC e baseado em experiências do gênero bem-sucedidas mundialmente.

São inúmeras as vantagens de se estudar a distância e somente por meio dessa modalidade de ensino são sanadas as dificuldades de tempo e espaço presentes nos dias de hoje. O aluno tem a possibilidade de administrar seu próprio tempo e gerenciar seu estudo de acordo com sua disponibilidade, tornando-se responsável pela própria aprendizagem.

O ensino a distância complementa os estudos presenciais à medida que permite que alunos e professores, fisicamente distanciados, possam estar a todo momento ligados por ferramentas de interação presentes na Internet através de nossa plataforma.

Além disso, nosso material didático foi desenvolvido por professores especializados nessa modalidade de ensino, em que a clareza e objetividade são fundamentais para a perfeita compreensão dos conteúdos.

A UNIVERSO tem uma história de sucesso no que diz respeito à educação a distância. Nossa experiência nos remete ao final da década de 80, com o bem-sucedido projeto Novo Saber. Hoje, oferece uma estrutura em constante processo de atualização, ampliando as possibilidades de acesso a cursos de atualização, graduação ou pós-graduação.

Reafirmando seu compromisso com a excelência no ensino e compartilhando as novas tendências em educação, a UNIVERSO convida seu alunado a conhecer o programa e usufruir das vantagens que o estudar a distância proporciona.

Seja bem-vindo à UNIVERSO EAD!

Professora Marlene Salgado de Oliveira

Reitora.

Sumário

Apresentação da disciplina 7

Plano da Disciplina 8

Unidade 1 - Introdução – Software e Engenharia de Software 11

Unidade 2 - Modelos Prescritivos de Processo de Desenvolvimento de Software .. 39

Unidade 3 - Modelos Ágeis de Desenvolvimento de Software 61

Unidade 4 - Gerenciamento de Sistemas de Informação 87

Unidade 5 - Projeto Arquitetural de Software 100

Unidade 6 - Análise de Requisitos de Software e de Sistemas 120

Considerações finais 144

Conhecendo o autor 145

Anexos 147

Apresentação da disciplina

Prezado(a) aluno(a),

Seja bem-vindo à disciplina de Engenharia de Software.

Este livro de estudo da disciplina traz, além dos conceitos básicos referentes aos processos, métodos e ferramentas para o desenvolvimento de software, aspectos referentes a disciplina de engenharia para a concepção, planejamento, construção e manutenção de um software.

Complementando tais aspectos, a disciplina pretende também, responder questões como: o que é um software? Qual o papel da Engenharia de Software? Quais os paradigmas e modelos prescritivos para o desenvolvimento e manutenção de software? Quais as vantagens, desvantagens e quando (ou não) devem ser utilizados?

Num contexto bem atualizado, traz conceitos de Metodologias Ágeis, apresentando os métodos mais conhecidos como o *eXtreming Programmig* (XP), Scrum, Crystal, FDD, DSDM e ASD. Além disso, apresenta sobre o Gerenciamento de Sistemas de Informação e dos Recursos de Informação, destacando a sua importância para as organizações em um meio onde a Informação é um bem intensamente valioso.

Por fim, nessa disciplina, abordaremos conceitos de Projeto Arquitetural de Software e Engenharia de Requisitos, processos fundamentais para o sucesso do desenvolvimento de qualquer sistema de software.

Vamos juntos aprimorar conhecimento parar agregar mais valor à sua formação e carreira! Conto com você!

Sucesso e bons estudos!

Plano da Disciplina

A disciplina tem como objetivo apresentar a visão geral e princípios fundamentais da Engenharia de Software. Conhecimentos básicos do ciclo de vida de software e seus vários estágios: requisitos de software, projeto de software, implementação de software e gerenciamento de software.

Unidade 1- Introdução – Software e Engenharia de Software

Nesta unidade, estudaremos os principais conceitos sobre o surgimento e importância da Engenharia de Software. A ideia inicial da Engenharia de Software era tornar o desenvolvimento de software um processo sistematizado, com técnicas e métodos definidos que seriam aplicados para manter o controle da qualidade e complexidade de grandes sistemas.

Objetivos da unidade:

O objetivo dessa unidade é fazer uma introdução aos conceitos de Software e Engenharia de Software, necessários para o entendimento dessa disciplina.

Unidade2 – Modelos Prescritivos de Processo de Desenvolvimento de Software

Nesta unidade estudaremos os principais conceitos sobre o processo de desenvolvimento de Software. Como foi dito na unidade anterior, no início o desenvolvimento de software era desenvolvido de qualquer jeito, sem seguir critérios e padrões de qualidade e nenhum processo era aplicado de forma geral.

Objetivos da unidade:

O objetivo é apresentar a ideia do processo de desenvolvimento de software, bem como alguns modelos prescritivos desse processo

Unidade 3 - Modelos Ágeis de Desenvolvimento de Software

Nesta unidade estudaremos os principais conceitos sobre os Métodos Ágeis de Desenvolvimento de Software. Era comum de se pensar na década de 1980 e no início de 1990 de que a melhor maneira de conseguir um software de qualidade era por meio de um planejamento cuidadoso e detalhado do projeto, com uma formalização da qualidade da segurança, com uso de métodos de análise e projeto apoiado por ferramentas CASE (*Computer-aided Software Engineering*) e com o processo de desenvolvimento de software rigoroso e controlado

Objetivos da unidade:

O objetivo dessa unidade é apresentar conceitos sobre os Métodos Ágeis de Desenvolvimento de Software.

Unidade 4 - Gerenciamento de Sistemas de Informação

Nesta unidade estudaremos os principais conceitos sobre o Gerenciamento de Sistemas de Informação. Entender o papel da Informação e dos sistemas que dão apoio e suporte a essa informação dentro de uma organização, bem como saber o papel do gestor da informação e suas responsabilidades.

Objetivos da unidade:

O objetivo dessa unidade é introduzir conceitos sobre o Gerenciamento de Sistemas de Informação

Unidade 5 - Projeto Arquitetural de Software

Nesta unidade, estudaremos os principais conceitos sobre Projeto Arquitetural de Software, que consiste em um mapeamento do sistema de forma que sejam representadas as diferentes partes com suas interações e mecanismos de interconexões.

Objetivos da unidade:

O objetivo desta unidade é apresentar os principais conceitos de arquitetura e do projeto de arquitetura de software.

Unidade 6 – Análise de Requisitos de Software e de Sistemas

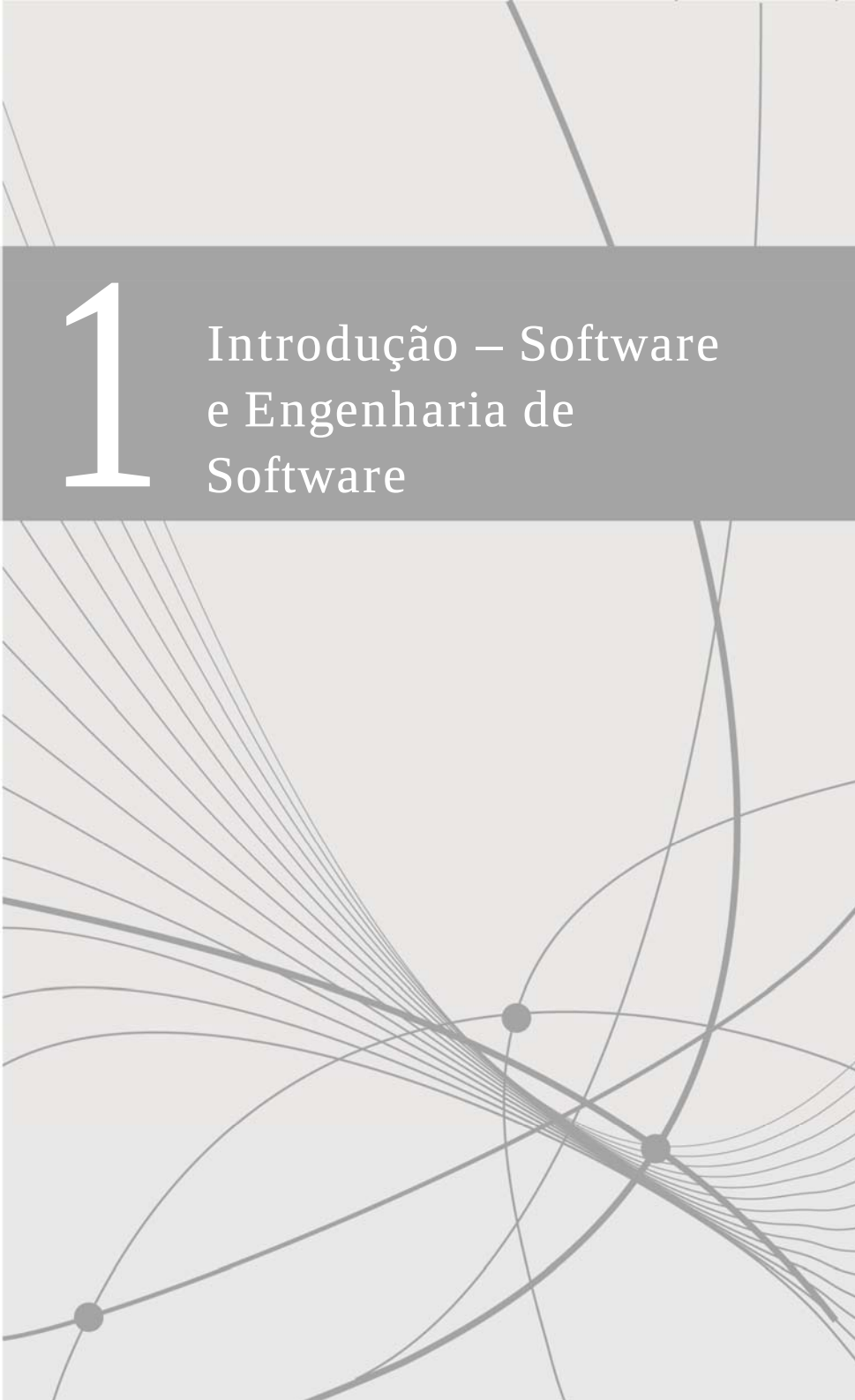
Nesta unidade, estudaremos os principais conceitos sobre Requisitos de um sistema de software. Os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento.

Objetivos da unidade:

O objetivo desta unidade é apresentar os principais conceitos sobre os requisitos de software e de sistema, além de discutir os processos envolvidos no levantamento e documentação desses requisitos.

Bons estudos!

1 Introdução – Software e Engenharia de Software



Caro(a) aluno(a),

Nesta unidade, estudaremos os principais conceitos sobre o surgimento e importância da Engenharia de Software. A ideia inicial da Engenharia de Software era tornar o desenvolvimento de software um processo sistematizado, com técnicas e métodos definidos que seriam aplicados para manter o controle da qualidade e complexidade de grandes sistemas.

Desenvolver software é uma atividade que pode ser considerada complexa, pois não existe uma única solução para um dado cenário que será desenvolvido. Além de tecnologias, quando se fala em desenvolvimento de software lidamos com pessoas e processos, ambos também tem suas particularidades e complexidades. No propósito de conectar tecnologias, pessoas e processos surgiu a Engenharia de Software, a qual pode ser definida como a aplicação de uma abordagem sistemática, disciplinada e quantificável, para o desenvolvimento, operação e manutenção do software.

Sommerville (2011), um dos grandes nomes quando se fala em Engenharia de Software, destaca a importância de se educar profissionais para o entendimento de tal disciplina. De acordo com o autor, o software além de permitir explorar diversas áreas como a internet, medicina e outras ciências, até mesmo explorar o espaço ainda é (e continuará sendo) uma tecnologia de muita importância para a humanidade. Tal importância está relacionada ao conjunto de desafios que temos: mudanças climáticas, diminuição de recursos naturais, aumento na população mundial que precisa de alimentos, abrigo, saúde, transportes, segurança etc., melhoria na qualidade de vida de forma geral. Certamente soluções para tais desafios envolvem tecnologias e o software tem um papel fundamental para auxiliar nas propostas de soluções. Então, Sommerville conclui que a Engenharia de Software é uma tecnologia crítica para o futuro da humanidade e que, devemos continuar a educar engenheiros de software para que possamos criar softwares cada vez mais complexos.

A Engenharia de Software é uma disciplina muito grande, para este material foram selecionados alguns tópicos importantes para o seu entendimento inicial, mas não se limite a estudar apenas o que apresentamos aqui.

Objetivos da unidade:

O objetivo dessa unidade é fazer uma introdução aos conceitos de Software e Engenharia de Software, necessários para o entendimento dessa disciplina. Com isso, espera-se que ao concluir tal disciplina você seja capaz de:

- Entender os principais conceitos de engenharia de software e sua importância;
- Entender que para desenvolver software, em um meio cada vez mais dinâmico, é necessário utilizar diferentes técnicas, as quais devem ser analisadas para se avaliar a mais adequada em cada situação.
- Compreender que, a engenharia de software depende de processos, pessoas e ferramentas e a interdependência entre esses pilares.

Plano da unidade:

- Software e Engenharia de Software
- Papel Evolutivo do Software
- Papel Evolutivo do Software

Bons estudos!

Software e Engenharia de Software

De acordo com o dicionário Michaelis, software pode ser definido como: 1. Qualquer programa ou grupo de programas o qual instrui o hardware (componentes físicos de um computador) sobre a maneira como ele deve executar uma tarefa, inclusive sistemas operacionais, processadores de texto e programas de aplicação; 2. Qualquer programa de computador, especialmente para uso com equipamento audiovisual. Para Sommerville (2011), softwares são programas de computadores e a documentação associada a eles, os quais podem ser desenvolvidos para um cliente específico ou para o mercado em geral.

O software está presente na rotina diária das pessoas, em diversas áreas e, desta forma, pode ser considerado como um componente fundamental do cotidiano. É difícil pensar no mundo moderno sem o uso de software; podemos citar a área de infraestrutura e serviços, sistemas elétricos, sistemas financeiros, a área de entretenimento (jogos, músicas, cinema e televisão), como exemplos de áreas que o funcionamento depende e fazem uso intensivo de software.

Por um bom tempo, pode-se falar nas primeiras décadas da computação, o principal desafio quando se falava em tecnologia era desenvolver hardware que reduzisse o custo de processamento e armazenamento de dados. Com avanço dos anos e da tecnologia, o poder computacional aumentou e o custo do hardware diminuiu, consideravelmente. Então, o desafio agora mudou para melhorar a qualidade (e reduzir custo também) de soluções implementadas com software.

As aplicações de software podem ser divididas em:

- **Software Básico**: conjunto de programas para dar apoio a outros programas; possuem forte interação com hardware, operações concorrentes, uso por múltiplos usuários, compartilhamento de recursos e múltiplas interfaces.
- **Software de Tempo Real:** são programas que monitoram, analisam e controlam eventos do mundo real.

- **Software Comercial:** são aplicações que gerencial as operações comerciais com o propósito de facilitar o gerenciamento do negócio e auxiliam na tomada de decisões.
- **Software Científico e de Engenharia:** caracterizados por algoritmos de processamento numérico, dependentes de coleta e processamento de dados para diversas áreas do conhecimento.
- **Software Embutido:** desenvolvidos para executarem atividades específicas e são inseridos em produtos inteligentes.
- **Software de Computador Pessoal:** são desenvolvidos para uso pessoal, tais como jogos, editores de texto, planilhas eletrônicas, entre outros.
- **Software de Inteligência Artificial:** faz uso de algoritmos não numéricos para resolver problemas complexos.

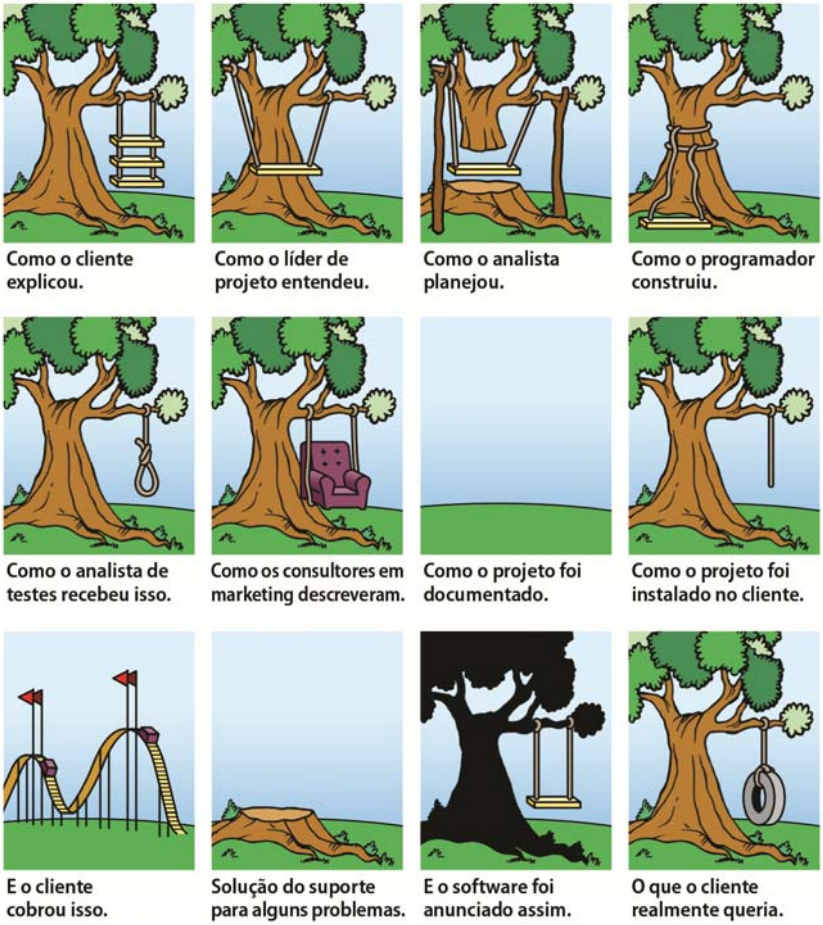
O que as pessoas esperam de um software atualmente?

Figura 1.1: O que as pessoas esperam de um software?

Para se desenvolver softwares que atendam à necessidade de seus usuários é preciso, além de contratar profissionais com grande experiência, implantar e utilizar processos, métodos e ferramentas para que o desenvolvimento seja efetivo dentro das necessidades e produza software que, de fato, desempenham o papel ao qual foram especificados. Nesse contexto, a Engenharia de Software desempenha um papel fundamental para desenvolver softwares com qualidade.

A Figura 1.2 é uma imagem clássica da Engenharia de Software que apresenta uma crítica de como funciona e os principais problemas enfrentados no processo de desenvolvimento de software.

Figura 2.2: Como os projetos realmente funcionam?



Fonte: Adaptada de *Project Cartoon: How Projects Really Work* (1.5)

A Figura 1.2 apresenta diversos cenários que envolvem os principais envolvidos e interessados em um projeto. Tal figura destaca como os interesses e entendimentos podem divergir em um projeto. Assim, reforça a necessidade de minimizar tais divergências com uso de métodos, ferramentas e processos bem definidos e estruturados.

Algumas causas para justificar a imagem acima podem ser destacadas como:

- Falta de dedicação de tempo e esforço para coletar dados sobre o desenvolvimento de software, que resultam em estimativas mal feitas.
- Falhas na comunicação entre o cliente e a equipe de desenvolvimento.
- Falta de testes sistemáticos e completos.
- Planejamento, projeto e construção executados por equipe sem formação específica ou com baixa capacidade técnica.
- Falta de investimentos (treinamentos, especialização, equipe, ferramentas etc.).
- Falta de métodos e de processos automatizados.

Kechi Hirama (2012) destaca que "um estudo recente apontou que a Tecnologia da Informação (TI) como sendo uma das áreas de trabalho mais estressantes. Exigem-se dos profissionais que desenvolvem software capacidade cognitiva de gerar soluções aderentes ao negócio, de cumprimento de prazos e custos rigorosos, etc.". Assim, os softwares são em geral liberados sem a qualidade desejada e são melhorados após várias manutenções, corrigindo-se os defeitos. O papel da Engenharia de Software é minimizar tal situação (HIRAMA, 2012).

A Engenharia de Software segue os mesmos princípios de uma disciplina de engenharia tradicional, baseando-se em adequar a relação custo/benefício do produto, por meio de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção. Porém, é uma disciplina muito abrangente. Há uma considerável frequência no surgimento de novos métodos, processos e ferramentas e, tal frequência é justificada pela evolução rápida dos modelos de negócio e, em especial, das tecnologias, que exigem softwares cada vez mais complexos.

De acordo com a IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos), a Engenharia de Software é a aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, operação e manutenção de software. Pode ser considerada sistemática por que parte do princípio de que existe um processo de desenvolvimento definindo as atividades a serem executadas; e disciplinada por

que parte do princípio que os processos definidos serão seguidos; e quantificável por que se deve definir um conjunto de medidas a serem extraídas do processo durante o desenvolvimento de forma que as tomadas de decisão sejam embasadas em dados reais.

De acordo com Sommerville (2011) a Engenharia de Software é uma disciplina cujo foco está em desde os estágios iniciais da especificação até a manutenção de um software.

A Engenharia de Software apresenta como principais objetivos:

- Definir, aumentar e garantir a qualidade do software.
- Aumentar a produtividade no desenvolvimento, operação e manutenção de software.
- Permitir que profissionais tenham controle sobre o desenvolvimento de software.
- Construir softwares dentro dos prazos, custos e níveis de qualidade estabelecidos e desejados.

Sommerville (2011) destaca que a Engenharia de Software é importante por dois motivos:

1. Cada vez mais, indivíduos e sociedade dependem dos sistemas de software avançados. É necessário produzir sistemas confiáveis, de forma econômica e rápida.
2. Geralmente é mais barato, a longo prazo, usar métodos e técnicas de Engenharia de Software para sistemas ao invés de produzi-lo de forma artesanal, especialmente quando se vai dar manutenção (maior parte do custo é mudar o software depois que ele começa a ser usado).

A abordagem sistemática usada na Engenharia de Software é chamada de Processo de Software. Um processo de software é uma sequência de atividades que leva à produção de um produto de software, que são, basicamente: especificação do software, desenvolvimento, validação e evolução do software (SOMMERVILLE, 2011).

O cenário de desenvolvimento de software atual e o cenário idealizado na Engenharia de Software ainda não estão congruentes. Podem ser citados dois motivos para tal situação:

1. O não uso dos fundamentos da Engenharia de Software para apoiar as atividades do desenvolvimento.
2. O mal uso dos fundamentos da Engenharia de Software.

É importante destacar também, que existem também fatores que afetam a prática da Engenharia de Software. Existem alguns fatores que afetam a viabilidade de entrega dos produtos de software. Tais fatores são:

- Maior disponibilidade de computadores pessoais, os quais estão cada vez mais potentes.
- Mudanças na economia, no modelo de negócio e nas tecnologias são frequentes.
- *Time to Market*: necessidade de entregar produtos de software de acordo com as exigências do mercado.
- Redes de Telecomunicações extensíveis.
- Grande disponibilidade de infraestrutura de redes de computadores.
- Disponibilidade e adoção de tecnologias orientada a objetos, características e aspectos.
- Maior preocupação e necessidade com interfaces gráficas voltadas aos usuários.

Um estudo realizado em 1995 pelo Standish Group, considerando 350 empresas e cerca de 8 mil projetos de software, revelou que:

- 16,2 % dos projetos são finalizados com sucesso, ou seja, cobre todas as funcionalidades em tempo e dentro do custo previsto.
- 52,7% dos projetos são considerados problemáticos, ou seja, não atendem a todas funcionalidades exigidas, tem custo aumentado ou está atrasado.

- 31,1% dos projetos fracassam (são cancelados durante o desenvolvimento).

O Standish Group realizou uma análise sobre os fatores críticos para o sucesso dos projetos de software e os principais são: requisitos incompletos, falta de envolvimento do usuário, falta de recursos, expectativas irreais, falta de apoio executivo, mudanças de requisitos e especificações, falta de planejamento e sistemas que não são mais necessários (falta de análise de viabilidade).

Então, baseado nesse estudo, percebe-se a necessidade de aplicar a Engenharia de Software de forma adequada para se ter maior sucesso no processo de desenvolvimento de software.

Sabendo da existência de diferentes tipos e aplicações de software, deve-se deixar claro que não existe um método ou uma técnica universal de Engenharia de Software que se aplique a todos. No entanto existem três aspectos fundamentais que afetam vários tipos diferentes de software (SOMMERVILLE, 2011):

1. Heterogeneidade: o desafio é desenvolver técnicas para construir softwares confiáveis que sejam flexíveis o suficiente para lidar com diferentes tipos de dispositivos e tecnologias (como dispositivos móveis), fazer integração em diferentes plataformas e com diversas linguagens de programação.
2. Mudança de negócio e social: A medida que a sociedade evolui, economias emergentes vão se desenvolvendo e novas tecnologias vão se tornando disponíveis deve ser possível alterar o software existente ou desenvolver um novo rapidamente.
3. Segurança e confiança: é essencial que possamos confiar no software dada a sua importância na nossa vida atualmente.

Além disso, Sommerville (2011) cita fundamentos da Engenharia de Software que se aplicam a todos os tipos de sistemas de Software:

1. Eles devem ser desenvolvidos em um processo gerenciado e compreendido.
2. Confiança e desempenho são importantes para todos os tipos de sistemas.

3. É importante entender e gerenciar a especificação e requisitos de software.
4. Deve ser feito o melhor uso possível dos recursos existentes.

Papel Evolutivo de Software

Com a evolução do poder computacional do hardware (mais barato, mais rápido e maior capacidade de armazenamento) também surgiu à necessidade de sistemas baseados em computadores adequados a essa evolução: mais complexos e mais sofisticados.

No início da era da computação softwares não eram produzidos em grande escala, sendo assim mais simples de controlar o que estava sendo feito. Nessa época não existiam métodos para controlar o desenvolvimento, nem pessoas especializadas (ou equipes direcionadas) para realizar um controle no que estava sendo produzido e o desenvolvimento era realizado sem planejamento e administração.

Segundo Pressman (2011), durante as três primeiras décadas da era do computador o desafio principal era diminuir o custo do hardware e aumentar o processamento e a capacidade de armazenamento de dados. E isso ocorreu ao longo da década de 1980, período em que os avanços na microeletrônica possibilitaram uma redução no custo e um aumento no poder de processamento.

A evolução do software pode ser apresentada, em resumo, como:

1950 a 1960: a primeira era do desenvolvimento de software.

- O desenvolvimento de software era considerado uma arte. Programação com processo "artesanal", na qual cada programador fazia da sua maneira.
- Software orientado a batch (programas em lotes).
- O hardware estava em processo de contínuas mudanças e era o centro das atenções.

- Distribuição limitada, com softwares customizados (adequado às necessidades do usuário final) ou destinados a uma aplicação bem específica.
- Não havia administração específica e havia poucos métodos sistemáticos para o desenvolvimento.
- Praticamente não havia documentação ou se houvesse nenhum padrão era seguido, todas as informações necessárias sobre o software estavam na cabeça das pessoas que o desenvolveram (*one's head*).

1960 a 1970: a segunda era do desenvolvimento de software.

- Software multiusuário interativo: nessa época surgiu a multiprogramação e os sistemas multiusuários.
- Utilização de Sistemas em Tempo Real.
- Desenvolvimento de técnicas de IHM (Interação Homem Máquina) ou IHC (Interação Humano Computador).
- Banco de Dados e surgimento da 1 ª Geração de Sistemas Gerenciadores de Banco de Dados (SGBDs).
- Surgimento das "*Software Houses*" e os produtos e software.
- O software era produzido para ampla distribuição em um mercado multidisciplinar, em várias áreas de conhecimentos.
- Surgimento do conceito de biblioteca de software.
- Devido à falta de metodologias de desenvolvimento e de documentação, a manutenção era algo impraticável.

1970 a 1980: terceira era do desenvolvimento de software.

- Surgimento de sistemas distribuídos e paralelos.
- Hardware de baixo custo que levou ao aumento do consumo de computadores.
- Criação dos computadores de uso pessoal (PC – *personal computers*) e estações de trabalho (*workstations*).

- Uso generalizado de microprocessadores.
- Desenvolvimento de redes locais e globais de computadores.
- Impacto de consumo, pois os computadores tornaram-se acessíveis a um grande público que antes não tinham acesso ao uso de computadores.
- Aumento da necessidade de demanda por acesso imediato a dados por parte dos usuários
- "Inteligência" embutida.

1980 a 2000: a quarta era do desenvolvimento de software

- Sistemas de Desktop poderosos;
- Surgimento das técnicas de 4ª Geração (4GT);
- Surgimento dos ambientes cliente-servidor;
- Surgimento dos sistemas multimídia e da realidade virtual;
- Desenvolvimento do paradigma orientado a objetos, da linguagem de modelagem unificada (UML) e do processo unificado;
- Desenvolvimento dos sistemas especialistas e de software de inteligência artificial utilizados em sistema do mundo real;
- Ferramentas CASE;
- Surgimento do conceito de reutilização.
- Surgimento das Redes Neurais Artificiais, computação biológica e da computação vestível (*wearable computers*);

2000 até os dias atuais: a quinta era do desenvolvimento de software.

- Surgimento da Realidade Virtual e Aumentada.
- Surgimento da Computação Pervasiva, Móvel e Ubíqua.
- A capacidade de construir software não pode acompanhar o ritmo da demanda de novos programas.

- A sofisticação do software ultrapassou a capacidade de construir um software que seja capaz de extrair o potencial do hardware.
- A capacidade de manter o software existente é dificultada por projetos ruins e recursos inadequados.

No princípio do desenvolvimento de software considera-se que a programação era uma forma de arte, na qual cada desenvolvedor fazia o código a sua maneira, sem se preocupar com padrões (que não existiam), documentação ou método de desenvolvimento formal. Nos primeiros anos poucos métodos formais eram aplicados, o esquema de desenvolvimento de software era baseado em tentativas e erros. Não havia (ou se houvesse era pouco) planejamento ou um projeto estruturado para análise de viabilidade do software.

O software é, geralmente, o item de maior custo. E, mesmo com métodos, ferramentas e processos bem definidos estruturados ainda há demora na execução e conclusão, ainda existem erros e dificuldade em mensurá-los.

É interessante observar que vários desafios são os mesmos do surgimento da Engenharia de Software, como o desenvolvimento de software rápido e de forma confiável. Mas, atualmente, os maiores desafios estão no processo de desenvolvimento de software em ambientes fisicamente distribuídos, visando ganhos de produtividade, redução de custos, diluição de riscos e melhorias de qualidade.

Paradigmas da Engenharia de Software

Segundo Pressman (2011), os paradigmas são os modelos de processo que possibilitam ao gerente controlar o processo de desenvolvimento de sistemas de software e ao desenvolvedor auxilia na obtenção da base para produzir (de forma eficiente) o software que satisfaça os requisitos estabelecidos. Os paradigmas estabelecem o conjunto de atividades e a ordem na qual devem ser executadas cada atividade pertencente ao conjunto.

O processo de software é o conjunto de atividades que constituem o desenvolvimento de um sistema computacional. Tais atividades são agrupadas em fases e em cada fase são definidas, além das suas atividades, as funções e responsabilidades de cada membro da equipe. O que diferencia um processo de software de outro é a ordem em que as fases vão ocorrer, o tempo e a ênfase a cada atividade da fase, e os artefatos que são gerados em cada uma delas.

Os paradigmas tem como propósito diminuir os problemas decorrentes do processo de desenvolvimento de software não estruturado, tais como entregas fora do prazo, aumento no custo previsto, não previsão de riscos, etc.

Existem vários paradigmas de engenharia de software, o qual deve ser escolhido com base na natureza do projeto e da aplicação e, de acordo com os métodos e as ferramentas que serão utilizadas. Deve-se preocupar também com os controles e os produtos que precisam ser entregues ao usuário na hora de definir qual paradigma utilizar.

O ciclo de vida clássico

O ciclo de vida é uma estrutura que contém processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software. Esse ciclo de vida abrange a vida do sistema, desde sua definição inicial até o seu uso efetivo.

O ciclo de vida clássico é o paradigma que utiliza um método sistemático e sequencial, no qual o resultado de uma fase é entrada para a próxima fase (que só começa quando a anterior terminar). Nesse paradigma há um processo de verificação (está sendo feito de forma correta?), validação (está sendo feito o produto certo?) e controle de qualidade.

O ciclo de vida clássico é o paradigma mais antigo e mais utilizado da Engenharia de Software. São atividades desse paradigma:

- **Concepção ou Estudo da viabilidade**: na fase de concepção são tomadas as decisões de se construir ou não o software. Nessa atividade são definidos os requisitos iniciais do sistema, determinando o que o cliente realmente necessita, quais os problemas nas atividades dos usuários, o que é possível fazer, etc. Nessa atividade também podem ser

definidas algumas restrições do software (restrições de ambiente, interoperabilidade, plataformas, restrições orçamentárias e de tempo, etc.). O resultado dessa atividade é a descrição dos requisitos funcionais (o que deve ser feito) e os não-funcionais (requisitos de ordem técnica, econômica, da organização, etc.). Nessa fase também deve haver uma estudo de viabilidade do software, que visa verificar se o software é viável (de forma técnica e econômica) e se os benefícios advindos do seu uso são compensadores. Associado a essa atividade devem ser realizadas estimativas de custos e prazos que visa determinar gastos e prazos a partir de dados e experiências anteriores. Também deve ser realizada a análise de riscos para verificar se existem possibilidades de algo que possa sair errado, como estourar o orçamento ou possibilidade de dificuldades técnicas.

- **Levantamento, Análise e Especificação de Requisitos:** entendimento dos requisitos que irão compor o sistema, das necessidades reais dos clientes e demais interessados. Os requisitos devem ser levantados, analisados e documentados, para que, em seguida, sejam revistos junto com o cliente antes de começar a execução do projeto.

- **Projeto**: o projeto de software é um processo de múltiplos passos que se concentra em: definições arquiteturais, de estrutura de dados, detalhes procedimentais e caracterização das interfaces. Assim como os requisitos, o projeto é documentado e torna-se parte da configuração do software.

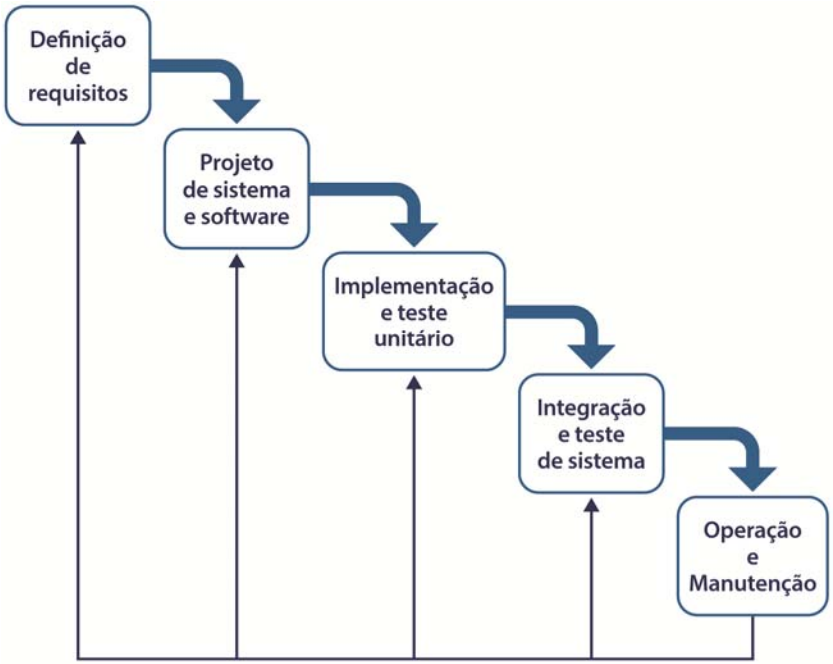
- **Codificação ou Implementação:** a etapa de codificação executa a transformação do projeto em programa com uso de uma linguagem de programação. Ou seja, essa etapa é a tradução do projeto em uma forma que seja legível pela máquina.

- **Testes e Integração do Sistema**: nessa atividade deve-se juntar as unidades (cada uma das partes) construídas e testar cada parte separada e de forma integrada em busca de erros. O resultado final deve estar de acordo com o projeto ou resultado estabelecido nas fases iniciais. Depois de testado (validado e verificado) o software pode ser entregue ao usuário.

- **Manutenção**: após a entrega e início do uso do sistema, inicia-se a fase de manutenção, na qual mudanças (alterações ou inclusões de novas funcionalidades) começam a surgir. A manutenção pode ser corretiva (com erros não detectados na fase anterior), adaptativa (adaptação da aplicação às mudanças do ambiente) e evolutiva (adição de novas funcionalidades e qualidade no software).

A Figura 1.3 apresenta um exemplo de um paradigma clássico ou modelo em Cascata proposto por Sommerville.

Figura 1.3: Paradigma Clássico (ou Modelo em Cascata)



Fonte: SOMMERVILLE, 2011, p. 20.

Dentre as vantagens pode-se citar é que esse paradigma tem uma abordagem disciplinada e dirigida a documentação. E, apesar de ser o mais antigo e o mais amplamente usado da Engenharia de Software, o ciclo de vida clássico apresenta alguns problemas:

- Divisão (inflexível) do projeto em fases sequenciais.
- Projetos reais raramente seguem o fluxo sequencial que o modelo propõe.
- Dificuldade de acomodar mudanças de requisitos do cliente (e usuários).
- Esse modelo exige paciência do cliente, pois uma versão do software só estará disponível no final do cronograma. Se houver um erro grave pode ser desastroso se a descoberta for tardia.
- Pode ser difícil para o cliente declarar tudo que ele necessita logo no início do projeto, não existindo nesse tipo de paradigma outro momento específico para o cliente interagir além do início e da entrega.

Prototipação

A prototipação é um paradigma usado para identificar os requisitos e têm seu uso indicado especialmente quando o cliente tem apena a ideia geral do que precisa, sem requisitos detalhados.

A prototipação é uma abordagem operacional cuja ideia principal é criar um protótipo que permita uma prévia da ideia do sistema proposto.

As etapas da prototipação são:

1. Levantamento dos Requisitos: analistas e clientes (*steakholders*) definem os objetivos gerais do sistema a ser desenvolvido; identificam quais requisitos são conhecidos e quais ainda necessitam de definições adicionais; identificam prioridades, riscos, funcionalidades e restrições.

2. Projeto rápido: representação dos aspectos do software que são visíveis ao usuário (abordagens de entrada e formatos de saída).

3. Construção do Protótipo: implementação rápida do projeto.

4. Avaliação do Protótipo: cliente e demais interessados avaliam o protótipo (atende as necessidades reais?).

5. Refinamento dos requisitos: cliente e demais interessados refinam os requisitos do software a ser desenvolvido.

6. Construção do Produto: identificados os requisitos, o protótipo deve ser descartado (ou melhorado, de acordo com a abordagem de protótipo escolhida e que será detalhada melhor na próxima unidade) e a versão de produção deve ser construída considerando os critérios de qualidade estabelecidos.

A vantagem da prototipação é a facilidade para se determinar os requisitos iniciais e aumentar as possibilidades de alcançar as reais necessidades do cliente, enquanto as desvantagens são os custos de sua construção (tempo e recursos) e a tendência de utilizar o protótipo como produto final.

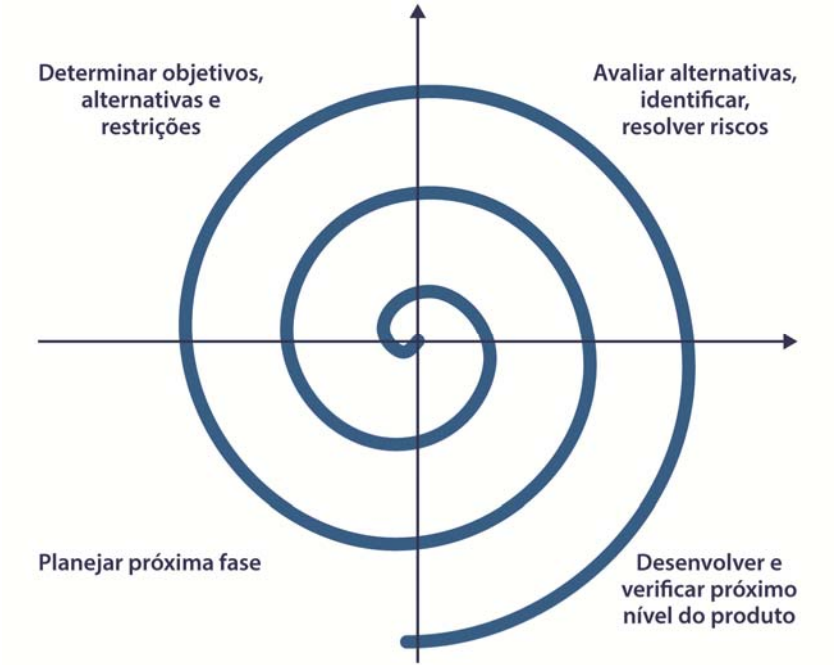
O Modelo Espiral

O Modelo Espiral foi desenvolvido considerando as melhores características do modelo de ciclo de vida clássico e prototipação, acrescentando a análise de riscos. O modelo espiral é considerado a abordagem mais realista para o desenvolvimento de sistemas de software em grande escala. A Figura 1.4 apresenta o Modelo Espiral proposto por Boehm.

O Modelo Espiral define quatro importantes atividades:

- **Definição dos Objetivos**: determinação dos objetivos, alternativas e restrições.
- **Análise de Riscos**: análise de alternativas e identificação/resolução de riscos.
- **Desenvolvimento e Validação**: desenvolvimento e validação do produto.
- **Planejamento:** avaliação dos resultados.

Figura 1.4: Modelo em espiral de processo de software de Boehm.



Fonte: Adaptado de SOMMERVILLE, 2011, p. 33.

Com cada interação ao redor da espiral, iniciando-se ao centro e avançando de dentro para fora, versões são progressivamente construídas e cada vez mais completas. No primeiro ciclo da espiral são definidos os objetivos, as alternativas e as restrições. Nesse primeiro ciclo também são identificados e analisados os riscos (principal característica desse paradigma que o difere dos outros). Se a análise dos riscos indicar que há incertezas nos requisitos, a prototipação pode ser usada para auxiliar.

O modelo Espiral utilizar a prototipação como um mecanismo de redução de riscos e mantém uma abordagem de passos sugerida pelo ciclo de vida clássico. Além disso, para refletir melhor o mundo real, acrescentar uma estrutura iterativa.

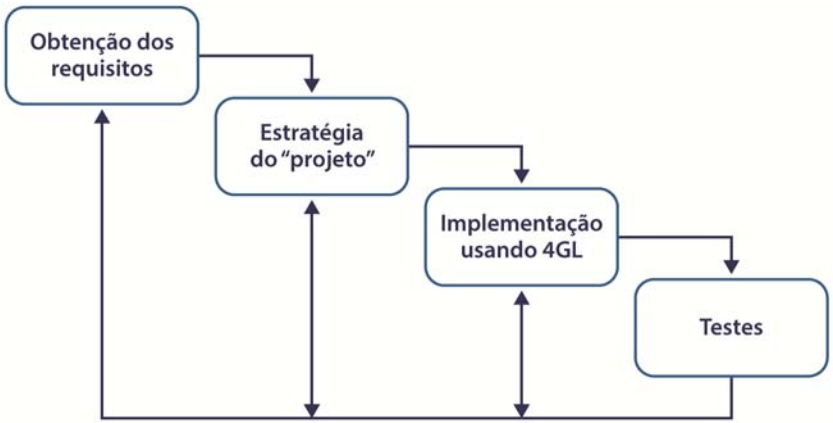
Técnicas da Quarta Geração (4GT)

O paradigma Técnicas de Quarta Geração (4GT) da Engenharia de Software concentra-se na capacidade de se especificar software a uma máquina em uma linguagem próxima a linguagem natural ou de se usar uma notação que comunique uma função significativa. O termo "Técnicas de quarta geração" abrange um amplo conjunto de ferramentas de software que possibilitam que o desenvolvedor de software especifique alguma característica do software num nível elevado.

O ambiente de desenvolvimento que sustente o paradigma 4GT inclui linguagens de programação não procedimentais para consulta de banco de dados, geração de códigos e capacidade gráfica de alto nível. Tal paradigma também é caracterizado pelo suporte automatizado à especificação de requisitos.

O paradigma 4GT está representado na Figura 1.5.

Figura 1.5: Paradigma 4GT



Fonte: Adaptado de PRESSMAN, 2011.

Assim como os outros paradigmas, o 4GT inicia-se com a etapa de coleta de requisitos, que continua sendo por meio de uma coleta com um diálogo entre cliente e equipe que irá desenvolver o software. A descrição dos requisitos pelo cliente são traduzidas para um protótipo operacional.

Para pequenas aplicações, dependendo da complexidade, é possível passar diretamente da etapa de levantamento de requisitos para a implementação, utilizando uma linguagem de quarta geração (4GL). Entretanto, para esforços maiores é preciso desenvolver uma estratégia de projeto para o sistema.

O uso de 4GT sem planejamento causará os mesmos problemas e dificuldades que os outros paradigmas: má qualidade, manutenibilidade difícil e baixa aceitação do cliente. Para o sucesso efetivo de um projeto utilizando 4GT devem ser realizados testes cuidadosos, desenvolver uma documentação significativa e executar todas as demais atividades de "transição" que são exigidas em outros paradigmas.

Ainda há bastante discussão sobre o uso do paradigma 4GT, onde os proponentes reivindicam uma considerável redução no tempo de desenvolvimento de software e produtividade elevada. Já os opositores afirmam que as atuais ferramentas 4GT não são tão fáceis de usar se comparadas as linguagens de programação e que o código produzido por tais ferramentas é ineficiente, além da manutenibilidade de grandes sistemas desenvolvidos pelas técnicas de 4GL está aberta a questionamentos.

Leituras Complementares

"No Silver Bullet: Essence and acidentes of software engineering." Brooks, F. P. IEEE Computer, v. 20, n. 4, abr. 1987.

"Software engineering code of ehtics is approved." Gotternarn, D., Miller, K., Rogerson, S. ACM, vol. 42, n. 10.

Nesta unidade, você estudou sobre os conceitos básicos de software e da Engenharia de Software. Na próxima unidade irá aprender mais sobre os Modelos prescritivos de processo de Desenvolvimento de Software.

É hora de se avaliar

Lembre-se de realizar as atividades desta unidade de estudo. Elas irão ajudá-lo a fixar o conteúdo, além de proporcionar sua autonomia no processo de ensino-aprendizagem.

Exercícios – Unidade 1

1. Softwares que são caracterizados por algoritmos de processamento numérico, dependentes de coleta e processamento de dados para diversas áreas do conhecimento são classificados como:

a) Software de Tempo Real

b) Software Embutido

c) Software Matemático

d) Software de Inteligência Artificial

e) Software Científico e de Engenharia

2. Complete a lacuna de forma adequada:

De acordo com a IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos), a Engenharia de Software é a aplicação de uma abordagem _____, disciplinada e quantificável no desenvolvimento, operação e _____ de software.

a) Sistemática – manutenção

b) Sistemática – entrega

c) Estruturada – implementação

d) Incremental – manutenção

e) Incremental – implementação

3. Sobre a evolução do software analise as assertivas a seguir e marque a alternativa correta.

I. O surgimento do conceito de biblioteca de software se deu na terceira era do desenvolvimento de software.

II. Os primeiros SGBD's se deram na primeira era do desenvolvimento de software, na década de 1960.

III. Na primeira era de desenvolvimento de software se tinha um processo considerado artesanal e produção de programas em lotes (batch).
IV. A quarta era do desenvolvimento de software foi marcada pelo surgimento das técnicas 4GT e dos ambientes cliente-servidor.
V. As chamadas "Software *Houses*" surgiram na segunda era do desenvolvimento de software.

Estão corretas, apenas e respectivamente:

a) I, II e III.

b) II, IV e V.

c) III, IV e V.

d) I, II e IV.

e) II, III e V.

4. A Engenharia de Software tem como um dos seus propósitos garantir que um software seja construído de forma que possa evoluir para satisfazer as diferentes necessidades dos clientes. Isso é uma característica essencial, pois a mudança no software é uma exigência inevitável. Com base nisso, assinale a alternativa que apresenta corretamente o nome dessa característica.

a) Adaptabilidade

b) Portabilidade

c) Aceitabilidade

d) Confiabilidade

e) Manutenibilidade

5. A Engenharia de Software consiste em:

a) Uma forma de programação de computadores em que todos os programas podem ser traduzidos às estruturas de sequência, decisão e repetição.
b) Utilizar de princípios de engenharia para a especificação, desenvolvimento e manutenção de sistemas de software.

c) Estudos de técnicas, ferramentas e métodos para desenvolvimento de sistemas de software por meio de modelos matemáticos e algoritmos.
d) Uma sequência de passos para realizar uma tarefa ou resolver um problema, utilizando recursos de softwares.
e) Representação gráfica do fluxo de dados por meio de um sistema de informação, fornecendo a visão estruturada do que será implementado.

6. Analise a definição dada a seguir e escolha a alternativa a qual ela se refere.

"É uma disciplina que se ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema, depois que ele entrou em operação. Seu principal objetivo é fornecer uma estrutura metodológica para a construção de software com alta qualidade".

a) Programação Orientada a Objetos
b) Engenharia de Software
c) Análise de Sistemas
d) Ciência da Computação
e) Ciclo de vida do software

7. Qual paradigma da Engenharia de Software concentra-se na capacidade de se especificar software a uma máquina em uma linguagem próxima a linguagem natural?

a) Técnicas de Quarta Geração
b) Documentos de Requisitos
c) Especificação de Requisitos
d) Paradigma Clássico
e) Modelo Espiral

8. O Modelo Espiral foi desenvolvido considerando as melhores características do modelo de ciclo de vida clássico e prototipação, acrescentando _____.

Assinale a alternativa que complete a lacuna corretamente.

a) A análise de viabilidade.

b) A análise estruturada.

c) A documentação técnica.

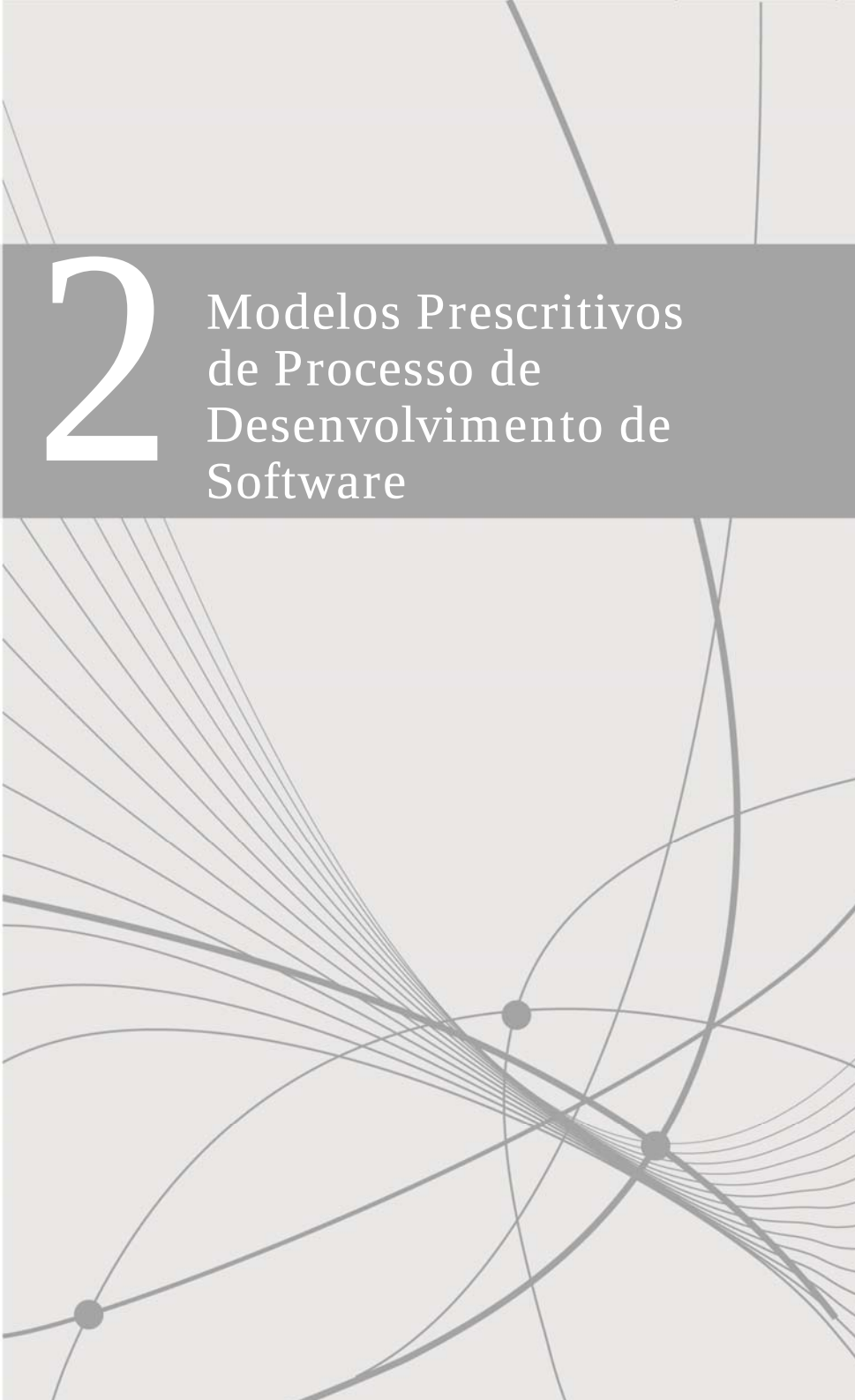
d) A análise de riscos.

e) O projeto estruturado.

9. Em um projeto, dependendo de sua complexidade e da quantidade de pessoas envolvidas, os interesses e entendimentos podem ser divergentes. E isto reforça a necessidade de minimizar tais divergências com uso de métodos, ferramentas e processos bem definidos e estruturados. Quais são as principais causas das divergências nos projetos solicitados pelos clientes e nos entregues pela equipe de desenvolvimento?

10. Apresente as ideias fundamentais na Engenharia de Software que se

2 Modelos Prescritivos de Processo de Desenvolvimento de Software



Caro(a) aluno(a),

Nesta unidade estudaremos os principais conceitos sobre o processo de desenvolvimento de Software. Como foi dito na unidade anterior, no início o desenvolvimento de software era desenvolvido de qualquer jeito, sem seguir critérios e padrões de qualidade e nenhum processo era aplicado de forma geral. Sabe-se, que isso trouxe muitas consequências, tais como, o não cumprimento dos prazos estabelecidos, falta de qualidade, muitos projetos falharam e nem chegaram a ser concluídos, entre outros. Nesta unidade você estudará sobre o que é um processo de desenvolvimento de software e conhecerá alguns modelos de processos já existentes.

Um modelo prescritivo de processo de desenvolvimento de software é um conjunto de elementos que incluem ações de engenharia de software, produtos de trabalho e mecanismos que garantam a qualidade e controle de modificações em cada projeto que são necessárias para o desenvolvimento de um software. A principal função dos modelos prescritivos é auxiliar, colocando métodos, processos e definindo uma estrutura inicial para melhorar o desenvolvimento de software.

Objetivos da unidade:

O objetivo é apresentar a ideia do processo de desenvolvimento de software, bem como alguns modelos prescritivos desse processo. Com isso, espera-se que ao concluir tal parte deste curso você seja capaz de:

- Compreender os conceitos e modelos de processo de software;
- Conhecer o modelo Cascata, suas características, aplicações, vantagens e desvantagens.
- Conhecer os modelos incrementais, entender suas aplicações, vantagens e desvantagens.
- Compreender sobre as técnicas de prototipação e suas aplicações.
- Compreender os modelos evolucionários, entender suas aplicações, vantagens e desvantagens.
- Compreender o modelo RAD, suas vantagens e desvantagens.

Plano da unidade:

- Modelo em Cascata
- Modelos Incrementais
- Prototipação
- Modelos Evolucionários
- Modelo RAD

Bons estudos!

Um processo de software é um conjunto de atividades relacionadas as quais são utilizadas para a produção de software. Um processo pode ser considerado um roteiro (guia) com o intuito de ajudar na criação de software (produto ou sistema) com qualidade satisfatória e cumprindo os prazos estabelecidos entre as partes interessadas. A principal importância de um processo está relacionada à estabilidade, controle e organização que se estabelece ao se executar uma atividade, a qual, sem devido direcionamento e controle, pode não ser efetuada dentro das expectativas e qualidade desejadas.

Apesar de existirem diversos processos de software distintos, sabe-se que basicamente tais processos devem incluir quatro atividades básicas, as quais são consideradas fundamentais de acordo com as principais referências da Engenharia de Software (SOMMERVILLE, 2011; PRESSMAN, 2011):

- Especificação do Software: definição das funcionalidades e restrições relacionadas ao seu funcionamento.
- Projeto e Implementação de Software: a produção do software deve seguir as definições especificadas. Portanto, ao analisar tal especificação, deve-se projetar "o que" e "como", que indica a forma na qual o software será implementado.
- Validação do Software: o software deve ser validado e verificado se foi implementado de acordo com a especificação. Ou seja, deve-se verificar se atende às demandas do cliente, dos usuários ou demais interessados (steakholders).
- Evolução do Software: ao se projetar e criar um software é importante pensar que, em algum momento, esse irá evoluir para atender à novas necessidades de mercado ou mudanças solicitadas pelo clientes.

De acordo com Sommerville (2011) os processos de software podem ser considerados complexos e, como todos os processos intelectuais e criativos, dependem de pessoas para idealizá-los, projetá-los e implementá-los. Importante também destacar que não existe um processo ideal e que seja um padrão para desenvolver qualquer tipo de software. Normalmente as organizações desenvolvem seus próprios processos, analisando as melhores práticas, encaixando

partes de processos diferentes que sejam mais adequados aos seus funcionários e modelos de negócio. Pois, um processo para ser efetivo precisa ser utilizado pelas pessoas e, não adianta uma organização adotar processos se as pessoas não o seguirão. Desta forma, os processos têm evoluído de forma a tirar melhor proveito das capacidades das pessoas, bem como são adaptados às características específicas do sistema em desenvolvimento.

Para que um processo seja efetivamente adaptável às necessidades da organização, aconselha-se em sua definição contar com a ajuda da equipe de análise, desenvolvimento, testes, gerentes, entre outros que possam colaborar com a definição ou construção de um software.

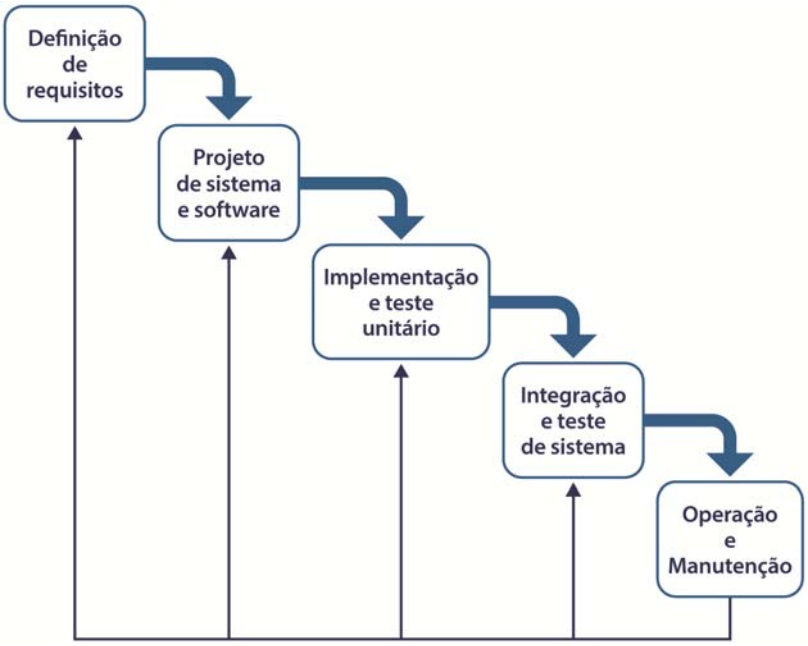
Modelo em Cascata

O modelo conhecido como "Cascata", devido ao encadeamento entre uma fase e outra, foi o primeiro modelo de processo de desenvolvimento de software a ser publicado (SOMMERVILLE, 2011). Encontra-se na literatura o modelo Cascata referido também como Ciclo de Vida Clássico/Tradicional ou Ciclo de Vida de Software. Tal modelo é considerado o mais simples dentre os modelos de desenvolvimento de software, estabelecendo uma ordenação linear no que se refere à realização de diferentes etapas.

Esse modelo, ilustrado na Figura 2.1, é um exemplo de processo dirigido a planos, ou seja, antes de começar a executar qualquer atividade, devem ser analisadas, planejadas e estruturadas todas as atividades.

O modelo cascata geralmente é utilizado quando os requisitos do problema em questão são conhecidos, estáveis e claros. Como por exemplo, ao se efetuar correções ou com a necessidade de aperfeiçoar um sistema já existente.

Figura 3.1: Modelo Cascata



Fonte: SOMMERVILLE, 2011, p. 20.

Como pode ser visto na Figura 2.1 este modelo sugere uma abordagem sistemática e sequencial para o desenvolvimento de software. As principais fases do modelo em cascata refletem diretamente as atividades consideradas fundamentais no desenvolvimento de software (SOMMERVILLE, 2011):

1. **Análise e Definição de Requisitos:** as funcionalidades, restrições e objetivos dos sistemas (requisitos) são estabelecidos por meio de consulta aos usuários, clientes e/ou especialistas. Basicamente esse levantamento de requisitos ou necessidades junto aos clientes consiste em determinar e delimitar o escopo de quais serviços devem ser fornecidos, quais as limitações e restrições e, por fim, o que se tem como meta ao desenvolver o sistema. Após o levantamento, o que foi requisitado é detalhado e funciona como uma especificação do sistema. Nesta etapa também, além da documentação, é realizado o estudo de viabilidade do projeto, que leva a determinação do processo de início do desenvolvimento do software.

2. **Planejamento e Projeto de Sistema e Software:** durante a fase do processo de desenvolvimento de software que se projeta os sistemas é definida uma arquitetura geral na qual se busca contemplar o que foi requisitado no processo de análise e definição dos requisitos. Compreende nesta etapa de planejamento a definição de cronogramas, com a estimativa de tempo de cada tarefa definida, as quais são baseadas nos requisitos estabelecidos. Também faz parte do processo de projeto a modelagem, a qual pode ser considerada uma prévia da próxima etapa de implementação (ou construção) do sistema. Na modelagem define-se as estruturas de dados, interfaces, arquitetura, ferramentas, etc.

3. **Implementação e Testes Unitários**: nessa fase do processo, cada unidade de programa ou conjunto de programas que compõem o projeto do software é desenvolvido. Também compreende parte do processo nessa fase a execução de testes unitários, que tem como foco a verificação de cada unidade para validar se essas atendem ao objetivo na quais foram especificadas.

4. **Integração e Testes de Sistemas:** as unidades desenvolvidas são integradas e testes também são realizados para validar a integração e assegurar que os requisitos foram implementados conforme especificados e verificarem a conformidade de acordo com a necessidade dos usuários. Após a execução desses testes o sistema pode ser implantado, ou seja, entregue ao cliente ou usuário final.

5. **Operação e Manutenção:** após o sistema ser entregue e começar efetivamente sua utilização, inicia-se a fase de manutenção (normalmente, mas não necessariamente, a fase mais longa do ciclo de vida). Em tal fase do processo ocorre o acompanhamento do uso do sistema, para que, se necessário, efetuar correções em erros que podem surgir ou terem passados despercebidos durante as fases anteriores, bem como, efetuar alguma melhoria nas funcionalidades que só foram percebidas no dia-a-dia do usuário, ou ainda, incrementar as funcionalidades, acrescentando novos requisitos que podem surgir com o tempo ou que foram desconsiderados previamente.

De forma geral, cada estágio gera um artefato, normalmente um ou mais documentos. Uma característica do modelo Cascata é que a fase subsequente só se inicia com a conclusão da fase anterior, mas na prática sabe-se que tais fases se sobrepõem e se interagem sendo uma para outra fonte de informações. Por exemplo, problemas com requisitos ou regras de negócio podem ser encontrados durante a fase de projeto ou então, durante a codificação problemas de projeto podem ser identificados, necessitando uma reavaliação do que foi planejado, e assim por diante.

São características do modelo cascata:

- Há uma divisão fixa, sequencial e inflexível do projeto em estágios distintos.
- A fase seguinte só deve iniciar quando a anterior for concluída e aprovada pelas partes envolvidas. O projeto só começa quando os requisitos forem completamente levantados, definidos, detalhados e aprovados.
- Oferece maior previsibilidade de prazos e custos e, por consequência, facilita o planejamento e gerenciamento.

Apesar de ser o paradigma mais antigo da Engenharia de Software, o modelo Cascata ainda é utilizado. Vários autores questionam sua eficácia e os principais problemas encontrados no modelo cascata são:

- Na realidade, os projetos raramente seguem um fluxo sequencial que o modelo Cascata pressupõe. A interação é sempre necessária quando o processo está sendo executado de fato e, então, cria-se problemas na aplicação do modelo.
- É difícil para o cliente conhecer e estabelecer de forma explícita todas as suas necessidades. O modelo Cascata é baseado nisso e suas maiores dificuldades estão em se adequar às incertezas que existem, de forma natural, no início dos projetos.
- Exige bastante paciência dos clientes, o que não costuma acontecer. No modelo cascata um versão para uso só estará disponível próximo ao final do projeto.

- Se houver algum erro grave nas etapas iniciais (como uma especificação mal compreendida ou mal especificada) e se este for identificado apenas nas etapas finais do projeto, isso pode levar a um resultado desastroso, como, em um caso extremo, a necessidade de descartar tudo e recomeçar todas as etapas.
- Pode ocorrer um bloqueio ou uma longa espera em alguns membros da equipe que precisam esperar que os outros completassem as suas tarefas que para que eles consigam dar sequência as atividades que estão realizando.

A realidade que é visível atualmente é que o ritmo está bastante acelerado no desenvolvimento de software e cada vez mais, com a evolução dos modelos de negócio, ferramentas e tecnologias, aumenta o potencial de ocorrer mudanças que surgem desde as necessidades do negócio, necessidades dos clientes ou até mesmo por exigências impostas por órgãos regulamentadores, leis governamentais ou imposições das organizações. Neste cenário o modelo Cascata se torna inapropriado, pois ele é útil em situações nas quais os requisitos são estáveis, fixos, bem conhecidos e o trabalho deve ser executado e finalizado de forma linear.

O modelo Cascata aplica-se bem em situações em que o sistema a ser desenvolvido é relativamente simples, os requisitos são bem conhecidos e não previsíveis de alterações grandes, a tecnologia utilizada é bem acessível e os recursos para o desenvolvimento estejam disponíveis.

Modelos Incrementais

O desenvolvimento incremental, de acordo com Sommerville (2011), é baseado na ideia de desenvolver uma implementação inicial, apresenta-las aos usuários ou clientes para que esses possam efetuar comentários e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido. Entende-se por sistema adequado aquele que atende aos requisitos do cliente e/ou dos usuários e os satisfaçam da melhor forma possível. É uma característica dos modelos que seguem

um desenvolvimento incremental que as atividades de especificação, desenvolvimento e validação sejam intercaladas (diferentes do sequencial que tem tais atividades separadas), com rápido *feedback* entre todas as atividades.

Destaca-se como vantagem do uso de um modelo incremental o fato que, se assemelha muito com a forma com que resolve geralmente um problema real: Sommerville (2011) explica que "desenvolvimento incremental reflete a maneira como resolvemos os problemas. Raramente elaboramos uma completa solução do problema com antecedência; geralmente movemo-nos passo a passo em direção a uma solução, recuando quando percebemos que cometemos um erro.". Além disso, desenvolver software de forma incremental facilita alterações, inclusões de funcionalidades, mudanças durante o desenvolvimento são mais fáceis e tem menor impacto, com isso podem ocorrer reduções no custo de desenvolvimento, comparado a execução das mesmas mudanças em uma abordagem sequencial.

Destacam-se, também, como vantagens do modelo incremental quando comparado ao modelo cascata:

- O custo de acomodar as mudanças nos requisitos de cliente é reduzido, diminuindo o retrabalho em alterar documentação ou em refazer o processo de análise.
- A obtenção de *feedback* com clientes é mais fácil, pois há uma maior participação desses desde o começo. Assim, os clientes podem fazer comentários sobre demonstrações do software e acompanhar o quanto já foi implementado. Apenas com documentações e relatórios de status *repport* os clientes sentem dificuldade de acompanhar a evolução do que está sendo feito.
- Mesmo se toda a funcionalidade não for concluída é possível obter uma versão de entrega e implementação rápida de um software útil ao cliente. Desta forma, os clientes podem utilizar antes do que em um processo Cascata.

O desenvolvimento incremental é uma das abordagens mais utilizada para o desenvolvimento de sistemas. Tal abordagem pode ser dirigida a planos, ágil, ou, mais comum, por meio de uma combinação dessas abordagens. Em uma abordagem dirigida a planos, os incrementos do sistema são identificados, previamente, se uma abordagem ágil for utilizada, os incrementos iniciais são identificados, mas o desenvolvimento de incrementos posteriores depende do progresso e das prioridades dos clientes (SOMMERVILLE, 2011).

Do ponto de vista de gerenciamento a abordagem incremental tem dois problemas:

1. Processo não é visível.
2. A estrutura do sistema tende a se degradar com a adição de novos incrementos.

Durante o processo de desenvolvimento de software incremental se faz necessário adaptar e refinar o sistema, e com isso, softwares grandes e complexos podem ser finalizados diferentes do que projetados inicialmente. Especialmente se as partes dos sistemas forem desenvolvidas por equipes diferentes. Outra desvantagem comum são as diversas alterações no escopo que podem surgir a cada incremento, aumentando ou diminuindo requisitos, ou até mesmo alterando funcionalidades já criadas.

O Modelo Incremental apresenta diversas vantagens para o desenvolvimento de um software, especialmente se os requisitos não estão claros inicialmente. Nesse modelo as versões do software são fornecidas após cada interação e, dessa forma, o usuário pode perceber mudanças necessárias antes da finalização do projeto. Além disso, é um modelo flexível e fácil de gerenciar, especialmente os riscos, pelo fato do cliente (usuário) participar a cada versão e validar se está de acordo com o que está sendo feito. As inconformidades são corrigidas antes da entrega da próxima versão, normalmente. Outras vantagens são:

- Quando se constrói algo menor o risco é reduzido se comparado a construir algo maior.

- Quando ocorre um erro em um incremento, os demais (anteriores) nem sempre são diretamente afetados. Em casos de erros graves, apenas o último incremento seria descartada (pior caso) ou ajustada.
- Reduzindo o tempo de desenvolvimento de um sistema é esperado que se reduzisse as chances de mudanças nos requisitos do usuário no final, visto que esse já participa validando cada versão.
- A quantidade de análise e documentação a ser refeita é menor quando comparada ao modelo de desenvolvimento em cascata.
- Nesse modelo é previsto que os clientes (usuários) façam comentários/feedbacks sobre o que foi desenvolvido. Normalmente as pessoas têm dificuldades de avaliar a evolução apenas por meio de documentos de projeto de software.

Prototipação

De acordo com Sommerville (2011) um protótipo "é uma versão inicial de um sistema de software, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções". Um das principais motivações do uso de protótipos é para ajudar a entender a real necessidade do cliente, auxiliar no levantamento de requisitos e validar requisitos já estabelecidos. Também é bastante utilizado para estudar soluções específicas de software e para estudar a interface projetada para avaliar se atende ao usuário de forma satisfatória.

Espera-se que o desenvolvimento de um protótipo seja rápido, isso é essencial para que os custos sejam controlados e os usuários possam fazer experiências com o protótipo no início do processo de software. Os protótipos de sistema permitem que os usuários realizem experiências para ver como o sistema pode apoiar seu trabalho. Com o uso de protótipos os usuários podem identificar pontos positivos e negativos do software antes dele ser de fato implementado, a partir daí sugerir alterações, exclusões e inclusões, diminuindo custo e o impacto de alterações se comparado a um modelo cascata.

O protótipo pode revelar erros e omissões nos requisitos propostos. Isso é muito importante, pois muitas vezes, em uma especificação em linguagem natural uma funcionalidade pode parecer bem definida e em palavras, pode parecer mais útil do que realmente é.

Os protótipos podem ser disponibilizados em diversas formas/tipos:

- ***Wireframes* e Rascunhos:** protótipos de baixa fidelidade, rápidos para se desenvolver e modificar. Esses protótipos não mostram detalhes visuais ou interações de tela, mas ajudam a validar regras de negócio e requisitos. Protótipos feitos com papel e caneta são exemplos desse tipo.
- **Visuais**: normalmente criados com softwares de edição gráfica, tais protótipos tem apelo mais visual, mas nem sempre possuem interações com tela e demandam mais tempo para fazer ajustes e melhorias. São indicados quando se precisar dar ênfase em estética e usabilidade, e quando os requisitos já foram estendidos. Softwares como *Photoshop* e *Corel Draw* podem ser utilizados para desenvolvê-los.
- **Interativos**: são protótipos mais completos e representativos. Além da parte visual podem ser incluídos detalhes interativos para proporcionar uma experiência mais real e ativa com o usuário. Porém, esse tipo de protótipo demanda uma equipe com maior conhecimento técnico e demoram muito mais tempo para serem criados ou alterados. Podem ser feitos em HTML, CSS, *Javascript*, *CSS Twitter Bootstrap*, *Adobe Dreamweaver*, entre outras ferramentas.

A experiência adquirida no desenvolvimento do protótipo é muito útil nas etapas posteriores do desenvolvimento do sistema real, permitindo reduzir o seu custo e resultando num sistema.

No processo de desenvolvimento de Software podem ter dois tipos de prototipação:

- **Prototipação evolucionária:** uma abordagem para o desenvolvimento do sistema na qual o protótipo é produzido e refinado, durante várias etapas, até chegar ao sistema final. Nessa abordagem o desenvolvimento do protótipo começa com os requisitos de melhores compreendidos.

- **Prototipação descartável**: uma implementação prática do sistema é produzida para ajudar a identificar os problemas com os requisitos e depois é descartado. Nessa abordagem o desenvolvimento do protótipo começa com os requisitos que ainda não estão bem compreendidos, para que possam ser validados.

Uma das desvantagens do uso de protótipos é que em longo prazo a manutenção pode ser cara, pois a continuidade de mudanças tende a corromper a estrutura do protótipo do sistema. Outra coisa é que, as vezes, para se construir um protótipo são necessárias habilidades de especialistas, os quais podem não estar disponíveis (questões contratuais/valores) na equipe de desenvolvimento.

Modelos Evolucionários

Modelos evolucionários de desenvolvimento de software se baseiam na ideia de que um processo no qual o software deve ser desenvolvido de forma a evoluir a partir de uma implementação inicial, que pode ser um protótipo. Tal abordagem também expõe os resultados aos comentários de usuários, que são refinados gerando verões até que seja desenvolvido o sistema adequado. Essa abordagem é mais indicada para sistemas de pequeno e médio porte.

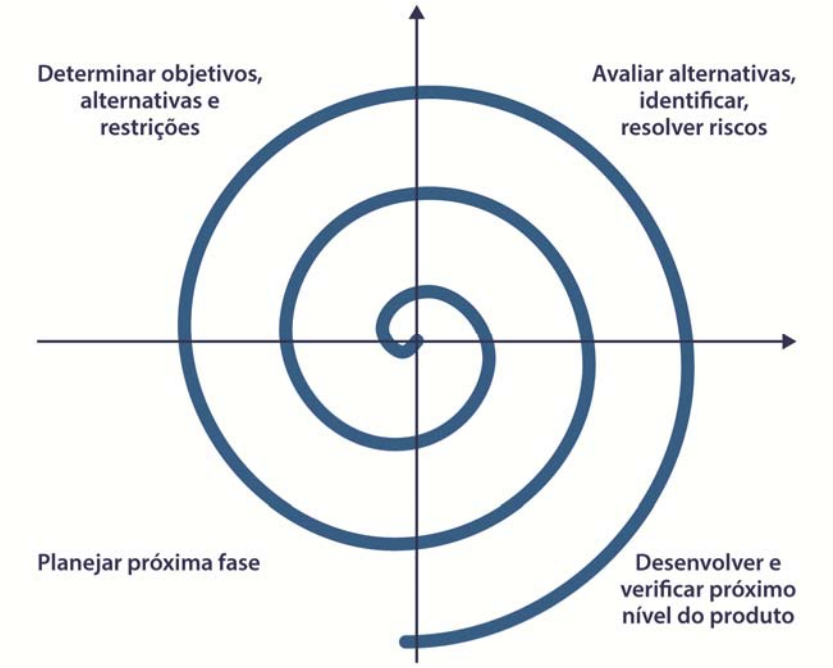
O modelo espiral proposto pro Boehm (1988) é um modelo de processo de software evolucionário. O processo de software nesse modelo é representado como se fosse uma espiral, e não como uma sequência de atividades como em outros modelos. Cada volta na espiral representa uma fase do processo de desenvolvimento de software. Assim, a volta mais interna pode ser um protótipo para validar a viabilidade do sistema, por exemplo; o ciclo seguinte voltado para a definição dos requisitos; o seguinte para o projeto do sistema e assim por diante (Sommerville, 2011).

O modelo espiral foi desenvolvido combinando as melhores práticas e características dos modelos lineares e prototipação. Porém, como destaque nesse modelo, foi acrescentado um novo recurso: a análise de riscos. Entende-se como risco qualquer coisa que possivelmente ocasionará algum erro ou problema durante o desenvolvimento de software.

A principal diferença entre o modelo espiral e outros modelos de processo de software é o reconhecimento explicito do risco. Um ciclo espiral começa com a definição de objetivos, como desempenho e funcionalidade. Em seguida, são enumeradas formas alternativas de alcançar tais objetivos e de lidar com as restrições de cada um deles. Cada alternativa é avaliada em função de cada objetivo, e as fontes de risco do projeto são identificadas. O próximo passo é resolver esses riscos por meio de atividades de coleta de informações, como análise mais detalhada, prototipação e simulação.

A Figura 2.2 apresenta o Modelo Evolucionário proposto por Boehm.

Figura 2.2: Modelo em espiral de processo de software de Boehm.



Fonte: Adaptado de SOMMERVILLE, 2011, p. 33

De acordo com Sommerville (2011), cada volta da espiral é dividida em quatro setores:

1. Definição de objetivos: objetivos específicos para essa fase do projeto são definidos; restrições ao processo e ao produto são identificadas e um plano de gerenciamento detalhado é elaborado; os riscos são identificados. Estratégias alternativas podem ser planejadas em função desses riscos.
2. Avaliação e Redução de Riscos: para cada um dos riscos identificados do projeto é feita uma análise detalhada e, então medidas são tomadas para a redução dos riscos.
3. Desenvolvimento e validação: após avaliação dos riscos é selecionado um modelo de desenvolvimento para o sistema.
4. Planejamento: o projeto é revisado e uma decisão é tomada a respeito da continuidade do modelo com mais uma volta da espiral. Caso seja decidido pela continuidade, planos são elaborados para a próxima fase do projeto.

São consideradas vantagens desse modelo espiral (evolucionário):

- Estimativas são mais reais.
- Maior versatilidade para lidar com mudanças.
- Mais fácil de testar e de definir quando testar.
- Melhora no tempo de implementação do sistema.
- Não há distinção do que é desenvolvimento e o que é manutenção.

Já as desvantagens são:

- Muita ênfase na parte funcional.
- Avaliação dos riscos exige experiência.
- Sua aplicação é melhor somente em sistemas de larga escala.
- Modelo relativamente novo, porém não muito utilizado.

Modelo RAD

Rapid Application Development (RAD) ou, em Português, Desenvolvimento Rápido de Software, é um modelo de desenvolvimento de software que possui um processo sequencial e linear que enfatiza um ciclo de desenvolvimento de projetos curtos (até 3 meses). Neste modelo o projeto deve ser fracionado em subprojetos e direcionado a equipe de desenvolvimento, que utiliza uma base de componentes com o intuito de agilizar a construção do produto.

São pontos positivos do RAD:

- Enfatiza um ciclo de desenvolvimento curto (60 a 90 dias);
- Cada módulo pode se direcionado a uma equipe de desenvolvimento e depois, completados e juntos, formarão um todo.

São considerados como pontos negativos:

- Se o projeto for grande, por exemplo, o número de equipes crescerá demais e a atividade de integração será muito complexa;
- A organização tem de ter recursos humanos suficientes para acomodar as várias equipes (alto custo);
- O sistema deve ser passível de modularização de modo que cada função principal seja completada em menos de 90 dias, senão o uso é desaconselhado.
- Não é aconselhável o uso se houver riscos técnicos consideráveis, por exemplo com novas tecnologias ou quando o software exige alto grau de interoperabilidade com programas já existentes.
- Clientes e desenvolvedores precisam estar comprometidos com as atividades do processo a fim de finalizar a construção do produto num prazo curto (sendo um risco). O envolvimento com o usuário tem que ser bastante ativo para a RAD funcionar como se propõe.

- Prazo curto pode impactar na qualidade do que está sendo desenvolvido, exigindo maior habilidade de quem está desenvolvendo, testando ou dando suporte.
- Projetos RAD são mais difíceis de acompanhar e normalmente não são utilizados métodos formais (diminuindo a precisão científica).

O processo RAD é apropriado quando a aplicação é do tipo "*stand alone*", ou seja, programas autossuficientes, que não necessitam de software auxiliar, como um interpretador, por exemplo. Outro fator também que viabiliza o uso desse modelo são softwares em que se pode utilizar API's (componentes prontos ou classes pré-existentes) ou se a distribuição do produto é pequena e com âmbito do projeto restrito.

Nesta unidade você estudou sobre os Modelos prescritivos de processo de Desenvolvimento de Software. Na próxima unidade irá aprender mais sobre os Modelos Ágeis de Desenvolvimento de Software.

É hora de se avaliar

Lembre-se de realizar as atividades desta unidade de estudo. Elas irão ajudá-lo a fixar o conteúdo, além de proporcionar sua autonomia no processo de ensino-aprendizagem.

Exercícios – Unidade 2

1. O modelo cascata é um dos paradigmas mais antigos da engenharia de software. Este ciclo de desenvolvimento de software, também conhecido como ciclo de vida clássico, é constituído pelas seguintes fases, em ordem de execução:

a) Levantamento de Requisitos, Implementação, Testes e Implantação.

b) Comunicação, Projeto, Codificação, Testes e Implantação.

c) Comunicação, Planejamento, Modelagem, Construção e Implantação.

d) Planejamento, Execução, Verificação, Implementação e Homologação.

e) Definição dos Requisitos, Projeto, Implementação, Testes e Manutenção.

2. Qual das seguintes características não se aplica a um processo de desenvolvimento de software baseado no Modelo em Cascata?

a) É adequado para projetos em que os requisitos já estão bem definidos e estruturados.
b) Representa um processo linear, no qual as atividades de uma fase só iniciam a partir da finalização das atividades da fase imediatamente anterior.
c) Devido aos incrementos é possível receber feedback dos clientes no final de cada fase.
d) Representa um processo dirigido a planejamento, no qual as atividades de cada fase são definidas com antecedência.
e) É adequado para projetos em que os requisitos não tendem a sofrer mudanças.

3. O modelo em Cascata é caracterizado por:

a) Possuir uma tendência de progressão sequencial.

b) Dividir o desenvolvimento do software em ciclos.

c) Descartar a fase de levantamento de requisitos.

d) Produzir uma versão executável ao final de cada fase.

e) Utilizar um processo de desenvolvimento ágil.

4. Dadas as seguintes informações, assinale a alternativa que apresenta o modelo de processo de desenvolvimento de software mais adequado:

1. Nesse projeto os requisitos tendem a mudar constantemente.
2. O cliente é ansioso para ver resultados e gostar de avaliar progressos constantemente.
3. As necessidades do negócio mudam constantemente.
4. Deve-se priorizar o desenvolvimento de versões cada vez mais completas.

a) Cascata

b) Evolucionário

c) RAD

d) Modelo em V

e) Scrum

5. A respeito de prototipação na engenharia de software, assinale a opção correta.

a) Tanto a prototipação evolucionária quanto a descartável apoiam o gerenciamento da qualidade dos sistemas.
b) Na impossibilidade de serem especificadas interfaces utilizando-se um modelo estático, as interfaces com o usuário devem ser desenvolvidas com a prototipação.

c) Os protótipos são muito úteis na validação de interfaces, mas não contribuem efetivamente na identificação e validação dos requisitos do sistema.
d) Os usuários não podem ser capacitados com o uso de protótipos, sejam estes do tipo evolucionário ou descartável.
e) Protótipos descartáveis precisam ser executáveis para terem utilidade no processo de engenharia de requisitos.

6. Dada as seguintes características assinale a alternativa que apresenta o modelo de processo de software descrito:

1. Construção baseada em componentes.
2. Desenvolvimento curto (máximo de 90 dias).
3. Desenvolvimento de software incremental.

a) Espiral
b) Evolucionário
c) Cascata
d) Incremental
e) Modelo RAD

7. Na engenharia de software, o modelo de processo de desenvolvimento de software incremental que enfatiza um ciclo de desenvolvimento extremamente curto é conhecido como modelo:

a) RAD
b) Espiral
c) Linear
d) CMM
e) Protótipo

8. Os modelos de processo em que o sistema é dividido em pequenos subsistemas funcionais que, a cada ciclo, são acrescidos de novas funcionalidades são denominados:

a) Lineares

b) Incrementais

c) Sequenciais

d) Unificados

e) Clássicos

9. Explique o modelo espiral proposto por Boehm, destacando os quatro quadrantes da divisão da espiral.

_____

_____

_____

_____

_____

_____

10. Explique quais as vantagens de se adotar um processo de desenvolvimento de software incremental.

_____

_____

_____

_____

_____

_____

3 Modelos Ágeis de Desenvolvimento de Software

Caro(a) aluno(a),

Nesta unidade estudaremos os principais conceitos sobre os Métodos Ágeis de Desenvolvimento de Software. Era comum de se pensar na década de 1980 e no início de 1990 de que a melhor maneira de conseguir um software de qualidade era por meio de um planejamento cuidadoso e detalhado do projeto, com uma formalização da qualidade da segurança, com uso de métodos de análise e projeto apoiado por ferramentas CASE (*Computer-aided Software Engineering*) e com o processo de desenvolvimento de software rigoroso e controlado. Tal percepção surgiu da comunidade de engenharia de software, responsável pelo desenvolvimento de sistemas de software grandes e duradouros, como sistemas aeroespaciais e de governo.

Objetivos da unidade:

O objetivo dessa unidade é apresentar conceitos sobre os Métodos Ágeis de Desenvolvimento de Software. Com isso, espera-se que ao concluir tal parte deste curso você seja capaz de:

- Compreender sobre o manifesto ágil e as diferenças entre desenvolvimento ágil e desenvolvimento dirigido a planos.
- Conhecer melhor sobre o *Extreming Programming* (XP) e *Scrum* para gerenciamento ágil de projetos.
- Conhecer a Metodologia Crystal, FDD, DSDM, ASD.

Plano da unidade:

- O Manifesto ágil
- Extreming Programming (XP)
- Scrum e Agile
- Metodologias Crystal
- FDD, DSDM, ASD

Bons estudos!

O Manifesto Ágil

Em 2001 foi assinado o "Manifesto para o Desenvolvimento Ágil de Software" (*Manifest for Agile Software Development*) por Kent Beck e outros 16 desenvolvedores, autores e consultores, que ficaram conhecidos como "*Agile Alliance* - Aliança dos Ágeis". Tal manifesto tinha como premissas (PRESSMAN, 2011):

- Indivíduos e interações acima de processos e ferramentas.
- Software operacional acima de documentação completa.
- Colaboração dos clientes acima de negociação contratual.
- Respostas a mudanças acima de seguir um plano.

O Manifesto Ágil reconhece que a utilização de processos, ferramentas, documentação, contratos e planos pode ser importante para o sucesso do projeto. Porém, valorizar os indivíduos e interações mais que processos e ferramentas levam em conta que os indivíduos que são os responsáveis por gerar os produtos, com suas habilidades e talento. Atividades que dependam de um ser humano para serem executadas então, o valor devem ser atribuído as questões humanas para que se tenha sucesso.

Os métodos ágeis se desenvolveram baseado em percepções reais de situações consideradas "fraquezas" ou problemas na engenharia de software convencional. Com as constantes mudanças no mercado e nos modelos de negócio, não é possível prever como uma aplicação (software) irá evoluir com o tempo. Com tantas mudanças, as necessidades do usuário se alteram e novas ameaças competitivas surgem a todo o momento (PRESSMAN, 2011). Devido ao ambiente fluído de negócios e as alterações das necessidades do usuário nem sempre é possível levantar corretamente os requisitos antes que se inicie o desenvolvimento do projeto.

Mas o que é agilidade? Agilidade no contexto de engenharia de software vai além de uma resposta rápida a mudanças. A agilidade propõe e incentiva uma estruturação de equipes e que os membros tenham atitudes que tornam a comunicação mais fácil (não só entre eles, mas também com todos os demais interessados no projeto). Agilidade também enfatiza a entrega rápida do software operacional e diminui a importância dos artefatos intermediários. Outro fato é que quando se fala em agilidade no contexto de desenvolvimento de software até mesmo o próprio cliente é visto como parte da equipe de desenvolvimento (ele conhece o negócio e as regras, testa e valida).

A *Agile Aliance* estabelece 12 princípios de agilidade para quem quiser ter agilidade (PRESSMAN, 2011):

1. A maior prioridade é satisfazer o cliente! Faça entregas adiantadas e contínua de software.

- Mas a documentação não é importante? Sim, a documentação é importante. Mas, é importante entender que o cliente não se importante com documentos e diagramas modelados com UML. Para o cliente o importante é o que você está entregando ou não do software para ele a cada ciclo de implantação. Entregar valor para o cliente é algo que é mais fácil de falar do que fazer, pois o que o cliente deseja nem sempre vai de encontro com o processo que está sendo seguido consegue fazer.

2. Mesmo com o desenvolvimento atrasado receba bem os pedidos de mudanças. Os processos ágeis se aproveitam das mudanças como uma vantagem competitiva na relação com o cliente.

- As constantes mudanças nas tecnologias e modelo de negócio causam podem ser vistas tanto quanto ameaças a serem evitadas ou como oportunidades a serem abraçadas. Sempre que surgir alguma alteração a ser feita ou mudança de escopo, ao invés de resistir a elas, a proposta do desenvolvimento ágil é encontrar uma forma de acomodá-las de maneira mais fácil e eficiente possível, mas ciente das consequências.

3. Entregue software em funcionamento frequentemente, em intervalos curtos.

- Entregas interativas e incrementais são essenciais em projetos ágeis. Aconselha-se um rápido ciclo interno de entregas que permite que todos avaliem e aprendam com o produto em crescimento.

4. Equipe de negócio e desenvolvedores devem trabalhar em conjunto diariamente durante todo o projeto.

5. Construa projetos com uma equipe motivada. Para isso é importante que se dê a eles um ambiente adequado e o apoio necessário e, também, que se confie neles para ter o trabalho bem feito.

- Mesmo om todas as ferramentas, tecnologias e processos, no final são as pessoas que fazem a diferença. Sem pessoas para utilizar as ferramentas, tecnologias e processos não servem para nada. É importante maximizar o fator humano e ganhar a confiança da equipe.

6. Uma conversa aberta e de forma presencial é o método mais eficiente e efetivo de transmitir informações para uma equipe de desenvolvimento.

- "Conhecimento tácito não pode ser transferido extraindo-o da cabeça das pessoas para o papel", escreveu Nancy Dixon em *Common Knowledge* (*Hardvard Business School Press*, 2000)." Conhecimento tácito não é somente fatos, mas relacionamentos entre fatos, isto é, a maneira que as pessoas podem combinar certos fatos para lidar com uma situação específica.

7. Software em desenvolvimento é a principal medida de progresso/evolução.

- Normalmente se vê equipes de projetos que não percebem os riscos até pouco ponto antes da entrega.

8. Os processos ágeis promovem desenvolvimento sustentável. É necessário que todos os envolvidos esteja capacitados para se manter um ritmo constante.

- É mais comum do que deveria ser, ver em projetos pessoas trabalhando noites e finais de semana tentando desfazer os erros dos planejamentos mal feitos. E a agilidade depende de pessoas atentas e criativas e que conseguem manter a atenção e a criatividade durante todo o processo.

Desenvolver de forma sustentável significa encontrar um ritmo de trabalho que a equipe consiga sustentar durante todo o tempo e permanecer saudável.

9. Agilidade é aumentada quando se tem atenção contínua para com a excelência técnica e para com bons projetos.

- Abordagens ágeis enfatizam a qualidade da solução porque para se manter a qualidade do projeto é essencial se ter uma solução de qualidade

10. Simplicidade é essencial!

- Os métodos ágeis sugerem o uso de abordagens simples, as quais são mais fáceis de usar. É mais fácil alterar algo em um processo simples do que em um complexo.

11. As melhores arquiteturas, requisitos e projetos surgem de equipes que se auto organizam.

- Os melhores projetos, tanto arquiteturas quanto requisitos, surgem do desenvolvimento iterativo e uso ao invés de planos antecipados. Outro ponto é que quando se precisa fazer algo com certa emergência equipes auto organizadas com alta interatividade e poucas regras de processo, fazem melhor.

12. Intervalos regulares, a que se avalia para ver como tornar-se mais eficiente, então sintonizam e ajustam seu comportamento de acordo.

- Qualquer equipe ágil deve parar para refletir durante todo o processo em busca de uma constante melhoria nas práticas de acordo com cada situação.

Importante

Importante destacar que nem todo modelo de processo ágil aplica esses 12 princípios de forma igual e, alguns modelos relevam a importância de alguns deles.

Defensores do desenvolvimento de software ágil dizem que "o desenvolvimento ágil foca talentos e habilidade de indivíduos, moldando o processo de acordo com as pessoas e as equipes específicas". São características de indivíduos que atuam em um processo ágil:

- Competência: ter aptidão, conhecimento e capacidade para executar, cumprir e/ou concluir uma determinada tarefa ou função.
- Foco comum: os indivíduos devem ter foco unânime e um objetivo em comum.
- Colaboração: é a capacidade executar um trabalho feito em comum com uma ou mais pessoas; também significa cooperação, ajuda e auxílio para concluir determinada tarefa.
- Habilidade na tomada de decisão: processo pelo qual se escolhe um plano de ação dentre vários disponíveis (baseados em diversos fatores e ambientes) para uma situação específica.
- Habilidade de solução de problemas confusos: capacidade de lidar com problemas em diversos níveis de complexidade e confusão, propondo soluções adequadas a cada um deles.
- Confiança mútua e respeito: a confiança mútua e o respeito são características essenciais para fortalecer as relações e tornar os ambientes mais produtivos.
- Auto-organização: é um processo dinâmico e adaptativo de se organizar para adequar a determinada situação.

Extreming Programming (XP)

O *Extreme Programming* (XP) é considerado o método ágil mais conhecido e o mais utilizado. Sua ideia central, que justifica seu nome de "extremo", por exemplo, é que várias novas versões de um sistema de um sistema podem ser desenvolvidas, integradas e testada em um único dia por programadores.

Em XP os requisitos são expressos como cenários (chamados de estórias do usuário), que são implementados diretamente como uma série de tarefas. Os programadores normalmente trabalham em pares e desenvolvem testes para cada tarefa antes de escreverem o código. Quando um novo código é integrado ao sistema, todos os testes devem ser executados com sucesso. Há um curto intervalo entre os releases do sistema.

A Figura 3.1 apresenta o processo XP para a produção de um incremento de um sistema.

Figura 3.1: O ciclo de um release em Extreme Programming.

Fonte: Sommerville, 2011, p. 44.

Extreme Programming envolve uma série de práticas que refletem os princípios dos métodos ágeis (SOMMERVILLE, 2011):

1. O desenvolvimento incremental é sustentado por meio de pequenos e frequentes releases do sistema. Os requisitos são baseados em cenários ou em simples estórias de clientes, usadas como base para decidir a funcionalidade que deve ser incluída em um incremento do sistema.
2. O envolvimento do cliente é sustentado por meio do engajamento contínuo do cliente com a equipe de desenvolvimento. O representante do cliente participa do desenvolvimento e é responsável por definir os testes de aceitação para o sistema.
3. Pessoas — não processos — são sustentadas por meio de programação em pares, propriedade coletiva do código do sistema e um processo de

desenvolvimento sustentável que não envolve horas de trabalho excessivamente longas.

4. As mudanças são aceitas por meio de releases contínuos para os clientes, do desenvolvimento *test-first*, da refatoração para evitar a degeneração do código e integração contínua de nova funcionalidade.

5. A manutenção da simplicidade é feita por meio da refatoração constante que melhora a qualidade do código, bem como por meio de projetos simples que não antecipam desnecessariamente futuras mudanças no sistema.

Kent Back, um dos percursores do XP, define um conjunto de cinco valores que estabelecem as bases para todo trabalho realizado como parte da XP:

- **Comunicação**: XP enfatiza a colaboração estreita (verbal), entre clientes e desenvolvedores, *feedback* contínuo e evitar documentação volumosa.
- **Simplicidade**: XP restringe os desenvolvedores a projetar apenas para as necessidades imediatas ao invés de necessidades futuras. O intuito é criar um projeto simples que possa ser facilmente implementado em código.
- ***Feedback*** (realimentação ou retorno): no XP o *feedback* provém de 3 fontes - do próprio *software* implementado (usam testes unitários e de aceitação), do cliente e de outros membros da equipe.
- **Coragem** (disciplina): o projeto deve ser feito para hoje, reconhecendo que necessidades futuras podem mudar e exigir uma quantidade substancial de retrabalho em relação ao projeto e ao código implementado.
- **Respeito**: Conforme consegue entregar com sucesso incrementos de software, a equipe desenvolve cada vez mais respeito pelo processo XP.

O Processo da XP envolve uma abordagem de utilizar o paradigma de desenvolvimento orientado a objetos e um conjunto de quatro atividades metodológicas: planejamento, projeto, codificação e testes. As principais atividades do XP são:

- **Planejamento**: levantamento de requisitos que capacita os membros técnicos da equipe XP a entender o ambiente de negócio do software e possibilita uma percepção mais ampla sobre os fatores principais e funcionalidades. Tal levantamento leva a criação de um conjunto de "histórias" (histórias de usuários) que descreve o resultado, as características e as funcionalidades que são requisitos para a construção do software. Conforme o desenvolvimento vai ocorrendo, o cliente pode acrescentar histórias, mudar a prioridade (valor) das já existentes, dividir ou até mesmo eliminá-las.

- **Projeto**: O projeto XP segue rigorosamente o princípio KIS (*Keep it simple*): "Preserve a simplicidade". O XP incentiva o uso de cartões CRC (classe-responsabilidade-colaborador) para identificar e organizar as classes orientadas a objetos relevantes para o incremento de software que está sendo desenvolvido no momento. Se um problema mais complexo de projeto for encontrado como parte o projeto de uma história, a XP recomenda a criação imediata de um protótipo operacional dessa parte do projeto. O objetivo é reduzir o risco para quando se for de fato implementar e validar as estimativas originais para a história contendo o problema de projeto.

- **Codificação**: Após desenvolver as histórias e elaborar o projeto, a próxima etapa é desenvolver os testes de unidade correspondente a cada uma das histórias a ser inclusas na versão corrente. No XP é recomendada a programação em dupla, em que duas pessoas trabalham juntas em uma mesma estação de trabalho para criar código para uma história. Isso tem o intuito de manter os desenvolvedores focados no problema em questão e duas pessoas pensando ao mesmo tempo, com revisão de código, espera-se melhoria na qualidade do que está sendo feito. Na prática cada uma das pessoas assume um papel diferente, por exemplo, enquanto um pensa nos detalhes da codificação de determinada parte do projeto a outra assegura que padrões de codificação sejam seguidos. Assim que uma dupla termina a codificação, o código é integrado ao trabalho de outros (em alguns casos há uma equipe específica para tal

integração). A estratégia de integração contínua ajuda a evitar problemas de compatibilidade.

- **Testes**: A criação de testes de unidade antes de iniciar a codificação é um elemento-chave na abordagem XP. A ideia é que a metodologia utilizada para tais testes permita que o processo seja automatizado (executados de forma repetitiva facilmente). No XP os testes de integração e validação do sistema ocorrem diariamente, o que dá a equipe XP uma indicação contínua do progresso e também permite alertar caso algo não esteja indo bem no início. "Corrigir pequenos problemas em intervalos de poucas horas leva menos tempo do que corrigir problemas enormes próximo ao prazo de entrega (PRESSMAN, 2011)". Os testes de aceitação (testes de cliente) são especificados pelo cliente e mantêm o foco nas características e na funcionalidade do sistema total.

SCRUM e Agile

Concebido no início da década de 1990 por Jeff Sutherland e sua equipe, o *Scrum* é um método de desenvolvimento ágil com princípios consistentes com o manifesto ágil. O *Scrum* é uma das metodologias mais difundida devido ao seu formato dinâmico de como as etapas do projeto são desenvolvidas. Alguns autores citam o *Scrum* como um framework para organizar e gerenciar trabalhos complexos. Apoiam essa ideia dizendo que o *Scrum* não é um processo ou uma técnica para construir produtos e sim um framework dentro do qual se podem empregar várias técnicas e processos.

Para entender melhor sobre o *Scrum* algumas definições são importantes:

- ***Sprint***: no *Scrum* os projetos ao divididos em ciclos conhecidos com *Sprints*. O Sprint representa um espaço de tempo no qual conjuntos de atividades devem ser executadas.
- ***Sprint Planning*:** reunião de planejamento realizada no início de cada Sprint. Nesta reunião é definida como o trabalho da equipe será realizado dentro do período estabelecido.

- ***Product Backlog*:** lista de funcionalidades a serem implementadas.
- ***Product Owner*:** atua como "dono" do projeto, sendo o responsável por definir as prioridades a serem desenvolvidas em cada Sprint e, também, é o responsável por intermediar a área de negócio com a equipe *Scrum*.
- ***Scrum Master*:** atua como um líder que assegura que a equipe siga a metodologia *Scrum*. É também responsável por remover obstáculos que possam prejudicar o desenvolvimento realizado pela equipe, ajudando-os a concretizar suas tarefas com a melhor performance possível.
- ***Scrum Team*:** é a equipe de desenvolvimento que atua em um projeto *Scrum*.
- ***Sprint Backlog*:** é uma lista de tarefas que o *Scrum Team* se compromete a executar dentro de um Sprint. Os itens dessa lista são extraídos do *Product Backlog* com base nas prioridades definidas pelo *Product Owner*.
- ***Daily Scrum:*** reunião diária para cada integrante da equipe dizer o que fez no dia anterior, o que tem programado para fazer no dia e se tem algum impedimento. Normalmente não ultrapassa 15 minutos.
- **Sprint Review:** momento em que é demostrado o que foi construído no Sprint.
- **Sprint Retrospective:** momento em que a equipe se reunem para refletir sobre os aprendizados que tiveram no Sprint (tanto positivos como negativos).

Basicamente o ciclo do *Scrum* é no começo de cada Sprint se realiza um Sprint *Planning Meeting*, no qual o *Product Owner* avalia os itens do *Product Backlog* e junto com sua equipe seleciona quais atividades serão incluídas no Sprint que está se iniciando. As tarefas selecionadas passam para o *Sprint Backlog*. Todos os dias a equipe se reúne no *Daily Scrum*.

O método *Scrum* pode ser usado para diversos tipos de projetos, mas ele tem mais sentido em ser utilizado quando se há um produto concreto sendo produzido. A ideia é que as pessoas possam resolver problemas complexos utilizando um framework leve, e simples de entender.

O *Scrum* se fundamenta em 3 pilares do empirismo:

- **Transparência**: dois pontos são importantes quando se fala em transparência. (1) Garantir que os aspectos significativos do processo estejam visíveis a todos os interessados. (2) Garantir que todos falem a "mesma língua", de forma que o entendimento sobre o processo e o que está sendo visto por todos seja o mesmo.
- **Inspeção**: inspecionar deve ser uma tarefa frequente no processo *Scrum*. Mas tal frequência tem que ser cautelosa a ponto de não atrapalhar a própria execução das tarefas. Tais inspeções trazem mais benefícios quando realizadas por inspetores especializados.
- **Adaptação**: Se em algum momento foi identificado que o que está sendo feito precisa ser ajustado (por exemplo, se alguma inconformidade foi encontrada durante uma inspeção) o processo ou o que está sendo produzido deve ser ajustado. Para minimizar impactos, tais ajustes devem ser realizados o mais cedo possível.

Metodologias Crystal

Alistair Cokburn e Jim Highsmith criaram a família Crystal de métodos ágeis que possuem foco em adaptabilidade (*maneuverability*). Para conseguir adaptabilidade, foi definido um conjunto de metodologias com elementos essenciais comuns a todas, mas com papéis, padrões de processos, produtos de trabalho e prática únicos para cada uma delas. As metodologias Crystal são um conjunto de exemplos de processos ágeis que provaram ser efetivos para diferentes tipos de projetos.

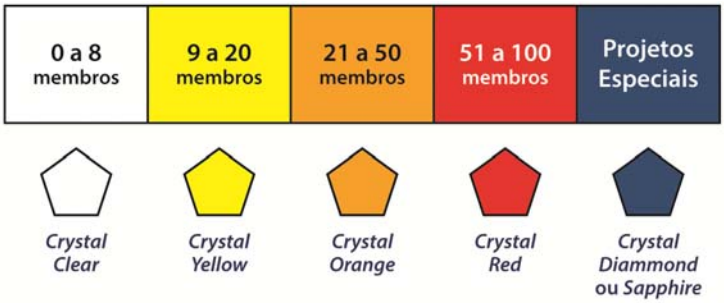
A metodologia Crystal utiliza dois parâmetros para adequar-se a ao projeto de software: (1) métrica o número de pessoas envolvidas e (2) métrica o nível crítico.

Cada método Crystal é caracterizado por uma cor, a qual está relacionada com o número de envolvidos. Tal representação pode ser vista na Figura 3.2.

- ***Crystal Clear*:** metodologia leve, para equipes de até 8 pessoas (em casos especiais pode chegar a 12).

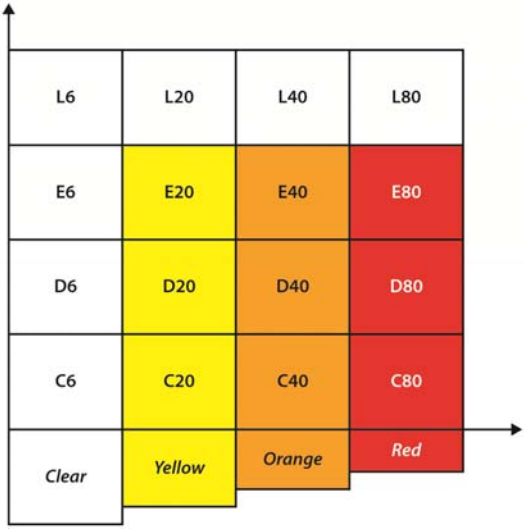
- ***Crystal Yellow***: para equipes com 9 a 20 membros.
- ***Crystal Orange*:** para equipes de 21 a 50 membros.
- ***Crystal Red***: para equipes de 51 a 100 membros.
- ***Crystal Diammond ou Sapphire*:** projetos especiais.

Figura 3.2: Representação de potenciais perdas causadas por uma falha.



Além das cores, o Crystal possui alguns códigos para representar potenciais perdas causadas por falhas no processo de desenvolvimento de software. A Figura 3.3 apresenta um exemplo de tal representação.

Figura 3.3: Representação de potenciais perdas causadas por uma falha.



Conforme pode ser observado na Figura 3.3:

- C (*Confort* - Conforto): casos em que a falha ocasiona perda de credibilidade do usuário.
- D (*Discretionary money* - Dinheiro não disponível para uso conforme necessário): casos em que a falha ocasiona perda financeira (mas de valor inexpressivo).
- E (Essencial Money - Dinheiro essencial): casos em que a falha ocasiona perda de uma quantia indispensável, grandes valores.
- L (Life - Vida): casos em que a falha ocasiona a perda de vidas.

Independente da metodologia Crystal escolhida, existem alguns princípios chaves:

- **Entrega Frequente:** os interessados pelos projetos querem ver resultados o mais breve possível. Para isso, devem ser realizadas entregas em um curto espaço de tempo. Independente do tamanho do projeto (ágil ou não) deve-se garantir entrega funcionais e código testado a cada curto espaço de tempo. As vantagens são muitas: feedback sobre a taxa de progresso do projeto e da equipe para os patrocinadores e demais interessados; os usuários podem ver e avaliar os produtos em funcionamento ainda durante a fase de desenvolvimento para ver se está de acordo com suas necessidades; desenvolvedores com curto tempo para realizar as tarefas mantêm seu foco; e a equipe por meio das entregas (conclusão do que precisa ser feito) se mantêm motivada.
- **Melhoria Reflexiva**: capacidade do projeto reverter as falhas com sucesso (especialmente as catastróficas). A equipe se reúne para discutir o que pode ser feito para melhorar e como serão feitas as mudanças na próxima interação.
- **Feedback Contínuo e Comunicação Constante::** a equipe do projeto deve se reunir regularmente para discutir as atividades do projeto e para buscar garantias de que o projeto está caminhando na direção certa (esperada) e para que se comunique qualquer coisa que pode afetar o

desenvolvimento do projeto. O ideal é que a equipe esteja na mesma sala (ou próximos) para que facilitar a comunicação.

- **Segurança Pessoal**: se refere a possibilidade de se comunicar, reclamar ou dizer algo sem medo de represálias. Isso é um fator fundamental para lidar com o comportamento e qualidade da equipe.
- **Foco**: o foco é fundamental para equipe trabalhar, sabendo lidar com as prioridades e focar naquilo que realmente é importante. Para manter o foco é preciso manter as pessoas naquilo que elas estão alocadas para atuar, evitando retirá-las para atuar em outras tarefas incompatíveis.
- **Fácil acesso para Usuários:** isso permite que a equipe possa realizar testes e entregas frequentes e, além disso, permite que se há alguma dúvida essas podem ser sanadas rapidamente.
- T**estes Automatizados e Integração Contínua:** ferramentas de controle devem ser postas em prática para apoiar a versão que está sendo entregue.

O ciclo de vida Crystal é baseado nas seguintes práticas:

- **Edição e Revisão:** construção, demonstração e revisão dos objetivos do incremento.
- **Inspeções de usuários:** são sugeridas duas a três inspeções feitas por usuários a cada incremento.
- ***Local matters*:** são os procedimentos a serem aplicados, que variam de acordo com o tipo de projeto.
- **Monitoramento**: O processo é monitorado com relação ao progresso e estabilidade da equipe. É medido em marcos e em estágios de estabilidade.
- **Paralelismo e fluxo:** Em *Crystal Orange* as diferentes equipes podem operar com máximo paralelismo. Isto é permitido através do monitoramento da estabilidade e da sincronização entre as equipes.

- ***Staging***: Planejamento do próximo incremento do sistema. A equipe seleciona os requisitos que serão implementados na iteração e o prazo para sua entrega.
- ***Standards* (padrões):** padrões de notação, convenções de produto, formatação e qualidade usadas no projeto.
- **Tools:** Ferramentas mínimas utilizadas. Para *Crystal Clear*, são compiladores, gerenciadores de versão e configuração, ferramentas de versão, programação, teste, comunicação, monitoramento de projeto, desenho e medição de desempenho.
- ***Work Products* (Produtos de Trabalho):** sequência de lançamento, modelos de objetos comuns, manual do usuário, casos de teste e migração de código. Especificamente para o *Clear*, são casos de uso e descrição de funcionalidade e, especificamente para o Orange, são documentos de requisitos.
- ***Workshops* refletivos:** são reuniões que ocorrem antes e depois de cada interação, com objetivo de analisar o progresso do projeto;

FDD, DSDM, ASD

O desenvolvimento dirigido a funcionalidade (*Feature Driven Development* - FDD) foi concebido originalmente por Peter Coad e seus colegas como um modelo de processos prático para a engenharia de software orientada a objetos.

O FDD adota abordagens ágeis que:

- Enfatiza a colaboração entre pessoas da equipe FDD.
- Gerencia problemas e complexidade de projetos utilizando a decomposição baseada em funcionalidades, seguida pela integração dos incrementos de software.
- Comunicação de detalhes técnicos usando meios verbais, gráficos e textos.

- Enfatiza atividades de garantia da qualidade de software por meio do encorajamento de uma estratégia de desenvolvimento incremental, com uso de inspeções de código e do projeto, aplicação de auditorias para garantia da qualidade de software, coleta de métricas e o uso de padrões para análise, projeto e construção.

No contexto do FDD, a ênfase na definição de funcionalidades gera os seguintes benefícios:

- As funcionalidades forma pequenos blocos que podem ser entregues e descritos facilmente pelo usuário. Blocos pequenos facilitam a compreensão de como as funcionalidades se relacionam entre si e permite uma revisão melhor para evitar ambiguidades, erros e omissões.
- As funcionalidades podem ser organizadas em um agrupamento hierárquico relacionado com o negócio.
- Com uma funcionalidade é o incremento de software do FDD que pode ser entregue, a equipe desenvolve funcionalidades operacionais a cada duas semanas.
- Pelo fato dos blocos de funcionalidades serem pequenos, os projetos FDD são mais fáceis de inspecionar.
- O planejamento, cronograma e acompanhamento do projeto são guiados pela hierarquia de funcionalidades.

Para definir uma funcionalidade em FDD é sugerido o seguinte modelo:

<ação> o <resultado> <por | para quem | de | para que> um <objeto>

Exemplo de funcionalidades:

- Adicione o produto ao carrinho
- Mostre as especificações técnicas do produto
- Armazene as informações de envio para o cliente

O FDD define cinco processos:

- Desenvolver um modelo geral
- Construir uma lista de funcionalidades
- Planejar por funcionalidade
- Projetar por funcionalidade
- Desenvolver por funcionalidade

Um aspecto interessante do FDD em relação a outros métodos ágeis é que ele dá maior ênfase às diretrizes e técnicas de gerenciamento de projetos do que outros métodos ágeis disponíveis.

Já o Método de Desenvolvimento de Sistemas Dinâmicos (DSDM – Dynamic System Development Method) é uma metodologia ágil que oferece um processo para construir e manter sistemas que atendam restrições de prazo apertado pode meio do uso de prototipagem incremental.

O DSDM é um processo iterativo no qual cada incremento possui somente a quantidade de trabalho suficiente. No DSDM é definido três ciclos iterativos diferentes que são precedidos por duas atividades de ciclo de vida adicionais:

- Estudo da viabilidade: que estabelece requisitos básicos de negócio e restrições associados à aplicação a ser construída e depois avalia se a aplicação é um candidato viável para o processo DSDM.
- Estudo do negócio: que é responsável por estabelecer os requisitos funcionais e de informação que permitirão à aplicação agregar valor de negócio e, também tem-se a definição básica da arquitetura da aplicação e a identificação dos requisitos de facilidade de manutenção para a aplicação.

Os três ciclos iterativos são:

- Modelos Funcionais: é produzido um conjunto de protótipos incrementais que demonstram a funcionalidade para o cliente.
- Projeto e Desenvolvimento: os protótipos desenvolvidos durante a iteração de modelos funcionais são revisitados para que se possa assegurar que cada um tenha passado por um processo de engenharia

para capacitá-los a oferecer aos usuários finais valor de negócio em termos operacionais.

- Implementação: é colocada a última versão do incremento de software (protótipo operacionalizado) no ambiente operacional.

O Método ASD (Adaptive Software Development – Desenvolvimento de Software Adaptativo) foi proposto por Jim Higsmith como uma técnica para a construção de software e sistemas altamente complexos. Esse modelo basicamente se concentra na colaboração e auto organização das equipes.

O ASD tem em seu ciclo de vida três fases: especulação, colaboração e aprendizagem.

- **Especulação**: o projeto é iniciado e tem-se o planejamento de ciclos adaptáveis, que faz uso das informações contidas no início do projeto como: a missão do cliente, restrições do projeto e os requisitos básicos. Os requisitos básicos são utilizados para a definição do conjunto de ciclos da versão, ou seja, os incrementos do software. Após completar o ciclo o plano é revisto e ajustado.

- **Colaboração**: envolve a confiança, críticas sem animosidade, auxílio, trabalho árduo, comunicação de problemas ou preocupações de forma a conduzir ações efetivas, dentre outras características. É um fator importante no levantamento das necessidades e especificações de requisitos.

- **Aprendizado**: considerado um elemento chave para que se possa conseguir uma equipe auto organizada. Os desenvolvedores superestimam o próprio entendimento quanto à tecnologia, processo e até mesmo quanto ao projeto. Portanto, o aprendizado auxilia os desenvolvedores a aumentar os níveis reais de entendimento. As equipes ASD aprendem por meio de três maneiras: grupos focados, revisões técnicas e revisões investigativas (autópsias) de projetos.

Nesta unidade, você estudou sobre os Modelos Ágeis de Desenvolvimento de Software. Na próxima unidade irá aprender mais sobre o Gerenciamento de Sistemas de Informação.

É hora de se avaliar

Lembre-se de realizar as atividades desta unidade de estudo. Elas irão ajudá-lo a fixar o conteúdo, além de proporcionar sua autonomia no processo de ensino-aprendizagem.

Exercícios – Unidade 3

1. No Scrum os papéis são bem definidos. Assinale a alternativa a qual o trecho abaixo se refere:

"Tem como função primária remover impedimentos para que a equipe consiga entregar o objetivo do Sprint. Além dessa função, a pessoa nesse papel tem a função de assegurar que as práticas do Scrum sejam utilizadas corretamente."

a) Scrum Product Owner

b) Scrum Manager

c) Scrum Project Manager

d) Scrum Master

e) Scrum Lider

2. Sobre o Scrum, assinale a alternativa correta.

a) o product owner tem, entre outras atribuições, a de indicar quais são os requisitos mais importantes a serem tratados em cada sprint . É responsável por conhecer e avaliar as necessidades dos clientes.

b) um dos conceitos mais importantes é o sprint , que consiste em um ciclo de desenvolvimento que, em geral, tem duração de 4 a 7 dias.

c) o scrum master é um gerente no sentido dos modelos prescritivos. É ele quem decide quais requisitos são mais importantes.

d) as funcionalidades a serem implementadas em cada projeto são mantidas em uma lista chamada de scrum board.

e) o scrum team é a equipe de desenvolvimento, necessariamente dividida em papéis como analista, designer e programador. Em geral o scrum team tem de 10 a 20 pessoas.

3. O conceito de sprint aplica-se ao modelo ágil do processo de engenharia de software denominado:

a) XP

b) DAS

c) DSDM

d) Scrum

e) Crystal

4. Qual opção não é um exemplo de metodologia ágil?

a) XP

b) Scrum

c) FDD

d) RUP

e) Crystal

5. Em relação as metodologias ágeis, julgue os itens a seguir, marcando com (V) a assertiva verdadeira e com (F) a assertiva falsa:

( ) O conceito de sprint aplica-se ao modelo ágil do processo de engenharia de software conhecido como Scrum.

( )RUP, XP e DSDM são alguns exemplos de metodologias de desenvolvimento de software consideradas ágeis.

( )A Feature Driven Development (FDD) é uma metodologia ágil de desenvolvimento de software que mantém seu foco apenas na fase de modelagem. ( ) Na extreme programming, os requisitos são expressos como cenários e implementados diretamente como uma série de tarefas.

A alternativa que apresenta, respectivamente, a alternativa correta é:

a) V – F – V – V

b) V – F – F – V

c) F – F – F – V

d) F – V – F – V

e) V – V – V – V

6. Cada método Crystal é caracterizado por uma cor, a qual está relacionada com o número de envolvidos. Qual cor se refere a equipes de 21 a 50 membros.

a) Crystal Clear

b) Crystal Red

c) Crystal Yellow

d) Crystal Orange

e) Crystal Diammonds

7. "Nessa metodologia os requisitos são expressos como cenários (chamados de estórias do usuário), que são implementados diretamente como uma série de tarefas. Os programadores normalmente trabalham em pares e desenvolvem testes para cada tarefa antes de escreverem o código". Qual metodologia esse trecho se refere:

a) Scrum

b) Crystal

c) XP

d) FDD

e) ASD

8. "Os modelos ágeis de desenvolvimento de software têm menos ênfase nas definições de atividades e mais ênfase na pragmática e nos fatores humanos do desenvolvimento. Um destes modelos enfatiza o uso de orientação a objetos e possui apenas duas grandes fases: 1 - Concepção e Planejamento e 2 - Construção. A fase de Concepção e Planejamento possui três disciplinas (chamadas de processos): Desenvolver Modelo Abrangente, Construir Lista de Funcionalidades e Planejar por funcionalidade. Já a fase de Construção incorpora duas disciplinas (processos): Detalhar por Funcionalidade e Construir por Funcionalidade".

O texto acima apresenta a metodologia ágil conhecida como:

a) FDD

b) Scrum

c) Crystal

d) ASD

e) DSDM

9. O Método ASD (Adaptive Software Development – Desenvolvimento de Software Adaptativo) foi proposto por Jim Higsmith como uma técnica para a construção de software e sistemas altamente complexos. Descreva as três fases do ciclo de desenvolvimento ASD.

10. Defensores do desenvolvimento de software ágil dizem que "o desenvolvimento ágil foca talentos e habilidade de indivíduos, moldando o processo de acordo com as pessoas e as equipes específicas". Descreva as principais características de indivíduos que atuam em um processo ágil.

4 Gerenciamento de Sistemas de Informação

Caro(a) aluno(a),

Nesta unidade estudaremos os principais conceitos sobre o Gerenciamento de Sistemas de Informação. Entender o papel da Informação e dos sistemas que dão apoio e suporte a essa informação dentro de uma organização, bem como saber o papel do gestor da informação e suas responsabilidades.

Objetivos da unidade:

O objetivo dessa unidade é introduzir conceitos sobre o Gerenciamento de Sistemas de Informação. Com o estudo dessa unidade, espera-se que, você:

- Compreenda os conceitos iniciais sobre Gerenciamento de Sistemas
- Entenda os aspectos principais sobre Gerenciamento dos Recursos de Informação

Plano da unidade:

- Gerenciamento de sistemas
- Gerenciamento dos recursos de informação

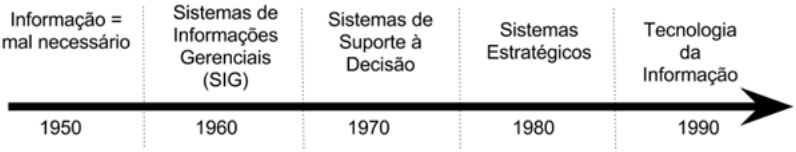
Bons estudos!

Gerenciamento de sistemas

Gerenciar sistemas é uma ação coordenada por uma série planejada de atividades relacionadas para alcançar um objetivo específico. O gerenciamento de sistemas está na aplicação de conhecimento, habilidades, ferramentas e técnicas de modo a alcançar objetivos específicos dentro de uma determinada organização.

Há uma perceptível variação no enfoque dado ao papel da informação e dos sistemas de informações dentro das organizações ao longo do tempo. A Figura 4.1 apresenta uma linha do tempo da evolução da informação dentro das organizações.

Figura 4.1: Evolução da Informação dentro das Organizações



Na Figura 4.1 temos que:

- 1950 e início de 1960: informação era um mal necessário, associada à burocracia do projeto, fabricação e distribuição de um produto ou serviço. Os sistemas de informação eram focados na redução de custo do processamento rotineiro de papéis, especialmente de contabilidade.
- Na década de 1960: as organizações começaram a reconhecer que informações poderiam ser usadas como suporte para o gerenciamento. Os sistemas de informações das décadas de 60 e 70 eram chamados de Sistemas de Informações Gerenciais (SIG).
- Na década de 70: informações, bem como os sistemas que as coletaram, armazenaram e processaram eram vistos como provedores de uma padronização do gerenciamento geral da organização. Os sistemas de informações que surgiram durante este período eram chamados de sistemas de suporte a decisão e sistemas de suporte executivo.

- Na década de 80: a informação era considerada um recurso estratégico, um potencial recurso para aquisição da vantagem competitiva, ou uma arma estratégica na competição. Os tipos de sistemas construídos para dar suporte a este conceito de informação são chamados de sistemas estratégicos. Os sistemas de informações estratégicos possibilitam a rápida aquisição e armazenagem de grande quantidade de informações, além de proporcionar o agrupamento das informações vindas de diversas fontes. Porém, tais sistemas levam a uma necessidade contínua de revisão dos mesmos, para se manterem aderentes as necessidades e tecnologias. Com isso, o pessoal deve ser continuamente treinado, para que consigam trabalhar com tais sistemas.
- Na década de 90: o desafio mais importante é o uso da Tecnologia da Informação para reestruturar os negócios e as organizações, garantindo uma crescente eficiência e eficácia.

Fatores que influenciam o gerenciamento de sistemas de Informação:

1. **Crescimento tecnológico acelerado:** criação contínua de aplicações para acompanhar o avanço tecnológico exige das organizações a reconstrução de seus sistemas para se adequarem e não perderem a competitividade em um curto espaço de tempo.
2. **Descentralização**: mesmo com a existência de centrais de dados praticamente todas as funções dentro de uma organização usam os sistemas para acessar informações e melhorar o desempenho de suas funções.
3. **Integração e Conexão**: para adquirir a integração necessária na organização é necessário um fluxo contínuo de informações que une os sistemas de automatização dos ambientes de trabalho, de processamento de dados e de telecomunicações.
4. **Escassez de recursos humanos qualificados**: a demanda que o mercado exige está maior que o número de profissionais qualificados disponíveis, que consigam conectar o conhecimento tecnológico com o conhecimento organizacional.

5. **Mudanças no ambiente**: mudanças no ambiente interno e externo da organização trazem reflexos imediatos no uso dos sistemas de informação.
6. **Gerenciamento dos sistemas internacionais**: devido a globalização, as grandes empresas estão expandindo seus mercados mundialmente. E isso faz necessário realizar adaptações para legislações de culturas de outros países.
7. **Um novo papel para os sistemas**: cada organização usa a tecnologia de sua própria maneira.

Baseado nesses fatores, as propostas utilizadas por grande parte das empresas para solucionar problemas em sistemas de informação são:

- Implantar o controle organizacional e o planejamento estratégico de sistemas.
- Estabelecer o valor dos sistemas e escolher alternativas.
- Gerenciar os projetos de sistemas.
- Atrair e manter bons profissionais em sistemas.
- Gerenciar as transições estratégicas.

De uma forma geral, o responsável pela Gestão dos Sistemas de Informação é feita por um gestor. Tal gestor é quem faz a conexão entre a administração da organização e os sistemas.

Gerenciar sistemas de informação implica, entre outras coisas em:

- Fazer parte do processo de planejamento estratégico da organização, mostrando como a informação e a tecnologia da informação pode contribuir para a redução dos custos, com o aumento da produtividade, a melhoria da qualidade, o desenvolvimento de novos produtos e serviços, expansão e exploração de novas áreas de mercado e, por consequência, para maior competitividade da organização.

- Fazer parte do processo e definição dos dados corporativos da organização e assumir responsabilidade pela sua administração, segurança, integridade e confiabilidade.
- Desenvolver, propor e negociar a implantação de normas e padrões que possa evitar o caos causado pela aquisição descentralizada e distribuída de recursos de tecnologia e pelo desenvolvimento de aplicações pelos usuários, quando não existem normas e padrões.
- Administrar a rede de telecomunicações da organização que deve fornecer infraestrutura não apenas para transmissão de dados.
- Lidar com executivos, gerentes, pessoal técnico, pessoas especializadas, etc.
- Dar suporte a usuários.
- Administrar conflitos.

Gerenciamento dos recursos de informação

Os recursos quando falamos de sistemas de informação podem ser pessoas, ferramentas (tecnologias) e os processos. Além disso, a própria informação pode ser considerada um recurso que necessita ser gerenciado. Savic (1992) listou 6 motivos pelos quais a informação deve ser vista como um recurso:

- Algo com valor fundamental, como capital, bens, trabalho ou suprimentos;
- Algo com características mensuráveis e específicas;
- Um ativo de entrada e saída de custos agregados.
- Algo que possa ser capitalizável ;
- Algo que possa ter seus custo padronizado e contabilizado.
- Algo que forneça subsídios estratégicos diversos para a tomada de decisões.

De forma geral, o conceito de Gestão de Recursos de Informação pode ser entendido como gestão de todos os recursos relacionados ao processo de ciclo da informação dentro de uma organização. Então, a Gestão de Recursos de Informação se refere aos recursos do qual se vale a organização para planejamento, orçamentação, capacitação de pessoas e tecnologias para tratamento da informação para alcançar seus objetivos.

A Gestão dos Recursos de Informação (GRI) é baseada em 3 pilares: pessoas, processos e tecnologias (ferramentas). A Figura 4.2 apresenta esses pilares.

Figura 4.2: Pilares da Gestão dos Recursos da Informação (GRI).

Como pode ser visto na Figura 4.2, pessoas, processos e tecnologias estão integrados e relacionados e, juntos, são recursos importantes para o planejamento estratégico de uma organização. Cabe ao gerente de recursos de informação a coordenar e integrar criticamente esses diversos meios.

A rapidez com que a Tecnologia da Informação passou a fazer parte das organizações e, também, sua constante evolução, gera consideráveis desafios para o gerenciamento. Um grande problema é que muitas organizações só se preocupam com as evoluções tecnológicas (hardware e software) e esquece-se de um fator primordial: as pessoas que os utilizam!

As informações efetivas sobre gerenciamento da informação devem começar pela verificação de como as pessoas utilizam a informação e não de como as pessoas utilizam as ferramentas.

O gerenciamento dos recursos de informação está relacionado a diversos aspectos gerenciais da organização, especialmente aqueles relacionados a custo,

qualidade e uso eficaz da informação. Quando se refere a tecnologia é considerado as novas tecnologias de informação disponíveis e que possam apoiar o processo de decisão da organização com informações, como comunicação verbal, comunicação de dados, entre outros.

Existem diversas fontes de informação, como pessoas, mídias e até mesmo sistemas. A decisão quanto à importância a ser atribuída à tecnologia e à informação, é outro fator a ser considerado e o gerente de recursos de informação deve se preocupar tanto com a gerência da informação quanto com a gerência da tecnologia da informação.

O avanço da tecnologia nas organizações trouxe uma grande variedade de serviços, ferramentas tecnologias e sistemas para o ambiente de TI, que aumentam a complexidade na gestão, mas requerem ainda mais cuidados no gerenciamento para garantir que todos os recursos estejam sempre disponíveis e seguros.

Os cuidados com o gerenciamento de recursos são necessários para evitar uma série de incidentes e surpresas desagradáveis. Alguns aspectos devem ser considerados quando se fala em gerenciamento de recursos de informação:

- **Criação de um modelo de governança**: é útil para identificar os pontos mais importantes a serem gerenciados, quais serviços necessitam de mais atenção e ajuda a definir planos de recuperação em casos de incidentes.
- **Monitoramento**: gerenciamento começa com o monitoramento dos recursos, que é a maneira de ser ter uma visão constante e em tempo adequado sobre o desempenho de cada recurso.
- **Métricas**: para medir e saber se há uma melhora na produtividade e resultados é necessário estabelecer métricas claras que permitirão os gestores acompanharem os resultados e buscarem melhorias. As métricas podem estrar relacionadas ao desempenho dos recursos, tempo de disponibilidade ou quantidade de requisições atendidas por um determinado serviço, por exemplo.

Mas que tipo de pessoa pode ser um Gestor de Recursos de Informação? O que se espera de um líder que exerce tal função? O mínimo que se espera é uma boa visão estratégica. O gerente de recursos de informação nem sempre é um tomador de decisões, mas geralmente provoca mudanças e apoia decisões, identifica as informações importantes e as encaminha com rapidez àqueles que as necessitam.

Nesta unidade você estudou sobre o Gerenciamento de Sistemas de Informação. Na próxima unidade irá aprender mais sobre Projeto Arquitetural de Software.

É hora de se avaliar

Lembre-se de realizar as atividades desta unidade de estudo. Elas irão ajudá-lo a fixar o conteúdo, além de proporcionar sua autonomia no processo de ensino-aprendizagem.

Exercícios – Unidade 4

1. Avalie as assertivas abaixo, classificando-as como verdadeiras (V) ou falsas (F).

( )É função do gestor de sistemas de informação gerenciar conflitos e dar suporte aos usuários.

( )O gestor dos sistemas de informação é quem faz a conexão entre a administração da organização e os sistemas.

( )O gestor dos sistemas de informação faz parte do processo e definição dos dados corporativos da organização e assumir responsabilidade pela sua administração, segurança, integridade e confiabilidade.

( )É função do Gestor de Sistemas de Informação administrar a rede de telecomunicações da organização que deve fornecer infraestrutura não apenas para transmissão de dados.

Assinale a alternativa que apresenta a alternativa correta:

a) V - V - V - V

b) V - F - F - F

c) V - F - V - F

d) F - V - V - V

e) F - V - F – F

2. Como é conhecido o princípio básico do gerenciamento de sistemas de informação que estabelece os objetivos globais e as metas do sistema a ser desenvolvido?

a) Oportunidades Tecnológicas

b) Planejamento Estratégico

c) Tecnologia da Informação

d) Gerenciamento de Dados

e) Gerenciamento de Projetos

3. Assinale a alternativa que completa a lacuna corretamente.

"O gerente de recursos de informação nem sempre é um tomador de decisões, mas geralmente provoca _____ e apoia _____, identifica as informações importantes e as encaminha com rapidez àqueles que as necessitam.

a) Conflitos - soluções

b) Conflitos - decisões

c) Mudanças - conflitos

d) Competitividade - soluções

e) Mudanças – decisões

4. Quais os principais aspectos devem ser considerados quando se fala em gerenciamento de recursos de informação?

a) Criação de um modelo de governança; Monitoramento; Métricas.

b) Gestão de Dados; Gestão de Informação; Gestão de Pessoas.

c) Gestão de Pessoas; Gestão de Processos; Gestão de Recursos;

d) Pessoas, Processos e Informação.

e) Pessoas, Tecnologia e Processo.

5. Assinale a alternativa que completa a lacuna corretamente.

"O gerenciamento dos recursos de informação está relacionado a diversos aspectos gerenciais da organização, especialmente aqueles relacionados a _____, _____ e uso eficaz da informação".

a) Dados e segurança

b) Dados e qualidade

c) Custo e qualidade

d) Segurança e custo

e) Segurança e qualidade

6. As propostas utilizadas por grande parte das empresas para solucionar problemas no gerenciamento de sistemas de informação são, exceto:

a) Implantar o controle organizacional e o planejamento estratégico de sistemas.

b) Estabelecer o valor dos sistemas e escolher alternativas.

c) Gerenciar os projetos de sistemas.

d) Validar a qualidade dos dados da informação.

e) Atrair e manter bons profissionais em sistemas.

7. Quais os Pilares da Gestão dos Recursos da Informação?

a) Competitividade, qualidade e informação.

b) Eficiência, eficácia e efetividade.

c) Gerente, pessoas e processos.

d) Pessoas, processos e tecnologia.

e) Dados, informação e tecnologia.

8. Assinale a alternativa que completa a lacuna corretamente.

"Um grande problema é que muitas organizações só se preocupam com as evoluções tecnológicas (hardware e software) e esquece-se de um fator primordial: _____".

a) A informação

b) As pessoas

c) Os dados

d) As ferramentas

e) A qualidade

9. Quais os pilares da Gestão dos recursos da Informação? Explique a relação entre eles.

10. Os recursos quando falamos de sistemas de informação podem ser pessoas, ferramentas (tecnologias) e os processos. Além disso, a própria informação pode ser considerada um recurso que necessita ser gerenciado. Apresente motivos pelos quais a informação deve ser vista como um recurso.

5 Projeto Arquitetural de Software

Caro(a) aluno(a),

Nesta unidade, estudaremos os principais conceitos sobre Projeto Arquitetural de Software, que consiste em um mapeamento do sistema de forma que sejam representadas as diferentes partes com suas interações e mecanismos de interconexões. Um projeto arquitetural facilita o desenvolvimento de um sistema, prevê possíveis problemas que poderiam ter impacto no futuro; estrutura o sistema e como seus componentes trabalham em conjunto.

Objetivos da unidade:

O objetivo desta unidade é apresentar os principais conceitos de arquitetura e do projeto de arquitetura de software. Espera-se que, estudando essa unidade, você:

- Compreenda por que o projeto de arquitetura de software é importante.
- Compreenda as decisões necessárias sobre a arquitetura de sistema durante o processo de projeto de arquitetura.
- Conheça os principais padrões de arquitetura: arquitetura em camadas, de repositório e cliente-servidor.

Plano da unidade:

- Padrões de Arquitetura
- Arquitetura em Camadas
- Arquitetura de Repositório
- Arquitetura Cliente-Servidor

Bons estudos!

O Projeto Arquitetural de Software consiste em um mapeamento do sistema de forma que sejam representadas as diferentes partes com suas interações e mecanismos de interconexões. Portanto, representa a estrutura dos componentes de dados e programas necessários para a construção de um sistema.

O projeto da arquitetura do software descreve a organização fundamental do sistema, identificando os diversos módulos, suas relações entre si e a relação desses com o ambiente, para que alcancem os objetivos propostos pelo cliente. Além da escolha dos algoritmos e estruturas de dados, a arquitetura envolve decisões sobre as estruturas de controle, protocolos de comunicação, sincronização e acesso de dados, distribuição lógica e física dos elementos, atributos de qualidade (escalabilidade, desempenho, etc) e seleção das alternativas de projeto.

O projeto de arquitetura tem foco na maneira com o que o sistema deve ser organizado e como será sua estrutura geral. O resultado do processo de projeto de arquitetura é uma descrição de como o sistema está organizado em um conjunto de componentes de comunicação. Tal descrição é o modelo de arquitetura.

De acordo com Sommerville (2011) é possível projetar as arquiteturas de software em dois níveis de abstração:

1. **Arquitetura em pequena escala**: preocupada com a arquitetura de programas individuais; preocupa-se com a maneira como um programa individual é decomposto em componentes.
2. **Arquitetura em grande escala**: preocupa-se com a arquitetura de sistemas corporativos complexos, que incluem outros sistemas, programas e componentes de sistemas.

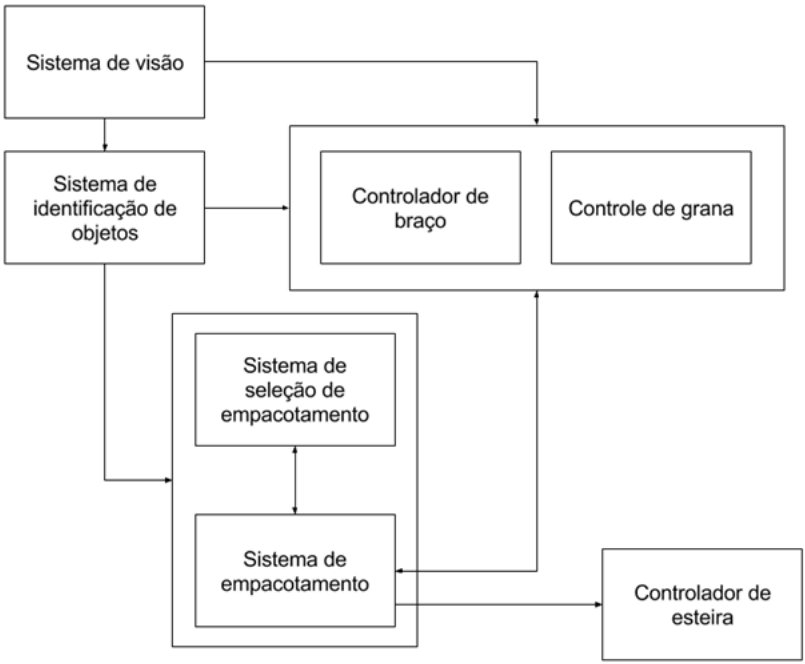
Sommerville também apresenta vantagens de se projetar e documentar, explicitamente, a arquitetura de software:

- A arquitetura é uma apresentação de alto nível do sistema e pode ser usada como um foco de discussão entre os interessados (*steakholders*).
- A arquitetura afeta na decisão e na avaliação das possibilidades do sistema atender ou não aos requisitos críticos, como desempenho, confiabilidade e manutenibilidade.

- Como normalmente se utiliza a mesma arquitetura para sistemas com requisitos semelhantes, o projeto arquitetural pode apoiar o reuso do software em grande escala.

Geralmente as arquiteturas de sistema são modeladas por meio de diagramas de blocos simples, conforme Figura 5.1. No diagrama, cada caixa representa um componente, as caixas dentro das caixas representam os subcomponentes (componentes decompostos) e as setas indicam o fluxo de dados.

Figura 5.4: A arquitetura de um sistema de controle robotizado de empacotamento.



Fonte: SOMMERVILLE, 2011, p. 104.

O projeto de arquitetura pode ser considerado um processo criativo no qual se projeta uma forma de organizar o sistema para que sejam satisfeitos os requisitos. Por ser considerado um processo criativo a definição do projeto arquitetural depende do sistema a ser desenvolvido, da formação e experiência do arquiteto e dos requisitos específicos para o sistema. Portanto, Sommerville (2011) destaca que o projeto de arquitetura é uma série de decisões e não apenas uma sequência de atividade.

Sommerville (2011) apresenta alguns questionamentos fundamentais sobre o sistema que devem ser realizados pelos arquitetos de sistemas durante as tomadas de decisões, são eles:

1. Existe uma arquitetura genérica de aplicação que pode atuar como um modelo para o sistema que está sendo projetado?
2. Como o sistema será distribuído por meio de um número de núcleos ou processadores?
3. Que padrões ou estilos de arquitetura podem ser usados?
4. Qual será a abordagem fundamental para se estruturar o sistema?
5. Como os componentes estruturais do sistema serão decompostos em subcomponentes?
6. Que estratégia será usada para controlar o funcionamento dos componentes do sistema?
7. Qual a melhor organização de arquitetura para satisfazer os requisitos não funcionais do sistema?
8. Como o projeto de arquitetura será avaliado?
9. Como a arquitetura do sistema deve ser documentada?

Mesmo os sistemas sendo únicos, ainda assim podem existir, quando se há o mesmo domínio de aplicação, sistemas com arquiteturas similares. A arquitetura de um sistema de software pode ser basear em um determinado padrão ou estilo de arquitetura. Um padrão de arquitetura é uma descrição de uma organização do sistema; capturam a essência de uma arquitetura que tem sido usada em diferentes sistemas de software. Ao tomar decisões sobre a arquitetura de um sistema, deve-se conhecer os padrões comuns, bem como saber onde podem ser usados e quais os pontos fortes e fracos (SOMMERVILLE, 2011).

Sommerville (2011) diz que o padrão ou estilo de arquitetura escolhido deve depender dos requisitos não funcionais do sistema:

1. **Desempenho**: Se o desempenho for um requisito crítico, a arquitetura deve ser projetada para localizar as operações críticas dentro de um pequeno número de componentes, com todos esses componentes implantados no mesmo computador, em vez de distribuídos pela rede.

2. **Proteção**: Se a proteção for um requisito crítico, deve ser usada uma estrutura em camadas para a arquitetura, com os ativos mais críticos protegidos nas camadas mais internas, com alto nível de validação de proteção aplicado a essas camadas.

3. **Segurança**: Se a segurança for um requisito crítico, a arquitetura deve ser concebida de modo que as operações relacionadas com a segurança sejam localizadas em um único componente ou em um pequeno número de componentes. Isso reduz os custos e os problemas de validação de segurança e torna possível fornecer sistemas de proteção relacionados que podem desligar o sistema de maneira segura em caso de falha.

4. **Disponibilidade**: Se a disponibilidade for um requisito crítico, a arquitetura deve ser projetada para incluir componentes redundantes, de modo que seja possível substituir e atualizar componentes sem parar o sistema.

5. **Manutenção**. Se a manutenção for um requisito crítico, a arquitetura do sistema deve ser projetada a partir de componentes autocontidos de baixa granularidade que podem ser rapidamente alterados.

Padrões de Arquitetura

Um padrão é um *template* (formulário) de solução para um problema recorrente que seja útil em um determinado contexto. Ou seja, um padrão expressa uma solução que pode ser reutilizada e é descrita para um contexto específico, ligando um problema e uma solução:

- **Contexto**: apresenta situações de ocorrência do problema a ser solucionado.
- **Problema**: estabelecimento dos aspectos do problema que devem ser considerados na solução.
- **Solução**: apresenta como resolver o problema.

Padrões de arquitetura é uma descrição abstrata, estilizada, de boas práticas experimentadas e testadas em diferentes sistemas e ambientes. Um padrão deve conter informações de quando seu uso é adequado (ou não), seus pontos fortes e fracos. Tais padrões são formulários prontos que solucionam problemas arquiteturais recorrentes.

Padrões no contexto da Engenharia de Software permitem que desenvolvedores possam recorrer a soluções já existentes para solucionar problemas decorrentes do desenvolvimento de software.

Buschmann (1996) agrupa padrões de arquitetura de acordo com as características dos sistemas nos quais eles são mais aplicáveis, conforme apresentado na Tabela 5.1.

Tabela 5.1: Agrupamento de Padrões de Arquitetura

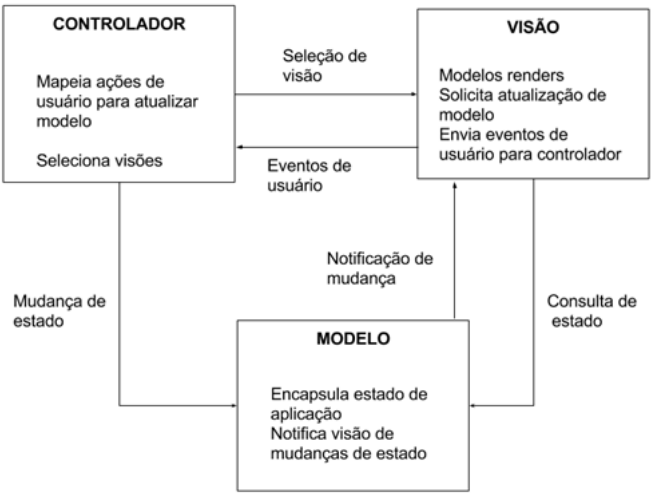
Categoria	Padrão
Estrutura	Camadas
	Pipes e Filtros
	Quadro-negro
Sistemas Distribuídos	Broker
Sistemas Interativos	MVC
	AAC
Sistemas Adaptáveis	Reflexo
	Microkernel

Sommerville (2011) apresenta a Tabela 5.2 que descreve o padrão MVC (*Model-View-Controller* - Modelo-Visão-Controlador), o qual é base do gerenciamento de interação em muitos sistemas baseados em Web. A descrição estilizada de padrão inclui o nome do padrão, uma breve descrição e um exemplo do tipo de sistema em que o padrão é usado.

Tabela 5.2: O padrão MVC (SOMMERVILLE, 2011, p. 109)

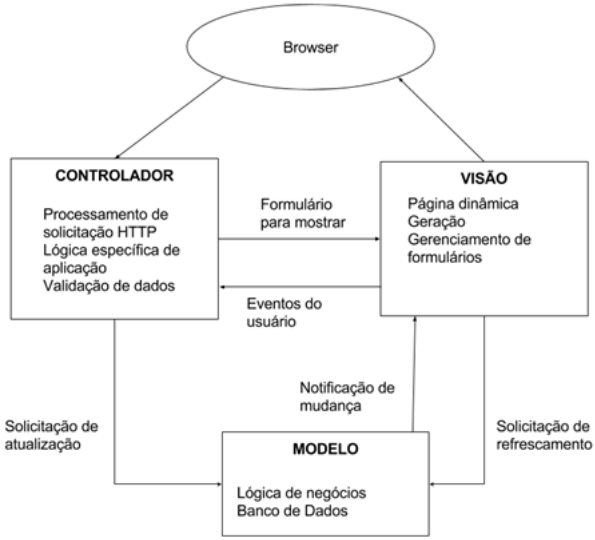
Nome	MVC (Modelo-Visão-Controlador)
Descrição	Separa a apresentação e a interação dos dados do sistema. O sistema é estruturado em três componentes lógicos que interagem entre si. O componente "Modelo" gerencia o sistema de dados e as operações associadas a esses dados. O componente "Visão" define e gerencia como os dados são apresentados ao usuário. O componente "Controlador" gerencia a interação do usuário (por exemplo, teclas, cliques do mouse etc.) e passa essas interações para a "Visão" e o "Modelo". Como pode ser visto na Figura 5.2.
Exemplo	A Figura 5.3 mostra a arquitetura de um sistema aplicativo baseado na Internet, organizado pelo uso do padrão MVC.
Quando é usado	É usado quando existem várias maneiras de se visualizar e interagir com dados. Também quando são desconhecidos os futuros requisitos de interação e apresentação de dados.
Vantagens	Permite que os dados sejam alterados de forma independente de sua representação, e vice-versa. Apoia a apresentação dos mesmos dados de maneiras diferentes, com as alterações feitas em uma representação aparecendo em todas elas.
Desvantagens	Quando o modelo de dados e as interações são simples, pode envolver código adicional e complexidade de código.

Figura 5.5: A organização do MVC



Fonte: SOMMERVILLE, 2011, p. 109

Figura 5.6: Arquitetura de aplicações Web usando padrão MVC



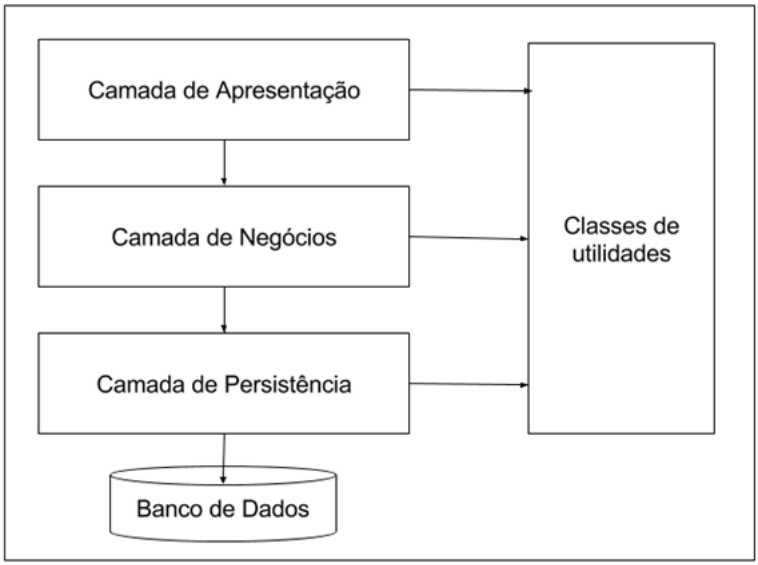
Fonte: SOMMERVILLE, 2011, p. 110

Arquitetura em Camadas

Arquitetura em camadas pode ser definida como um processo de decomposição de sistemas complexos em camadas para facilitar a manutenção, bem como a compreensão desses sistemas. Arquitetura em camadas visa a criação de aplicativos modulares, de forma que a camada num nível mais acima se comunica com a camada num nível mais baixo e assim por diante, fazendo com que uma camada seja dependente apenas da camada imediatamente inferior.

A decomposição dos sistemas em camadas foi baseada na arquitetura de computadores, que utiliza camadas para chamadas do sistema operacional, *device drivers*, instruções do processador, etc. A Figura 5.4 apresenta um exemplo de camadas.

Figura 5.4: Exemplo de arquitetura em camadas.



Os benefícios de separar sistemas em camadas são:

- Estimula a organização da arquitetura do sistema em um conjunto de camadas coesas com fraco acoplamento entre elas. Separando em camadas, o desenvolvimento pode ser feito em etapas.
- Cada camada possui um propósito bem definido. Software das camadas tende a ser mais limpos (fáceis de entender).
- É possível compreender uma única camada coerentemente como um todo, sem a necessidade de muito conhecimento das outras camadas.
- As camadas podem ser substituídas por implementações alternativas dos mesmos serviços básicos.
- As camadas são bons lugares para padronização.
- Separação de código relativo a interface com o usuário, comunicação, negócio e dados.
- Se surgir algum problema é mais fácil encontrá-lo e eliminá-lo, pois como as camadas são bem "isoladas" (focadas em funcionalidades específicas)
- Permite a mudança de implementação de uma camada sem afetar outra, desde que a interface entre as mesmas seja mantida.
- Possibilita que uma camada trabalhe com diferentes versões de outra camada.

Desvantagem:

- Aumento no número de classes existentes no sistema.

O surgimento de sistemas cliente/servidor trouxe destaque para o uso de camadas de software. Nesse tipo de sistema o cliente mantinha a interface de usuário e outros códigos da aplicação e o servidor era como se fosse um banco de dados relacional.

A divisão em camadas representa um agrupamento de forma ordenada de funcionalidades, na qual as camadas superiores contêm funcionalidades especificadas da aplicação, as camadas intermediárias abrangem funcionalidades do domínio da aplicação e as camadas mais inferiores funcionalidades mais

específicas do ambiente. O número e a composição das camadas dependem da complexidade do domínio do problema e do espaço para a solução.

O MVC (*Model-View-Controller*) é um padrão que pode ser utilizado em vários tipos de projetos (web, mobile e desktop). A comunicação entre interfaces e regras de negócios é definida por meio de um controlador. Por exemplo, quando se executa um clique na interface gráfica (clique em um botão) a interface irá se comunicar com o controlador, que se comunicará com as regras de negócio. O controlador é responsável por definir o comportamento da aplicação, interpretar as ações do usuário e mapear as chamadas do modelo.

O principal objetivo do padrão MVC é separar dados ou lógica de negócios (MODEL) da interface do usuário (VIEW) e do fluxo da aplicação (CONTROL). Essa ideia é o que permite que uma mesma lógica de negócio possa ser acessada e visualizada por interfaces distintas.

São vantagens do uso do padrão MVC:

- Possibilita a existência de várias interfaces com o usuário que podem ser modificadas sem que haja alteração das regras de negócio (flexibilidade e reuso).
- Torna a aplicação escalável.

Desvantagens do modelo MVC:

- Requer uma quantidade maior de tempo para analisar e modelar o sistema.
- Não é aconselhável para pequenas aplicações
- É necessário conhecer o padrão para desenvolver.

Arquitetura de Repositório

A arquitetura em camadas e padrões MVC são exemplos nos quais a visão apresentada é a organização conceitual de um sistema (SOMMERVILLE, 2011). O padrão Repositório, descrito na Tabela 5.3, descreve como um conjunto de componentes que interagem podem compartilhar dados.

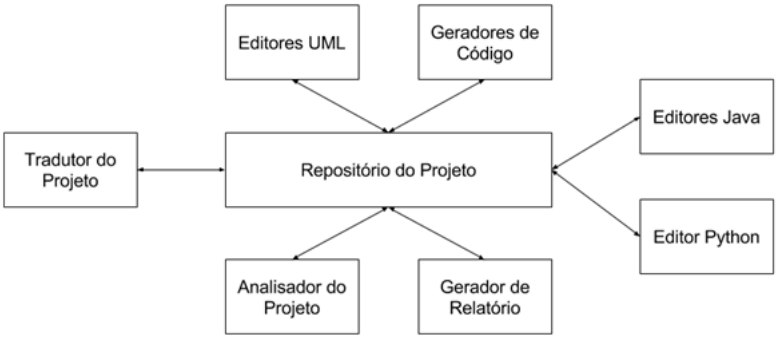
Tabela 5.3: O padrão Repositório (SOMMERVILLE, 2011, p. 112)

Nome	Repositório
Descrição	Todos os dados em um sistema são gerenciados em um repositório central, acessível a todos os componentes do sistema. Os componentes não interagem diretamente, apenas por meio do repositório.
Exemplo	A Figura 5.5 é um exemplo de um IDE em que os componentes usam um repositório de informações sobre projetos de sistema; Cada ferramenta de software gera informações que ficam disponíveis para uso por outras ferramentas.
Quando é usado	É usado quando tem um sistema no qual grandes volumes de informações são gerados e precisam ser armazenados por um longo tempo. Pode ser usado também em sistemas dirigidos a dados, nos quais a inclusão dos dados no repositório dispara uma ação ou ferramenta.
Vantagens	Os componentes podem ser independentes – não precisam saber da existência dos outros. As alterações feitas a um componente podem propagar-se para todos os outros. Todos os dados podem ser gerenciados de forma consistente.
Desvantagens	O repositório é um ponto único de falha, assim, problemas no repositório podem afetar todo o sistema. Pode haver ineficiências na organização de toda a comunicação por meio do repositório.

De acordo com Sommerville (2011) a maioria dos sistemas que usam grandes quantidades de dados é organizada em torno de um banco de dados ou repositório compartilhado. Esse modelo é, portanto, adequado para aplicações nas quais os dados são gerados por um componente e usados por outro.

A Figura 5.5 ilustra uma situação na qual um repositório pode ser usado, mostrando um IDE que inclui diversas ferramentas para apoiarem o desenvolvimento dirigido a modelos. O repositório nesse caso pode ser um ambiente controlado de versões que mantém o controle de alterações de software e permite a reversão para verões anteriores (SOMMERVILLE, 2011).

Figura 5.5: Uma arquitetura de repositório para uma IDE.



Fonte: SOMMERVILLE, 2011, p. 112

Arquitetura Cliente-Servidor

Um sistema que segue o padrão cliente-servidor é organizado como um conjunto de serviços e servidores associados e clientes que acessam e usam os serviços. De acordo com Sommerville (2011) os principais componentes desse modelo são:

- Um conjunto de servidores que oferecem serviços a outros componentes.
- Um conjunto de clientes que podem chamar os serviços oferecidos pelos servidores.
- Uma rede que permite os clientes acessar esses serviços. A maioria dos sistemas cliente-servidor são implementados como sistemas distribuídos, conectados por meio de protocolos de Internet.

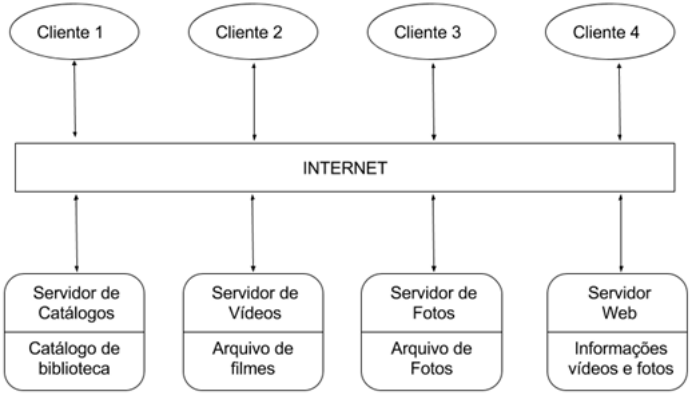
Sommerville (2011) descreve o padrão Cliente-Servidor na Tabela 5.4.

Tabela 5.4: O padrão Cliente-Servidor (SOMMERVILLE, 2011, p. 113)

Nome	Cliente-Servidor
Descrição	Em uma arquitetura cliente-servidor, a funcionalidade do sistema está organizada em serviços – cada serviço é prestado por um servidor. Os clientes são os usuários desses serviços e acessam os servidores para fazer uso deles.
Exemplo	A Figura 5.6 é um exemplo de uma biblioteca de filmes e vídeos/DVDs, organizados como um sistema cliente-servidor.
Quando é usado	É usado quando os dados em um banco de dados compartilhado precisam ser acessados a partir de uma série de locais. Como os servidores podem ser replicados, também pode ser usado quando a carga em um sistema é variável.
Vantagens	A principal vantagem é que os servidores podem ser distribuídos por meio de uma rede. A funcionalidade geral (por exemplo, um serviço de impressão) pode estar disponível para todos os clientes e não precisa ser implementada por todos os serviços.
Desvantagens	Cada serviço é um ponto único de falha suscetível a ataques de negação de serviço ou de falha do servidor. O desempenho, bem como o sistema, pode ser imprevisível, pois depende da rede. Pode haver problemas de gerenciamento se os servidores forem propriedade de diferentes organizações.

No exemplo da Figura 5.6, tem-se um sistema baseado no modelo cliente-servidor, multiusuário, baseado na Internet para fornecimento de uma biblioteca de filmes e fotos. Tal sistema é gerenciado por vários servidores. O catálogo deve ser capaz de lidar com uma variedade de consultas e fornecer links para o sistema de informação da Internet que incluam dados sobre os filmes. O programa do cliente é uma interface de usuário integrada, construída usando-se um browser da Internet para a acesso a esses serviços.

Figura 5.6: Uma arquitetura Cliente-Servidor para uma biblioteca de filmes.



Fonte: SOMMERVILLE, 2011, p. 114.

A principal vantagem do modelo cliente-servidor é que se trata de uma arquitetura distribuída. O uso efetivo pode ser feito por sistema em rede com muitos processadores distribuídos. Sommerville também destaca que é fácil adicionar um novo servidor e integrá-lo com o resto do sistema ou atualizar os servidores de forma transparente, sem afetar outras partes do sistema.

Nesta unidade você estudou sobre Projeto Arquitetural de Software. Na próxima unidade irá aprender mais sobre Análise de Requisitos de Software e de Sistemas.

É hora de se avaliar

Lembre-se de realizar as atividades desta unidade de estudo. Elas irão ajudá-lo a fixar o conteúdo, além de proporcionar sua autonomia no processo de ensino-aprendizagem.

Exercícios – Unidade 5

1. *Model-View-Controller* (MVC), ou traduzido para Modelo-Visão-Controlador, é um padrão de arquitetura de software que separa a lógica de negócios (*Model*), da interface do usuário (*View*) e do fluxo da aplicação (*Control*). São vantagens da arquitetura MVC, exceto:

a) Permitir a distribuição dos elementos da aplicação.

b) Permitir interfaces mais sofisticadas com o usuário.

c) Possuir melhor separação de responsabilidades.

d) Poder ser utilizada em aplicações simples, sem sobrecarregar o desenvolvimento com componentes desnecessários.

e) Garantir um crescimento constante da aplicação.

2. O componente *Controller* do padrão de arquitetura MVC:

a) Define o comportamento da aplicação, as ações do usuário para atualizar os componentes de dados e seleciona os componentes para exibir respostas de requisições.

b) É onde são concentradas todas as regras de negócio da aplicação e o acesso aos dados.

c) Envia requisições do usuário para o controlador e recebe dados atualizados dos componentes de acesso a dados.

d) Responde às solicitações de queries e encapsula o estado da aplicação.

e) Notifica os componentes de apresentação das mudanças efetuadas nos dados e expõe a funcionalidade da aplicação.

3. Qual é a arquitetura de software muito utilizada para desenvolvimento de aplicação web, onde a lógica da aplicação é implementada em uma camada separada da interface do usuário (entrada de dados e apresentação) e onde a comunicação entre as camadas se dá através de uma camada controladora?

a) Arquitetura 3 camadas: cliente, servidor de aplicação e banco de dados

b) Arquitetura Cliente / Servidor: cliente e banco de dados

c) Arquitetura de comunicação multicamadas

d) Arquitetura MVC

e) Arquitetura SOA

4. Assinale a alternativa correta sobre arquitetura em camadas. Uma arquitetura em camadas..

a) Possui apenas 3 camadas, cada uma realizando operações que se tornam progressivamente mais próximas do conjunto de instruções da máquina.

b) Tem, na camada mais interior, os componentes que implementam as operações de interface com o usuário.

c) Pode ser combinada com uma arquitetura centrada nos dados em muitas aplicações de bancos de dados.

d) Tem, como camada intermediária, o depósito de dados, também chamado de repositório ou quadro- negro.

e) Tem, na camada mais externa, os componentes que realizam a interface com o sistema operacional.

5. As seguintes atividades não fazem parte da fase de projeto de um software:

a) Estabelecer uma forma de organização interna que permita ao sistema atender aos diversos requisitos especificados.

b) Elaborar estudos de viabilidade técnico econômica do sistema.

c) Definir a arquitetura e o modelo de controle que serão empregados.

d) Escolher os frameworks e arquiteturas de referência que serão utilizados.

e) Elaborar diagramas utilizando a linguagem de modelagem unificada (*unified modeling language* - uml).

6. Há um tipo de padrão de projeto de software denominado arquitetural sobre o qual é correto dizer que, prioritariamente,

a) Contempla o relacionamento entre subsistemas e componentes do software.

b) Define a forma de elaboração do cronograma do projeto.

c) Define o editor de texto a ser utilizado nos documentos do projeto.

d) Estabelece o número de projetistas do projeto

e) Somente é utilizado na fase de teste do software.

7. Considerando o desenvolvimento de um projeto de software orientado a objetos, projetar a arquitetura do sistema envolve

a) Identificar as classes de objetos que compõem o sistema.

b) Desenvolver o modelo de projeto do sistema.

c) Especificar as interfaces entre os diversos componentes do sistema.

d) Identificar os principais componentes do sistema e suas interações.

e) Definir as interações entre o sistema e o ambiente em que está inserido.

8. O projeto arquitetural de software é um processo em que se visa a estabelecer uma organização de sistema que satisfaça os requisitos funcionais e não-funcionais do software em questão. Durante esse processo, o projetista deve tomar decisões que afetam diretamente o sistema e o seu processo de desenvolvimento, tal como a

a) Escolha da linguagem de programação.

b) Definição dos critérios de verificação e validação.

c) Adoção de modelos de arquitetura de referência.

d) Corretude das unidades estruturais.

e) Nenhuma das alternativas anteriores

9. Apresente as principais vantagens e desvantagens de se utilizar uma arquitetura cliente-servidor.

10. Sobre o padrão MVC aborde seu principal objetivo, vantagens e desvantagens de seu uso.

6 Análise de Requisitos de Software e de Sistemas

Caro(a) aluno(a),

Nesta unidade, estudaremos os principais conceitos sobre Requisitos de um sistema de software. Os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento. Tais requisitos refletem as necessidades dos clientes para um sistema. O processo de descobrir (levantar ou elicitar), analisar, documentar e verificar esses requisitos é chamado de Engenharia de Requisitos.

Objetivos da unidade:

O objetivo desta unidade é apresentar os principais conceitos sobre os requisitos de software e de sistema, além de discutir os processos envolvidos no levantamento e documentação desses requisitos. Espera-se que após o estudo dessa unidade, você seja capaz de:

- Compreender os conceitos de requisitos de usuário e de sistema (bem como suas diferenças).
- Entender a diferença entre requisitos funcionais e não-funcionais.
- Compreender como os requisitos podem ser documentados.
- Compreender as principais atividades do levantamento, análise e validação dos requisitos e as relações entre essas atividades.
- Entender sobre o gerenciamento de requisitos.

Plano da unidade:

- Engenharia de Sistemas de Computador
- Engenharia de Requisitos de Software
- Atividade de Análise de Requisitos de Software
- Problemas na Análise de Requisitos

Bons estudos!

Engenharia de Sistemas de Computador

Para melhor compreensão de aspectos relativos à Engenharia de Sistemas de Computador ou Engenharia de Sistemas Computacionais são relevantes algumas definições sobre um sistema computacional.

Um sistema computacional (ou sistema, dado ao contexto) é um conjunto de componentes que se relacionam para alcançar um determinado objetivo. Sistemas são construídos para automatizar ou apoiar atividades humanas por meio de tecnologias e processamento de informações.

A Figura 6.1 apresenta os principais componentes de um sistema:

Figura 6.1: Principais componentes de um sistema**.**

São componentes de um sistema:

- **Software**: conjunto de componentes lógicos de um computador (programas de computador) ou sistema de processamento de dados.

- **Hardware**: conjunto dos componentes físicos (material eletrônico, placas, monitor, equipamentos periféricos etc.) de um computador.
- **Pessoas**: fundamentais para utilizar os sistemas.
- **Banco de Dados/Informações:** conjunto de arquivos relacionados entre si. São coleções organizadas de dados que se relacionam de forma a criar algum sentido (Informação).
- **Documentação**: conjunto de todos documentos, fontes contendo informações que ajudem a tomar decisões, comuniquem decisões tomadas, registrem assuntos de interesse da organização.
- **Procedimentos**: modo como algo é executado, ou seja, como é feito o processo.

Os sistemas não são independentes, eles dependem do ambiente no qual está inserido, das pessoas que irão utilizá-lo, das regras de negócio e do contexto que ele faz parte, dentre outros. As funções de um sistema podem, inclusive, fazer alterações no ambiente, assim como o ambiente pode afetar o funcionamento do sistema, como, por exemplo, se faltar eletricidade o sistema pode não funcionar.

A Engenharia de Computador é composta pela Análise do Sistema e pela Engenharia do Sistema. A Análise de Sistemas é uma atividade composta por tarefas como a identificação das necessidades, estudo de viabilidade, análise técnica e econômica, definição dos requisitos e planejamento do desenvolvimento. A Engenharia do Sistema é composta pela Engenharia do Hardware (equipamentos utilizados, infraestrutura, redes etc.), Engenharia de Software (ferramentas, tecnologias e processos) e Engenharia de Processos, Informações e Pessoas.

O principal papel da Engenharia de Computador é a resolução de problemas e sua principal atividade é identificar, analisar e atribuir aos componentes do sistema as funções desejadas. Outro ponto importante é a multidisciplinaridade e interdisciplinaridade que compõe a Engenharia de Computador. Além do Software e Hardware, sistemas necessitam também de outros especialistas, que podem ser: especialistas em Engenharia Elétrica, Mecânica, Mecatrônica, Hidráulica, Químicos, Físicos, Telecomunicações, Médicos, entre outros.

Para conceber um sistema, a ação que determinar o ponto de partida é analisar!

Os objetivos da análise de sistemas são:

- Identificar as reais necessidades do usuário;
- Avaliar a viabilidade do sistema tanto técnica como econômica;
- Atribuir funções aos elementos do sistema (software, hardware, pessoas, banco de dados etc.);
- Estabelecer restrições de prazos e custos;
- Criar uma definição do sistema;
- Elaborar o documento "Especificação do Sistema".

A Análise de Sistemas é composta dos seguintes passos:

- Estudo da necessidade:
    - Reunião com clientes e demais interessados para entender as necessidades e desejos para então definir as funções, metas globais, futuras extensões, escopo e não escopo, confiabilidade e outras restrições.
    - Como resultado desse estudo se tem o Documento Conceitual do Sistema.
- Estudo da Viabilidade do Sistema:
    - Análise da viabilidade econômica e técnica.
    - Análise de aspectos jurídicos;
    - Análise de alternativas e possíveis soluções.
    - Como resultado desse estudo se tem o Documento de Estudo de Viabilidade.
- Análise de Requisitos:
    - Processo que estabelece funcionalidades necessárias e restrições de operações e desenvolvimento.

- Participam dessa fase o cliente, usuários, especialistas e o analista de sistemas. O analista de sistemas é quem define os elementos para um sistema específico.
- Reconhecimento do problema: entender a especificação do sistema, o que faz e o que não faz parte do escopo.

Determinada a necessidade e a viabilidade do Sistema, passa-se então para a Análise dos Requisitos e restrições formuladas pelo cliente. Tais requisitos e restrições permitirão delinear inicialmente as funções do sistema, questões como desempenho, interfaces com usuário e suas restrições, estruturas das informações a serem processadas, etc.

Os tipos de requisitos são:

- **Requisitos do usuário:** declarações em linguagem natural, com uso de diagramas dos serviços que o sistema deverá fornecer e as restrições desses serviços. Estes requisitos são escritos para os clientes, utilizando uma modelo de linguagem que eles compreendem.
- **Requisitos de sistema:** são descrições mais detalhadas dos requisitos do usuário. Compõem um documento estruturado estabelecendo descrições detalhadas das funções do sistema, serviços e restrições operacionais. Esse documento serve como definição do que deve ser implementado, podendo ser utilizando como parte de um contrato.

Os requisitos de software são classificados como:

- **Requisitos funcionais:** declarações de serviços/funcionalidades do que o sistema deve fornecer, formas que o sistema deve reagir a entradas específicas e como o sistema deve se comportar dada uma situação. Os requisitos funcionais descrevem o que um sistema deve fazer. Tais requisitos dependem dos usuários e do ambiente (organização) no qual ele está inserido.
- **Requisitos não funcionais:** são restrições que são impostas aos serviços ou funcionalidades do sistema. Normalmente são regras que incidem sobre o sistema como um todo: desempenho, confiabilidade, segurança, disponibilidade, restrições legais ou governamentais, entre outros.

Engenharia de Requisitos de Software

Entender os requisitos de um problema está entre as tarefas mais difíceis dentre as desempenhadas por um engenheiro de software. Ao parar para analisar essa não deveria ser uma tarefa difícil, pois na teoria o cliente sabe o que ele precisa e o usuário final entende bem as características e funcionalidades que ele irá utilizar. Não é mesmo?

Na prática isso é um pouco diferente. E mesmo que os clientes e usuários tivesse certeza do que querem, as coisas evoluem, as necessidades e prioridades se alteram com o passar do tempo, então, em um projeto algo que é verdade é que: vai mudar!

Existem diferentes visões quando se trata de construir software. Alguns defendem que as coisas só ficarão claras após a entrega de alguns incrementos, logo, não se deve preocupar com detalhar os requisitos no começo e, sim, começar a construir logo para que o cliente e usuário final possam avaliar e verem se o que está sendo atende às suas necessidades. Outros, numa linha mais tradicional, defendem uma análise e projeto de requisitos bem detalhados, defendendo que o custo de mudança e o tempo de alterações com o passar do tempo de projeto aumenta.

Engenharia de Requisitos é o conjunto de técnicas e tarefas para o entendimento de requisitos de um projeto. Tal conjunto fornece mecanismos apropriados para o entendimento das necessidades do cliente, avaliando o que ele deseja a viabilidade disso e, se necessário, formas de negociação com o cliente para uma solução mais adequada ao problema de projeto proposto.

A Engenharia de Requisitos possui ferramentas para especificar uma solução de software ausente de ambiguidades, validar a especificação e gerenciar as necessidades à medida que são construídas.

Os principais objetivos da Engenharia de Requisitos são:

- **Investigação de objetivos, funções e restrições:** compreender necessidades, prioridades e grau de satisfação; associar os requisitos com vários agentes de domínio; estimar custos, riscos e cronogramas; assegurar a completude.

- **Especificação do software:** integrar formas de representar e as múltiplas visões; avaliar estratégias de soluções alternativas; obter uma especificação completa, consistente e livre de ambiguidade; validar a especificação para validar se ela satisfaz os requisitos.
- **Gerenciamento da evolução:** reutilização de requisitos durante a evolução e desenvolvimento de softwares similares; reconstrução de requisitos.

A Engenharia de Requisitos abrange 7 tarefas distintas (PRESSMAN, 2011):

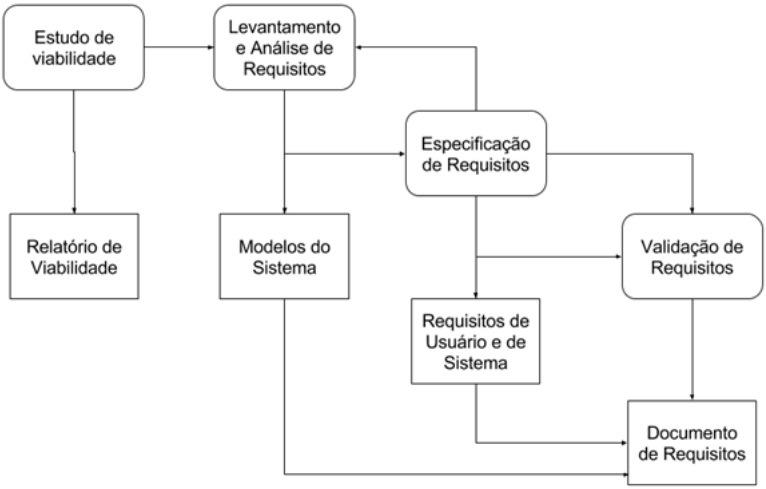
- **Concepção**: geralmente um projeto se inicial com a identificação da necessidade do negócio ou quando é descoberto um novo serviço ou mercado potencial. Na concepção do projeto se estabelece um entendimento básico do problema, quem são os principais interessados, a equipe, métodos de comunicação e colaboração, entre outros.
- **Levantamento**: etapa em que se levanta com os clientes, usuários e especialistas o que deve (ou não deve) ser feito, os objetivos para o sistema ou produto e como o sistema atende às necessidades de cada usuário. E isso é uma tarefa muito difícil! Podem surgir problemas tais como: problemas de escopo, como indefinições por parte do cliente; problemas de entendimento, especialmente quando clientes e usuários não sabem ao certo o que precisam ou tem um conhecimento inadequado ou insuficiente das regras de negócio, do domínio do problema ou de seus ambientes computacionais; problemas de volatilidade, pois os requisitos mudam com o tempo.
- **Elaboração**: nessa etapa se faz o refinamento e expansão das informações obtidas durante as etapas de concepção e levantamento. Essa tarefa tem como principal papel desenvolver um modelo de requisitos que seja capaz de identificar diversos aspectos funcionais, comportamentais e de informações do software.
- **Negociação**: É mais comum do que deveria usuários e clientes pedirem mais que pode ser feito. E, mais ainda, usuários e clientes proporem necessidades conflitantes, especialmente quando a lista de interessados é grande. Nessa etapa devem-se conciliar esses conflitos por meio de

processo de negociação, no qual se deve ordenar e priorizar os requisitos, discutir em termos de prioridade e avaliar custos e riscos. O importante é tratar conflitos internos, eliminar, combinar e/ou modificar os requisitos necessários de forma que se alcance um nível de satisfação dos interessados.

- **Especificação**: No contexto de software o termo especificação pode ter conotações diferentes: um documento por escrito, um conjunto de modelos gráficos, um modelo matemático formal, um conjunto de cenários de uso, um protótipo ou qualquer combinação desses. A melhorar abordagem a ser utilizada para especificar o projeto deve ser avaliada de acordo com a necessidade e complexidade do que será executado.
- **Validação**: etapa em que se examina a especificação para garantir que todos os requisitos de software estejam livres de ambiguidades; que as inconsistências, omissões e erros tenham sido detectados e corrigidos e que os artefatos estejam de acordo com os padrões estabelecidos para o processo, projeto e produto.
- **Gestão**: gestão de requisitos é um conjunto de atividades que auxiliam a equipe de projeto a identificar, controlar e acompanhar as necessidades e suas mudanças durante todo o ciclo de vida do projeto.

Sommerville (2011), resume essas tarefas em 4 atividades básicas, apresentadas na Figura 6.2.

Figura 6.2: Engenharia de Requisitos



Fonte: Pressman, 2011.

A Figura 6.2 apresenta as quatro atividades básicas: Estudo da Viabilidade, Levantamento e Análise de Requisitos, Especificação de Requisitos e Validação de Requisitos. Apresenta também o resultado de cada atividade, por exemplo, após o estudo de viabilidade tem-se o Relatório de Viabilidade; após o levantamento de requisitos têm-se os Modelos de sistema que são bases para o documento de requisitos, o qual é criado na especificação de requisitos e finalizado após a validação.

A Engenharia de Requisitos deve envolver a documentação de funções, do desempenho, interfaces externas e internas e atributos de qualidade de Software. Pode-se dizer que é uma área que tem por finalidade interpretar as observações do mundo real e transformá-las em uma notação apropriada para se tornar um sistema.

São citados como benefícios da Engenharia de Requisitos:

- Um meio para chegar a um acordo (concordância) entre clientes, usuários e equipe de projeto sobre o trabalho a ser executado e quais os critérios de aceitação.

- Uma base mais precisa de estimativa de recursos (humanos, financeiro, de tecnologias e ferramentas necessárias, prazos e infraestrutura).
- Melhoria na usabilidade, manutenibilidade e outras qualidades do sistema.

Uma boa especificação de requisitos deve ser clara e não ambígua, além de completa, correta e de fácil compreensão. Outras características importantes são a confiabilidade, consistência e concisão do que está especificado.

Atividade de Análise de Requisitos de Software

Como apresentado na Seção 6.2, os processos da Engenharia de Requisitos podem incluir tarefas desde estudo da viabilidade, levantamento e entendimento dos requisitos, especificação em algum formato e validação.

Sommerville (2011) apresenta uma visão espiral do processo de Engenharia de Requisitos, apresentada na Figura 6.3.

Após o início do projeto é realizada a especificação dos requisitos de negócio e levantamento de requisitos não funcionais em alto nível, além de requisitos do usuário. Posteriormente, o processo consiste em levantar e compreender os requisitos de sistema de forma detalhada.

Figura 6.3: Uma visão espiral do processo de engenharia de requisitos.

Fonte: Adaptado de SOMMERVILLE, 2011, p. 70.

Estudo da Viabilidade

De acordo com Sommerville (2011), no processo de Engenharia de Requisitos, a primeira atividade é o estudo de viabilidade do sistema a ser concebido. Tal estudo inicia-se com uma descrição geral do sistema e do ambiente ao qual ele será inserido. A descrição do ambiente, como por exemplo, a forma como ele será utilizado dentro de uma organização, tem impacto importante para os resultados da análise da viabilidade.

Como resultado dessa atividade tem-se um relatório que traz indicações se é viável ou não realizar o processo de Engenharia e Requisitos e, em seguida, o processo de desenvolvimento do sistema.

O processo de estudo de viabilidade deve ser rápido e direcionado. É importante que essa atividade responda os seguintes questionamentos:

- O sistema contribui para os objetivos gerais da organização?
- Obedecendo as restrições de custo e prazo o sistema proposto pode ser implementado utilizando as tecnologias disponíveis?
- Existe a possibilidade de integrar esse sistema da forma como está sendo proposto com outros já em operação?

Avaliar se o sistema contribui (ou não) para os objetivos da empresa é essencial, pois caso a resposta seja negativa, tal sistema não irá agregar valor, não tendo sentido investir recursos em sua construção.

Alguns exemplos de questionamentos que podem ser feitos durantes a análise de viabilidade:

- Qual seria o comportamento da organização se esse sistema não fosse implementado?
- Qual a real contribuição desse sistema para os objetivos da empresa?
- Quais os problemas com os processos atuais e como esse sistema ajudaria a solucionar esses problemas?
- É possível transitar informações entre esse novo sistema e os já existentes na organização?

- O que precisa ser compatível com o sistema?
- O sistema requer o uso de tecnologia que ainda não tenha sido utilizada pela organização?

Para responder tais questões os principais interessados (cliente, especialistas, usuários finais) devem ser consultados e, então, o relatório de estudo de viabilidade deverá ser elaborado.

Levantamento e Análise de Requisitos

A primeira atividade após a análise da viabilidade é o levantamento e análise de requisitos. Nessa atividade os analistas de requisitos (ou engenheiros de software ou o profissional alocado para a atividade de levantamento de requisitos) conversam com os clientes, usuários finais, especialistas e demais interessados no sistema para obter informações sobre o domínio da aplicação, os serviços que o sistema deve oferecer e as restrições.

São atividades da atividade do processo de levantamento e análise de requisitos:

- **Levantamento dos requisitos:** atividade onde os analistas conversam com os interessados para levantamento dos requisitos.
- **Classificação e organização dos requisitos:** os requisitos coletados na descoberta são listados de forma não estruturada e, então, a atividade de classificação e organização, agrupa os requisitos relacionados e os organiza em grupos.
- **Priorização e negociação de requisitos:** quando há várias pessoas interessadas envolvidas podem ocorrer conflitos entre os requisitos. Essa atividade está relacionada em priorizar requisitos e buscar uma solução para os conflitos que surgirem por meio de uma negociação.
- **Especificação de requisitos:** os requisitos são documentados (formal ou informal).

Descoberta dos Requisitos

A descoberta de requisitos é o processo de reunir informações sobre o sistema. São fontes de informações sobre o sistema: documentação, *steakholders* (clientes, usuários finais e demais interessados) e especificações de sistemas com contextos similares.

A interação com os *steakholders* podem ocorrer por meio de observação e entrevistas que serão apresentadas a seguir. Além deles, os requisitos podem vir a partir do domínio da aplicação e de outros sistemas que interagem com o que está sendo especificado.

A seguir serão apresentadas algumas técnicas para levantamento dos requisitos. Existem outras: Cenários, Prototipação, Casos de Uso, Brainstorming, Questionários, etc. É importante e indicado que você leia sobre elas!

Entrevistas

Uma entrevista é uma abordagem, seja formal ou informal, para levantamento de informações junto a uma pessoa ou a um grupo de pessoas por meio de uma conversa com o entrevistado. Nesse processo o entrevistador faz perguntas ao entrevistado e as respostas são documentadas.

As entrevistas podem ser dois tipos:

1. **Entrevistas Estruturadas (ou fechadas):** os interessados respondem a um conjunto pré-definido de perguntas.
2. **Entrevistas Não estruturadas (ou abertas):** não existe uma agenda ou questões pré-definidas, o entrevistador e o entrevistado discutem abertamente os tópicos.

O nível de compreensão do domínio por parte do entrevistado, experiência do entrevistador na condução de entrevistas, habilidade do entrevistador em documentar, a boa vontade do entrevistado em responder os questionamentos e a empatia são fatores que definem o sucesso de uma entrevista.

São vantagens do uso de entrevistas:

- Técnica simples e direta que pode se usada em diversos ambientes e situações;

- Permite que o entrevistador e o entrevistado discutam e reflitam sobre as perguntas e respostas. Dúvidas sobre contextos, perguntas e respostas podem ser esclarecidas.
- O entrevistador tem possibilidade de refazer perguntar, até mesmo de outra forma, para confirmar informações.
- Permite aos entrevistados expressarem opiniões de forma individual e privada.
- Permite observação de outros aspectos, por exemplo, não verbais.

São desvantagens do uso de entrevistas:

- Requer dedicação, envolvimento e interesse por parte dos participantes.
- Difícil de alcançar um consenso quando é realizada em grupo.
- É necessário que o entrevistador tenha treinamentos para conduzir entrevistas efetivas.
- Entrevistadores precisam ter algumas habilidades para captar as informações de forma correta e adequada.
- A documentação e análise dos dados da entrevista pode ser um processo caro e complexo.

Etnografia

Etnografia é uma técnica de observação que pode ser usada para compreender os processo s operacionais e auxiliar na extração dos requisitos de apoio a esses processos. Tal método é utilizado pela antropologia na coleta de dados e se baseia no contato entre o antropólogo e seu objeto de estudo, geralmente um grupo social constituído formalmente.

Na engenharia de software a etnografia é caracterizada como uma técnica de observação utilizada para mapear requisitos implícitos que refletem processos reais dentro de um ambiente sistêmico. Compreender requisitos sociais e organizacionais, promover um entendimento dos aspectos culturais que regem o ambiente sistêmico direcionam os procedimentos etnográficos.

De acordo com Sommerville (2011) a Etnografia é eficaz par descobrir dois tipos de requisitos:

1. Requisitos derivados da maneira como as pessoas realmente trabalham, e não da forma como as definições dos processos dizem que deveriam trabalhar.
2. Requisitos derivados da cooperação e conhecimento das atividades de outras pessoas.

Documentação/Especificação dos Requisitos

Após o levantamento e análise dos requisitos a próxima atividade é a criação do documento de requisitos. De acordo com Pressman (2011) uma especificação pode ser um documento escrito, um modelo gráfico ou matemático formal, um protótipo ou a combinação desses. Não há um padrão determinado para a documentação de requisitos e diversas propostas de metodologias e modelos são propostas, porém, é consenso que a documentação deve ser coerente com o projeto e os riscos, os quais definem o nível de detalhamento.

O analista de requisitos é o responsável pela transcrição das informações coletadas no levantamento de requisitos para o documento. Tal documento contém informações do que o sistema deve fazer, das reais necessidades dos usuários e as restrições impostas para que o sistema execute o que foi proposto.

Validação dos Requisitos

A validação é uma atividade que tem como objetivo verificar o documento de requisitos para garantir que a necessidade real do usuário esteja descrita corretamente. Obviamente é melhor encontrar um erro nesta fase em que se está entendendo o que precisa ser feito do que durante a construção e implementação, ou até mesmo, depois de entregue o software. Por isso essa tarefa é importante!

O processo de validação auxilia encontrar, caso existam, inconsistências, ambiguidades ou até mesmo perceber omissões. São propostas por Sommerville (2011) algumas técnicas de validação, tais como, revisões de requisitos (por outras pessoas), prototipação e geração de casos de teste.

São exemplos de verificações da validação:

- Quais serviços são necessários? -- Verifica a validade.
- Existe algum conflito entre os requisitos? -- Verifica a consistência.
- Todos os requisitos foram documentados? -- Verifica a completude.
- Os requisitos podem ser implementados? -- Verifica o realismo.
- É possível escrever um conjunto de testes que demostrem o sistema que será entregue? – Verificabilidade.

Existem técnicas de validação de requisitos, as quais podem ser utilizadas individualmente ou em conjunto:

- **Revisões de Requisitos**: verificação por meio de revisores para buscar erros e inconsistências.
- **Prototipação:** um modelo executável do sistema é criado e apresentado aos usuários finais e clientes, que podem experimentar o modelo para verificar se ele atende a suas reais necessidades.
- **Geração de Casos de Teste:** os requisitos devem ser testáveis. Se for difícil conceber um teste a partir dos requisitos, logo é difícil implementá-lo da forma como está descrito.

De acordo com Sommerville (2011) não se deve subestimar as dificuldades no processo de validação de requisitos. A validação de requisitos não consegue descobrir todos os problemas, por isso, mesmo depois de ter sido aceito a versão final do documento, modificações podem surgir para corrigir falhas e omissões por falta de compreensão inicial.

Gerenciamento dos Requisitos

Gerenciamento de Requisitos é o processo de compreensão e controle das mudanças nos requisitos do sistema, que deve ser realizado por meio de um processo formal de mudanças e, também, deve ser possível avaliar o impacto das mudanças nos requisitos.

Normalmente softwares são desenvolvidos para solucionar problemas que não podem ser completamente definidos. Logo, isto faz com que os requisitos não sejam completos. Além disso, durante o processo de desenvolvimento do software, o entendimento dos interessados a respeito do problema pode mudar e essas novas percepções também traz alterações nos requisitos. Outro cenário que traz alterações nos requisitos é o uso efetivo do software pelos usuários finais, que com o tempo, percebem novas necessidades e prioridades.

Sommerville (2011) apresenta três razões pelas quais as mudanças são inevitáveis:

1. A mudança no ambiente técnico e de negócio, como por exemplo, um novo hardware, interface e integração com um novo sistema, novas legislações ou regulamentos as quais o sistema deve respeitar etc.
2. As pessoas que solicitam (e pagam) pelo sistema normalmente não são os usuários finais. Logo, podem surgir conflitos de interesses.
3. Sistemas de grande porte normalmente têm muitos envolvidos e interessados que podem ser conflitantes e contraditórios.

No processo de gerenciamento de requisitos o primeiro estágio essencial é o planejamento. Os requisitos devem ser identificados unicamente; deve-se ter definido um processo de gerenciamento de mudanças (conjunto de atividades que avaliam o impacto e o custo da mudança); devem-se definir as políticas de rastreabilidade; e as ferramentas de apoio.

Outro estágio essencial é o gerenciamento das mudanças para definir se os benefícios da implementação de novos requisitos justificam os custos. Existem três estágios principais (SOMMERVILLE, 2011):

1. Análise do problema e especificação de mudanças.
2. Análise de mudança e custos.
3. Implementação de mudanças.

Problemas na Análise de Requisitos

Sommerville (2011) como as principais dificuldades de se levantar requisitos como:

- As pessoas podem fazer exigências inviáveis ou desnecessárias, por muitas vezes não saberem articular o que querem de um sistema computacional.
- Analistas de requisitos podem ter dificuldades de entender as regras de negócio expressadas por pessoas com conhecimento implícito do seu próprio trabalho.
- Quando se tem muitas pessoas interessadas, tais pessoas podem expressar requisitos diferentes ou os mesmos de várias maneiras. O analista precisa identificar as semelhanças e conflitos.
- Com o passar do tempo do projeto novos requisitos podem surgir com alterações no modelo de negócio, ou até mesmo com alterações (inclusões ou exclusões) de pessoas interessadas. Pessoas diferentes podem ter visões diferentes de necessidades e funcionalidades.

Estamos encerrando a unidade. Sempre que tiver uma dúvida entre em contato com seu tutor virtual através do ambiente virtual de aprendizagem e consulte sempre a biblioteca virtual.

É hora de se avaliar

Lembre-se de realizar as atividades desta unidade de estudo. Elas irão ajudá-lo a fixar o conteúdo, além de proporcionar sua autonomia no processo de ensino-aprendizagem.

Exercícios – Unidade 6

1. Os requisitos de sistema de software são frequentemente classificados como requisitos:

a) Funcionais e não funcionais.

b) Externos e organizacionais.

c) De reuso e de interatividade.

d) De caso de uso e de sistema.

e) De usuários.

2. No que diz respeito à Engenharia de Software, um processo é um conjunto de atividades e resultados associados, cujo objetivo é o desenvolvimento e a produção do software. Existem quatro atividades fundamentais de processo, duas das quais são definidas a seguir.

I. O software é modificado para se adaptar às mudanças dos requisitos do cliente e do mercado.

II. O software é testado para garantir que o produto gerado é o que o cliente deseja.

As atividades I e II são denominadas, respectivamente:

a) Evolução do Software e Homologação do Software
b) Especificação do Software e Homologação do Software
c) Evolução do Software e Validação do Software
d) Especificação do Software e Validação do Software
e) Verificação do Software e Validação do Software

3. Sobre os requisitos de software, é correto afirmar que:

a) Quando os requisitos são documentados, não há problemas de ambiguidade.
b) A área de estudo de requisitos de software está relacionada apenas ao levantamento, análise e validação de requisitos.
c) A maioria das falhas relacionadas aos requisitos em projetos de software se devem às dificuldades em entender o que o usuário quer e a descrições incompletas e mudanças não controladas nos requisitos.
d) Os requisitos definem, em princípio, o que o software deve fazer. Não é preciso que fique claro, em nenhum momento, como as operações serão realizadas.
e) Durante o ciclo de vida de um software, os requisitos não devem sofrer influência de pessoas ou de grupos de pessoas para que não haja inconsistências no desenvolvimento.

4. Que requisitos no documento de especificação de requisitos descrevem como dados recebidos pelo software devem ser transformados em respostas (saídas). Estes requisitos descrevem as ações fundamentais que devem ser realizadas pelo software.

a) Requisitos Funcionais
b) Requisitos de Interface
c) Requisitos do Cliente
d) Requisitos de Processamento Primário
e) Requisitos de Interface de Processamento

5.Assinale a alternativa que complete corretamente a lacuna.

"_____ é uma abordagem, seja formal ou informal, para levantamento de informações junto a uma pessoa ou a um grupo de pessoas por meio de uma conversa."

a) Questionário
b) Etnografia.

c) Levantamento de Requisitos

d) Entrevista

e) Requisitos

6. "Em uma das etapas da Engenharia de Requisitos há a preocupação em se observar a especificação produzida, visando verificar que os requisitos tenham sido declarados, por exemplo, sem ambiguidades."

O texto refere-se à etapa de :

a) Gestão dos requisitos.

b) Elicitação dos requisitos.

c) Negociação dos requisitos.

d) Levantamento dos requisitos.

e) Validação dos requisitos.

7. O que possibilita seguir um requisito, a partir de sua ori-gem, passando por seu desenvolvimento e especifica-ção, inclusive o projeto correspondente, é denominado:

a) Refinamento de requisitos.

b) Qualidade de requisitos.

c) Engenharia de requisitos

d) Rastreabilidade de requisitos

e) Mudança de requisitos

8. Assinale a alternativa que complete corretamente as lacunas.

"Existem técnicas de validação de requisitos, as quais podem ser utilizadas individualmente ou em conjunto: _____ que é a verificação para buscar erros e inconsistências; _____ que é um modelo executável do sistema é criado e apresentado aos usuários finais e clientes, que podem experimentar o modelo para verificar se ele atende a suas reais necessidades."

a) Revisões de Requisitos - Prototipação

b) Geração de Casos de Testes - Modelo

c) Geração de Casos de Teste - Prototipação

d) Revisão de Requisitos - Modelo

e) Testes de Sistemas – Protótipos

9. Quais as principais dificuldades encontradas no processo de levantamento de requisitos?

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

10. O que é Etnografia? Explique sobre esse conceito e suas aplicações.

Considerações finais

Aluno(a),

Chegamos ao final dos estudos da disciplina de Engenharia de Software.

Agora que seus conceitos fundamentais foram assimilados, já é possível entender melhor sobre o processo de desenvolvimento de software, bem como os artefatos, ferramentas, metodologias, padrões e melhores práticas adotadas pra se construir software com qualidade.

Procurei organizar a disciplina de maneira a abordar o conteúdo programático, com base nos autores clássicos e referências da área.

Espero que este material tenha contribuído para sua formação e capacitação profissional, e que o mesmo seja um estímulo na continuidade do referido curso.

Parabéns e sucesso!

Luciane de Fátima Silva

Conhecendo o autor

Luciane de Fátima Silva

Doutoranda em Ciência da Computação com ênfase em Inteligência Artificial aplicada à Educação na Universidade Federal de Uberlândia. Obteve o título de Mestre em Ciência da Computação com ênfase em Engenharia de Software pela Faculdade de Computação da Universidade Federal de Uberlândia em 2014. Graduada em Ciência da Computação pela Universidade Federal de Uberlândia (2010). Trabalha como docente no Centro Universitário do Triângulo nos cursos de Sistemas de Informação e Ciência da Computação, ministrando disciplinas de Engenharia de Software, Qualidade de Software, Teste e Manutenção de Software, Fundamentos de Banco de Dados, Controle e Avaliação de Sistemas e Lógica para Ciência da Computação desde 2014. Trabalha como Coordenadora de Programa de Estágio na Neppo Tecnologia da Informação atuando nessa empresa desde 2010.

Referências

BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAND, P. e STAHL, M. **Pattern-Oriented Software Architecture - A System of Patterns**, Nova York, NY: John Wiley and Sons, Inc., 1996.

COCKBURN, A. **Escrevendo Casos de Uso Eficazes - Um Guia Prático para Desenvolvedores de Software**. Bookman, 2005.

FILHO, W. P. P. **Engenharia de Software: Fundamentos, Métodos e Padrões**, Editora LTC, 2009.

HIRAMA, K. **Engenharia de software: Qualidade e produtividade com tecnologia**. Editora: Elsevier, 2012.

MACHADO, F. N. **Análise e Gestão de Requisitos de Software: Onde Nascem os Sistemas**, Erica, 2011.

PFLEEGER, S. L. **Engenharia de Software**, Editora Prentice-Hall, 2004.

PRESSMAN, R. **Engenharia de Software**, 6a edição, McGraw Hill, 2006.

SAVIC, D. Evolution of information resources management. Journal of librarianship andiInformation science, v. 2, n.3, 1992.

SBROCCO, J. H. T. C. **Metodologias Ágeis: Engenharia de Software Sob Medida**. 1ª Ed. Erica, 2012.

SOMMERVILLE, I. **Engenharia de Software**. 8a. edição, São Paulo: Pearson Prentice-Hall, 2012.

TONSIG, S. L. **Engenharia de software: análise e projeto de sistemas**. Rio de Janeiro: Ciência Moderna, 2008.

Anexos

Gabaritos

Exercícios – Unidade 1

1) e

2) a

3) c

4) e

5) b

6) b

7) a

8) d

9) As principais causas são: Falta de dedicação de tempo e esforço para coletar dados sobre o desenvolvimento de software, que resultam em estimativas mal feitas. Falhas na comunicação entre o cliente e a equipe de desenvolvimento. Falta de testes sistemáticos e completos. Planejamento, projeto e construção executados por equipe sem formação específica ou com baixa capacidade técnica. Falta de investimentos (treinamentos, especialização, equipe, ferramentas etc.). Falta de métodos e de processos automatizados.

10) Existem quatro atividades fundamentais comuns a todos os processos de software: especificação do software (em que clientes e engenheiros definem o software a ser produzido e as restrições de operação), desenvolvimento de software (em que o software é projetado e programado), validação (o software é verificado para garantir que é o que o cliente quer) e evolução do software (o software é modificado para refletir a mudança de requisitos do cliente e do mercado). Além disso, existem ideias fundamentais que se aplicam a todos os tipos de sistemas de software: Eles devem ser desenvolvidos em um processo gerenciado e compreendido; Confiança e desempenho são importantes para todos os tipos de sistemas; É importante entender e gerenciar a especificação e requisitos de software; Deve ser feito o melhor uso possível dos recursos existentes.

Exercícios – Unidade 2

1) e

2) c

3) a

4) b

5) b

6) e

7) a

8) b

9) O modelo espiral proposto pro Boehm é um modelo de processo de software evolucionário. O processo de software nesse modelo é representado como se fosse uma espiral, e não como uma sequência de atividades como em outros modelos. Cada volta na espiral representa uma fase do processo de desenvolvimento de software. Assim, a volta mais interna pode ser um protótipo para validar a viabilidade do sistema, por exemplo; o ciclo seguinte voltado para a definição dos requisitos; o seguinte para o projeto do sistema e assim por diante. Cada volta da espiral é dividida em quatro setores: (1) Definição de objetivos: objetivos específicos para essa fase do projeto são definidos; restrições ao processo e ao produto são identificadas e um plano de gerenciamento detalhado é elaborado; os riscos são identificados. Estratégias alternativas podem ser planejadas em função desses riscos. (2) Avaliação e Redução de Riscos: para cada um dos riscos identificados do projeto é feita uma análise detalhada e, então medidas são tomadas para a redução dos riscos. (3) Desenvolvimento e validação: após avaliação dos riscos é selecionado um modelo de desenvolvimento para o sistema. (4) Planejamento: o projeto é revisado e uma decisão é tomada a respeito da continuidade do modelo com mais uma volta da espiral. Caso seja decidido pela continuidade, planos são elaborados para a próxima fase do projeto.

10) O Modelo Incremental apresenta diversas vantagens para o desenvolvimento de um software, especialmente se os requisitos não estão claros inicialmente. Nesse modelo as versões do software são fornecidas após cada interação e, dessa forma, o usuário pode perceber mudanças necessárias antes da finalização do projeto. Além disso, é um modelo flexível e fácil de gerenciar,

especialmente os riscos, pelo fato do cliente (usuário) participar a cada versão e validar se está de acordo com o que está sendo feito. As inconformidades são corrigidas antes da entrega da próxima versão, normalmente. Outras vantagens são: Quando se constrói algo menor o risco é reduzido se comparado a construir algo maior; Quando ocorre um erro em um incremento, os demais (anteriores) nem sempre são diretamente afetados. Em casos de erros graves, apenas o último incremento seria descartada (pior caso) ou ajustada; Reduzindo o tempo de desenvolvimento de um sistema é esperado que se reduzisse as chances de mudanças nos requisitos do usuário no final, visto que esse já participa validando cada versão; A quantidade de análise e documentação a ser refeita é menor quando comparada ao modelo de desenvolvimento em cascata; Nesse modelo é previsto que os clientes (usuários) façam comentários/feedbacks sobre o que foi desenvolvido. Normalmente as pessoas têm dificuldades de avaliar a evolução apenas por meio de documentos de projeto de software.

Exercícios – Unidade 3

1) d

2) a

3) d

4) d

5) b

6) d

7) c

8) a

9) O ASD tem em seu ciclo de vida três fases: especulação, colaboração e aprendizagem. Especulação: o projeto é iniciado e tem-se o planejamento de ciclos adaptáveis, que faz uso das informações contidas no início do projeto como: a missão do cliente, restrições do projeto e os requisitos básicos. Os requisitos básicos são utilizados para a definição do conjunto de ciclos da versão, ou seja, os incrementos do software. Após completar o ciclo o plano é revisto e ajustado; Colaboração: envolve a confiança, críticas sem animosidade, auxílio, trabalho árduo, comunicação de problemas ou preocupações de forma a conduzir ações

efetivas, dentre outras características. É um fator importante no levantamento das necessidades e especificações de requisitos; Aprendizado: considerado um elemento chave para que se possa conseguir uma equipe auto organizada. Os desenvolvedores superestimam o próprio entendimento quanto à tecnologia, processo e até mesmo quanto ao projeto. Portanto, o aprendizado auxilia os desenvolvedores a aumentar os níveis reais de entendimento. As equipes ASD aprendem por meio de três maneiras: grupos focados, revisões técnicas e revisões investigativas (autópsias) de projetos;

10) As principais características de indivíduos que atuam em um processo ágil são: Competência: ter aptidão, conhecimento e capacidade para executar, cumprir e/ou concluir uma determinada tarefa ou função. Foco comum: os indivíduos devem ter foco unânime e um objetivo em comum. Colaboração: é a capacidade executar um trabalho feito em comum com uma ou mais pessoas; também significa cooperação, ajuda e auxílio para concluir determinada tarefa. Habilidade na tomada de decisão: processo pelo qual se escolhe um plano de ação dentre vários disponíveis (baseados em diversos fatores e ambientes) para uma situação específica. Habilidade de solução de problemas confusos: capacidade de lidar com problemas em diversos níveis de complexidade e confusão, propondo soluções adequadas a cada um deles. Confiança mútua e respeito: a confiança mútua e o respeito são características essenciais para fortalecer as relações e tornar os ambientes mais produtivos. Auto-organização: é um processo dinâmico e adaptativo de se organizar para adequar a determinada situação.

Exercícios – Unidade 4

1) a

2) b

3) e

4) a

5) c

6) d

7) d

8) b

9) As pessoas, processos e tecnologias são os pilares da Gestão dos Recursos da Informação e, tais pilares, estão integrados e relacionados e, juntos, são recursos importantes para o planejamento estratégico de uma organização. Cabe ao gerente de recursos de informação a coordenar e integrar criticamente esses diversos meios.

10) A informação pode ser vista como um recurso, pois: Algo com valor fundamental, como capital, bens, trabalho ou suprimentos; Algo com características mensuráveis e específicas; Um ativo de entrada e saída de custos agregados. Algo que possa ser capitalizável ; Algo que possa ter seus custo padronizado e contabilizado. Algo que forneça subsídios estratégicos diversos para a tomada de decisões.

Exercícios – Unidade 5

1) d

2) a

3) d

4) c

5) b

6) a

7) d

8) c

9) Vantagens: A principal vantagem é que os servidores podem ser distribuídos por meio de uma rede. A funcionalidade geral (por exemplo, um serviço de impressão) pode estar disponível para todos os clientes e não precisa ser implementada por todos os serviços. Desvantagens: Cada serviço é um ponto único de falha suscetível a ataques de negação de serviço ou de falha do servidor. O desempenho, bem como o sistema, pode ser imprevisível, pois depende da rede. Pode haver problemas de gerenciamento se os servidores forem propriedade de diferentes organizações.

10) O principal objetivo do padrão MVC é separar dados ou lógica de negócios (MODEL) da interface do usuário (VIEW) e do fluxo da aplicação (CONTROL). Essa ideia é o que permite que uma mesma lógica de negócio possa ser acessada e

visualizada por interfaces distintas. São vantagens do uso do padrão MVC: Possibilita a existência de várias interfaces com o usuário que podem ser modificadas sem que haja alteração das regras de negócio (flexibilidade e reuso). Torna a aplicação escalável. Desvantagens do modelo MVC: Requer uma quantidade maior de tempo para analisar e modelar o sistema. Não é aconselhável para pequenas aplicações. É necessário conhecer o padrão para desenvolver.

Exercícios – Unidade 6

1) a

2) c

3) c

4) a

5) d

6) e

7) d

8) a

9) Sommerville (2011) como as principais dificuldades de se levantar requisitos como: As pessoas podem fazer exigências inviáveis ou desnecessárias, por muitas vezes não saberem articular o que querem de um sistema computacional. Analistas de requisitos podem ter dificuldades de entender as regras de negócio expressadas por pessoas com conhecimento implícito do seu próprio trabalho. Quando se tem muitas pessoas interessadas, tais pessoas podem expressar requisitos diferentes ou os mesmos de várias maneiras. O analista precisa identificar as semelhanças e conflitos. Com o passar do tempo do projeto novos requisitos podem surgir com alterações no modelo de negócio, ou até mesmo com alterações (inclusões ou exclusões) de pessoas interessadas. Pessoas diferentes podem ter visões diferentes de necessidades e funcionalidades.

10) Etnografia é uma técnica de observação que pode ser usada para compreender os processo s operacionais e auxiliar na extração dos requisitos de apoio a esses processos. Tal método é utilizado pela antropologia na coleta de

dados e se baseia no contato entre o antropólogo e seu objeto de estudo, geralmente um grupo social constituído formalmente. Na engenharia de software a etnografia é caracterizada como uma técnica de observação utilizada para mapear requisitos implícitos que refletem processos reais dentro de um ambiente sistêmico. Compreender requisitos sociais e organizacionais, promover um entendimento dos aspectos culturais que regem o ambiente sistêmico direcionam os procedimentos etnográficos. De acordo com Sommerville (2011) a Etnografia é eficaz par descobrir dois tipos de requisitos: Requisitos derivados da maneira como as pessoas realmente trabalham, e não da forma como as definições dos processos dizem que deveriam trabalhar. Requisitos derivados da cooperação e conhecimento das atividades de outras pessoas.