UNIP

UNIVERSIDADE PAULISTA

Engenharia de Software

Autor: Prof. Edson Quedas Moreno
Colaboradoras: Profa. Vanessa Lessa
Profa. Larissa Rodrigues Damiani

Professor conteudista: Edson Quedas Moreno

Mestre em Engenharia de Produção (2002) pela Universidade Paulista (UNIP), com ênfase em Sistemas de Informação, tem pós-graduação em Novas Tecnologias de Ensino-Aprendizagem (2011) pela Unítalo e Graduação em Engenharia Eletrônica (1983) pela Faculdade de Engenharia Industrial (FEI), com ênfase em Computação. Exerce há 40 anos atividades na área da educação e do ensino. Atuou por mais de 25 anos na área corporativa. Em aspectos comunitários: de 2005 a 2006 foi gerente do Departamento de Extensões de Assuntos Comunitários pela Unítalo (DEAC), em 2006 projetou e efetivou o Projeto Farol Verde com apoio do Centro Universitário Ítalo Brasileiro e Prefeitura de São Paulo em parceria com o 12º BPM/M SP, e em 2015, projetou e efetivou um Hackathon (maratona de programação de software por 30h contínuas com a participação de 200 alunos), promovido pela Secretaria Municipal de Promoção da Igualdade Racial da Prefeitura de São Paulo (SMPIR), em conjunto com o Banco Interamericano de Desenvolvimento (BID), o que rendeu em 2016 a comenda "Prêmio Excelência e Qualidade Brasil 2016" pela Braslider.

Dados Internacionais de Catalogação na Publicação (CIP)

M843e Moreno, Edson Quedas.

Engenharia de Software / Edson Quedas Moreno. – São Paulo: Editora Sol, 2024.

136 p. : il.

Nota: este volume está publicado nos Cadernos de Estudos e Pesquisas da UNIP, Série Didática, ISSN 1517-9230.

1. Software. 2. NBR. 3. Humano-computador. I. Título

681.3

U519.70 – 24

Sumário

Engenharia de Software

Unidade II

Unidade III

APRESENTAÇÃO

O computador surgiu durante a Segunda Guerra Mundial e se tornou público em 1947. Os primeiros computadores eram enormes, ocupavam uma sala inteira e tinham capacidade bem reduzida em comparação aos computadores atuais. A partir daí, o mundo testemunhou uma crescente dependência do software e sua adoção em diversas áreas do conhecimento. A demanda por software cresceu exponencialmente, levando ao surgimento de novos produtos e soluções nesse campo. Tem-se nesse momento o início da era da computação, com a dualidade entre o software e o hardware alavancando a criação de novas tecnologias, que se estende até os dias atuais.

No entanto, apesar desse avanço contínuo, a produção de software ainda não conseguiu acompanhar plenamente a demanda dos consumidores, o que levou à busca por novas abordagens e metodologias de produção. Com o computador, o ser humano automatizou o conhecimento. Ademais, a humanidade prosperou muito nesses últimos 80 anos. O software foi o responsável por essa transformação positiva.

Desde os anos 1970, assistimos ao surgimento dos primeiros computadores pessoais, como o Altair 8800 e o Apple I. Esses sistemas eram voltados para uso doméstico e pequenos negócios. A invenção do microprocessador, como o Intel 4004 em 1971, tornou os computadores menores e mais acessíveis, levando ao "boom" da computação pessoal nos anos 1980.

No início dos anos 1990 surgiu a internet e, em consequência, o e-mail, a agenda de grupo on-line e a informação em alto estilo com a *world wide web* (www), componentes de um grande marco e um dos avanços mais significativos da história da humanidade. Vários outros sistemas de software foram criados e a tecnologia da informação ganhou um novo impulso.

Na passagem para os anos 2000, a questão passou a ser a "velocidade". Os empresários precisavam ter rapidez em estratégias, processos, transações comerciais, em questões logísticas e no acesso às informações. A tecnologia e os sistemas produtivos melhoraram, e começaram a ser usadas ferramentas digitais para controlar atividades básicas de produção e gestão dos negócios. A tecnologia da informação proporciona alta velocidade do conhecimento para competir no mundo dos negócios. É o começo de um novo ciclo na história da humanidade, a era da informação e do conhecimento.

A engenharia de software neste novo ciclo fornece às empresas mecanismos de acesso instantâneo às informações, velocidade para lançar produtos antes da concorrência, estatísticas atualizadas em tempo real para cada produto em qualquer lugar do mundo e diversos outros meios facilitadores que possibilitam ao gestor ou ao operacionalizador se manterem na vanguarda dos negócios. A indústria se automatiza, eletroeletrônicos se tornam digitais, e a informação liga o planeta e as pessoas.

Acompanhando toda essa evolução, a engenharia de software viu uma explosão de ferramentas e técnicas que revolucionaram a forma de desenvolver software. Elas ajudaram a melhorar a produtividade, a qualidade e a eficiência na construção de software, alguns exemplos são a engenharia de requisitos de software para a concepção do produto, a gestão do projeto com PMBOK (*Project Management Body of Knowledge*), as IDEs (*Integrated Development Environment*) modernas – que oferecem recursos avançados de edição de código, depuração e integração, facilitando o desenvolvimento e

a colaboração entre desenvolvedores. Para acompanhar a produção de software, são abordados os controles de versionamento e os meios que permitem que desenvolvedores rastreiem e colaborem no desenvolvimento de software, com referência a diversos projetos e arquiteturas de sistemas de software, apoiadas em métodos organizacionais de alta produtividade, tais como as metodologias ágeis, técnicas de customização, componentização e os principais modelos e padrões de qualidade do software, tais como a NBR ISO/IEC 12207, SPICE, CMMI, NBR ISO/IEC 9000, e a integração com as NBR ISO/IEC 9126, 14598 e 25000.

A área de conhecimento da engenharia de software é muito extensa, e este livro-texto em questão traz o conteúdo necessário para ingressar nessa área desafiadora, criativa e de conquistas pessoais e da equipe. À medida que navegamos pelas complexidades do mundo digital, esse livro-texto vai desvendar alguns mistérios da inovação tecnológica.

Tenha bons estudos!

INTRODUÇÃO

A engenharia de software é uma disciplina que surgiu como resposta à crescente demanda por sistemas de software cada vez mais complexos e eficientes. No início da era da computação, a criação de software era frequentemente uma tarefa artesanal, realizada por indivíduos sem muita formalização ou padronização. No entanto, à medida que a tecnologia evoluiu e a dependência de sistemas de software se tornou ubíqua em praticamente todas as esferas da vida, ficou evidente que abordagens mais sistemáticas e organizadas eram necessárias para garantir a qualidade e a confiabilidade desses sistemas.

A organização da disciplina mostra a evolução da construção do produto ao longo do ciclo de vida do desenvolvimento de software. A disciplina apresenta conceitos no seu contexto de aplicação, exemplos reais que contribuem para a compreensão das técnicas de desenvolvimento de software, gerenciamento de projetos em empresas, papéis das equipes de desenvolvimento em cada estágio do processo e mudanças que afetam a prática da engenharia de software.

Por se tratar de uma área de engenharia, a proposta desta disciplina é capacitar o aluno no conhecimento e nas práticas profissionais do projeto e desenvolvimento de software, fornecendo uma base suficiente de aprendizado para iniciar na profissão.

Este livro-texto está dividido em três unidades.

A unidade I conceitua essa engenharia apresentando o produto software, identifica a dualidade entre a engenharia de software e a engenharia de sistemas, os processos de desenvolvimento e sua forma de produção, bem como suas características, aplicações, suporte e manutenção. Nesse momento inicia-se a concepção do software com abordagens sobre a engenharia de requisitos, destacando o estudo para análise dos requisitos: do usuário, funcionais, do sistema e não funcionais. Aborda como a produção de software evoluiu de uma atividade artesanal para um processo altamente estruturado. O desempenho, a escalabilidade, a segurança e a manutenção são preocupações centrais ao criar software, pois a qualidade de um sistema de software não está apenas ligada à sua funcionalidade, mas também à sua capacidade de lidar com demandas e alterações futuras. Em destaque está a explosão da conectividade global provocada por uma forte demanda de sistemas de informação para internet, que abrange desde aplicativos de uso pessoal até sistemas empresariais de larga escala, o que leva a uma especial atenção no desenvolvimento de software para esta área. Apresenta a organização e o processo para desenvolver software com base em modelos e técnicas eficientes. Discorre sobre as formas de customização apoiada por metodologias, tais como a análise de pontos por função e testes e diagnósticos que eliminam ou mitigam defeitos.

A unidade II caracteriza a engenharia de domínio com a criação de um conjunto de artefatos, como modelos, padrões e componentes, que podem ser reutilizados em diferentes projetos de software dentro de um determinado domínio. Nesse contexto é explorada a métrica reusabilidade do software, o principal objetivo da engenharia de software, porque determina a capacidade de utilizar componentes de software existentes em novos contextos, economizando tempo e recursos. Descreve a importância da gestão de projetos e a utilização do Guia PMBOK (*Project Management Body of Knowledge*), editado e publicado pela PMI (*Project Management Institute*), que é um conjunto de melhores práticas amplamente reconhecido

por diversas empresas desenvolvedoras, assumindo o status de principal guia para o gerenciamento de projetos. Em complemento, explora também os conceitos das principais metodologias ágeis para melhoria da produtividade na construção de software. Essas metodologias enfatizam a colaboração, a entrega incremental, a adaptabilidade e a resposta às mudanças dos requisitos do cliente.

A unidade III tem foco na interface homem-computador, bem como nos padrões e modelos de qualidade que acompanham essa disciplina. A Interação Humano-Computador (IHC), também chamada de Interface Humano-Computador, concentra-se no design de interfaces entre seres humanos e sistemas computacionais. Os princípios da IHC incluem a usabilidade, a eficiência, a eficácia, a aprendizagem, a satisfação do usuário e a acessibilidade. Por fim, mostra como o software evolui, como ocorre o processo de melhoria contínua e automatização dos negócios pela reengenharia, como se dá a manutenção do software e as tendências do desenvolvimento de software.

A linguagem utilizada para a elaboração deste livro-texto é estritamente técnica. Os estudos, pesquisas e práticas apresentadas são resultados em campo das atividades de engenharia de software aplicadas no projeto, construção, manutenção e evolução do software.

Este livro-texto não apresenta a engenharia de software em sua completude, mas orienta para isso, com diversas leituras, links de acesso, exemplos e casos aplicados, dicas de ferramentas de desenvolvimento, tecnologia ao nosso alcance e melhor capacitação profissional.

Inicie nesta carreira. É criativa, é de responsabilidade, é organizada, é desafiadora e é gratificante.

Unidade I

1 FUNDAMENTOS DA ENGENHARIA DE SOFTWARE

1.1 Definição de engenharia de software

A engenharia de software projeta e constrói o produto software de computador. Abrange programas que executam em computadores de qualquer tamanho e arquitetura o processamento de dados e informações, que incluem formas impressas e virtuais, combinam caracteres dos mais variados tipos, incluindo representações de informação no formato de imagens, vídeos e áudios.

As tecnologias atuais resultam da convergência de sistemas de computadores, diversas mídias, sistemas de comunicação e sistemas mecatrônicos. O que suscita questões complexas para os engenheiros de software são: hardware diferente, diversos ambientes operacionais, regras do negócio que mudam constantemente, adaptações a novas tecnologias e interfaces, e a ligação de todos estes elementos.

O software é uma tecnologia indispensável para os negócios, entretenimento, ciência e engenharia. O desenvolvimento de um sistema computacional, seja ele voltado para negócios, web, aplicação científica, automação industrial, ou outras aplicações, inicia pelo software, que atende a necessidade (ou resolve um problema) de cálculo, de automação, de manufatura, do negócio empresarial, da engenharia e da ciência. Segundo Roger Pressman, "Software de computadores continua a ser a tecnologia única mais importante no cenário mundial" (2011).

O software é desenvolvido ou passa por um processo de engenharia, não é um produto industrializado. Seu custo está concentrado no trabalho de engenharia, que contempla os seguintes itens:

- Especificação, documentação e procedimentos;
- Análise, projeto, codificação, implementação, testes, diagnósticos e implantação;
- Suporte ao cliente/usuário.

Observação

O software é classificado em duas categorias:

1. Software aplicativo: permite aos usuários realizarem suas tarefas computacionais.
2. Software de sistema: permite aos especialistas de TI gerenciar e desenvolver o sistema de software.

De acordo com Ian Sommerville, "produzir software que apresente uma boa relação custo/benefício é essencial para o funcionamento das economias nacionais e internacionais" (2011).

1.1.1 Relação das engenharias de software e de sistemas

Algumas disciplinas, por terem similaridades com a engenharia de software, se confundem, como é o caso da engenharia de sistemas no aspecto profissional, pois boa parte dos métodos e técnicas são semelhantes. Contudo, são disciplinas distintas. Esse assunto será discutido no próximo tópico.

A engenharia de software usa o conhecimento e resultados de diversas áreas, fornece outros problemas de estudo, bem como auxilia na resolução de problemas. De acordo com Ian Sommerville:

> O mundo moderno não poderia existir sem o software. Infraestruturas e serviços nacionais são controlados por sistemas computacionais, e a maioria dos produtos elétricos inclui um computador e um software que o controla (2011).

Estudo de caso: modelagem de um sistema de bomba de insulina

Do ponto de vista da engenharia de sistemas, a análise da bomba de insulina é apresentada em um diagrama de blocos, como mostrado na figura 1. O modelo apresenta os componentes de hardware e o fluxo de controle para a bomba de insulina.

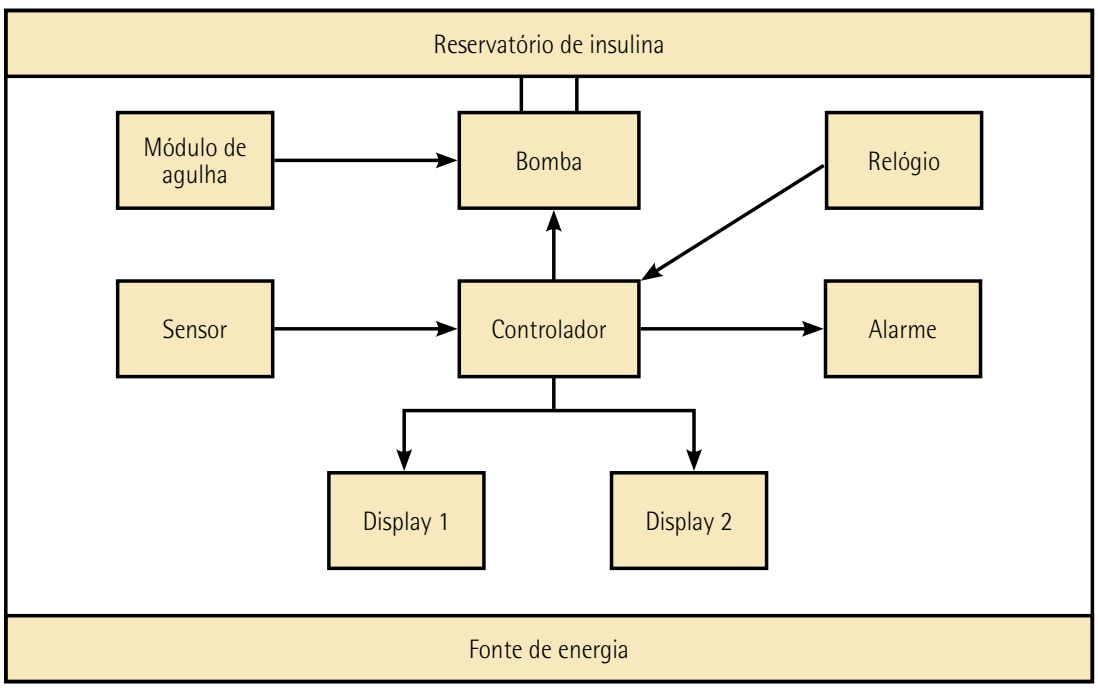

Figura 1 – Hardware de bomba de insulina

Fonte: Sommerville (2011).

Do ponto de vista da engenharia de software, a análise da bomba de insulina é apresentada em um modelo de negócio com um diagrama de atividades, como mostrado na figura 2. O modelo representa algumas funções de interface com o usuário e com o hardware que devem ser apresentadas pelas funcionalidades do software.

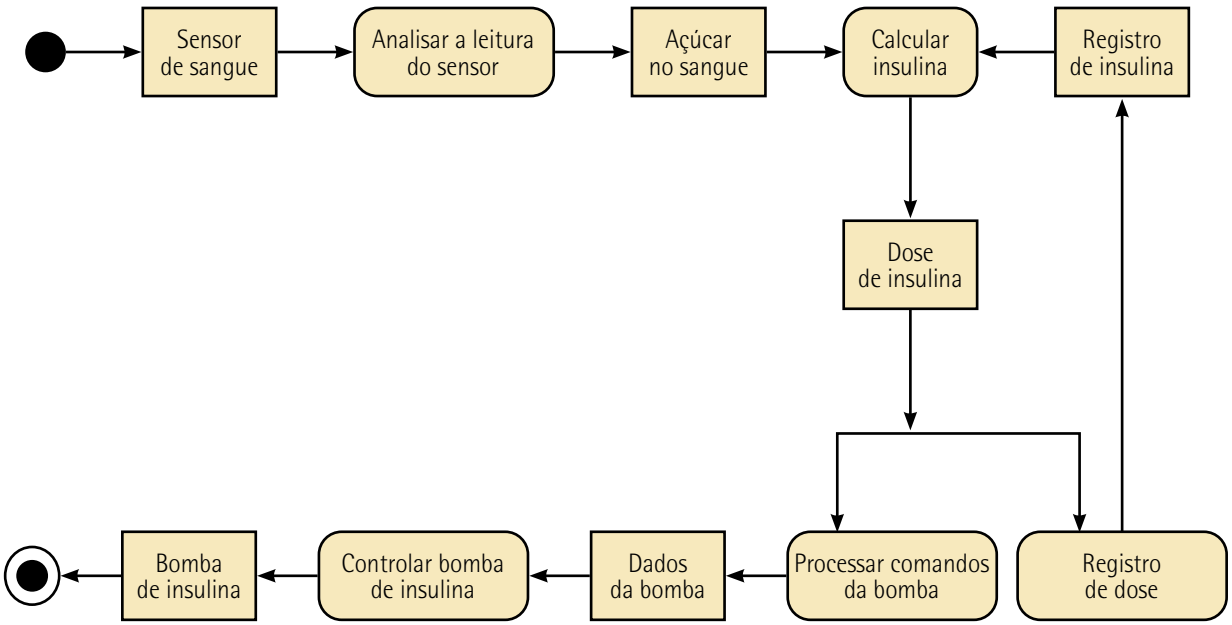

Figura 2 – Modelo de atividade da bomba de insulina

Fonte: Sommerville (2011).

A engenharia de sistemas é uma disciplina presente em muitos projetos de desenvolvimento de software. Ela focaliza diversos elementos e, nesse sentido, analisa, projeta e organiza esses elementos em um sistema para atender um objetivo específico, com a visão voltada para a interface e a ligação entre os elementos.

A engenharia de sistemas trata da integração dos principais elementos que compõem os sistemas computacionais, que são: software, hardware, computador pessoal, base de dados e redes de computadores. A integração desses elementos dará suporte ao software.

A figura 3 mostra a ligação dos principais elementos que compõem o sistema computacional em um diagrama de componentes e implantação e a forma que vão interagir, para que possam funcionar com eficiência quando reunidos em um determinado conjunto.

Nela são vistas as ligações dos elementos do sistema, representados pelos seus componentes, tais como: computadores, protocolos de rede, bibliotecas, linguagem de programação e detalhes de conexões.

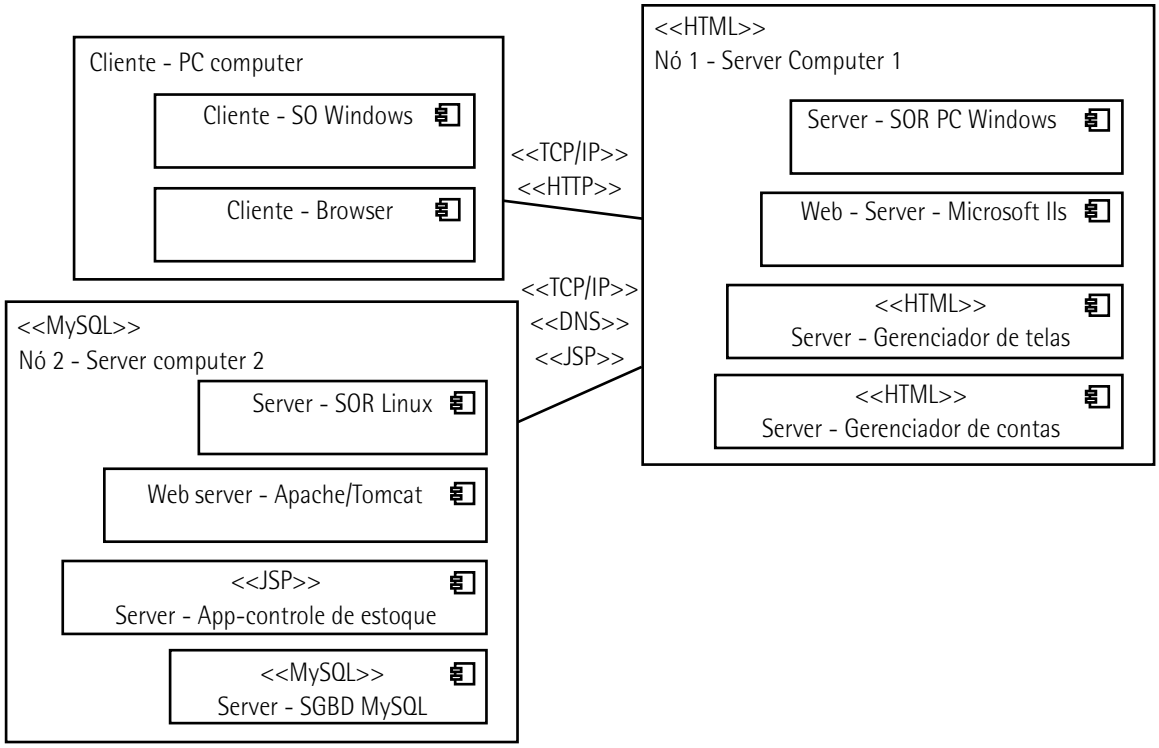

Figura 3 – Infraestrutura projetada pela engenharia de sistemas
para atender à aplicação de controle de estoques

As bases para a engenharia de sistemas são procedimentos e documentos com elementos do sistema que devem ser identificados e requisitos operacionais que devem ser atingidos, analisados, especificados, modelados, validados e gerenciados.

Exemplo de aplicação

Como já foi mencionado, a engenharia de software projeta e constrói software para computador. O exemplo aqui disposto mostra um problema comum de software em que a solução está na engenharia de sistemas.

Situação-problema: a influência da engenharia de sistemas em um projeto de software está, por exemplo, no teste de uma determinada funcionalidade, em que se desenvolve uma funcionalidade cuja particularidade é fazer múltiplos acessos a um banco de dados em uma determinada rede de computadores, porém o desenvolvimento do software é único e faz os testes do código em ambiente restrito. Por isso, os múltiplos acessos, que podem ocorrer também em diversas estações de computadores, podem degradar o desempenho de uso do software.

Solução: a engenharia de sistemas cuidará da implementação da banda de rede e da capacidade do serviço de banco de dados de atender as requisições de múltiplos usuários do software em variados ambientes operacionais.

1.2 Características do software

O software é o arranjo lógico de programas de computador organizados para atingir um objetivo específico do negócio, que é o resultado obtido pela interpretação do negócio por meio do processamento de dados, constituindo o principal elemento do ambiente operacional. A meta da engenharia de software é traduzir o desejo do cliente para um conjunto de capacidades definidas em um produto, que esteja em funcionamento.

O programa de computador é uma forma lógica de organizar intelectualmente instruções baseadas em linguagem de programação. Os programas de computador são responsáveis pelo processamento que transforma dados em informações ou retrabalha informações para gerar novas informações.

A hierarquia de análise do produto software que leva à sua construção é mostrada na figura 4. Os requisitos globais do produto são extraídos das necessidades do cliente e determinados pelo desenvolvedor. No caso, foram destacados os componentes software, hardware, dados e redes.

O objetivo da análise é verificar outros requisitos em cada componente que devem dar suporte ao software. Na figura 4, para cada requisito de processamento do software a ser produzido, impõe-se a necessidade de dados, função e comportamento do produto. Com base na necessidade do software podem ser considerados também outros requisitos de processamento, tais como controle, configuração do software, desempenho global, restrições de projeto e outras necessidades especiais.

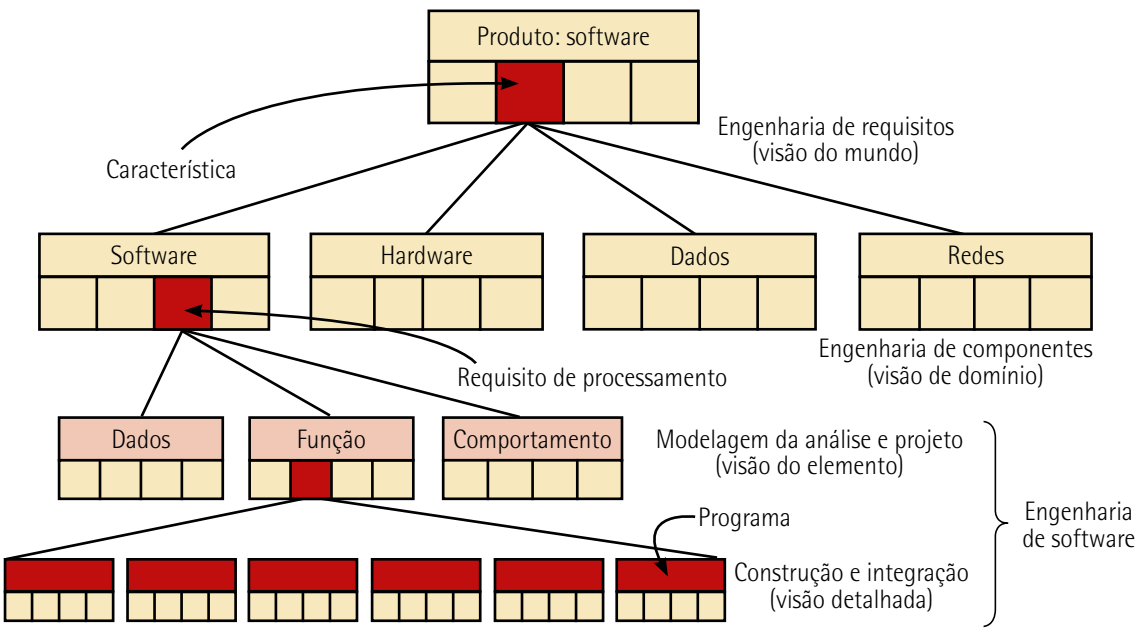

Figura 4 – Hierarquia e análise do produto software pela tecnologia da informação

Adaptada de: Pressman (2002).

1.2.1 Dualidade entre o software e o hardware

A dualidade entre o software e o hardware permite a construção de sistemas computacionais, mas o que realmente justifica a função do computador é o uso do software. Veja bem: não é possível um editor de textos funcionar se ele não for projetado e desenvolvido pela engenharia de software.

O software é desenvolvido ou passa por um processo de engenharia, não é manufaturado. Na manufatura, as mudanças necessárias nos produtos, sejam por necessidade, correções ou melhorias, são feitas por substituições ou manutenção de peças. Os problemas de qualidade ou manutenção do hardware podem ser corrigidos pela manufatura, o que não ocorre com o software.

O software não "se desgasta", mas "se deteriora"! O software não é suscetível aos males ambientais, que causam desgaste do hardware, mas sim devido às mudanças que ocorrem no ciclo de vida do software (Pressman, 2011).

Durante o ciclo de vida, o software passará por modificações (manutenção e/ou mudanças). Essas modificações causam a deterioração do software e, à medida que são feitas, é provável que novos defeitos sejam introduzidos, causando dente na curva de taxa de falhas, como mostrado no gráfico da direita da figura 5.

O gráfico da esquerda se refere à curva de falhas do hardware, também chamado na engenharia de "curva da bacia". Este gráfico mostra que o índice de falhas de um produto ao ser lançado no mercado ou adaptado a outros produtos (faixa da mortalidade infantil) é alto. No decorrer do tempo, estas falhas são reduzidas por processos revisionais e de manutenção, até chegarem ao nível de estabilidade e com baixo índice de falhas. A vida útil é a soma entre os tempos da região da mortalidade infantil e da região de estabilidade. Como o hardware fica sujeito ao desgaste, isso provoca o aumento do índice de falhas ao longo do tempo e, consequentemente, o aumento do custo de manutenção. As manutenções sucessivas crescem e o custo para manter o produto acaba por inviabilizá-lo.

O gráfico da direita se refere à curva de falhas do software, que, diferentemente do hardware, após a faixa de estabilidade começam a ocorrer mudanças no software, provocadas a pedido do cliente ou algum outro fator, como mudança no ambiente operacional (hardware, capacidade limitada do sistema, atualização do sistema operacional ou a inclusão/exclusão de outros drivers ou software). A cada mudança implementada, surge um "pico" de falhas, que leva novamente a processos revisionais e de manutenção. Nestas mudanças existe a possibilidade de serem acrescentados novos códigos e criadas redundâncias de programas e de dados.

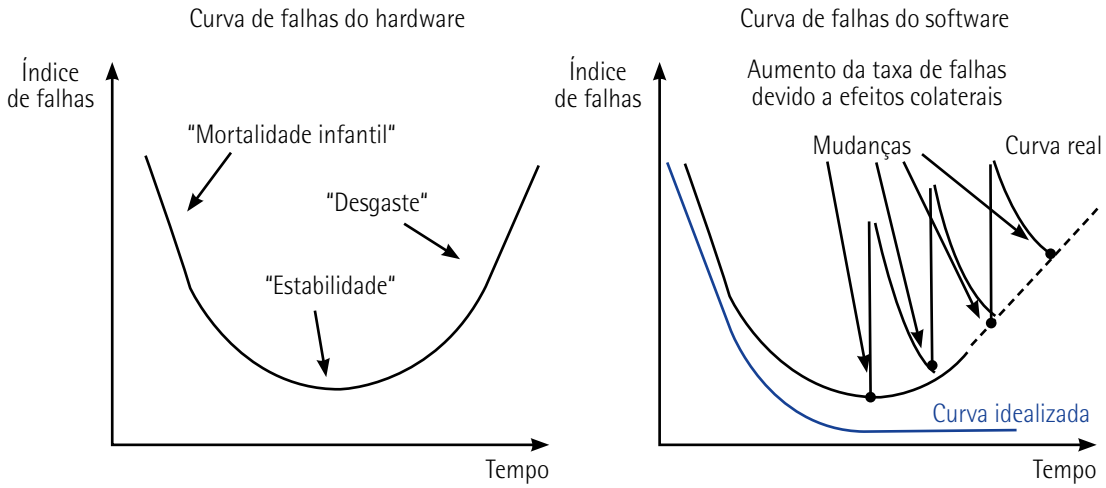

Figura 5 – Gráficos comparativos da curva de falhas do hardware em relação a curvas de falhas do software

Adaptada de: Pressman (2011).

Devido às mudanças causadas no software, são registradas versões e releases no acompanhamento de sua evolução. As versões são registros feitos sobre as mudanças que ocorrem na adaptação do software em relação às necessidades do cliente, adaptação a novos ambientes operacionais, correções de falhas ou quando o software está em desenvolvimento. O release (lançamento) é o registro da versão liberada para o usuário.

Após várias mudanças causadas no software, ele deve ser reestruturado.

Reestruturar o software significa:

- fazer limpeza dos dados;
- fazer limpeza dos códigos redundantes;
- atualizar hardware;
- atualizar o sistema operacional e as linguagens de programação com novas versões;
- gerar novos algoritmos;
- adaptar de forma correta as antigas e novas funcionalidades com base em uma nova arquitetura.

Observação

Na deterioração do software os usuários reclamam muito, por exemplo: "Isto está lento".; "Sumiu meu registro"; "Tinha um relatório aqui"; "Este campo está vazio, deveria ter a informação...".

Nesta situação, o volume de chamadas aos desenvolvedores e equipes de manutenção aumenta. Para corrigir de vez este problema e retomar um ciclo novo de vida do software, é necessário reestruturá-lo.

1.2.2 Crise do software

Em 2002, Pressman escreve: "Software: uma crise no horizonte", definindo crise como "um ponto decisivo no curso de algo". O termo se refere a um conjunto de problemas encontrados no desenvolvimento de software de computador.

Por haver um rápido crescimento da demanda por software, imaginava-se que haveria uma forte crise com toda a complexidade no seu desenvolvimento. Com a inexistência da engenharia de software nessa época, não havia técnicas estabelecidas para o desenvolvimento de sistemas que funcionassem adequadamente ou que pudessem ser validadas. Estes problemas considerados "a crise do software" ainda são os mesmos da atualidade.

Gerentes responsáveis pelo desenvolvimento de software concentram-se nas questões problemáticas de "primeiro plano":

- As estimativas de prazo e de custo frequentemente são imprecisas.
- A produtividade das pessoas da área de software não tem acompanhado a demanda por seus serviços.
- A qualidade de software às vezes não é adequada.

Observação

As questões problemáticas são a manifestação mais visível de outras dificuldades do software:

- Não é dedicado tempo para coletar dados sobre o processo de desenvolvimento de software. Sem qualquer indicação sólida de produtividade, não podemos avaliar com precisão a eficácia de novas ferramentas, métodos ou padrões.
- A insatisfação do cliente com o sistema "concluído" ocorre muito frequentemente. A comunicação entre o cliente e o desenvolvedor de software costuma ser muito fraca.
- A qualidade do software geralmente é suspeita. Somente agora começam a surgir conceitos quantitativos sólidos de confiabilidade e garantia de qualidade de software.

1.3 Processos de software

A estrutura organizacional orientada por um modelo de processo é a composição do ciclo de vida do desenvolvimento do software em uma coleção ordenada de conjuntos de atividades, métodos, práticas e transformações empregadas no seu desenvolvimento e manutenção. O modelo de processo de software fornece estabilidade, controle e organização para as atividades que, se deixadas sem controle, tornam-se caóticas.

O processo é um diálogo no qual o conhecimento que deve se transformar no software é reunido e embutido nele (Pressman, 2002).

Um modelo de processo de software é uma representação simplificada de um processo. Cada modelo representa uma perspectiva particular de um processo e, portanto, fornece informações parciais sobre ele (Sommerville, 2011).

Logo, um modelo de software deve ser aplicado em partes específicas do processo que atendam toda a organização. Em uma visão global, tanto um sistema como um processo são orientados por uma estrutura organizacional.

Os modelos de processo de software mais conhecidos são:

- **Modelos de processos tradicionais**: Cascata, Balbúrdia, Prototipagem, Incremental, RAD e Espiral.
- **Processo unificado**: RUP e Praxis.
- **Modelos de processos pessoal e de equipe**: PSP e TSP.

1.3.1 Modularidade: coesão e acoplamento

A modularidade consiste em dividir o sistema ou software em componentes, ou módulos, que trabalham em conjunto para desempenhar uma determinada atividade e atingir um objetivo.

Tanto o módulo como o componente do software são blocos isolados, que independem de outras partes do sistema, com endereçamento próprio e que permitem manutenções isoladas sem que estas afetem outras partes do sistema.

A diferença básica entre um componente e um módulo está associada a seu tamanho e complexidade. Veja bem: pode haver um componente de software que faça um cálculo de folha de pagamento e outro que gere relatórios. Contudo, os dois componentes podem ser integrados em um único módulo que faça estas duas operações.

Observe o modelo de infraestrutura de um sistema na figura 6. Os módulos A, B e C (componentes) têm uma alta dependência do módulo M (implantação), ou seja, têm um alto índice de acoplamento. O que não é bom, porque se o módulo M falhar, todos os outros módulos ficam comprometidos. A vantagem é a simplicidade e a consistência no tratamento dos dados, e como desvantagem há o risco de todo o sistema falhar.

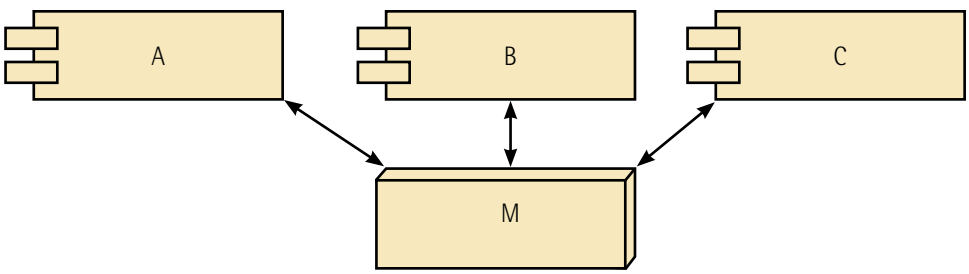

Figura 6 – Modelo de infraestrutura de um sistema com alto índice de acoplamento

Contudo, este outro modelo de infraestrutura da figura 7 apresenta baixo índice de acoplamento e alta coesão. Mesmo que um dos módulos apresente falha, os dois outros estarão coesos. Ele tem como vantagem a redução do risco de perda de dados, e como desvantagem há a complexidade no tratamento de dados.

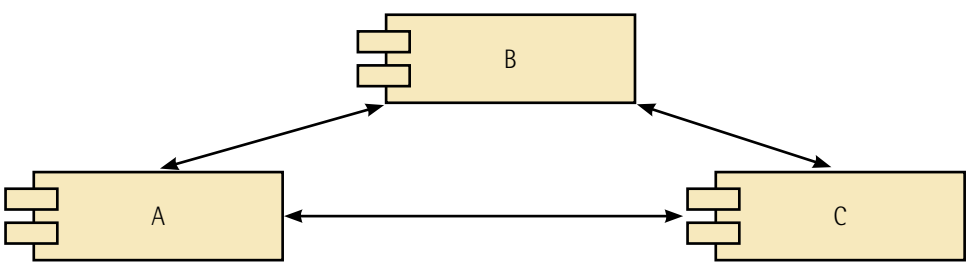

Figura 7 – Modelo de infraestrutura de um sistema com alto índice de coesão

1.4 Engenharia de requisitos

1.4.1 Conceito de processo da engenharia de requisitos

Entender os requisitos de um problema está entre as tarefas mais difíceis enfrentadas por um engenheiro de software. Quando começamos a pensar sobre isso, a engenharia de requisitos não precisaria ser tão difícil. Afinal de contas, o cliente não sabe o que é necessário? Os usuários finais não deveriam ter um bom entendimento das características e funções que vão oferecer benefícios? Mesmo que clientes e usuários finais sejam explícitos quanto às necessidades, elas vão mudar ao longo do projeto (Pressman, 2007).

As principais atividades do processo da engenharia de requisitos do software/sistema são:

- estudo da viabilidade do sistema;
- elicitação;
- análise;
- especificação;
- modelagem;
- validação.

A figura 8 mostra o fluxo de atividades do processo da engenharia de requisitos do software/sistema. Observe que o desenvolvimento só ocorre após a validação dos requisitos, e, caso os requisitos não sejam validados, ocorre uma iteração, o que significa que todo o processo será revisado, incluindo novas abordagens para o levantamento dos requisitos.

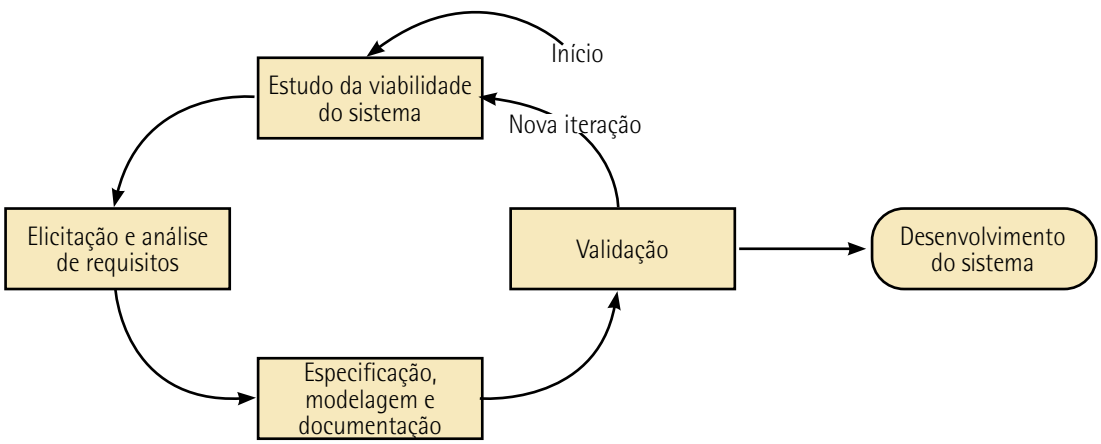

Figura 8 – Modelo do processo da engenharia de requisitos do software/sistema

1.4.2 Estudo da viabilidade do sistema

Identificação (concepção): a maioria dos projetos começa quando uma necessidade de negócio é identificada ou um mercado ou serviço novo é descoberto. Os desenvolvedores fazem uma série de perguntas com a intenção de estabelecer um entendimento básico do problema. Deve haver uma colaboração entre cliente e desenvolvedor.

Para todos os sistemas novos, o processo de engenharia de requisitos de sistema deve começar com o estudo de viabilidade, antes mesmo da obtenção e análise de requisitos. Os resultados deste estudo devem ser um relatório que recomenda se vale a pena ou não realizar o processo de engenharia de requisitos e o processo de desenvolvimento do sistema.

É um estudo breve, direcionado, que se destina a responder algumas perguntas:

1. O sistema proposto contribui para os objetivos gerais da organização?

2. O sistema pode ser integrado com os outros sistemas já em operação? Qual é o impacto das mudanças no ambiente operacional do cliente?

3. O que o cliente quer está dentro do orçamento e prazo para entrega do sistema?

4. O sistema pode ser implementado com a utilização de tecnologia atual dentro das restrições de custo e prazo?

1.4.3 Processos de requisitos do software

A elicitação dos requisitos é a tarefa de comunicar-se com os usuários e clientes para determinar quais são os requisitos. Nela, o analista deve ter habilidade e sutileza para extrair informações durante uma conversa. Os analistas podem empregar várias técnicas para elicitar os requisitos dos clientes.

As mais utilizadas são:

- Organizar reuniões, entrevistas ou grupos focais (workshops) e, então, a criação da lista de requisitos.

- Prototipação e casos de uso: o protótipo é recomendado para avaliar a interface do usuário, os problemas de comunicação com outros produtos e a possibilidade de atendimento aos requisitos de desempenho. Os casos de uso são úteis para identificar funcionalidades do software.

- Modelagem do Processo de Negócio (Business Process Management – BPM): semelhante ao diagrama de atividades da UML (Unified Modeling Language), é uma ferramenta útil para o cliente e os analistas na interpretação de como o negócio funciona.

O resultado final do trabalho da engenharia de requisitos é um documento com especificação e modelagem dos produtos que serão produzidos.

Em sistemas de software, o termo especificação serve como fundamento para as atividades de engenharia de software subsequentes. A partir dela, são descritas a função e o desempenho do sistema e as restrições que acompanharam seu desenvolvimento. A modelagem parte de um modelo gráfico, um modelo matemático formal, como casos de uso, protótipos, diagramas ou qualquer combinação desses elementos.

Os stakeholders por vezes interpretam os requisitos de maneiras diferentes. Os modelos auxiliam, criando a interpretação das várias ideias em uma linguagem comum (simbólica).

Os quatro principais grupos de requisitos para elaboração do contrato do software/sistema são:

- requisitos do usuário;
- requisitos do sistema;
- requisitos funcionais;
- requisitos não funcionais.

Os requisitos do usuário (RU) são declarações em linguagem natural, formulários e diagramas simples sobre as funções que o sistema/software deve fornecer e as restrições sobre as quais deve operar. Este documento pode ser elaborado pelo representante do usuário. Nesta fase, modelos de negócios e casos de uso são produzidos.

Os requisitos não funcionais (RNF) determinam a qualidade do software, que é o diferencial para uma aplicação ser boa. Uma forma de memorizar o que são requisitos não funcionais é saber que requisito não funcional é tudo aquilo que o cliente não pede. Mas se der problema, ele vai reclamar.

Exemplo de aplicação

Estudo de caso: situação de falha no envio de formulários

Situação problema: um determinado usuário levou uma hora preenchendo um formulário na rede local de um sistema servidor/cliente. Ao finalizar e salvar o formulário, o software emite um aviso de tela inteira: "sistema fora do ar".

Análise: se o software tiver o recurso de recuperação do arquivo, mesmo que seja no computador local, o usuário vai encarar isso de forma mais amena. O caso é diferente se ele perder todo o formulário já preenchido! Ele vai reclamar de um problema de confiabilidade.

Na figura 9 vemos os principais tipos de requisitos não funcionais (Sommerville 2003), formados pelos grupos: requisitos do produto, requisitos organizacionais e requisitos externos.

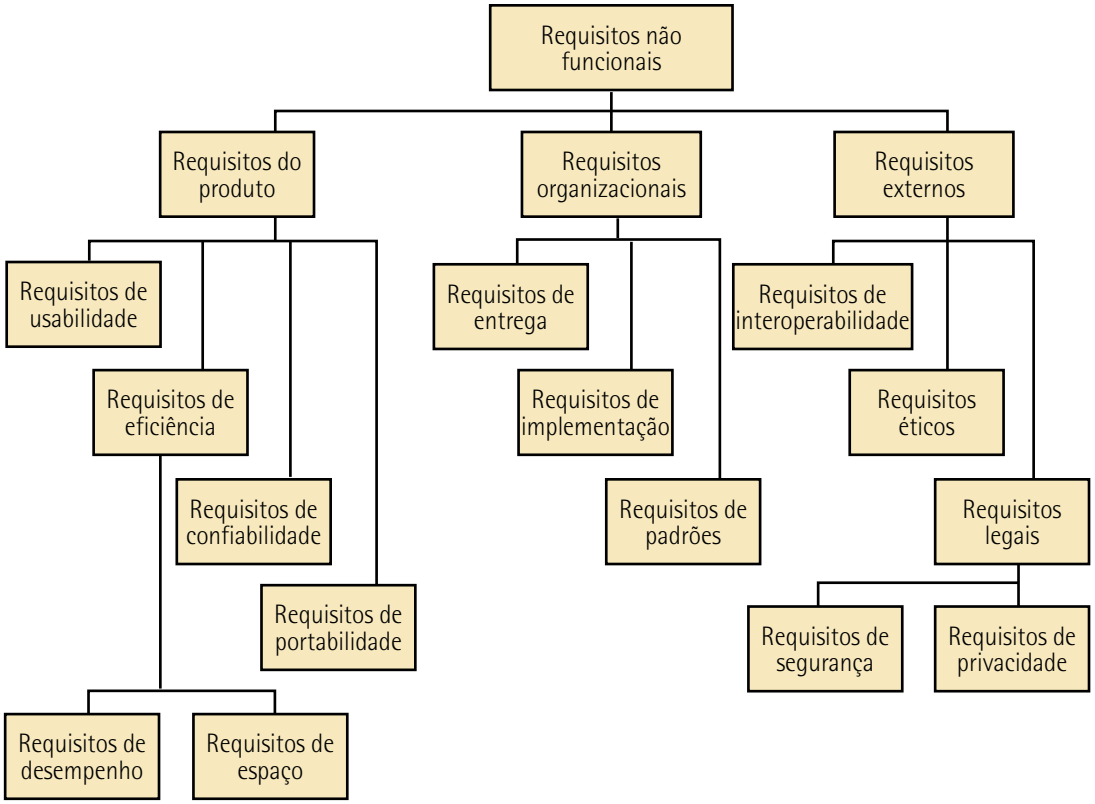

Figura 9 – Tipos de requisitos não funcionais (RNF)

Fonte: Sommerville (2003).

Acompanhe os conceitos a seguir com o observado na figura 9:

- **Grupo de requisitos do produto**: este grupo diz respeito ao atendimento direto das necessidades e expectativas dos clientes ou usuários. É composto pelos requisitos usabilidade, que se refere à interface do usuário; eficiência, que diz respeito a fazer correto e da melhor forma possível, considerando o tempo e o valor econômico; confiabilidade, que se trata de atender à expectativa de apresentar um determinado resultado de acordo com o esperado no requisito; e portabilidade, que é a capacidade de escalar o software em outros ambientes operacionais com base no desempenho e espaço de armazenamento.

- **Grupo de requisitos organizacionais**: se refere às necessidades, restrições, padrões e normas estabelecidas pela organização responsável pelo desenvolvimento do produto software. Nesse contexto, envolve a disponibilidade dos recursos necessários para a Entrega e Implementação do software, bem como os procedimentos para atender aos padrões organizacionais.

- **Grupo de requisitos externos**: diz respeito ao atendimento de regulamentações governamentais, normas da indústria de software, atendimento às expectativas dos stakeholders, competitividade e outros. Como principais destaques estão a interoperabilidade, que é a capacidade de efetuar operações em ambientes distribuídos; a ética, que pode ser relacionada ao cumprimento do

lançamento de um software na data prevista para o consumidor; e aspectos legais, que envolvem a segurança e privacidade, principalmente dispostos na LGPD (Lei Geral de Proteção de Dados).

Os requisitos funcionais (RF) são declarações mais detalhadas dos requisitos do usuário com uma especificação completa e consistente de toda a funcionalidade ou serviços que se espera que o sistema forneça. A atividade de elaboração dos requisitos funcionais consiste em extrair dos casos de uso as funções necessárias que vão operar no sistema. O objetivo é preparar um material detalhado para que possa ser utilizado na codificação.

É aconselhável que, para maior clareza dos objetivos das funções, seja feita uma tabela com a identificação e especificação das funções do software, como mostrado no quadro 1, com um código específico do requisito:

Quadro 1 – Especificação de Requisitos Funcionais (RF)

Requisito Funcional (RF)	Especificação
RF01	Chamada de menu: Relatórios – deverá exibir o menu de relatórios disponíveis
RF01.1	Função: Relatório de compras – relatório com as compras efetuadas no período desejado. Exibindo: Fornecedor, produto comprado, quantidade e valor
RF01.2	Função: Relatório de pagamentos efetuados – relatório com os pagamentos efetuados a fornecedores no período desejado. Exibindo: Fornecedor, produto comprado, quantidade e valor pago

Os requisitos do sistema (RS) são descrições mais detalhadas dos requisitos do usuário com o foco no sistema, com uma especificação completa e consistente de todo os componentes do sistema. Podem ser apresentados vários modelos que mostram objetos ou fluxo de dados. Servem como base para o contrato destinado à implementação do sistema.

Para a modelagem dos requisitos do sistema são utilizados basicamente os diagramas de componentes e o diagrama de implantação da UML.

Os componentes podem ser divididos em dois grupos:

- A visão estática da arquitetura promove a visão da organização e relações dos componentes do software com elementos de dados (banco de dados, arquivos-texto etc.) com o hardware e outros ambientes operacionais.

- A visão dinâmica da arquitetura promove a visão comportamental do sistema e de seus componentes. São definidos como os componentes reagem a eventos internos e externos e a forma como eles se comunicam (ou trocam mensagens).

Seguindo o mesmo padrão de documentação aplicada para requisitos funcionais, nos requisitos do sistema também é montada uma tabela com a identificação e especificação dos requisitos do sistema, conforme mostra o quadro 2, com um código para cada requisito:

Quadro 2 – Especificação de requisitos do sistema

Requisito do Sistema (RS)	Especificação
RS01	Computador de cliente, modelo PC com médio desempenho
RS02	Sistema operacional do computador de cliente – Windows. Ver RS01
RS03	Navegador para internet. A aplicação irá funcionar em uma rede intranet

A validação dos requisitos ocorre formalmente. Os produtos de trabalho resultantes da engenharia de requisitos são avaliados e aprovados. A atividade de validação é a última fase do processo da engenharia de requisitos, responsável por autorizar o desenvolvimento do sistema/software. O objetivo é aprimorar, incluir mudanças propostas e aprovar (aceite do cliente) o início do projeto e desenvolvimento do sistema de software.

1.4.4 Ergonomia cognitiva

A ergonomia cognitiva é uma área da ergonomia que se concentra no estudo e na melhoria da interação entre os seres humanos e os sistemas cognitivos, como a mente, a memória, a atenção e a percepção.

Um aspecto importante da ergonomia cognitiva é o design de interfaces de usuário intuitivas e fáceis de usar. Isso envolve a consideração de como as informações são apresentadas, como os comandos são solicitados e como os usuários interagem com os sistemas. Interfaces bem projetadas podem reduzir a carga cognitiva, facilitar a compreensão e minimizar erros e frustrações.

Uma das teorias mais conhecidas de design centrado no usuário é a engenharia cognitiva (Norman, 1986 – *apud* Souza *et al.*, 2000). Norman considera que o designer inicialmente cria o seu modelo mental do sistema, chamado modelo de design, com base nos modelos de usuário e tarefa. O modelo de imagem do sistema a ser implementado é construído a partir do projeto do modelo de design.

O usuário então interage com esta imagem do sistema e cria seu modelo mental da aplicação, chamado de modelo do usuário. Este modelo mental é o que permite ao usuário formular suas intenções e objetivos em termos de comandos e funções do sistema. A figura 10 mostra o processo de design na abordagem da engenharia cognitiva.

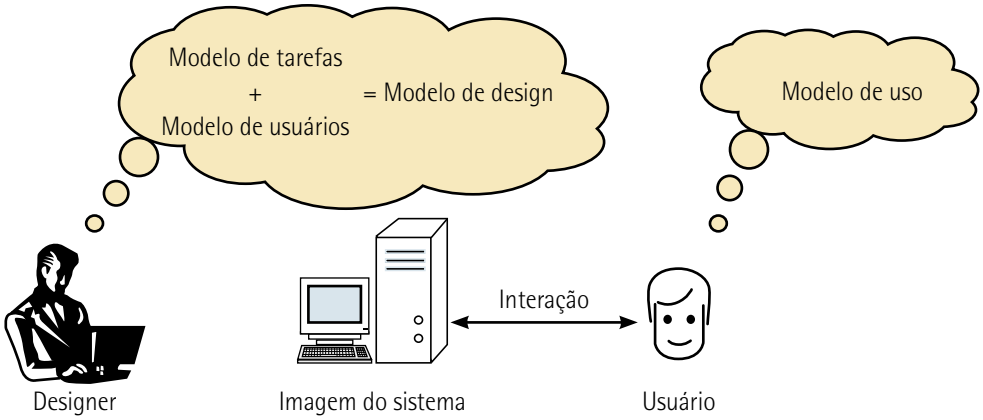

Figura 10 – Processo de design: modelo de interação da engenharia cognitiva

Fonte: Souza (2000).

Outro aspecto a considerar na ergonomia cognitiva são as características ditadas na norma ISO 9241 (1998), que se referem aos "Requisitos Ergonômicos para Trabalho de Escritórios com Computadores".

O tópico 11 dos da norma ISO 9241-11 aborda os benefícios de medir a usabilidade em termos de desempenho e satisfação do usuário. Eles são medidos pela extensão na qual os objetivos de uso pretendidos são alcançados, pelos recursos gastos e pela aceitação do usuário no uso do produto.

Na figura 11 os componentes da NBR 9241-11 e a forma que se relacionam ilustram o design do produto software, o resultado pretendido e o resultado de uso para atingir os objetivos de usabilidade.

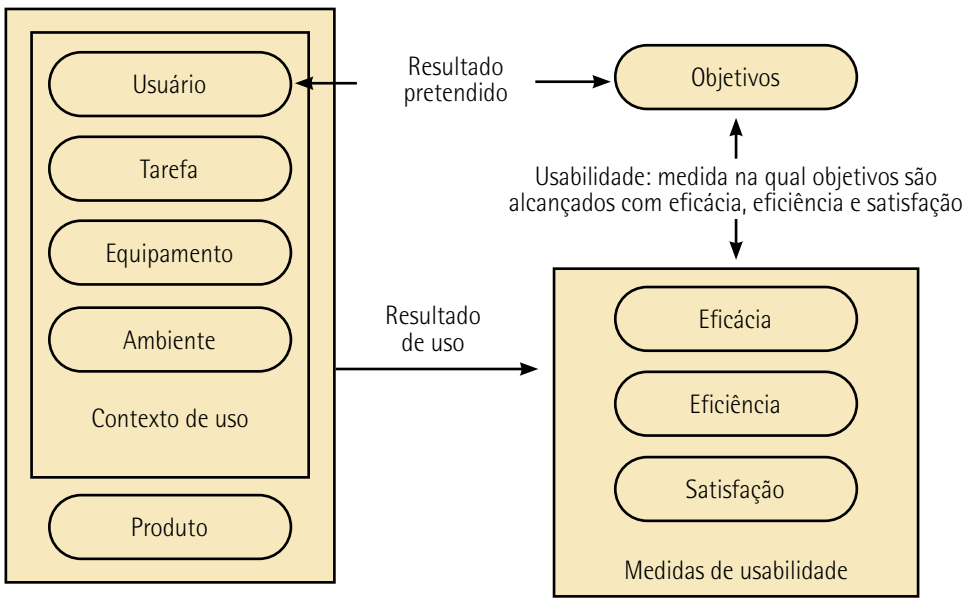

Figura 11 – Estrutura da ISO 9241-11 (2002:4)

Fonte: ABNT (2002).

2 CICLO DE VIDA DO SOFTWARE

2.1 NBR ISO/IEC 12207: Processos do ciclo de vida do software

2.1.1 Software Development Life Cycle (SDLC)

Ao elaborar um sistema é importante percorrer uma série de passos previsíveis, conhecidos como o Ciclo de Vida do Desenvolvimento do Software (Software Development Life Cycle – SDLC) (Stair, 2006).

O ciclo de vida do desenvolvimento do software é um roteiro que ajuda a criar a um resultado de alta qualidade. Os modelos de processos de software englobam um conjunto de atividades, métodos, práticas e transformações a serem empregadas no desenvolvimento e manutenção do software. Fornecem estabilidade, controle e organização para as atividades.

O ciclo de vida do desenvolvimento do software deve atender a uma estrutura organizacional, como a apresentada na figura 12, que mostra as fases da organização e suas formas de relacionamento.

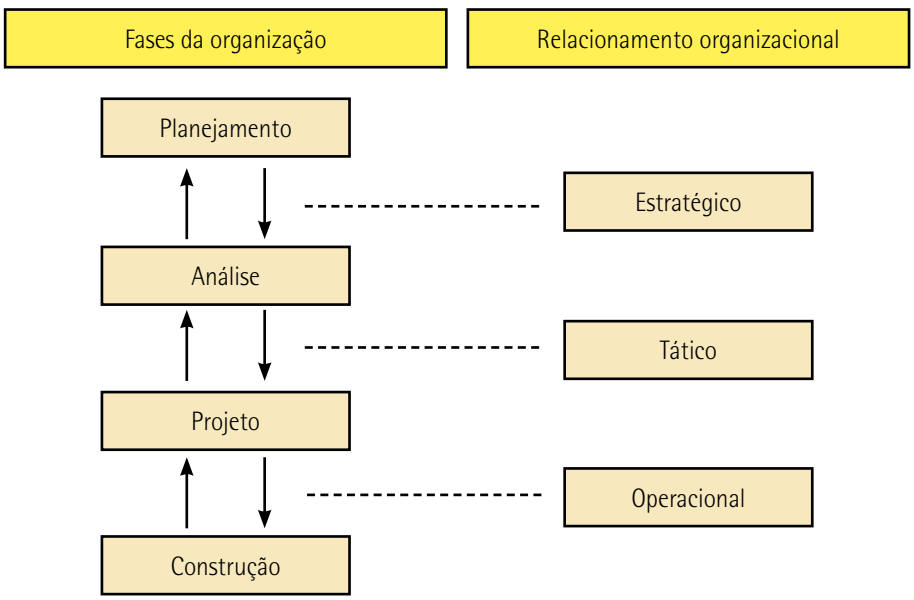

Figura 12 – Estrutura organizacional para o desenvolvimento de sistema/software com suas fases e formas de relacionamento

A estrutura organizacional é um fator ambiental da empresa que pode afetar a disponibilidade dos recursos e influenciar a maneira como os projetos são conduzidos (PMBOK, 2010).

2.1.2 Conceito e classificação dos Processos da ISO/IEC 12207

A norma NBR ISO/IEC 12207 – Processos do Ciclo de Vida do Software foi criada em 1995 com o objetivo de fornecer uma estrutura com base em uma linguagem comum para o adquirente, fornecedor, desenvolvedor, mantenedor, operador, gerentes e técnicos envolvidos com o desenvolvimento de

software. Esta linguagem comum é estabelecida na forma de processos bem definidos. A figura 13 mostra os processos classificados em três tipos: fundamentais, de apoio e organizacionais.

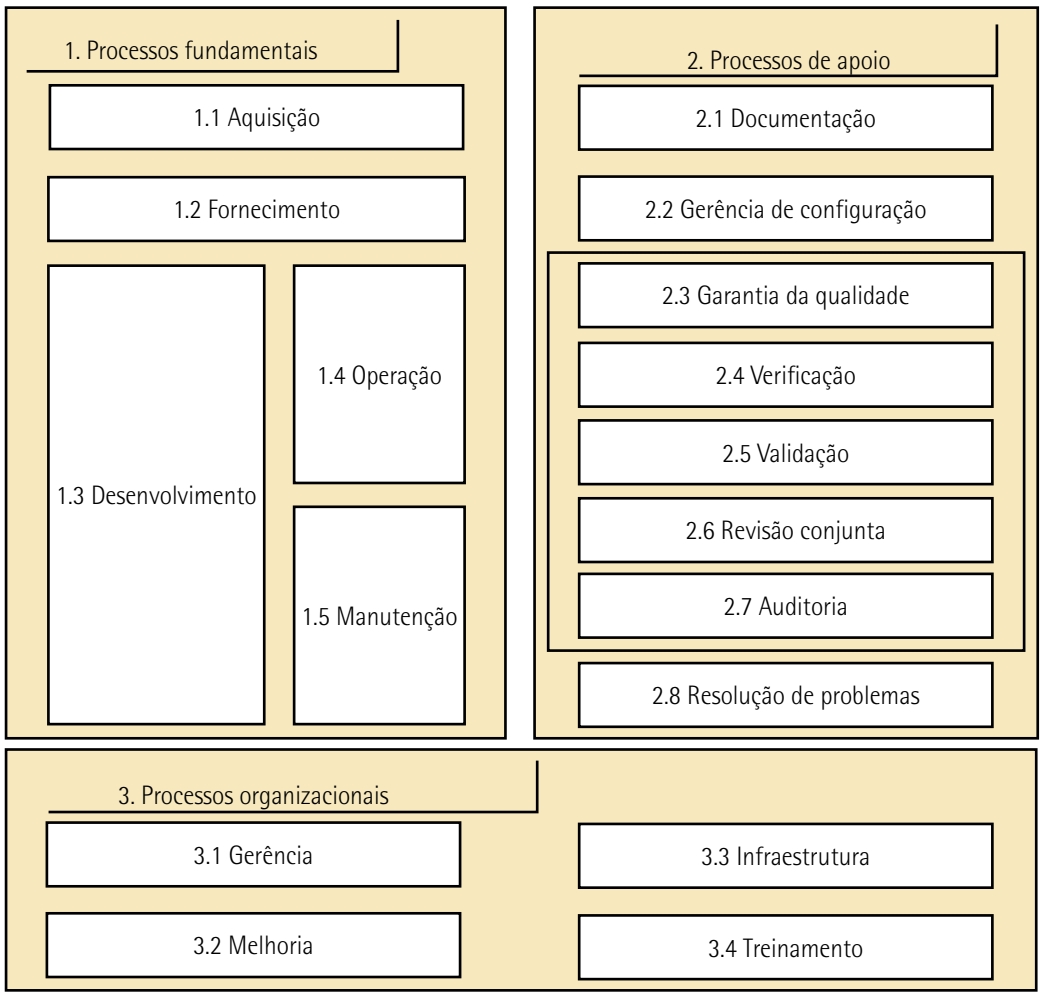

Figura 13 – NBR ISO/IEC 12207: Processos do Ciclo de Vida do Software

Fonte: ABNT (1998).

Observe na figura 13 o *framework* da ISO/IEC 12207, que fornece diretrizes para os processos envolvidos no desenvolvimento, manutenção e operação de sistemas de software, organizados em três categorias principais: processos fundamentais, processos de apoio e processos organizacionais, descritas a seguir:

- **Processos fundamentais**: esses processos estão diretamente relacionados à produção de software e entrega de produtos ou sistemas de software. Eles incluem as seguintes atividades:

  – Aquisição: refere-se à obtenção de software ou componentes de software de terceiros.

  – Fornecimento: refere-se ao fornecimento de software a outras partes interessadas, como clientes, usuários ou parceiros no desenvolvimento.

– Desenvolvimento: conjunto de atividades relacionadas à criação de software, da concepção à entrega, abrangendo atividades como especificação de requisitos, projeto, codificação e testes.

– Operação: é o acompanhamento da operação contínua de sistemas de software em ambientes de produção, incluindo o suporte e manutenção.

– Manutenção: refere-se às atividades de manutenção corretiva, adaptativa, evolutiva e preventiva em software.

- **Processos de apoio**: diretamente ligadas à produção do software, essas atividades garantem que os processos fundamentais sejam efetivados. Elas incluem as seguintes atividades:

  – Documentação: são atividades relacionadas a criação, manutenção e gerenciamento de documentação sobre o software.

  – Gerência de configuração: o gerenciamento das configurações de software inclui o controle de versões, releases e mudanças que ocorrem no ciclo de vida do software.

  – Garantia da qualidade: são definições para o monitoramento dos padrões e garantia da qualidade do software, para que sejam atendidos.

  – Verificação: refere-se ao rastreio do sistema de software para avaliar se o que foi proposto nos requisitos de software foi contemplado, acordado e testado.

  – Validação: segue na sequência da verificação, normalmente em conjunto pela especificação V&V (Verificação & Validação). Refere-se ao aceite do cliente em relação aos serviços dos desenvolvedores.

  – Revisão conjunta: é uma revisão formal e estruturada por parte dos stakeholders, que garantem a conformidade do software com o que foi estabelecido, contribuindo para o sucesso do projeto de software.

  – Auditoria: processo que envolve uma avaliação independente e sistemática dos processos e sistema de software para verificar sua conformidade com padrões, normas, procedimentos, diretrizes e requisitos estabelecidos.

  – Resolução de problemas: identifica defeitos, falhas, erros ou não conformidades no software.

- **Processos organizacionais**: são os processos de mais alto nível que afetam toda a organização. Eles incluem as seguintes atividades:

  – Gerência: as atividades relacionadas a esse processo envolvem a alocação de recursos materiais, humanos, tecnológicos e financeiros para o projeto de software.

– Melhoria: refere-se à melhoria contínua dos processos de software.

– Infraestrutura: refere-se aos vários elementos essenciais ou prioritários especificados pela gerência, abrangendo ferramentas para o desenvolvimento do software e arquiteturas, computadores, instalações físicas e demais recursos de hardware, equipes qualificadas e capacitadas, estruturas de serviços de dados e arquiteturas, regras e topologias de rede de computadores.

– Treinamento: o treinamento busca garantir que a equipe envolvida no desenvolvimento do software tenha habilidades e técnicas necessárias para desempenhar de forma competente suas funções.

No contexto brasileiro, a norma NBR ISO/IEC 12207 é reconhecida e utilizada como referência na indústria de software. É adotada por diversas organizações brasileiras, incluindo empresas de desenvolvimento de software, instituições acadêmicas e órgãos governamentais. Ela fornece diretrizes para a gestão e execução de projetos de software, abrangendo a concepção, especificação, implementação, implantação e manutenção do produto software.

2.2 Gerência de configuração e mudanças de software

2.2.1 Gerenciamento de configuração

O gerenciamento de configuração do software é o desenvolvimento e a aplicação de padrões e procedimentos para gerenciar um sistema em desenvolvimento. Esses procedimentos definem como registrar e processar as mudanças do sistema, como relacioná-los aos seus componentes e os métodos utilizados para identificar as diferentes versões desse sistema (Sommerville, 2011).

Como mostra o quadro 3, Sommerville (2011) sugere quatro principais atividades do gerenciamento de configuração.

Quadro 3 – Principais atividades do gerenciamento de configuração do software

Atividades	Descrição
1. Gerenciamento de mudanças	Com as constantes mudanças exercidas em cima dos softwares, elas devem ser registradas e aplicadas ao sistema de forma prática, econômica e que satisfaça o cliente
2. Gerenciamento de versões	Consiste em acompanhar e identificar o desenvolvimento das diferentes versões do sistema
3. Construção de sistemas	Processo de compilar e ligar componentes de software em um programa que é executado em uma configuração específica
4. Gerenciamento de releases	Envolve a preparação do software para o release externo e manter o acompanhamento das versões de sistema que foram liberadas para uso do cliente

Fonte: Sommerville (2011).

2.2.2 Gerenciamento de versões e releases

As mudanças ocorrem em todo o ciclo de vida do software. Quando aumenta o nível de complexidade e de confusão, é comprometido o controle na entrega do produto ou subproduto do software. O gerenciamento de versões acompanha e identifica o desenvolvimento das diferentes versões de um determinado sistema. Este processo é chamado de versionamento.

As versões são criadas para testes ou desenvolvimento interno e não são liberadas para o cliente. O quadro 4 mostra que podem ser adotadas algumas técnicas básicas no controle de versões para a devida identificação de componente.

Quadro 4 – Técnicas básicas de numeração e identificação da versão aplicadas no gerenciamento de versões

Técnicas básicas	Descrição
Numeração de versões	É o esquema de identificação mais comum. Atribui-se um número, explícito e único, de versão ao componente Exemplo: Versão 3.21 – o dígito (3) identifica mudança de estrutura, o dígito (2) indica inserção de uma funcionalidade e o dígito (1) indica que houve a implementação de uma mudança
Identificação baseada em atributos	Cada componente recebe um nome e um conjunto de atributos, que não é único em todas as versões. Exemplo: Versão TEC01_1 – TEC – representa o componente teclado, (01) inserção de um mapa de caracteres e (_1) indica que houve a implementação de uma mudança
Identificação orientada a mudanças	Além do anterior é associada uma ou mais solicitações de mudança Exemplo: CAR_04 – Carlos requisita uma quarta mudança

Exemplo de aplicação

Controle de versão baseada em numeração de versões.

Padrão de numeração de versão: X.Y.Zzzz, onde:

X corresponde a uma mudança de estrutura do software.

Y corresponde à inserção ou adaptação de novas funções ou funcionalidade.

Zzzz corresponde a adaptações, correções ou implementações no código-fonte.

Imagine que estamos agora abrindo o Windows 10. Hoje, dia 28/07/2020, verifica-se em informações do sistema a versão 10.0.18363. Se formos aplicar o conceito de identificação baseada em atributos, isto nos leva à seguinte análise:

X = 10, que indica a 10ª geração do projeto de arquitetura do Windows.

Y = 0, que indica que não houve inserções ou adaptações de novas funções ou funcionalidade no projeto original.

Zzzz = 18363, que indica as mudanças ocorridas no código-fonte.

Sempre existiram muitas versões de um sistema, mais do que releases, porque o release é a versão do software ou do sistema produzido autorizada para distribuir ao cliente. O lançamento do release é acompanhado de vários fatores técnicos e organizacionais, tais como:

- programa de instalação;
- manual técnico e do usuário;
- arquivos de configurações;
- bibliotecas, arquivos de dados e scripts de registros.

Exemplo de aplicação

Observe o grafo de controle de versões na figura 14, na qual o controle de versões (V 1.0, V 1.1 e assim por diante) acompanha as mudanças realizadas pelos desenvolvedores e os releases (Rel 1.0 e Rel 1.1) correspondem às versões aprovadas para distribuição ao usuário.

A numeração do release independe da numeração da versão, porém, neste caso, pode-se considerar o primeiro dígito uma mudança de estrutura.

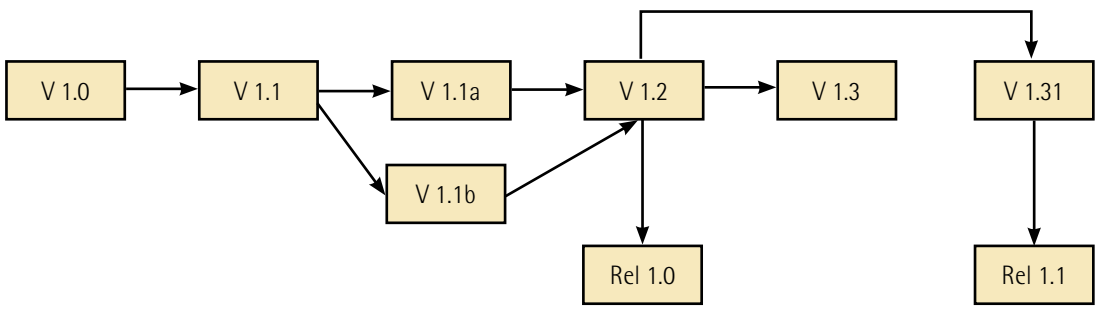

Figura 14 – Modelo de grafo de controle aplicado na numeração de versões e releases

2.2.3 Projeto e construção do software/sistema

O projeto e a construção do software/sistema seguem basicamente o princípio ditado por Pressman, como mostra a figura 15. Segundo o autor, "A engenharia de software é uma tecnologia em camadas e que deve estar fundamentada em um comprometimento organizacional com a qualidade" (2002).

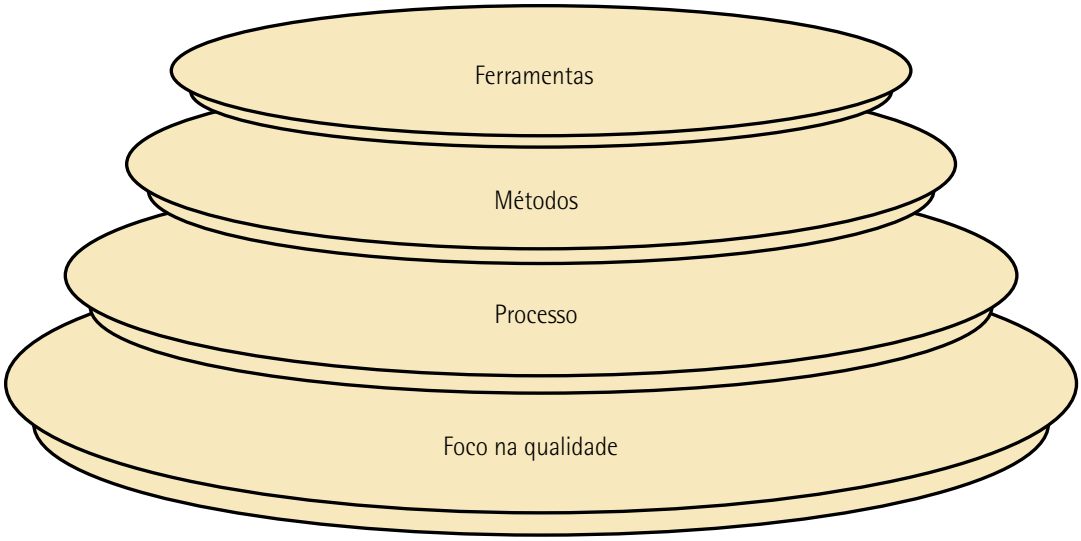

Figura 15 – Engenharia de software, uma tecnologia em camadas

Fonte: Pressman (2002).

Vamos analisar cada uma dessas camadas:

- **Foco na qualidade**: determina padrões e normas da qualidade a serem aplicadas pela engenharia de software. A qualidade leva à cultura de melhoria contínua dos processos e a abordagens cada vez mais efetivas. Devemos considerar que a qualidade é importante, mas se o usuário não está satisfeito, nada mais realmente importa.

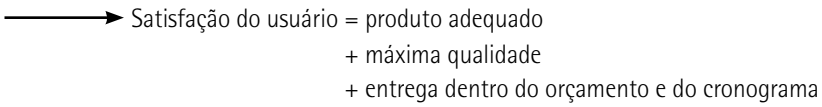

Figura 16

Observe alguns exemplos de normas e modelos da qualidade:

– NBR ISO/IEC 25010: Engenharia de sistemas e software – Requisitos e avaliação de qualidade de sistemas e software (SQuaRE). A norma foi alterada em 2013.

– NBR ISO 9001: Modelo de Garantia de Qualidade em projeto, instalação, desenvolvimento, produção, arquitetura e serviço. A norma foi alterada em 2015.

– ISO 12207: Processos do Ciclo de Vida do Software. A norma foi alterada em 2017.

– CMMI 2.0: *Capability Maturity Model Integration* (Modelo de Maturidade e Capacidade para Software). Foi lançado em março de 2018.

– ISO 15504: Spice – *Software Process Improvement and Capability Determination* (Melhoria do Processo de Software e Determinação da Capacidade). A norma foi alterada em 2003 como Spice Networking.

– MPS.BR: Melhoria do Processo de Software Brasileiro. A norma foi alterada em 2020.

- **Processo**: forma a estrutura básica para o projeto e desenvolvimento do software orientando-se pelos padrões e normas estabelecidos pela qualidade. O processo é responsável por manter integradas as camadas da tecnologia, permitindo o desenvolvimento racional e oportuno do software. O analista pode denominar um processo quando ele define e determina uma série organizada de eventos e atividades, com documentos e modelos que orientam a obtenção de resultados específicos, para um determinado serviço ou para a construção de um produto.

Uma estrutura genérica de um processo para a engenharia de software é composta basicamente por cinco atividades:

1. Elicitação: compreende o gerenciamento das relações com o cliente, a comunicação e a interpretação do negócio. A intenção é compreender os objetivos dos stakeholders (conjunto dos interessados).

2. Análise: partindo da interpretação do negócio, constrói-se um arquétipo do Plano de Projeto de Software que contemple a descrição do produto de software a ser produzido, as atividades e tarefas técnicas, o cronograma, a gestão do risco, os recursos e os custos empregados.

3. Modelagem: é a tarefa de criar modelos que representem ideias apresentadas em uma linguagem simbólica ou gráfica. Uma vez apresentada a ideia, é possível compreender, detalhar e refinar o modelo.

4. Construção: constitui a tarefa de codificação (geração do código), depuração (exame e limpeza dos dados, do código e da interface) e geração de diagnósticos.

5. Implementação/Implantação: é a entrega do software ao cliente, que por sua vez fornece feedbacks de avaliação, validação e aceite do produto de software.

A seguir, elencamos os principais modelos de processos de software:

– Modelo cascata (*Waterfall* ou Sequencial Linear): modelo clássico do ciclo de vida de desenvolvimento do software.

– Prototipagem: modelo iterativo com atividades para montagem de protótipos.

– Modelo incremental: modelo iterativo desenvolvido com o conceito de versões.

– Modelo espiral: modelo evolucionário que combina a natureza iterativa da prototipagem com os aspectos sistemáticos do modelo cascata.

– RUP – *Rational Unified Proc*ess (Processo Unificado Racional): processo proprietário de engenharia de software criado pela Rational Software Corporation e atualmente representado pela IBM.

– PSP – *Personal Software Process* (Processo de Software Pessoal): processo de desenvolvimento criado pelo SEI - Software Engeneering Institute (Humphrey, 1995), específico para engenheiros de software.

– TSP – *Team Software Process* (Processo de Software da Equipe): processo de desenvolvimento criado pelo SEI - Software Engeneering Institute (Humphrey, 1995), com enfoque na equipe de trabalho.

- **Método**: é o conjunto de procedimentos, regras e operações que fornecem uma técnica para chegar a uma determinada meta, um fim ou conhecimento. Os métodos incluem um amplo conjunto de tarefas, que abrange gestão de equipes, análises de requisitos, projeto, construção de programas, teste, entrega, suporte e manutenção. Formam um princípio básico que rege cada área da tecnologia, com atividades de modelagem e outras técnicas descritivas.

Vamos observar alguns exemplos de métodos:

– Metodologias ágeis aplicadas ao desenvolvimento de software: coleção de metodologias fundamentada na prática para modelagem efetiva de sistemas baseados em software, que podem ser aplicadas por profissionais no dia a dia. Uma destas metodologias que mais se destaca é o Scrum, que fornece um processo para projetos e desenvolvimento orientado a objetos.

– MR ou Matriz de Responsabilidades: é um método útil no processo organizacional para distribuição de responsabilidades entre os membros da equipe de desenvolvimento. A Matriz de Responsabilidades é orientada pelo PMBOK na gestão de stakeholders.

- **Ferramenta**: fornece apoio automatizado ou semiautomatizado para os processos e para os métodos. As tecnologias são apresentadas como ferramentas de trabalho e têm como objetivo agilizar, acompanhar, melhorar a compreensão e dar manutenção na elaboração e desenvolvimento software/sistema.

A seguir, elencamos alguns exemplos de ferramentas:

– Gestão de Projetos: o mercado disponibiliza aplicativos que têm como principais funções: elaborar distribuição de atividades e responsabilidades, elaborar cronogramas, planilhas de custos e modelar caminhos críticos no ciclo de desenvolvimento.

– Modelagem: essas ferramentas atendem aos requisitos do software/sistema. Servem para criar modelos da aplicação, dos dados e da infraestrutura da tecnologia da informação. Veja na figura 17 o uso da ferramenta de modelagem Visio da Microsoft para a construção de um diagrama de sequência da UML.

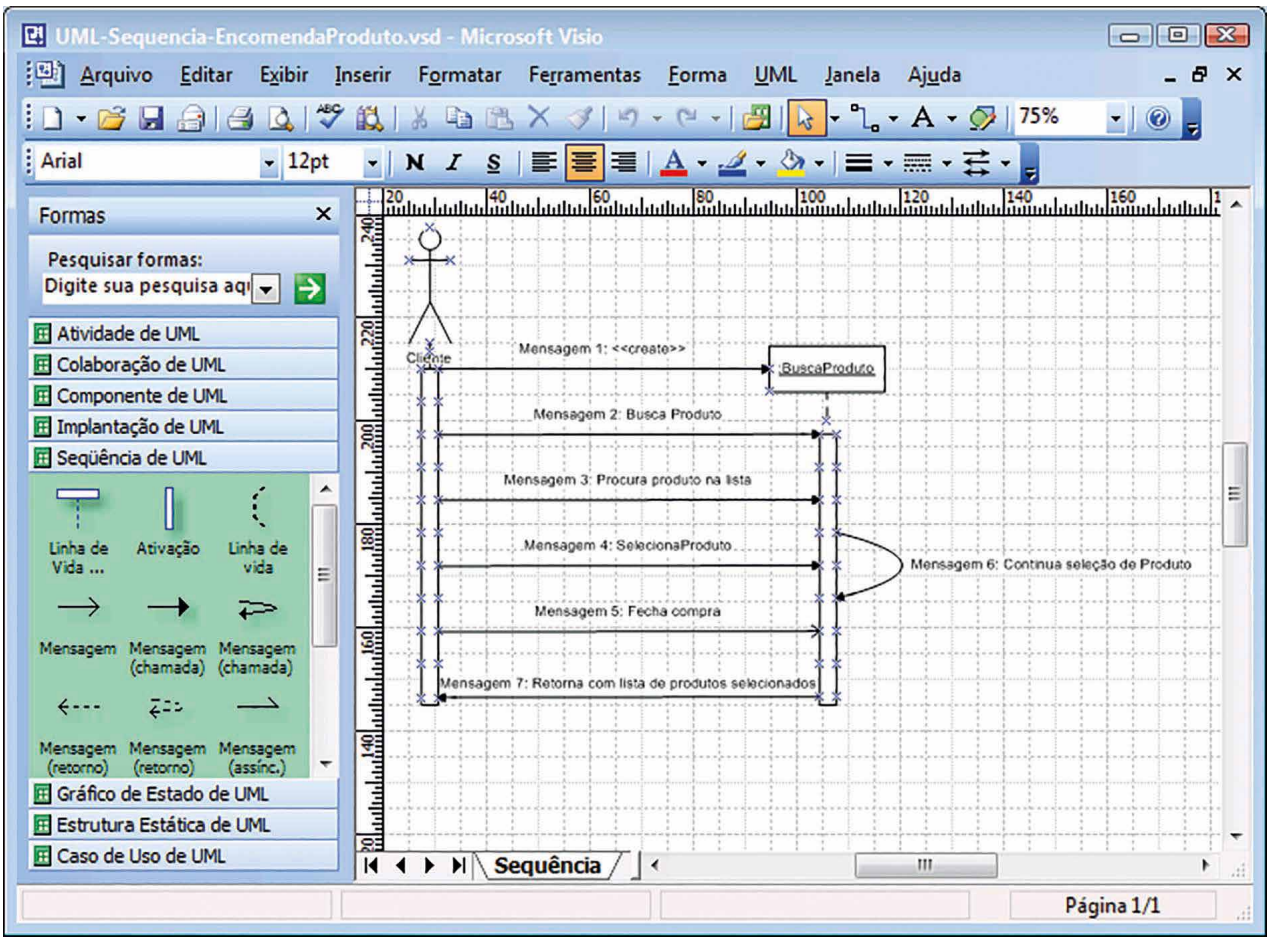

Figura 17 – Modelagem de software com diagrama de sequências elaborado pela ferramenta Visio da Microsoft

- Ferramenta Case: *Computer Aided Software Engineering* (Engenharia de Software Auxiliada por Computador) é um software de apoio que cria um ambiente de desenvolvimento para construir outro software. Esta ferramenta combina vários artefatos do software, hardware, estrutura dos dados e do sistema em um repositório. Contém informações sobre análise, projeto, construção de programas e teste.

- *Framework*: funciona no domínio da aplicação, provendo uma solução completa para uma família de problemas específicos de determinada funcionalidade, usando basicamente um conjunto de classes e interfaces que são divididas em quadros com seus respectivos códigos. Veja o exemplo de uso do framework NetBeans na figura 18 sendo usado para o desenvolvimento de classe.

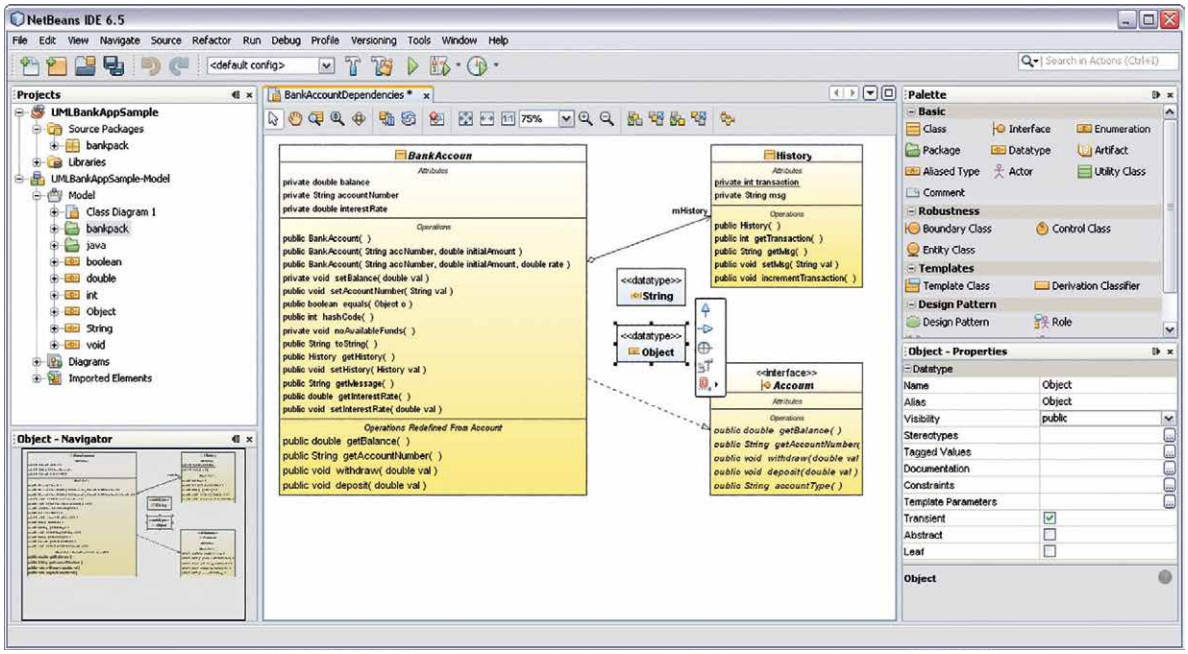

Figura 18 – Ambiente de trabalho na codificação e modelagem de software com o *framework* NetBeans

2.3 Gerenciamento de riscos do projeto

O risco pode ser conceituado como a combinação da probabilidade de um evento e suas consequências. É a medida do nível de incerteza associada à probabilidade de ocorrência de um evento e suas consequências.

Exemplo: Risco = Probabilidade × Impacto

O gerenciamento dos riscos do projeto tem por objetivo aumentar a probabilidade e/ou o impacto dos riscos positivos e diminuir a probabilidade e/ou o impacto dos riscos negativos, a fim de otimizar as chances de sucesso do projeto (PMBOK, 2017).

Os processos de gerenciamento dos riscos do projeto são:

1. Planejar o gerenciamento dos riscos.

2. Identificar os riscos.

4. Realizar a análise quantitativa dos riscos.

5. Planejar as respostas aos riscos.

6. Implementar respostas a riscos.

7. Monitorar os riscos.

Os processos de gerenciamento dos riscos do projeto apresentados na figura 19 são um *framework* que mostra processos discretos com interfaces definidas. Quando elas não são gerenciadas, os riscos têm o potencial de desviar o projeto do que foi planejado.

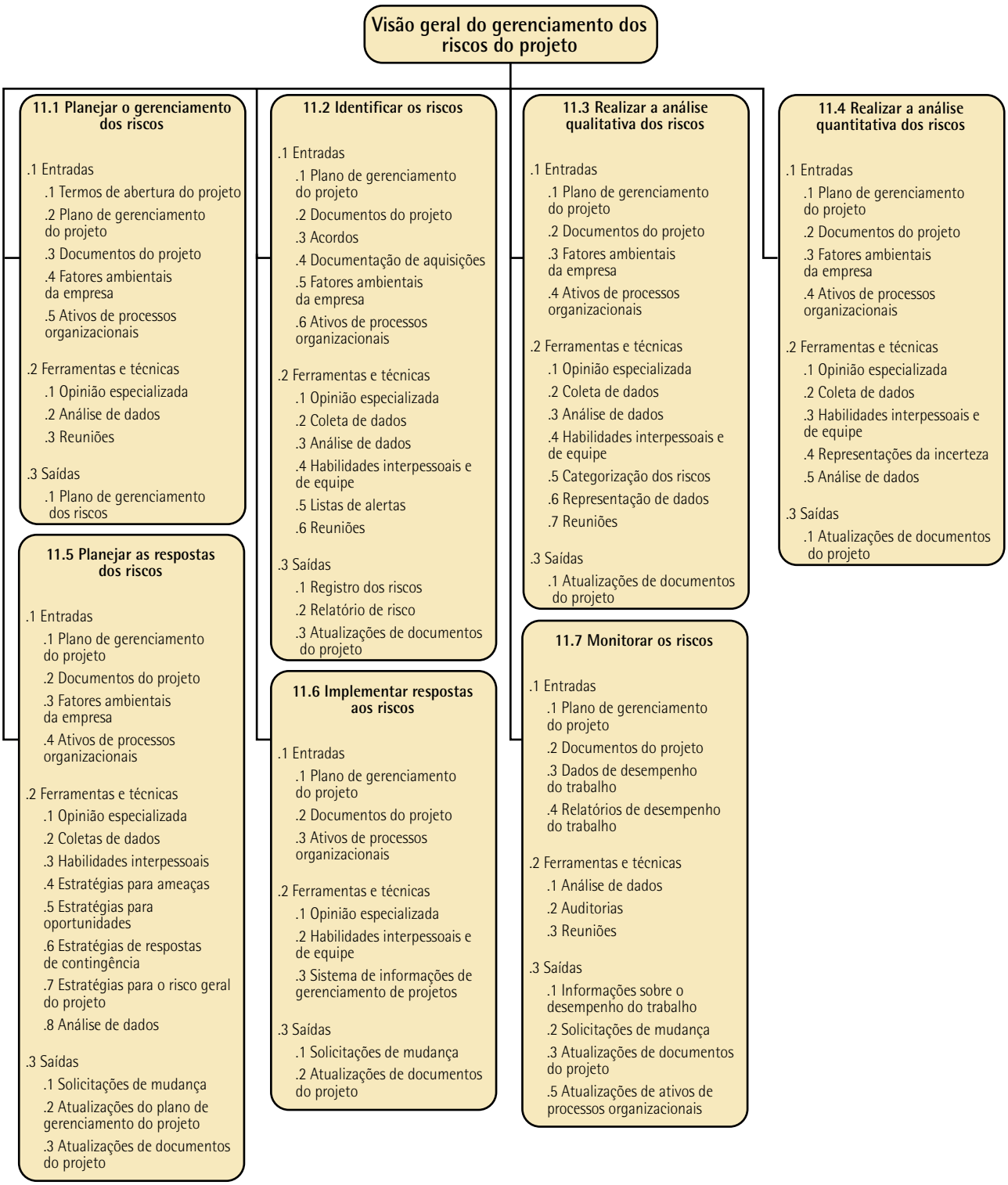

Figura 19 – Visão geral do gerenciamento dos riscos do projeto

Fonte: PMBOK (2017).

Saiba mais

Observe que o plano de gerenciamento de riscos é importante em qualquer projeto corporativo, seja pequeno ou grande. Veja no link a seguir como a TOTVS coloca em prática o gerenciamento dos riscos do projeto.

TOTVS. *Plano de Gerenciamento de Riscos: como elaborar?* Totvs, 2023. Disponível em: http://tinyurl.com/mk5h4e8y. Acesso em: 17 nov. 2023.

2.4 Desenvolvimento de sistemas de informação para internet

2.4.1 Desenvolvimento de software para internet

As páginas da web recuperadas por um browser constituem o software que incorpora instruções executáveis (p. ex. CGI, HTML, Pearl ou Java) e dados (p. ex. hipertexto e uma variedade de formatos visuais e de áudio). Com estes recursos tecnológicos surgem os sistemas e aplicações baseadas na web, as WebApps, que são pequenas aplicações embarcadas na internet. As mais diversas aplicações, que comumente precisavam de processamento local, são processadas em servidores que operam na internet.

Consistindo no principal uso dos sistemas de informação no mundo dos negócios, a amplitude desta área está classificada pelo termo Comércio Eletrônico (e-commerce), estruturado basicamente pelas tecnologias: B2B (business to business), B2C (business to consumer) e C2C (consumer to consumer).

A internet tem como base uma arquitetura servidor/cliente em ambiente distribuído. Por meio do browser ou uma aplicação específica (app), o cliente recebe documentos em vários formatos que podem ser manipulados por *plugins* ou *helpers*, sem ter que mudar o browser ou a aplicação. O servidor apenas entrega documentos e não se preocupa com as interfaces do usuário ou o formato dos documentos.

A figura 20 mostra uma arquitetura para internet útil para apresentar o projeto conceitual do ambiente web. É um modelo de abstração de alto nível sem detalhes específicos de como vai funcionar.

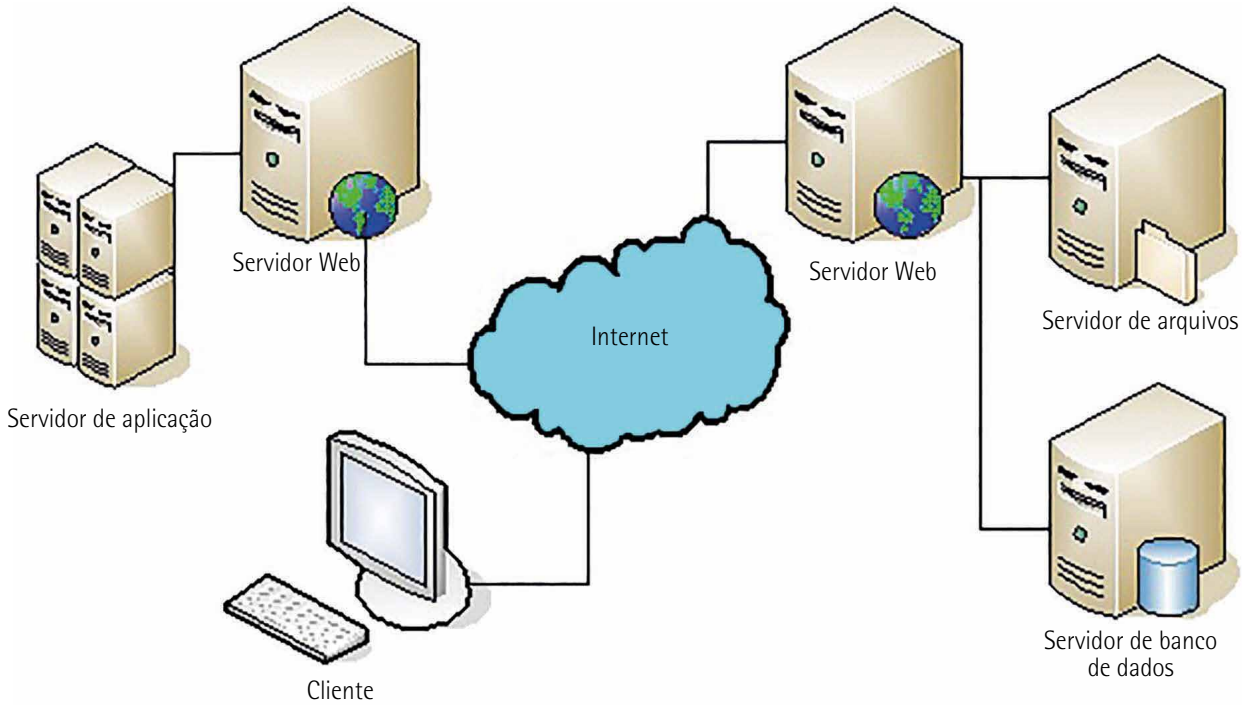

Figura 20 – Projeto conceitual de arquitetura de um sistema distribuído para internet

O projeto lógico apresentado na figura 21 mostra uma arquitetura com quatro nós e alguns detalhes de implementação, construído com os diagramas de implantação e de componentes da UML. É uma tecnologia com três camadas (apresentação, negócios e integração) em que são apresentados os componentes (blocos menores), a implantação (caixa) e os estereótipos de ligação <<TCP>>, <<HTTP>>, <<IP>> e <<DNS>.

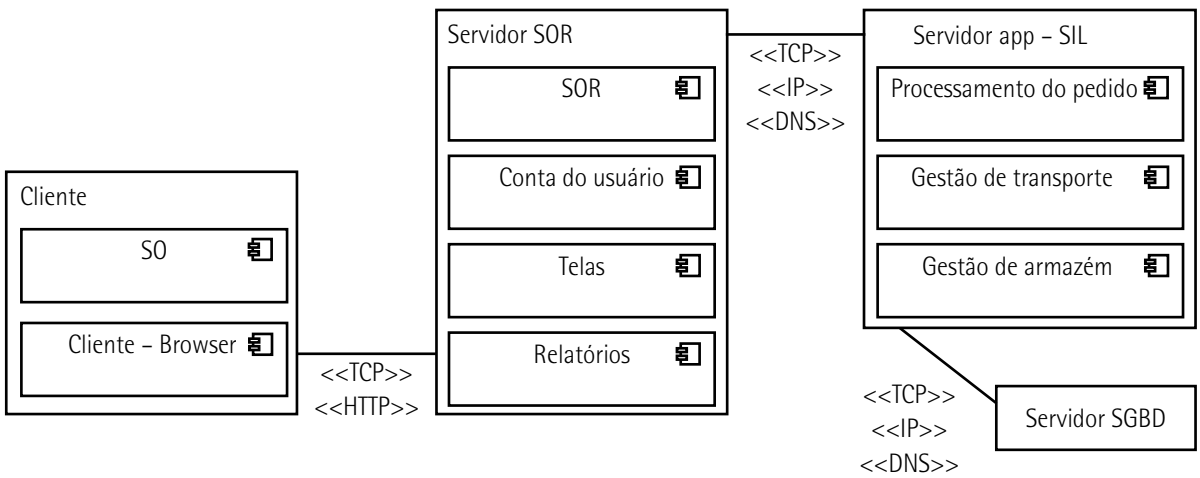

Figura 21 – Projeto lógico de arquitetura web de implantação, independente de ambiente operacional para cliente acessar SIL – Sistema de Informação Logístico

Sistemas e aplicações baseadas na web (WebApps) produzem uma complexa matriz de conteúdo e funcionalidade para uma ampla população de usuários finais. A Engenharia da Web (Web Enginnering – WebE) é o processo usado para criar WebApps de alta qualidade. A WebE aplica princípios científicos sólidos de engenharia e de gestão para o bem-sucedido desenvolvimento, implantação e manutenção de sistemas e aplicações baseadas na web.

As WebApps evoluem continuamente e devem ser estabelecidos mecanismos de controle de configuração, garantia de qualidade e suporte continuado.

As revisões e testes examinam uma ou mais das seguintes dimensões da qualidade em modelos WebE (Pressman, 2007):

- **Conteúdo**: avaliado em nível sintático e semântico.
- **Funcionalidade**: avaliação da conformidade com os requisitos do cliente.
- **Estrutura**: assegura o fornecimento apropriado de conteúdo e função da WebApp.
- **Usabilidade**: garante a cada categoria de usuário a interface necessária para navegação.
- **Navegabilidade**: testes para assegurar que toda a sintaxe e semânticas de navegação estão aceitáveis.
- **Desempenho**: é testado sobre uma variedade de condições de operações.
- **Compatibilidade**: teste da WebApp em uma variedade de diferentes configurações hospedeiras.
- **Interoperabilidade**: teste de interface com outras aplicações e/ou base de dados.
- **Segurança**: investigação de vulnerabilidades potenciais.

Observe o fluxo de testes dos elementos da pirâmide de teste da WebApp na figura 22. De cima para baixo são elementos visíveis ao usuário seguidos pelos elementos de projeto da infraestrutura.

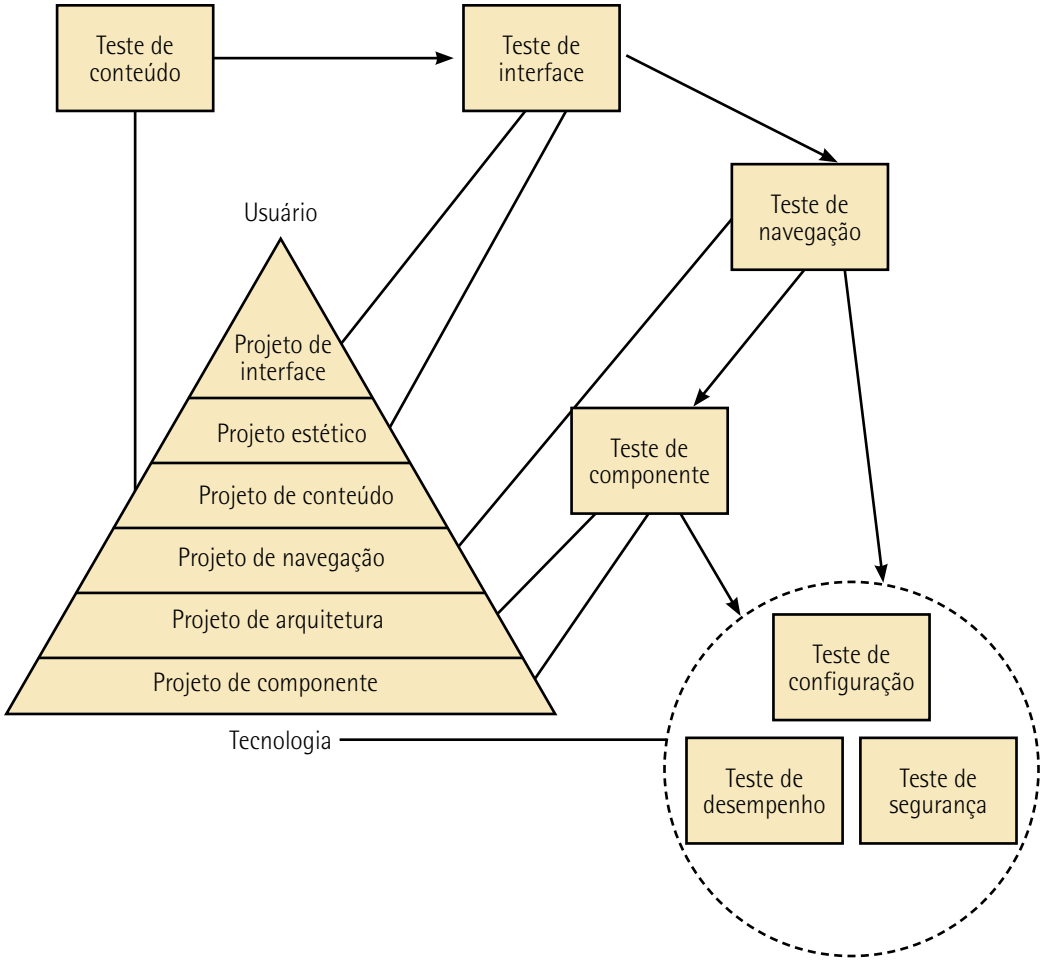

Figura 22 – Processo de teste da WebApp da pirâmide de projeto para WebApps

Adaptada de: Pressman (2011).

3 ORGANIZAÇÃO E O PROCESSO DE DESENVOLVER SOFTWARE

3.1 Joint Application Development (JAD)

De acordo com Fournie, o método Joint Application Development (JAD), que na tradução significa Desenvolvimento de Aplicação Conjunta, é uma metodologia criada pela IBM em 1977, cuja finalidade é reunir em uma equipe usuários e desenvolvedores com objetivos de alcançar acordos baseados na compreensão das funções do processo.

As sessões de JAD são usadas em desenvolvimento de software. Essas sessões facilitadas são focadas em reunir os especialistas em assuntos de negócio e a equipe de desenvolvimento para coletar requisitos e melhorar o processo de desenvolvimento de software (PMBOK, 2017).

Com essa metodologia é possível determinar as funções que serão aplicadas nas operações empresariais. Ela é comprovada para aumento de velocidade do processo de análise preliminar e ao mesmo tempo mantém uma abordagem estruturada mais flexível.

Participantes do JAD (Costa, 1994; Fournier, 1994):

- **Executivo patrocinador**: é a gerência de mais alto nível, pertencente às fileiras dos usuários e está fortemente comprometido com as atividades do processo. É ele quem fornece as diretrizes sobre as principais metas e objetivos de um projeto. Este executivo muitas vezes é o próprio cliente patrocinador do projeto.

- **Gerência funcional**: estas pessoas são peritas na matéria em discussão nas sessões JAD detalhadas, são as pessoas que precisam do sistema ou, ainda, representa o usuário final. Devem ser escolhidas pela sua capacitação profissional de descrever as práticas atuais de negócio melhor do que qualquer outra pessoa.

Lembrete

A gerência funcional é a que mais representa o sucesso do sistema de software. O JAD é uma técnica, talvez a melhor, para elicitar requisitos do software. Para relembrar estes requisitos, reveja o tópico1.4.3 Processos de requisitos do software.

- **Representantes do sistema de informação**: são algumas das poucas pessoas do desenvolvimento que levam conhecimentos técnicos das aplicações atuais de negócios pelo ponto de vista do sistema de informação e, para manter uma forte perspectiva de negócio durante as sessões, os requisitos do sistema são elaborados como os usuários e/ou os clientes os veem.

- **Líder da sessão**: responsável pelo controle das reuniões e interações entre os participantes da equipe. O líder da sessão deve garantir que o processo revisional prossiga conforme o planejado, controlar as interações entre os participantes e arbitrar quando necessário. Ele deve ser experiente na condução efetiva de reuniões envolvendo diversas pessoas e dominar o relacionamento interpessoal. Deve ser respeitado tecnicamente, ser independente e com algum envolvimento no projeto.

- **Secretário**: atividade semelhante ao *walktrough*, também se utiliza de ferramentas para organizar, documentar e apontar tendências dos assuntos pertinentes na discussão. São registradas apenas as informações relevantes de forma manual ou com o uso de software. O secretário JAD é muitas vezes citado como um especialista no uso de ferramentas CASE e de prototipação.

3.2 *Rational Unified Process* (RUP)

O *Rational Unified Process* (RUP), que na tradução significa Processo Unificado da Rational, é um processo proprietário de engenharia de software criado pela Rational Software Corporation, atualmente representado pela IBM, ganhando um novo nome, o IRUP, que agora é uma abreviação de IBM Rational Unified Process.

O RUP fornece às organizações um processo de software maduro, rigoroso e flexível. É distribuído on-line em formato eletrônico, utilizando tecnologia web, disponibilizando upgrades regulares de software pela Rational Software. Veja na figura 23 o ambiente operacional de desenvolvimento via web.

O RUP é configurado de acordo com as necessidades de cada organização. É documentado, desenhado, desenvolvido, distribuído e mantido como uma ferramenta de software, usando a UML (*Unified Modeling Language*).

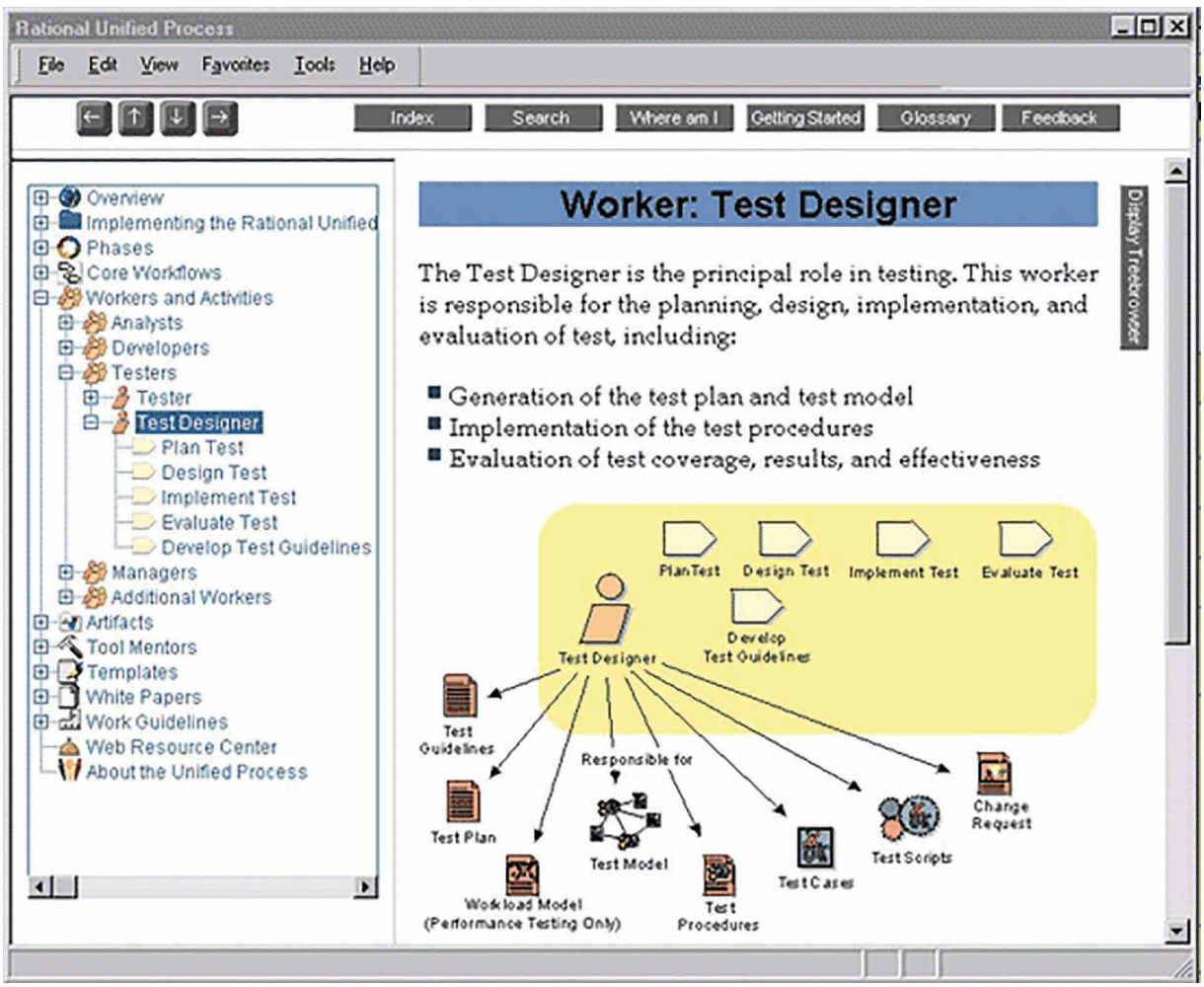

Figura 23 – Plataforma de trabalho RUP pela web

Fonte: Kruchten (2000).

Entre as características do RUP, podemos salientar a abordagem baseada em disciplinas para atribuir tarefas e responsabilidades dentro de uma organização de desenvolvimento, cuja meta é garantir que a produção de software de alta qualidade atenda às necessidades dos usuários dentro do cronograma e orçamentos previsíveis, e a capacidade de capturar as principais boas práticas modernas da engenharia de software: o desenvolvimento iterativo, a gestão de requisitos, a arquitetura baseada em componentes, o uso de software de modelos visuais, a verificação contínua da qualidade e a gestão e o controle de mudanças de software. A seguir estão as especificações e detalhes dos itens mencionados:

- Desenvolvimento iterativo

Diferentemente do método clássico de desenvolvimento, um grande sistema de software não permite que se defina o problema e se construa uma solução eficiente em um único passo. Dependendo da complexidade, os requisitos mudam com frequência, devido a vários fatores, tais como: restrições da arquitetura empregada, mudanças nas necessidades primárias do cliente e mudanças refletidas devido a um refinamento do problema inicialmente levantado. Segundo Kruchten, "cada iteração pode corresponder a um novo release do sistema, diminuindo assim o risco do projeto, permitindo ao cliente uma avaliação do progresso do projeto e de sua gerência" (2001).

- Gestão de requisitos

Documentar apropriadamente é crucial para o sucesso de um grande projeto. O RUP é conduzido por casos de uso e sustentado em outros diagramas da UML. Ele descreve como documentar as funcionalidades, restrições do sistema, restrições do projeto e requisitos através de casos de uso (*use cases*), desde a análise de requisitos até o teste do sistema finalizado. Sobre este tema, Kruchten afirma que "os casos de uso e cenários são exemplos de artefatos do processo, que se mostram altamente eficazes para documentar os requisitos funcionais" (2000).

- Arquitetura baseada em componentes

O RUP é fundamentado na arquitetura de componentes do sistema. Promove a definição inicial de uma arquitetura de software robusta, que facilita a parametrização do desenvolvimento, a reutilização e a manutenção.

Uma arquitetura baseada em componentes viabiliza um sistema extensível promovendo a reutilização de software. De acordo com Kruchten, "o RUP suporta essa sistemática de construção de sistema, focando-se numa arquitetura executável nas primeiras fases do projeto (2000).

- Uso de software de modelos visuais

Ao representar o projeto utilizando construções gráficas, o RUP apresenta de forma eficiente uma visão geral da solução, o que permite também que clientes sem nenhuma afinidade com a área de desenvolvimento de sistemas possam facilmente compreender o projeto e, dessa forma, desenvolverem-se mais com o projeto. Segundo Kurchten, "a linguagem UML se transformou em um padrão para as indústrias ao representar projetos, e é utilizada amplamente pelo RUP" (2000).

- Verificação contínua da qualidade

Assegurar a qualidade do software durante o desenvolvimento do projeto é algo intensivo feito pelo RUP, que assiste todo o processo envolvendo todos os membros no controle e planejamento da qualidade (Kruchten, 2000).

- Gestão e controle de mudanças do software

O RUP define métodos para controlar e monitorar as mudanças a fim de garantir que não venham a comprometer inteiramente o sucesso do projeto (Kruchten, 2000).

3.2.1 Fases e *workflow* (ou disciplinas de trabalho)

A figura 24 mostra o ciclo de vida do desenvolvimento de software praticado no RUP. É dividida em fases e a cada ciclo resulta numa nova geração do produto ou parte dele. Cada fase é dividida em iterações a definir em cada projeto concreto. O RUP tem quatro fases, seis disciplinas de trabalho e três disciplinas gerenciais.

A arquitetura do RUP apresenta um processo bidimensional. Na dimensão horizontal estão as fases do processo de desenvolvimento (estrutura dinâmica), representando mudanças no tempo de desenvolvimento, e na dimensão vertical estão os *workflows* ou disciplinas (estrutura estática), representando mudanças no esforço despendido no desenvolvimento.

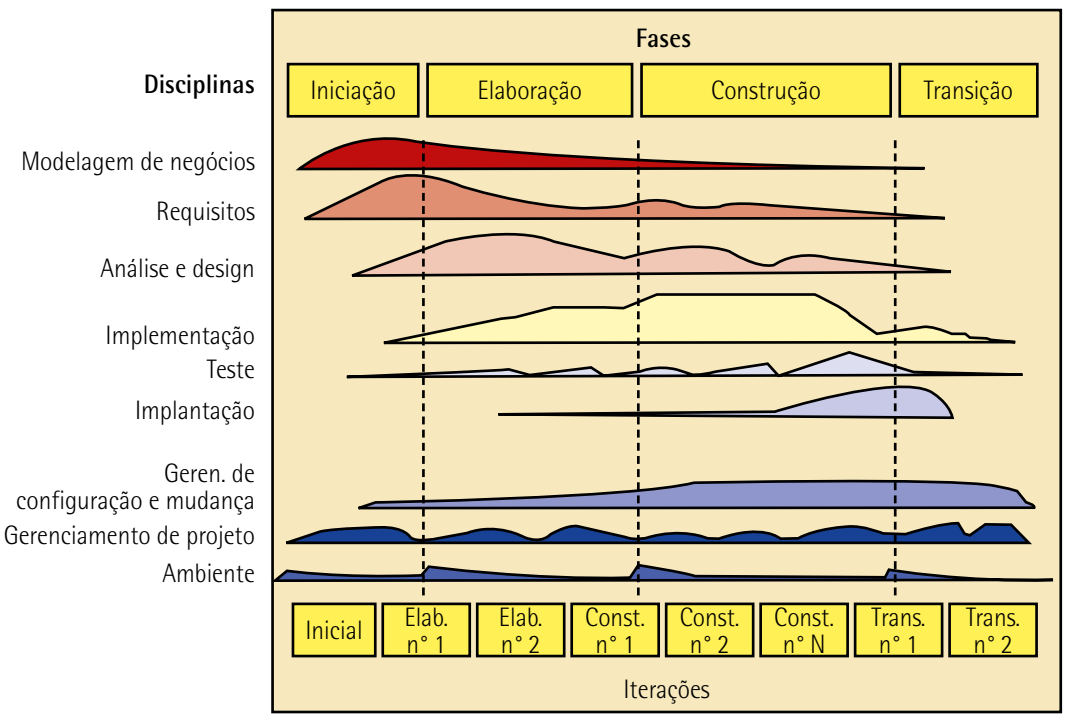

Figura 24 – Arquitetura do RUP com suas fases e *workflow*

Fonte: Krunchten (2000).

- Estrutura dinâmica

A estrutura dinâmica corresponde às quatro fases do RUP que equivalem ao ciclo de vida do desenvolvimento de um produto de software. Ao final de cada uma destas fases verificamos se os objetivos da fase foram concluídos. Após completar as quatro fases, uma versão release do software está completa, e, para cada nova versão, o software passará por todas as fases novamente (desenvolvimento evolucionário). Acompanhe a seguir as quatro fases:

1. Iniciação/Concepção (*Inception/Conception*)

   – Definir o escopo e as fronteiras do sistema (contrato com o cliente).

   – Elaborar casos de uso crítico ao sistema (atores e requisitos).

   – Elaborar o *business case* do sistema, que deve incluir os critérios de sucesso, avaliação de riscos, uma estimativa de recursos e um cronograma dos pontos mais importantes.

   – No final: obter concordância dos stakeholders, compreensão dos requisitos e credibilidade nas estimativas. O projeto pode ser cancelado ou fortemente reformulado se este marco falhar.

2. Elaboração (*Elaboration*)

   – Garantir que o projeto, a arquitetura e o conjunto de requisitos estejam estáveis para o cumprimento de prazos e custos.

   – Produzir protótipos evolucionários e exploratórios.

   – Eliminar os elementos de maior risco.

   – Descrever requisitos adicionais (não funcionais e com menor grau de importância).

   – Revisar o *business case*.

   – No final: obter estabilidade da arquitetura, demonstrar que o projeto é viável (tempo, custo) e que o projeto pode ser cancelado ou reformulado se falhar neste marco.

3. Construção (*Construction*)

   – Minimizar custos através do bom uso de recursos.

   – Desenvolver versões utilizáveis.

   – Completar os processos de análise, projeto, implementação e testes das funcionalidades do sistema.

– Desenvolver de maneira iterativa e incremental um produto que esteja pronto para ser entregue aos usuários, um manual e uma descrição da versão corrente.

– Decidir se o software, o site e os usuários estão prontos para a implantação do sistema.

– No final: a disponibilização pode ser adiada se os critérios não forem cumpridos.

4. Transição/Implantação (*Transition*): atividades de entrega do software

– Testar para validar o novo sistema.

– Treinar usuários e pessoas responsáveis pela manutenção.

– Iniciar as tarefas de marketing e distribuição.

– No final: obter satisfação dos clientes, gastos e custos dentro do aceitável.

- Estrutura estática: na estrutura estática, o RUP é representado por suas nove disciplinas (seis disciplinas de trabalho e três disciplinas gerenciais). Cada uma delas utiliza quatro elementos primários de modelagem: papéis, atividades, artefatos e *workflows*.

1. Papéis

– Descreve o comportamento e responsabilidades de um indivíduo ou grupo de indivíduos de uma equipe.

– O comportamento de cada papel é dado pelo conjunto de atividades desempenhadas por ele.

– As responsabilidades de cada papel estão associadas a artefatos que devem ser desenvolvidos ao longo do processo.

– Exemplo de papéis: especificador de casos de uso, arquiteto de softwares, projetista de interface, gerente de projetos.

2. Atividades

– Realizadas pelos papéis, consistem em ações que atualizam ou geram artefatos.

– A granularidade de uma atividade é expressa em horas ou dias.

– As atividades são divididas em etapas de tarefas: entendimento, realização, revisão.

Exemplo de aplicação

Atribuições de papéis e responsabilidades:

Quadro 5 – Papéis e responsabilidades

Atividade	Papel
Planejar uma iteração	Gerente de projetos
Determinar casos de uso e atores	Analista de sistemas
Executar teste de desempenho	Analista de testes de desempenho

3. Artefatos

– Informações produzidas, modificadas ou utilizadas ao longo do desenvolvimento de software.

– São utilizados como entrada para atividades e são produzidos como saída.

– Exemplos: modelos (de caso de uso, projetos), documentos (plano de negócios, conjunto de artefatos, plano de projeto) e código-fonte.

4. *Workflow*

– Define uma sequência de atividades que produzem resultados observáveis na forma de artefatos.

– Podem ser expressos em forma de diagramas de sequência, colaboração, atividades.

– O *workflow* é apresentado em seis disciplinas de trabalho e três disciplinas gerenciais, descritas a seguir.

Disciplinas de trabalho:

- Modelagem de negócio (*Business Modeling*): a intenção é melhorar o entendimento do negócio através do desenvolvimento de um modelo de negócios. Os documentos gerados são casos de uso de negócio e modelo de objetos de negócio.

- Requisitos (*Requirements*): possibilita melhor entendimento dos requisitos do sistema partindo de um acordo com o cliente, com objetivo de oferecer uma orientação aos desenvolvedores. É produzido um modelo de caso de uso detalhado e um protótipo do sistema.

- Análise e projeto (*Analysis and Design*): os requisitos capturados na disciplina anterior são transformados em projeto. Modelos de análise e projeto são gerados.

- Implementação (*Implementation*): nesta fase, o projeto é transformado em código. A estratégia é desenvolver o sistema em camadas, particionando em subsistemas. O resultado final é um componente testado que faz parte do produto final.

- Teste (*Test*): elaboração de testes e verificação se todos os requisitos foram cumpridos. Os documentos gerados são os modelos e casos de testes.

- Utilização (*Deployment*): torna o produto disponível para o usuário final. Responsável pelo empacotamento do produto, instalação, treinamento ao usuário e distribuição do produto.

Disciplinas gerenciais:

- Gerenciamento de configuração: controla as modificações e mantém a integridade dos artefatos do projeto.

- Gerenciamento de projeto: descreve várias estratégias para o trabalho com um processo iterativo.

- Ambiente: abrange infraestrutura necessária para o desenvolvimento do sistema.

3.3 Fatores e métricas da qualidade do software

3.3.1 Fatores, atributos e métricas da qualidade do software

No contexto de qualidade de software é preciso saber que a qualidade de software é uma mistura complexa de fatores que podem variar para cada aplicação e de acordo com o que os clientes desejam.

Existem três conceitos distintos para determinar a qualidade: fator de qualidade, atributo da qualidade e métrica da qualidade.

Fator de qualidade é um aspecto amplo e geral que influencia na qualidade de um produto ou serviço. Ele representa um conceito mais abstrato, sendo uma categoria ampla que agrupa atributos específicos e importantes para determinar a qualidade do produto ou serviço em questão. Os fatores de qualidade podem variar dependendo do domínio ou contexto em que estão sendo aplicados. O conjunto de fatores de qualidade do produto software permite controlar os níveis dos padrões e requisitos que estão sendo implementados.

Alguns exemplos de fatores de qualidade são funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.

Atributos da qualidade são características específicas e mensuráveis que contribuem para um determinado fator de qualidade. Eles representam um aspecto particular do produto, que é determinado em um fator de qualidade. Cada fator de qualidade pode ter vários atributos associados a ele.

Exemplo de aplicação

Análise da escolha dos atributos da qualidade para compor o fator de qualidade **usabilidade**.

Os atributos atratividade, eficácia, operabilidade, produtividade, segurança, satisfação e utilização contribuem para o fator de qualidade usabilidade. Porém, não é necessário usar todos esses atributos para compor o fator de qualidade usabilidade de uma aplicação específica. Normalmente são escolhidos de três a cinco atributos para compor um fator de qualidade.

Situação-problema: determinar os atributos da qualidade a serem escolhidos para o fator usabilidade de um ERP do departamento financeiro.

Para as operações de um software do departamento financeiro são importantes, por exemplo, garantias de privacidade e sigilo nas operações do usuário, precisão nos resultados obtidos pelo software, facilidade de operar o software e outros mais que podem ser determinados com base nos requisitos do usuário.

Ao analisar o contexto, pode-se determinar que:

a) Garantias de privacidade e sigilo nas operações do usuário: sem dúvida, o atributo segurança (Segu) deve ser escolhido, e para garantir que não haja a incerteza no uso de determinadas funções do software, pode ser escolhido o atributo operabilidade (Oper).

b) Precisão nos resultados obtidos pelo software: o atributo eficácia (EFCA) garante que os resultados estejam nos padrões e dimensões dos cálculos e resultados determinados pelo usuário.

c) Facilidade de operar o software: nesse caso poderiam ser determinados os atributos produtividade (Prod) e satisfação (Sati).

Dessa forma, para o controle da qualidade do ERP do departamento financeiro sugere-se construir o seguinte quadro:

Quadro 6 – Requisitos

	Requisito Não Funcional (RNF) Fator de qualidade: usabilidade do ERP do Financeiro				
	Atributos da qualidade				
Requisito Funcional (RF)	Segu	Oper	EFCA	Prod	Sati
Cadastros	X		X	X	
Comunicação	X				X
Consultas	X	X			X
Controle da conta	X	X			
Relatórios		X	X		

Métricas da qualidade são medidas usadas para quantificar e qualificar um determinado atributo da qualidade. Com as métricas da qualidade, é possível avaliar informações e dados objetivos sobre o desempenho do item em relação aos atributos da qualidade.

Exemplo de aplicação

Na sequência do exemplo anterior e com base na tabela, determine as métricas da qualidade do requisito funcional Cadastros.

Observe a tabela. Para medir os atributos da qualidade relativos ao requisito funcional Cadastros, devem ser contemplados os atributos: segurança (Segu), eficácia (EFCA) e produtividade (Prod). Veja como fica:

a) Segurança (Segu): medida indireta comparativa entre as LGPD com o que o software disponibiliza.

b) Eficácia (EFCA): medida direta sobre o índice de falhas nos resultados obtidos pelo software em confronto com o que é esperado nos requisitos do usuário, tais como falhas em cálculos, campos de casas decimais, campos vazios não identificados ou a omissão de um resultado esperado.

c) Produtividade (Prod): medida direta que pode contemplar medidas de taxa de transferência da rede, *troughout* do sistema, índice de falhas nas operações do sistema.

Em resumo, o fator de qualidade influencia diretamente na qualidade do produto software, o atributo da qualidade é uma característica específica do fator de qualidade e a métrica é uma ou mais medidas quantitativas usadas para avaliar um atributo e, consequentemente, o qualifica.

Os fatores de qualidade são usados também para detectar áreas com problemas, de maneira que soluções possam ser apresentadas e que o processo de software possa ser melhorado.

Os fatores de qualidade do software podem ser divididos em dois grupos:

- Fatores que podem ser medidos diretamente. Exemplo: defeitos por ponto de função.
- Fatores que podem ser medidos indiretamente. Exemplo: usabilidade ou mantenabilidade (ou manutenibilidade).

São coletados dados básicos de produtividade e qualidade pelos engenheiros de software, esses dados são analisados, comparados com médias anteriores e avaliados pelos gerentes de software para verificar se ocorreu melhoria ou não.

Exemplo de aplicação

Estudo de caso: LGPD (Lei n.13.709/2018). Esta legislação visa proteger a privacidade e os dados pessoais dos cidadãos brasileiros. Um dos principais fatores de qualidade a ser considerado é a conformidade com a lei e a segurança dos dados. Assim, vamos analisar os atributos e métricas relacionados a ela.

1. Fator de qualidade: a conformidade garante que o software atenda às obrigações estabelecidas por lei, como obter o consentimento adequado, objeto específico da coleta de dados, segurança da informação, acesso direto aos dados, atualização e remoção de dados.

2. Atributos da qualidade relacionados à LGPD:

- **Segurança dos dados**: proteção dos dados contra acessos não autorizados, uso indevido ou violações.
- **Privacidade**: garantia de privacidade dos dados e respeito aos direitos do titular.
- **Consentimento**: implementação de funções que permitem coleta, uso e armazenamento de dados pessoais.
- **Transparência**: fornecimento de informações claras e acessíveis aos usuários sobre as políticas de privacidade e tratamento de dados.

3. Métricas da qualidade para medir a implantação da LGPD:

- **Taxa de conformidade**: medir a porcentagem de requisitos da LGPD que foram implementados corretamente no software.
- **Nível de segurança**: avaliar a eficácia das medidas de segurança por meio de testes de invasão, avaliação de vulnerabilidades, entre outros.
- **Taxa de consentimento**: medir a proporção de usuários que forneceram consentimento adequado para o processamento de seus dados pessoais.
- **Tempo de resposta a solicitações**: avaliar o tempo necessário para responder e atender às solicitações de acesso, retificação ou exclusão de dados dos usuários.
- **Número de violações**: registrar o número de violações de dados ocorridas no sistema e implementação de medidas corretivas para reduzi-las.

4 CUSTOMIZAÇÃO DO SOFTWARE

4.1 Análise de Ponto de Função (APF)

A Análise de Ponto de Função (APF) é uma técnica de medição utilizada para estimar o tamanho funcional do software. Ela permite avaliar a complexidade e o esforço necessário para desenvolver, manter e testar um sistema de informação.

Os objetivos da APF são medir as funcionalidades do software requisitadas pelo cliente/usuário e medir projetos de desenvolvimento e manutenção de software independentemente da tecnologia que será implementada.

A métrica chamada de ponto de função (function points – FP) pode ser usada efetivamente como um meio para medir a funcionalidade fornecida por um sistema. Por meio de dados históricos, a métrica FP pode ser empregada para estimar o custo ou trabalho necessário para projetar, codificar e testar o software, prever o número de erros que serão encontrados durante o teste e prever o número de componentes e/ou o número de linhas projetadas de código-fonte no sistema implementado (Pressman, 2011).

De acordo com Pressman (2011), os valores do domínio de informações são definidos da seguinte maneira:

- Número de entradas externas (*number of external inputs* – EEs).
- Número de saídas externas (*number of external outputs* – EOs).
- Número de consultas externas (*number of external inquiries* – EQs).
- Número de arquivos lógicos internos (*number of internal logical files* – ILFs).
- Número de arquivos de interface externos (*number of external interface files* – EIFs).

Para calcular FP, usa-se a seguinte equação:

$$FP = \text{contagem total} \times (0{,}65 + 0{,}01 \times \Sigma (F_i^*))$$

* F_i são fatores de ajuste de valor. Por exemplo: $F_i = 25$, baixa complexidade; $F_i = 50$, média complexidade e $F_i = 75$, alta complexidade.

A aplicabilidade da APF atualmente é mantida pelo IFPUG (International Function Point Users Group), organismo internacional responsável pela manutenção e evolução do padrão de medição de pontos de função (Albert *et al.*, 2013).

De acordo com o Tribunal Regional do Trabalho do Paraná – TRTPR (2023), o processo de medição (figura 25) conduzido pela IFPUG determina a contagem de pontos de função com base nas atividades que identificam e classificam os componentes funcionais básicos:

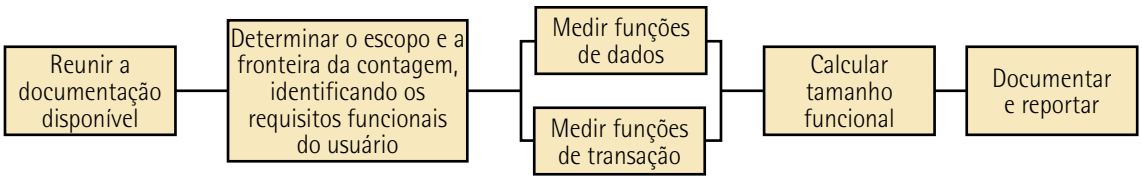

Figura 25 – Processo de medição da APF

Fonte: TRTPR (2023).

Componentes funcionais básicos:

- **Arquivo Lógico Interno (ALI)**: representam as informações mantidas internamente pelo sistema. Eles podem ser considerados grupos de dados relacionados. Por exemplo, em um sistema de cadastro de clientes, o arquivo de clientes seria um ALI. A complexidade de um ALI é avaliada com base em três características: número de campos, número de relacionamentos e complexidade de processamento.

- **Arquivo de Interface Externa (AIE)**: são as entradas e saídas do sistema que se comunicam com outros sistemas ou usuários. Eles são responsáveis por trocar informações com o ambiente externo. Por exemplo, em um sistema de integração com um serviço de pagamento online, o AIE seria a interface que recebe as transações de pagamento. A complexidade de um AIE é avaliada considerando-se características como número de campos, número de relacionamentos e complexidade de processamento.

- **Transações**: representam as funcionalidades do sistema que alteram o estado de um ou mais arquivos lógicos, isto é, as operações realizadas pelo software. Por exemplo, em um sistema de gerenciamento de estoque, uma transação pode ser a atualização do estoque após uma venda. As transações são avaliadas com base em características como complexidade de processamento, número de arquivos lógicos afetados e número de campos. As transações são medidas em três grupos:

  – Entradas Externas (EE): transações de entrada de dados pela fronteira da aplicação para processamento.

  – Saídas Externas (SE): transações de saída de dados processados para fora da fronteira da aplicação.

  – Consulta Externa (CE): transações que enviam dados para fora da fronteira da aplicação, sem a realização de operações de qualquer processamento, com objetivo apenas de apresentar a informação para o usuário.

No domínio da aplicação, as relações dos componentes funcionais básicos considerados na contagem da APF são mostradas no modelo da figura 26.

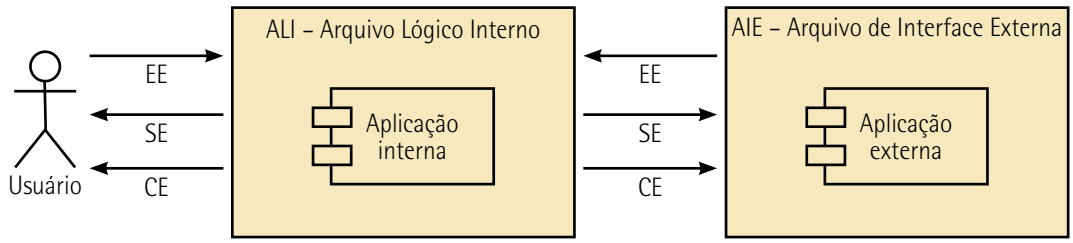

Figura 26 – Componentes funcionais básicos considerados na contagem da APF

Na contagem da APF são necessárias informações sobre as funções de dados e transacionais. Uma análise detalhada dos requisitos do usuário pode resultar na seguinte APF (TIMP, 2015), apresentada no quadro a seguir:

Quadro 7 – Análise de Pontos de Função (APF) detalhada

Função de dados ou de transação	Tipo de função	Complexidade (determinada pelos componentes das funções)	Pontos de função
Cliente	ALI	Média	10
Produto	ALI	Baixa	7
Fornecedor	AIE	Baixa	5
Incluir cliente	EE	Alta	6
Alterar cliente	EE	Média	4
Excluir cliente	EE	Baixa	3
Consultar cliente	CE	Baixa	3
Relatório 1 de cliente	SE	Baixa	4
Relatório 2 de cliente	SE	Média	5
Relatório 3 de cliente	SE	Baixa	4
Relatório 4 de cliente	SE	Alta	7
Incluir produto	EE	Média	4
Alterar produto	EE	Baixa	3
Excluir produto	EE	Baixa	3
Consultar produto	CE	Média	4
Relatório de produto	SE	Média	5
Consulta de fornecedor	CE	Baixa	3
Relatório de fornecedor	SE	Média	5
Tamanho funcional			**85**

Fonte: TIMP (2015).

Saiba mais

Conheça agora a ferramenta de software *open source* para contagem de pontos de função. É de domínio público e apresentada pelo Ministério do Planejamento, Desenvolvimento e Gestão do Brasil.

Desenvolvido pela Prime Informática, o APF Prime Light é uma ferramenta *open source* que auxilia a elaboração de estimativas de tamanho funcional de projetos de software, baseada na técnica de Análise de Pontos de Função estabelecida pelo IFPUG (International Function Point Users Group). Consulte o APF Prime Light no link a seguir:

BRASIL. *Software Público Brasileiro*. Brasília, 2011. Disponível em: http://tinyurl.com/3ray8bmn. Acesso em: 17 nov. 2023.

4.2 *Use-Case Points* (UCP)

Use Case Points (UCP) é uma técnica de contagem usada no desenvolvimento de software para estimar o tamanho e a complexidade de um sistema com base nos casos de uso.

Em uma abordagem de estimativa com casos de uso, diferente de estimar a contagem por linhas de código (do inglês *lines of code* – LOC) ou ponto de função (do inglês *function points* – FP), um determinado caso de uso pode precisar de meses de trabalho, enquanto o caso de uso de outra parte do sistema pode ser implementado em um ou dois dias (Pressman, 2011).

Os casos de uso representam um grau de abstração superior a LOC ou FP. Apesar de poder levar um tempo considerável, o caso de uso pode representar um domínio completo do software com características altas de reúso, que, uma vez estabelecidas, pode-se também obter estimativas de contagens de LOC ou FP.

O método UCP proposto por Gustav Karner (1993) *apud* Farahneh e Issa (2011) se baseia nos elementos descritos a seguir.

O cálculo ajustado de UCP (AUCP – *Adjusted* UCP) é um cálculo que corresponde à estimativa do tamanho e complexidade do sistema com base nos casos de uso, calculado pela equação:

AUCP = UUCP + TACF + EF

As descrições e comentários sobre as variáveis UUCP, TACF e EF estão a seguir:

1. Casos de Uso (UC – *Use Cases*): representam as interações entre o usuário e o sistema, descrevem o comportamento do sistema do ponto de vista do usuário. Cada caso de uso descreve uma funcionalidade específica que o sistema deve fornecer. Na contagem de pontos de casos de uso se considera:

- Peso não ajustado de casos de uso (UUCW – *Use Case Weights*): corresponde à soma dos pesos de todos os casos de uso. Tipos de peso: simples (1), mediano (2) e complexo (3).

Quadro 8 – Exemplo do cálculo de UUCW

Tipo de UC	Peso	N. de casos de uso	Total
Simples	1	4	4
Mediano	2	8	16
Complexo	3	2	6
		UUCW	26

- Peso não ajustado de atores (UAW – *Unadjusted Actor Weights*): corresponde à soma dos pesos de todos os atores. Tipos de peso: simples (1), mediano (2) e complexo (3).

Quadro 9 – Exemplo do cálculo de UAW

Tipo de ator	Peso	N. de casos de uso	Total
Simples	1	3	3
Mediano	2	1	2
Complexo	3	1	3
		UAW	8

Peso Total não Ajustado (UUCP – *Unadjusted Use Case Points*): é a soma de UUCW com UAW.

UUCP = UUCW + UAW

Logo: UUCP = 26 + 8 = 34

2. Fator de ajuste de complexidade técnica (TCAF – *Technical Complexity Adjustment Factor*): correspondem aos requisitos do sistema, requisitos não funcionais e processos de desenvolvimento do sistema. São critérios que ajudam a avaliar a complexidade técnica do sistema, considerando desempenho, eficiência, reutilização de código, facilidade de instalação entre outros. O cálculo do TCAF é:

TCAF = 0,6 + (0,01 × TFactor *)

*TFactor (peso × influência), que é a somatória dos fatores de influência técnica. No seu cálculo são consideradas as variáveis: tipos de peso, que podem ser simples (1), mediano (2) e complexo (3); e valores que correspondem ao nível de influência, que variam de 0 (não influencia) a 5 (tem forte influência).

Quadro 10 – Exemplo do cálculo de TCAF

Fator	Requisito	Peso	Influência	TFactor
TF1	Sistema servidor/cliente	2	2	4
TF2	Rede de computadores	2	4	8
TF3	Usabilidade	1	2	2
TF4	Portabilidade	3	5	15
TF5	Segurança	1	4	4
			TCAF	33

Logo: TCAF = 0,6 + (0,01 + 33) = 0,93

3. Fator ambiental (EF – *Environment Factor*): corresponde à disponibilidade de recursos para o desenvolvimento do projeto e do produto software. São fatores que refletem as condições externas ao projeto que podem afetar sua execução, como a experiência da equipe, a estabilidade dos requisitos, a complexidade do ambiente operacional, entre outros. O cálculo do EF é:

EF = 1,4 + (-0,03 × EFactor *)

* EFactor (peso × influência), que é a somatória dos fatores de influência ambiental. No seu cálculo são consideradas as variáveis: tipos de peso, que pode ser simples (1), mediano (2) e complexo (3); e valores que correspondem ao nível de influência, que variam de 0 (não influencia) a 5 (tem forte influência).

Quadro 11 – Exemplo do cálculo de TCAF

Fator	Requisito	Peso	Influência	EFactor
EF1	Programadores (Homens/hora)	3	4	12
EF2	Engenheiros de software (Homens/hora)	2	2	4
EF3	Treinamento dos desenvolvedores	1	3	3
EF4	Treinamento dos usuários	3	5	15
EF5	Recursos	2	5	10
			EF	44

Logo: EF = 1,4 + (-0,03 + 44) = 0,08

4. Estimativa do esforço para produzir de forma racional o caso de uso (do inglês, *Rationale's use case effort estimation*): esse cálculo permite finalizar o cálculo ajustado de UCP (AUCP) que, acordo com a equação a seguir:

AUCP = UUCP + TCAF + EF = 34 + 0,93 + 0,08 = 35,01

Uma aplicação desse resultado pode ser usada como o cálculo da Produtividade do sistema, que de acordo com Karner (1993) *apud* Farahneh e Issa (2011), demonstra que a produção de um caso de uso pode variar de 15 a 30 horas por AUCP.

Se for estimada a produção do caso de uso em 20hs por AUCP, temos que:

Produtividade = 35,01 × 20 = 700,2 h/sistema.

Com o valor da produtividade do sistema, com base na AUCP (ou simplesmente UCP), pode-se estimar, por exemplo, o custo total para produzir o sistema.

E se a empresa ou equipe desenvolvedora tiver um histórico de produção, o valor calculado de AUCP pode ser usado como medida de eficiência no desenvolvimento do sistema, redução ou aumento de implementações e custo do sistema, prazos de entrega e outros.

4.3 Testes de software

Testes de software são práticas fundamentais no desenvolvimento de programas de computador, visando garantir a qualidade e confiabilidade do produto final.

O teste é destinado a mostrar que um programa faz o que é proposto a fazer para descobrir os defeitos do programa antes do uso. Quando se testa o software, o programa é executado usando dados fictícios. Os resultados do teste são verificados à procura de erros, anomalias ou informações sobre os atributos não funcionais do programa (Sommerville, 2011).

Os testes podem ser manuais ou automatizados, envolvendo casos de teste elaborados para cobrir diversas situações, como testes unitários, de integração, de sistema, de aceitação (validação) e regressão.

Um princípio básico na realização de testes de software (principalmente em sistemas de média complexidade para cima) é diferenciar a equipe puramente de desenvolvimento da equipe especificamente de testes, ou seja, quem desenvolve não testa, e quem testa não desenvolve.

Diversas técnicas podem ser aplicadas em diferentes momentos e de diferentes formas para validar os aspectos principais do software. Neste livro-texto serão abordadas duas técnicas de destaque nos testes de software: os testes alfa e beta, e os testes caixa-branca e caixa-preta.

4.3.1 Testes alfa e beta

Quando um software é construído especificamente para um cliente, é normal ele passar por um teste de aceitação. Esse teste, por ser conduzido pelo próprio usuário, passará por uma bateria de testes levando às vezes semanas ou mesmo meses para ser finalizado. No entanto, se o software for feito para vários clientes/usuários, o teste de aceitação pode não ser suficiente para atender um cliente específico devido a diferenças regionais e diversos ambientes operacionais. Por isso, a melhor estratégia é aplicar os testes alfa e beta. Veja a seguir os pontos de vista diferentes para cada ambiente de uso.

O teste alfa exige um ambiente controlado, ou seja, os usuários são levados a testar o software desde seus estágios iniciais de instalação até a sua operação completa, e é conduzido internamente pela equipe de desenvolvimento. Nessa fase, o software é submetido a testes rigorosos, simulando cenários diversos e abrangendo funcionalidades específicas. O objetivo é identificar falhas, erros, omissões e possíveis melhorias antes de disponibilizá-lo para um grupo maior de usuários.

Por exemplo, uma empresa de web design pode reunir a equipe de desenvolvimento com alguns funcionários para o teste de uma aplicação web em relação à interface homem-computador. O teste será realizado num ambiente especial, em que são registradas todas as impressões dos desenvolvedores e funcionários, suas reações, comentários e outros.

O teste beta ocorre após as mudanças determinadas para o software no teste alfa. Ao contrário do teste alfa, são realizadas exclusivamente no hábitat do usuário, o ambiente de teste é descontrolado e sem a participação dos desenvolvedores. O objetivo é coletar feedbacks reais do público-alvo, identificar problemas em um ambiente mais próximo ao uso real e avaliar a usabilidade do software em diferentes ambientes operacionais.

Por exemplo, a mesma empresa de web design em outro caso de testes para a mesma aplicação web mencionada anteriormente permite usar a aplicação para diferentes perfis de usuários e em diversos ambientes operacionais (tais como sistemas operacionais e hardware diferente). Por meio de comunicação das questões mais frequentes (*frequently asked questions* – FAQ) e feedbacks automáticos gerados pela própria aplicação web, os desenvolvedores identificam a experiência do usuário (*user experience* – UX), possíveis falhas e sugestões de melhorias.

Lembrete

A menos que você seja um especialista na área, tenha cuidado ao instalar um software de versão beta em seu computador. Fique atento aos principais problemas que podem ocorrer: instabilidade, incompatibilidade com o ambiente operacional, segurança dos dados, falta de suporte técnico e uso ineficiente de recursos computacionais.

4.3.2 Testes caixa-preta e caixa-branca

Os testes caixa-preta e caixa-branca são duas abordagens distintas para realização de testes de software, representam, respectivamente, testes do ponto de vista externo ao software e do ponto de vista interno do software. São guiados pelo código-fonte, suas métricas de software aplicáveis e se realmente estão atendendo aos requisitos com um bom desempenho e segurança.

Observe a figura 27. O teste caixa-preta (também chamado de teste comportamental) é uma técnica em que os testadores avaliam o software sem ter conhecimento interno da sua estrutura ou lógica interna. O objetivo é validar se o software funciona corretamente em relação aos requisitos estabelecidos, sem se preocupar com a implementação interna.

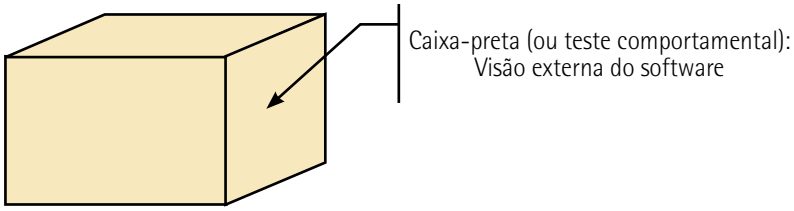

Figura 27 – Representação de caixa-preta em que o testador não consegue ver seu interior

Os testes são baseados exclusivamente na relação entre os requisitos do usuário (RU) com os requisitos funcionais (RF), focando nas entradas e saídas esperadas do software em relação ao usuário.

Por exemplo, em um aplicativo de e-commerce os testadores realizam testes caixa-preta para verificar se todas as funcionalidades declaradas e interface do usuário, tais como adicionar itens ao carrinho, concluir a compra e realizar o pagamento, estão claras para o usuário e funcionando corretamente, sem olhar para o código-fonte do aplicativo.

Observe a figura 28. O teste caixa-branca (também chamado de teste estrutural ou, como chama Pressman (2011), o teste da caixa-de-vidro) é focado nos possíveis erros internos de estrutura dos componentes do sistema. É uma abordagem em que os testadores têm acesso e conhecimento detalhado da estrutura interna do software. Os testes são elaborados com base na análise do código-fonte do programa, visando garantir que todas as instruções e caminhos lógicos tenham sido exercitados adequadamente.

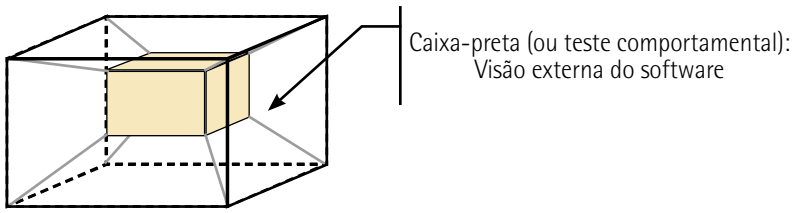

Figura 28 – Representação de caixa-branca em que o testador consegue ver seu interior

Os testes são baseados exclusivamente na relação entre os requisitos funcionais (RF) com os requisitos do sistema (RS), focando no código-fonte, chamadas ao banco de dados e interfaces de hardware e de rede.

Por exemplo, em um software de controle de estoque os testadores realizam testes caixa-branca para garantir que todas as condições lógicas do código, como situações de estoque mínimo ou máximo atingido, foram verificadas corretamente, percorrendo todas as ramificações do código. Nas situações de estoque mínimo ou máximo, pode haver a condição em que a lógica de cálculos esteja correta, porém a possível falha de acesso ou transferência de dados pode acarretar em dados omitidos no cálculo, o que acarretaria uma falha do software.

4.3.3 Teste unitário, de integração, aceitação e regressão

Como vimos anteriormente, os métodos de testes de software (testes alfa e beta; e testes caixa-preta e caixa-branca) são complementares, abrangendo todas as características e funcionalidades implementadas e essenciais para garantir a qualidade e a eficiência do software durante o processo de teste e validação.

Teste unitário, de integração, aceitação e regressão são as principais atividades da testabilidade do software, em que devem estar inclusos os principais métodos de testes de software. James Bach define testabilidade em 1994 como a facilidade com que um programa de computador pode ser testado (*apud* Pressman, 2011).

- Teste unitário

O teste unitário tem como objetivo avaliar um determinado componente do software. O foco é a verificação de pequenas unidades isoladas do código-fonte, como funções, métodos ou classes, para garantir que cada unidade funcione corretamente. Esses testes são realizados pelo próprio desenvolvedor durante o processo de codificação e são essenciais para identificar erros logo no início do desenvolvimento.

Por exemplo, em um sistema de gerenciamento de transportes, com base no método de teste caixa-branca, o desenvolvedor cria testes unitários para verificar se a função responsável pela roteirização está de acordo com a lógica de programação especificada no caso de teste do requisito.

- Teste de integração

Depois de passar todos os componentes de software por testes unitários, eles podem ser combinados para construir outros componentes, módulos ou um sistema completo.

Os testes de integração são realizados para verificar a troca de mensagens entre diferentes módulos ou componentes do software. O teste de integração visa garantir que as unidades integradas funcionem corretamente em conjunto, identificando possíveis problemas de comunicação e comportamentos inesperados.

Por exemplo, em fase de teste alfa (testes no ambiente do desenvolvedor), um analista de sistemas valida a implantação do componente de software "Emissão de Nota Fiscal". A validação ocorreu após verificar como corretas a codificação e testes do componente de software a ser implantado no cliente. Para validação do software por parte do cliente, é necessário integrar o componente "Emissão de Nota Fiscal" ao ambiente operacional dele e fazer os principais testes de integração, como transferência de dados, interface de hardware e outros.

- Teste de aceitação

O teste de aceitação ou teste de validação inicia após o teste de integração, quando cada componente integrado ao sistema já tiver sido aprovado, erros de interface identificados e corrigidos.

O teste de aceitação tem como objetivo verificar se o software atende aos requisitos do cliente ou do usuário final. Os testes são conduzidos com base em casos de uso ou especificações em requisitos. Em termos gerais, o principal objetivo é ter o aceite do cliente.

Por exemplo, em um software de e-commerce, na categoria B2C, os testes de aceitação verificam se os processos de compra, pagamento e entrega de pedidos estão funcionando conforme as expectativas do cliente.

- Teste de regressão

De acordo com Pressman (2011), cada vez que um novo módulo é acrescentado como parte do teste de integração, o software muda. Novos caminhos de fluxo de dados são estabelecidos, podem ocorrer novas entradas e saídas, novas lógicas de programação são implementadas.

Enfim, os testes de regressão são realizados para garantir que as mudanças no software, alterações feitas no código-fonte, correções de bugs ou implementação de novas funcionalidades não tenham introduzido novos problemas ou afetado o funcionamento de outras partes do software que já estavam funcionando corretamente.

Por exemplo, após identificar uma falha de cálculo em uma determinada função em um ERP do departamento financeiro, foi feito um teste de regressão aplicando engenharia reversa. Nele, a depuração do código deve ocorrer no sentido inverso para rastrear o código a partir da informação errada gerada em direção aos dados, algoritmos e entradas que construíram a informação.

Resumo

Na unidade I foram discutidas as bases da tecnologia da informação que sustentam a infraestrutura para a engenharia de software e a engenharia de sistemas, Além das características do produto software, o reúso, a demanda e a indústria de software. Também foi abordado de que forma o software é produzido e mantido, suas dificuldades e estratégias de solução.

A engenharia de software conduz o conhecimento do profissional, a saber, o que é um software bem escrito e o acompanhamento em todo o ciclo de vida de sua produção: sua concepção, implantação, evolução e mudanças que ocorrem.

Nessa unidade, o aprendizado gera capacidade para conceber o software por meio da engenharia de requisitos, não importando seu tamanho, complexidade ou nível exigido de qualidade. Também foram vistos os estudos de viabilidade, que garantem o início correto do projeto, os processos da engenharia de requisitos e os principais requisitos RU, RNF, RF e RS, bem como a interface homem-computador.

No ciclo de desenvolvimento de sistemas computacionais, inicialmente foram mostradas as diferenças do software, modelos e padrões da qualidade, tais como a NBR ISO/IEC 12207, que acompanha todo o projeto de software, suas ferramentas e técnicas de desenvolvimento e a importância da modularidade e componentização no reúso dos produtos de software.

Todo o software tem que ser mantido por meio da gerência de configuração do software, que acompanha todo o desenvolvimento, suporte e mudanças que ocorrem no ciclo de vida do software. Por isso, nesta unidade são abordados os principais aspectos do versionamento.

No cenário de gestão e processos, o estudo inicia-se pelo gerenciamento dos riscos do projeto, na identificação e contingência de onde o projeto pode falhar. Em seguida, são mostradas as características do desenvolvimento de sistemas para internet, como as equipes podem ser organizar por meio do JAD e o RUP, que apresenta o processo unificado, ou seja, que percorre todas as etapas do ciclo de vida do desenvolvimento do software. Elaborado pela Rational Software Corporation e aprimorado pela IBM, o RUP é sem dúvida um processo completo para conceber, desenvolver e implantar o software. Este tópico é complementado com os principais atributos da qualidade que acompanham todo o processo de construir software.

Por fim, nessa unidade são apresentadas as formas de customização (ou configuração) do software, para efeito de licitações e concepção do projeto. Esta etapa se baseia na análise de pontos por função (AFP) e pelo *use case points* (UCP). A unidade é finalizada com as etapas de testes e diagnósticos durante a implementação, tais como os testes alfa/beta, caixa-branca/caixa-preta e as formas de garantir e manter o software operacional e com boa capacidade.

Exercícios

Questão 1. (UFSM/2022, adaptada) Em relação à engenharia de requisitos de software, avalie as afirmativas.

I – Os requisitos funcionais descrevem as funções que o software deve executar, isto é, aquilo que ele deve fazer.

II – Os requisitos não funcionais descrevem restrições sobre os serviços ou as funções que o software oferece. Esses requisitos podem ser de vários tipos, como de eficiência, de confiabilidade, de portabilidade e de segurança.

III – O requisito "o sistema deverá fazer o cancelamento de consultas" é classificado como um requisito funcional.

IV – Para o levantamento de requisitos, as únicas técnicas conhecidas são a prototipagem e a entrevista.

É correto o que se afirma em:

A) I, apenas.

B) IV, apenas.

C) I e II, apenas.

D) I, II e III, apenas.

E) I, II, III e IV.

Resposta correta: alternativa D.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: os requisitos funcionais descrevem as funcionalidades específicas que o software deve fornecer, ou seja, as ações ou as operações que o sistema deve ser capaz de realizar. Portanto, esses requisitos definem o comportamento funcional do software.

II – Afirmativa correta.

Justificativa: os requisitos não funcionais descrevem restrições ou características adicionais do sistema que não estão relacionadas diretamente às funcionalidades específicas, mas, sim, a critérios como desempenho, qualidade ou segurança. Exemplos incluem requisitos de tempo de resposta, níveis de segurança e confiabilidade do sistema, entre outros.

III – Afirmativa correta.

Justificativa: o requisito "o sistema deverá fazer o cancelamento de consultas" descreve uma função específica que o software deve realizar. Como ele está diretamente relacionado à funcionalidade do sistema, é classificado como um requisito funcional.

IV – Afirmativa incorreta.

Justificativa: o levantamento de requisitos é uma etapa fundamental no desenvolvimento de software, e várias técnicas podem ser usadas para coletar informações e entender as necessidades dos usuários e do sistema.

Como dissemos, existem diversas técnicas de levantamento de requisitos. A prototipagem e as entrevistas são apenas duas delas. Outras técnicas comuns incluem reuniões, questionários, estudos de caso, análise de documentos existentes e grupos focais, entre outras atividades.

Questão 2. (Idecan/2022, adaptada) A disciplina de teste de software não busca apenas identificar falhas em um sistema. Ela busca, também, analisar a qualidade e garantir que o sistema será entregue de forma satisfatória, respeitando aspectos como requisitos acordados, utilização de padronizações e qualidade do código-fonte. Muitos conceitos estão presentes nas diferentes formas como um software pode ser testado. A respeito desses conceitos, avalie as afirmativas.

I – O teste de caixa branca é aquele em que o testador tem acesso à estrutura interna da aplicação. Há também o teste de caixa preta, no qual o testador desconhece o conteúdo interno da aplicação.

II – Os testes de caixa branca são conhecidos como testes comportamentais. São chamados assim porque têm o objetivo de validar saídas de acordo com as entradas, ou seja, garantir que os requisitos funcionais da aplicação são atendidos.

III – Os testes de caixa preta são conhecidos como testes estruturais. Eles têm o objetivo de verificar a lógica do programa, a cobertura de código e a qualidade do código-fonte em si.

É correto o que se afirma em:

A) I, apenas.

B) III, apenas.

C) I e II, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: Alternativa A.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: o teste de caixa branca é uma abordagem de teste na qual o testador tem conhecimento da estrutura interna do software, incluindo o código-fonte. Isso permite que o testador projete testes com base no conhecimento da lógica interna do programa. O teste de caixa preta, por sua vez, é um processo no qual o testador não tem conhecimento da estrutura interna do software e se concentra no seu comportamento funcional.

II – Afirmativa incorreta.

Justificativa: os testes de caixa preta, que se concentram no comportamento externo da aplicação, são chamados de testes comportamentais, não os de caixa branca.

III – Afirmativa incorreta.

Justificativa: os testes de caixa branca são conhecidos como testes estruturais, já que se concentram na análise da estrutura interna do código-fonte.