UNIP

UNIVERSIDADE PAULISTA

Engenharia de Software Ágil

????. ??

UNIP

UNIVERSIDADE PAULISTA

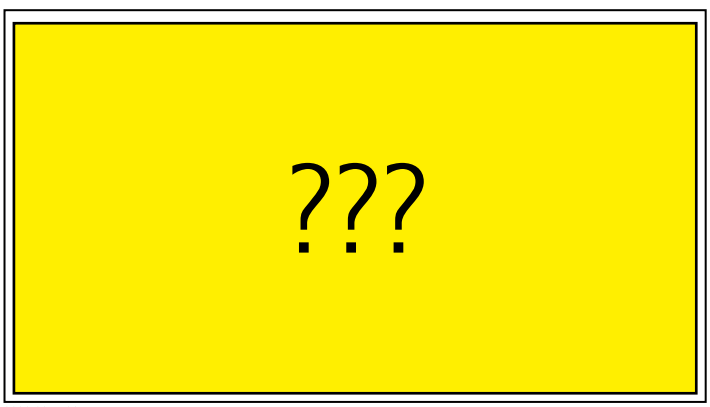
Engenharia de Software Ágil

Autor: Prof. Edson Quedas Moreno
Colaborador: Prof. Angel Antonio Gonzalez Martinez

Professor conteudista: Edson Quedas Moreno

Mestre em Engenharia de Produção pela Universidade Paulista (UNIP), com ênfase em Sistemas de Informação (2002), pós-graduado em Novas Tecnologias de Ensino-Aprendizagem pela UniÍtalo (2011) e graduado em Engenharia Eletrônica pela Faculdade de Engenharia Industrial (FEI – 1983). Desde 1989, é professor nos cursos de graduação e pós-graduação nas áreas de Negócios, Computação e Informação pela UNIP e pela UniÍtalo. Atuou por mais de 25 anos na área corporativa. De 2002 até 2021, foi diretor da empresa ASSERTI; de 2009 a 2012, foi sócio das empresas Consultoria EaD e ITGVBR. Também já trabalhou como analista de sistemas na Sisco e como programador na Digilab/Bradesco. Em 2006, projetou e efetivou o Projeto Farol Verde (educação para crianças carentes) e, em 2015, projetou um hackathon (maratona de programação de software por 30 horas contínuas com a participação de 200 alunos). Recebeu o Prêmio Excelência e Qualidade Brasil 2016 pela Braslider.

Dados Internacionais de Catalogação na Publicação (CIP)



????.?? – ??

Sumário

Engenharia de Software Ágil

APRESENTAÇÃO

A engenharia de software é uma disciplina que surgiu como uma 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 desta disciplina mostra a evolução da construção do produto ao longo das etapas do ciclo de vida do desenvolvimento de software. Na pesquisa de possíveis causas, caminhos e soluções, apresentam-se conceitos em seu contexto de aplicação, exemplos reais que vão contribuir para que você compreenda as técnicas de desenvolvimento e gerenciamento de projetos de software em empresas, os papéis das equipes de desenvolvimento em cada estágio do processo, as mudanças que afetam a prática da engenharia de software, tais como a implementação das metodologias, ferramentas e técnicas ágeis em sua construção.

Por se tratar de uma área de engenharia, a proposta desta disciplina é capacitar o aluno quanto ao conhecimento e às práticas profissionais ágeis do projeto e desenvolvimento de software, fornecendo uma base suficiente de aprendizado para iniciar na engenharia de software com a praticidade em metodologias ágeis.

Bons estudos!

INTRODUÇÃO

Os primeiros computadores eletrônicos surgiram durante a Segunda Guerra Mundial. Para ser mais exato, o marco da disponibilidade pública do computador eletrônico digital é associado ao ENIAC (Electronic Numerical Integrator and Computer), desenvolvido em 1945 e apresentado oficialmente em 1946. Esses computadores eram grandes e ocupavam grandes espaços, pesavam em torno de 30 toneladas, ocupando uma área de 167 m^2. Eles utilizavam válvulas eletrônicas para processamento, com capacidade e memória muito baixas, bem inferiores aos modelos atuais.

Desde então, o mundo presenciou um aumento exponencial da dependência e da integração do software em praticamente todas as áreas de conhecimento e da atividade humana, permitindo uma rápida evolução e criação de uma vasta gama de produtos, serviços e soluções inovadoras. Inicia-se aí a era da computação, quando a dualidade entre software e hardware permitiu ainda a criação de diversas tecnologias que transformam mercados e setores, substituindo até os modelos tradicionais, criando formas de interação e consumo.

Nos últimos 80 anos, a humanidade experimentou um progresso tecnológico sem precedentes, impulsionado pela evolução do software e sua integração em diversas áreas, como comunicação, saúde, educação, indústria, transporte, finanças, entretenimento, ciência e pesquisa. O avanço contínuo na tecnologia – que se estende até hoje – faz com que a produção de software enfrente desafios para atender plenamente à crescente demanda, diversificada e em constante evolução de usuários e consumidores.

Na década de 1970, presenciamos a invenção do microprocessador Intel 4004 (1971), o que possibilitou o surgimento dos primeiros computadores pessoais viáveis comercialmente, como o Altair 8800 (1975), com o microprocessador Intel 4004 e, depois, o Apple I (1976), com o microprocessador MOS Technology 6502. Esses sistemas de baixa capacidade em relação aos computadores atuais eram voltados para entusiastas e pequenos negócios. Esse é um marco histórico que permitiu a criação de máquinas menores e mais acessíveis, tornando a computação pessoal acessível além de grandes instituições e governos.

Os computadores passam a desempenhar um papel vital na automação do conhecimento, permitindo que informações sejam organizadas, processadas e acessadas de maneira mais rápida e precisa. Em meados dos anos 1990, a internet começa a se popularizar, trazendo consigo avanços como o e-mail, as ferramentas colaborativas com agendas de grupos online e a World Wide Web (WWW), que representaram um novo marco nos avanços mais significativos da história da humanidade. A capacidade de conectar pessoas e informações globalmente transformou a comunicação, o acesso ao conhecimento e a forma como os negócios eram conduzidos. A TI (Tecnologia da Informação) ganha novo impulso e conduz à criação de diversos sistemas de software, transformando a comunicação global e a forma de condução dos negócios.

No novo milênio, a rapidez tornou-se um fator crucial para o setor empresarial. Na gestão dos negócios, empresários precisaram de agilidade na escolha de estratégias, processos, transações comerciais, eficiência na cadeia logística de suprimentos e acesso instantâneo às informações. Nesse contexto, a engenharia de software alavancou as tecnologias na área industrial, fornecendo

ferramentas digitais para o controle e a automatização das atividades da produção. As redes sociais, os ambientes online e a inclusão digital permitem que indivíduos e comunidades divulguem novas ideias, participem de debates em eventos e promovam novas mudanças em escala global.

Este livro-texto é apresentado em quatro unidades e os seus principais aspectos envolvem:

- **Unidade I**: conceitua engenharia de software, define o software, identifica a dualidade entre as engenharias de software e a de sistemas. Nesse contexto, são abordadas as características, as aplicações, o emprego de metodologias ágeis no desenvolvimento, a implementação, o suporte e a manutenção do software. Depois, são acentuadas abordagens sobre a engenharia de requisitos e como ocorrem a viabilidade e o desenvolvimento do processo de requisitos do software, destacando técnicas ágeis de abstração da informação e estudos para análise.

- **Unidade II**: apresenta o emprego de metodologias ágeis nas atividades do processo de software, bem como modelos de processos individuais, de equipe e corporativos, que são colocados como técnicas eficazes e eficientes na disciplina de construção do software. Ainda destaca como alinhar e ligar os recursos do projeto, do software e dos stakeholders com uma seleção de artefatos que formam o arcabouço do processo de software da equipe de desenvolvimento.

- **Unidade III**: chega a hora de começar a desenvolver o software. Essa unidade mostra como associar os diversos recursos do software com o processo selecionado para percorrer o ciclo de vida do desenvolvimento de software. Na sequência, apresenta o RUP (Processo Unificado Racional), que, apesar de ter processo em seu nome, é uma metodologia de desenvolvimento de software baseada em melhores práticas para a engenharia de software.

- **Unidade IV**: aborda com exclusividade as metodologias ágeis aplicadas no desenvolvimento de software. Cada metodologia se integra com outras e também com modelos estruturados de desenvolvimento. Ela enfatiza a colaboração, a entrega incremental, a adaptabilidade e a resposta às mudanças dos requisitos do cliente. São acentuadas metodologias ágeis como XP (Extreme Programming – Programação Extrema), Scrum, FDD (Feature Driven Development – Desenvolvimento Dirigido a Funcionalidades), DSDM (Dynamic Systems Development Method – Método Dinâmico de Desenvolvimento de Sistemas), Crystal e AM (Agile Modeling – Modelo Ágil).

Tenha bons estudos.

Unidade I

1 ENGENHARIA DE SOFTWARE

Ao longo de sua história, a engenharia de software esteve sempre ligada à ciência, não só em relação à evolução dos computadores, mas servindo de apoio eficiente para a produção de conhecimento em praticamente todas as áreas atuantes da humanidade.

Com o surgimento dos computadores nos anos 1940, as linguagens de programação de alto nível ainda não existiam. A princípio, a programação acompanhava cada um dos hardwares, que eram computadores grandes, complexos e voltados para aplicações específicas. A programação se baseava em linguagem de máquina sem muitos métodos para desenvolver software: o programador era o próprio "escovador de bit".

Como as rotinas de programas de computadores se tornavam cada vez mais vinculadas aos elementos de hardware da arquitetura de Von Neumamm, ou seja, a dualidade entre um hardware específico e respectivo software em código de máquina, surgiu um elemento padronizado: para cada hardware específico existia um software específico, ou seja, a evolução de um necessita da evolução do outro.

Nos anos 1950 surgiram o sistema operacional UNIVAC OS e as primeiras linguagens de programação de alto nível, Fortran e Cobol. Já no fim dos anos 1960, a Otan propôs o termo engenharia de software como responsável pelo projeto e pela construção do software de computador.

Neste tópico, abordaremos os seguintes aspectos: definição de software, as características do software, suas aplicações e sua fábrica.

1.1 Definição de software

Ele é definido como um programa ou conjunto de programas de computador organizados intelectualmente para atender a um ou mais objetivos de um negócio.

A engenharia de software começou a se formar como uma disciplina nos anos 1960, quando o desenvolvimento de software passou a se tornar mais complexo em virtude de sua alta demanda e os métodos tradicionais de desenvolvimento se mostraram insuficientes para lidar com projetos de grande escala e alto nível de complexidade.

A qualidade e a produtividade estão intimamente ligadas e podem parecer conflitantes. No entanto, o desenvolvimento de software na busca por alta qualidade e alta produtividade é essencial para entregar produtos confiáveis, robustos, eficientes e que agreguem valor ao cliente.

Para atender a demanda de um determinado produto, a produção tem que acompanhar e, por vezes, até reduzir o custo da produção para que o produto continue competitivo no mercado.

Para aumentar a produção, deve-se elevar a quantidade de produto produzido. Para tal, deve-se melhorar a produtividade, o que implica, por exemplo, produzir atualmente o equivalente ao período anterior em uma quantidade superior.

A qualidade do software especificado como requisito não funcional determina normas, padrões e regras que estruturam a arquitetura do sistema (software) no ambiente em que será operado. Já a produtividade de software mede a eficiência com que a equipe de desenvolvimento transforma requisitos funcionais em funcionalidades entregues.

Algumas empresas já se adaptaram a essa situação, no entanto, para a grande maioria, o aumento da produção não é acompanhado pelos requisitos da qualidade. O desafio é elevar a produção em menor tempo a custos baixos. Para tal, algumas entidades abrem mão de alguns dos critérios da qualidade, como suporte, inspeção, manutenção, testes e diagnósticos. Consequentemente, o número de defeitos aumenta, encarecendo a logística reversa para suporte e manutenção e elevando o custo do produto.

Existe um equilíbrio natural entre demanda e produção. A primeira promove o produto e as vendas, e a segunda visa atender à demanda de forma exata, sem desperdícios e a um custo baixo: esse é o desafio do fornecimento.

1.2 Características do software

Qualidade de software refere-se à avaliação de um atributo próprio de um determinado componente de software para cumprir seus requisitos, livre de defeitos, com eficiência e entrega de valor ao usuário final.

De acordo com Pressman (2016), em 1977, McCall, Richards e Walters criaram uma proposta de categorização dos fatores que afetam a qualidade de software (figura 1). Esse modelo passou a ser um modelo clássico para definir os caminhos da qualidade do software.

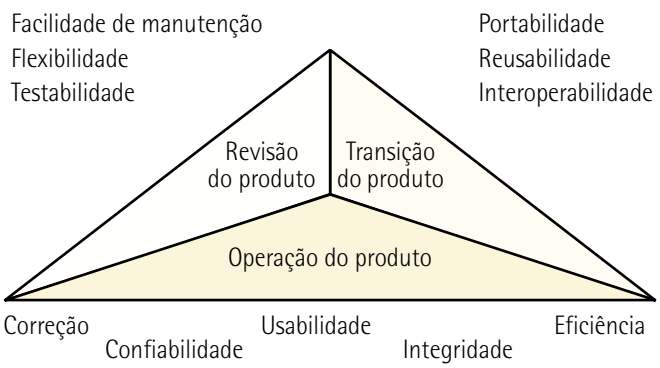


Figura 1 – Fatores de qualidade de software de McCall

Adaptada de: Pressman (2016).

A qualidade desejada se refere à definição dos RNF (Requisitos Não Funcionais) do software e que são determinados na construção do software.

Alguns dos principais RNF a serem avaliados são determinados pelos seguintes fatores:

- **Funcionalidade**: o software deve atender a todas as especificações referentes às funções requeridas em RF (Requisitos Funcionais). Uma funcionalidade é identificada quando possui um agrupamento de funções similares desenvolvidas com o emprego da mesma tecnologia.

- **Confiabilidade**: o software deve ser estável e, ponderando um marco de referência, apresentar o menor índice de falhas.

- **Usabilidade**: envolve a facilidade de uso, a compreensão, a atratividade e o aprendizado pelo software. A usabilidade é o principal critério a ser avaliado na IHC (Interface Humano-Computador).

- **Segurança**: engloba práticas e medidas com o objetivo de proteger o software contra ameaças, ataques e vulnerabilidades.

- **Eficiência**: refere-se ao desempenho das funcionalidades do software; além de fornecer resultados corretos, ela pode ser melhorada, por exemplo, ofertando os mesmos resultados "corretos" com redução de tempo e/ou custo.

- **Manutenibilidade**: representa o quão fácil é implementar mudanças para atualizações, correções e prevenção de falhas.

- **Portabilidade**: o quanto um determinado software é portável para outros ambientes operacionais.

Algumas práticas eficazes de desenvolvimento de software podem melhorar a relação entre qualidade e produtividade, como as elencadas a seguir:

- **Automação de testes**: reduz o tempo de testes manuais e aumenta a cobertura de testes, melhorando a qualidade sem sacrificar a produtividade.

- **Desenvolvimento ágil**: promove entregas frequentes e iterativas, permitindo ajustes rápidos e contínuos que mantêm a qualidade alta e a produtividade constante.

- **Reúso de código**: a reusabilidade é o principal fator de qualidade na engenharia de software. Componentes reutilizáveis podem acelerar o desenvolvimento e reduzir a probabilidade de erros, melhorando ambos os aspectos.

Nesse contexto, destacam-se também alguns processos e métodos para a garantia de qualidade do software:

- **Verificação e validação (V&V)**: trata-se de testes de software que incluem testes unitários, de integração, de sistema e de aceitação.

- **Revisões de código**: envolvem revisões formais ou informais, auditorias, walkthroughs e inspeções para identificar problemas precocemente.

- **Controle de versão**: engloba ferramentas e técnicas de versionamento que acompanham o desenvolvimento do software e entregas aos clientes. Ferramentas como Git (sistema de controle de versões distribuído) ajudam a gerenciar mudanças e colaborar de forma eficaz.

- **Documentação**: compreende manuais de usuário e registro técnico detalhado, de preferência em formato eletrônico.

- **Gestão do risco**: trata-se de definir regras e procedimentos que assegurem a qualidade do software, mesmo apresentando problemas. Ao mesmo tempo, deve-se definir um plano de contingência para mitigar riscos e, possivelmente, garantir a produtividade.

1.2.1 Dualidade do software com o hardware

Ela permite a construção de sistemas computacionais e o que realmente justifica a função do computador é o uso do software.

Atenção ao exposto: não é possível que um editor de textos funcione sem o apoio de um especialista do negócio, no caso editores de textos, e que irá determinar as funções e as regras desse negócio. Com essas ferramentas, o profissional irá especificar, projetar e desenvolver o software, conseguindo automatizar um negócio pela engenharia de software.

O software é desenvolvido ou passa por um processo de engenharia, não é manufaturado. Na manufatura, as mudanças necessárias a serem feitas 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, p. 5).

Na figura 2, o gráfico se refere à curva de falhas do hardware, a qual expressa o ciclo de vida de um determinado hardware. Representada pelos eixos Índice de falhas X Tempo, acentua a ocorrência do desgaste do hardware ao longo do tempo.

O gráfico mostra um produto que é lançado no mercado ou adaptado a outros produtos. Na região de mortalidade infantil (termo metafórico que representa um conceito biológico), vemos que o índice de falhas é alto e declina com o tempo, ou seja, quando um produto é lançado, a chance de acentuar defeitos é alta.

Em virtude do exposto, a indústria e o comércio dão revisões técnicas, garantia ou seguro do produto ou do serviço prestado.

Com o passar do tempo, após revisões e manutenções, as falhas são reduzidas, até chegarem à região de estabilidade (figura 2) e com baixo índice de falhas, o que representa a melhor faixa de uso do produto.

A vida útil de um produto é a soma entre os tempos das regiões da mortalidade infantil e da estabilidade. Essa faixa é usada pelas seguradoras para a customização do seguro do produto.

Como o hardware fica sujeito ao desgaste (figura 2), isso faz com que o produto saia da região de estabilidade, o que provoca, ao longo do tempo, a elevação do índice de falhas e, consequentemente, o aumento do custo de manutenção devido a consertos ou substituição de peças. As manutenções sucessivas crescem e, assim, o custo para manter o produto acaba por inviabilizá-lo: ou se faz um novo projeto ou se descarta o produto.

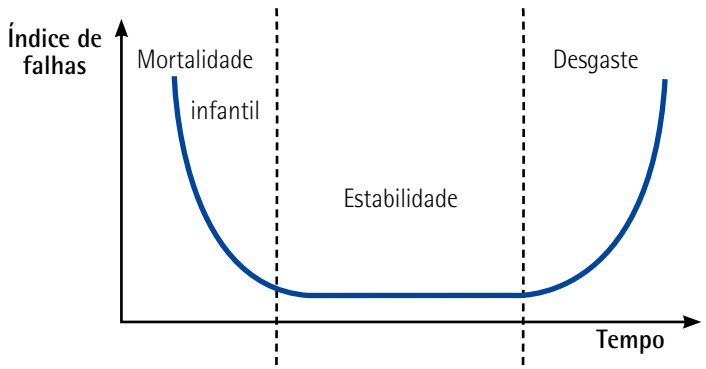


Figura 2 – Curva de falhas do hardware

Adaptada de: Pressman (2021).

Na figura 3, o gráfico se refere à curva de falhas do software. Ela representa o ciclo de vida de um determinado software e é similar à curva de falhas do hardware, exceto na região final (deterioração), que mostra a degradação do software.

A região de mortalidade infantil indica a implantação de um novo software ou de uma nova estrutura. Quando isso ocorre, é provável que o usuário identifique algumas falhas, tais como adaptação ao ambiente operacional, falta de alguma biblioteca de instalação, falhas com a interface de hardware, enfim, diversas situações que são corrigidas até que o software atinja sua estabilidade.

Ao término da região de estabilidade, o software passará por modificações (manutenção e/ou mudanças). À medida que elas são implementadas, é provável que novos defeitos sejam introduzidos, como acentuado na curva dos "dentes de serra", levando o software à deterioração.

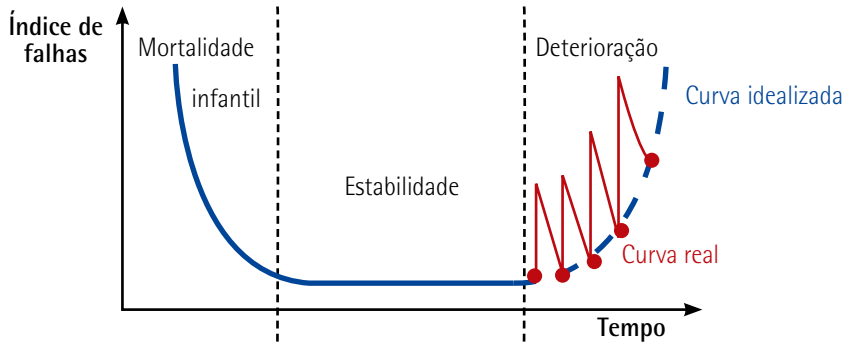


Figura 3 – Curva de falhas do software com "dentre de serra" com pontos de mudanças no software

Adaptada de: Pressman (2021).

Exemplo de aplicação

Considere o seguinte cenário: no módulo financeiro de um software ERP (Enterprise Resource Planning – Planejamento dos Recursos Empresariais), os operadores requisitaram mudanças nos registros de notas fiscais.

Após implementações e testes de novos campos e dados, passaram a observar a seguinte situação: os usuários passaram a reclamar da falta de alguns registros, alegaram a necessidade de reajustar alguns tamanhos de campos do registro. Então, observou-se uma pequena queda de desempenho no acesso aos dados.

Nessa situação, começa-se a observar a deterioração do software após a execução de novas mudanças, como: inserção de novos códigos, que passaram a exigir mais processamento, aumento da busca de pacotes pela rede, elevação de dados e inserção de novos ponteiros.

Tarefa: rever melhorias no algoritmo de busca dos dados, verificar exclusão de ponteiros no código-fonte e monitorar o desempenho do software para melhorar o sistema.

1.2.2 Como surgem versões, releases e novas estruturas?

A estrutura do software corresponde à sua arquitetura, que pode ser nova ou ocasionada pelo implemento de novas adaptações às já existentes.

A nova arquitetura parte da criação do projeto. Assim, são inseridas a funcionalidade e as interfaces do software, acompanhadas das formas pelas quais as normas e os padrões da qualidade foram determinados.

As implementações no projeto correspondem às mudanças ao longo do ciclo de vida do desenvolvimento do software. Elas são provocadas a pedido do cliente ou por algum 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 figura 4 acentua a região de deterioração do gráfico da figura 3. A cada mudança implementada surge um pico de falhas, que leva novamente a processos revisionais e de manutenção. Nessas mudanças, existe a possibilidade de acrescentar novos códigos e criar redundâncias de programas e de dados.

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

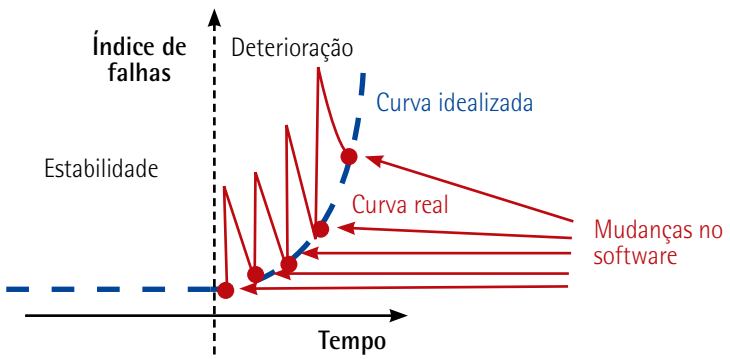


Figura 4 – Região de deterioração na curva de falhas do software, provocada pelas mudanças

Adaptada de: Pressman (2011).

Novas versões e releases são produzidos com base no projeto inicial e, com o passar do tempo, novas mudanças requisitadas pelo cliente são executadas. Assim, o hardware torna-se obsoleto, ocorrem atualizações do sistema operacional, mudanças de topologias, novas versões da linguagem de programação, enfim, diversas variações que contribuem para a queda de eficácia e eficiência do software.

Esses problemas acarretam o aumento dos custos de manutenção, que, por sua vez, inviabilizam o uso do software. Nessa situação, logicamente, antes de descartá-lo, deve-se atualizá-lo, o que implica reestruturar a arquitetura do software, o que significa:

- fazer limpeza de códigos redundantes e dos dados;
- implementar melhorias e novos algoritmos para funções, interfaces e ganhos no desempenho;
- atualizar o hardware, o sistema operacional e as linguagens de programação;
- adaptar as antigas e novas funcionalidades com base na nova arquitetura.

1.3 Aplicabilidade do software

Ela é ampla e diversificada. Atualmente, o software abrange boa parte dos aspectos da vida moderna e se utiliza das várias áreas do conhecimento para sua construção.

São observadas amplas pesquisas de software e algumas delas são bem demoradas, complexas e ficam aquém de seu objetivo operacional, tornando-se obsoletas mesmo antes de seu lançamento. "Quando é introduzida uma tecnologia bem-sucedida, o conceito inicial transforma-se em 'ciclo de inovação', razoavelmente previsível" (Pressman, 2016, p. 840).

Com esse ciclo de inovação, as demandas por software e por sua melhoria apontam algumas tendências na engenharia de software, acompanhadas de diversas outras áreas do conhecimento e isso se faz em constante evolução.

Quanto à aplicabilidade e em uma escala aproximada de aceitação das tecnologias no mercado de software, da mais amplamente adotada a mais emergente, as tecnologias acentuadas a seguir apontam tendências notáveis na engenharia de software.

Aplicativo para microcomputadores

Envolve processamento de textos, planilhas eletrônicas, computação gráfica, diversões, gerenciamento de banco de dados etc. De fato, o software de computador pessoal continua a representar os mais inovadores projetos de interfaces com seres humanos de toda a indústria de software. Nessa categoria, cita-se a criação de utilitários, que servem para manutenção do software.

Aplicações móveis

A proliferação de dispositivos móveis permitiu a criação de ferramentas indispensáveis para empresas e indivíduos em suas atividades diárias, proporcionando um desenvolvimento intensivo de aplicativos móveis.

Hoje, o mercado exige soluções eficientes, com interfaces de usuário intuitivas e funcionalidades robustas, em aplicativos que abrangem um grande número de setores, como:

- e-commerce;
- jornalismo e notícias;
- saúde e bem-estar, redes sociais e comunicação;
- entretenimento;

- serviços financeiros;
- transporte e mobilidade;
- educação e produção.

A popularização dos smartphones e tecnologias como 5G tem projetado um crescimento de aplicativos móveis para os próximos anos, com soluções digitais inovadoras, como a inteligência artificial (IA) e a realidade aumentada (RA).

Computação em nuvem (cloud computing)

Atualmente, é a parte básica para a infraestrutura de TI, apoiando sistemas de software de todos os tamanhos.

Os serviços oferecidos reúnem diversos recursos de processamento, aplicações e armazenamento em servidores, que podem ser acessados de qualquer lugar do mundo pela internet. A computação em nuvem, além de permitir que aplicativos e serviços sejam entregues pela internet, proporciona escalabilidade, flexibilidade e acessibilidade a partir de dispositivos variados.

Ela tem sido adotada por diversas empresas e alguns desses sistemas de software são disponibilizados ao público para uso geral. No Brasil, destacam-se o Google Drive e o Outlook Drive, da Microsoft:

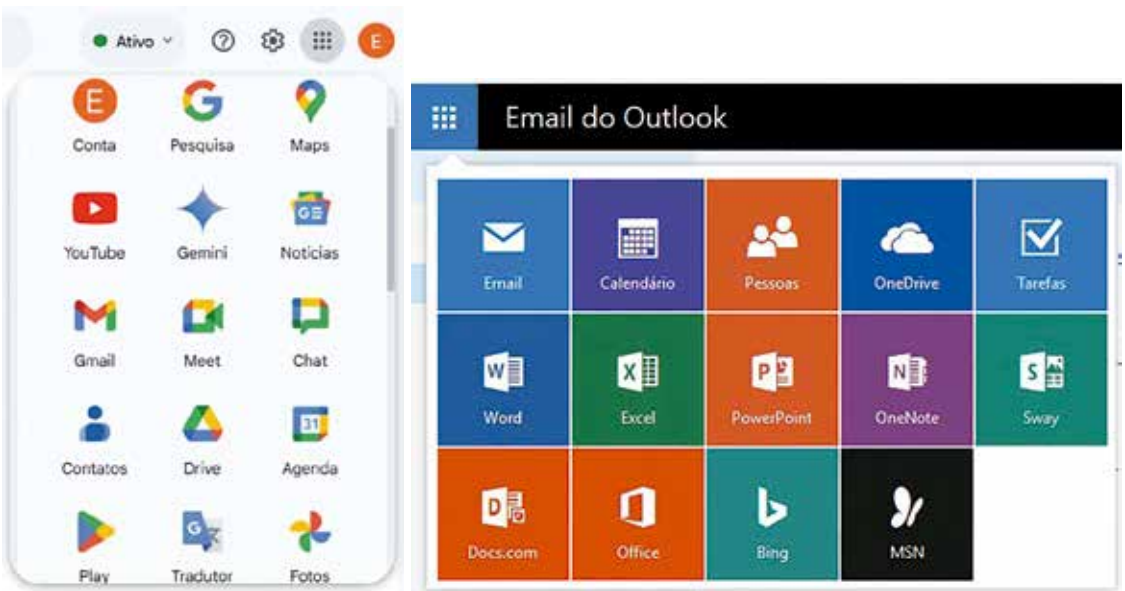


Figura 5 – Telas de software para computação em nuvem de uso público:
Google Drive (à esquerda) e Outlook Drive (à direita)

Software básico

É uma coleção de programas escritos para apoiar outros programas; normalmente, esse software tem uma forte interação com o hardware.

Em razão dessa interação, ele é chamado de firmware, usado para a codificação de drivers. É utilizado como algoritmo para:

- interface do hardware;
- processadores de telecomunicação;
- componentes do sistema operacional;
- computadores com intenso uso de múltiplos usuários;
- operações concorrentes que exigem escalonamento (schedule), compartilhamento de recursos, estrutura de dados complexas e múltiplas interfaces.

Observação

Para a construção do software básico, usa-se a linguagem ASSEMBLER (linguagem de máquina) ou, em casos específicos, a linguagem C.

Esse tipo de software pode estar embutido, conhecido popularmente como software embarcado (embedded software), ou seja, fica em memória ROM e é usado para controlar produtos e sistemas para mercados industriais e de consumo. Pode ser usado para controlar também funções muito limitadas, tais como controle de um forno de micro-ondas e funções digitais para automóveis.

Ele é útil também para construir aplicações de tempo real (real time). Nesse caso, de um software que monitora e analisa eventos do mundo real, deve haver um controle/saída que responde ao ambiente externo e um componente de monitoração que coordena todos os demais elementos para obter resposta em tempo real. Tipicamente, varia de 1 milissegundo até 1 minuto para ser mantida. O termo tempo real difere do interativo ou do time-sharing, que podem exceder o tempo de resposta sem resultados desastrosos.

E-commerce (comércio eletrônico)

É uma das áreas mais desenvolvidas e com maior aceitação, especialmente após a digitalização massiva das vendas.

Essa área pode apresentar as seguintes classificações:

- **B2B (Business to Business)**: específico para transações entre empresas, o que envolve um ciclo de vendas mais longo, a integração com sistemas CRM (Customer Relationship Management – Gerenciamento das Relações com o Cliente), um grande volume de pedidos, os contratos personalizados e a integração com sistemas ERP.

- **B2C (Business to Consumer)**: são as lojas virtuais (Magazine Luiza, Submarino, Amazon e outros) e seu foco é o consumidor final. Assim, suas características de usabilidade, como preferências e comportamento dos consumidores individuais, envolvem transações de alta frequência e baixo valor, UX (User Experience – Experiência do Usuário), marketing, logística e diversidade de produtos.

- **C2C (Consumer to Consumer)**: o principal aspecto aqui é a interação direta entre consumidores para a compra e venda de produtos. Os principais processos são: utilização de plataformas de mercado (por exemplo: Mercado Livre, OLX, leilões virtuais e outros), diversidade de produtos, monitoramento das transações comerciais e feedbacks, flexibilidade e autonomia para o controle das transações.

Observação

Existem várias outras siglas tecnológicas associadas ao e-commerce. No entanto, as tecnologias citadas (B2B, B2C e C2C) fornecem base para o desenvolvimento das demais. Aspectos de infraestrutura de TI que dão suporte a alguns dos sistemas aqui citados serão abordados ao longo deste livro-texto.

Segurança cibernética

Ela se tornou uma prioridade global diante do cenário crescente e sofisticado das ameaças digitais. O desenvolvimento de software seguro passou a incorporar técnicas avançadas de criptografia, métodos de autenticação de um usuário, um dispositivo ou sistema. O intuito é garantir que operações online e acesso a recursos de dados sejam seguros e confiáveis.

A segurança cibernética evolui constantemente para proteger sistemas computacionais, garantindo a integridade, a confidencialidade e a disponibilidade das informações digitais. Empresas e usuários adotam soluções como monitoramento contínuo, IA para detecção de ameaças e testes de penetração, reduzindo vulnerabilidades.

Além disso, a conformidade com regulamentações como LGPD (Lei Geral de Proteção de Dados), no Brasil, e GDPR (Regulamento Geral sobre a Proteção de Dados), na Europa, fortalece a proteção de dados, exigindo práticas rigorosas de segurança.

Desenvolvimento ágil e DevOps

São métodos amplamente aceitos no desenvolvimento de software, promovendo maior desempenho e qualidade na realização dos processos. A abordagem ágil permite um desenvolvimento mais iterativo e colaborativo, enquanto o DevOps integra desenvolvimento e operações para acelerar a entrega e melhorar a qualidade do software.

Análise de dados e inteligência de negócios

Trata-se de uma área crucial para a tomada de decisões empresariais. A capacidade de coletar e analisar dados em tempo real ajuda as empresas a tomar decisões informadas e prever tendências futuras.

Nesse contexto, destaca-se o BI (Business Intelligence – Inteligência de Negócios), que normalmente está integrado ao Big Data, ferramenta que permite a análise em tempo real para a análise de dados em larga escala.

O BI é um exemplo avançado de sistema integrado de gestão, tendo em vista que é composto de várias ferramentas que combinam diversos tipos de dados para análise, comparação e simulação de resultados que levam diretamente à tomada de decisão. O BI proporciona ao gestor ferramentas para extrair dados de forma ágil e confiável, fornecendo uma visão global do andamento dos negócios da empresa.

Exemplo de aplicação

Tem-se a seguinte situação-problema: uma loja virtual que trabalha com tecnologia B2C está com dificuldades para melhorar sua estratégia de mercado. Ela tem uma grande quantidade de dados de diversas fontes, originárias das transações de negócios, feedbacks de clientes e redes sociais. Contudo, esses dados estão dispersos e a empresa não consegue extrair insights consistentes para a tomada de decisão.

Solução

Implementar no software o módulo de sistema BI para concentrar todos os dados e permitir em uma única plataforma integrada uma análise mais eficiente. Dessa forma, estariam inclusos os processos de: coleta e integração de dados, análise, segmentação de clientes, automação de marketing dos produtos, monitoramento e ajustes automáticos em tempo real.

IA e aprendizado de máquina

Em rápida adoção pelo mercado consumidor, com aplicações que vão desde chatbots até a análise de dados que usa técnicas estatísticas, a IA usa algoritmos não numéricos para resolver problemas complexos que não sejam favoráveis à computação ou à análise direta (Pressman, 2011).

O ML (Machine Learning – Aprendizado de Máquina) é uma ramificação da IA, cujo foco é o desenvolvimento de algoritmos e técnicas que permitem aos computadores aprender com grandes volumes de dados, analisar para identificar padrões, tomar decisões ou fazer previsões com base nesses padrões.

A IA e o ML estão transformando a maneira como o software é desenvolvido e usado. Algoritmos de IA são incorporados em aplicativos para melhorar a automação, a análise de dados e a interação com os usuários.

Por exemplo, na IA, a área mais ativa é a dos sistemas especialistas, também chamados de sistemas baseados em conhecimento. Eles resolvem problemas capturando o conhecimento para um domínio muito específico e limitado da perícia.

IoT (Internet of Things – Internet das Coisas)

Com ampla aplicação na automação residencial e industrial, o controle e a interconexão pela internet de dispositivos inteligentes estão impulsionando a demanda por software, de forma a coletar, analisar e atuar sobre dados gerados por esses dispositivos em setores como casas inteligentes, manufatura e cidades inteligentes.

Blockchain

Essa tecnologia continua a expandir e tem se consolidado nos setores financeiro e logístico, oferecendo segurança e rastreabilidade aprimoradas para transações e processos, garantindo transparência e reforçando a segurança de dados ou objetos que não podem ser alterados após sua criação.

Com uma abordagem descentralizada e segura, o blockchain continua a transformar diversos setores financeiros, como criptomoedas, ativos digitais e serviços financeiros peer-to-peer sem necessidade de intermediários, contratos inteligentes, sistemas de pagamento, identidade digital e outros.

Na logística, essa ferramenta oferece melhor rastreabilidade em tempo real, transparência na cadeia de suprimentos, redução de fraudes e falsificações, melhoria na gestão de estoques, processos alfandegários e segurança alimentar. Sem dúvida, o blockchain é uma tecnologia revolucionária que está expandindo suas aplicações em votação eletrônica, direitos autorais, IoT e muito mais.

Interfaces de usuário avançadas

Cada vez mais difundidas, interfaces de usuário como VR (Virtual Reality – Realidade Virtual), AR (Augmented Reality – Realidade Aumentada) e interfaces de voz estão se tornando mais comuns, proporcionando interações mais imersivas e naturais.

Diferentemente da VR, a RA permite integrar informações digitais da VR ao ambiente real. Com isso, é possível ter a percepção do mundo real em 3D e acesso a recursos característicos dessa interface.

Outras tecnologias servem de apoio na construção de interfaces 3D, como a estereofotogrametria, que permite escanear um corpo externo em movimento e depois renderizá-lo. Já o OptiTrack é um sistema de captura de movimentos que rastreia pontos brancos em tempo real e registra a posição e o movimento dos pontos no espaço tridimensional. Por sua vez, o scanner 3D possibilita a criação de um modelo digital detalhado. Nesse contexto, inclui-se também um SGBD, que combina um modelo 3D com os deslocamentos de pontos digitalizados, replicando com precisão no ambiente virtual, como o Datamine, que não deve ser confundido com data mining. Apesar de ambos serem utilizados na análise de dados, eles têm conceitos e aplicações diferentes, e a data mining pode ser uma etapa do processo de uso do Datamine.

Saiba mais

Na reportagem indicada a seguir, é possível, por exemplo, observar arquiteturas de prédios residenciais e em seu interior também. Em uma única imagem pode-se analisar uma peça de motor virtual integrada a um motor real, visualizar elementos geológicos em todas suas dimensões, incluindo os que estão abaixo da linha de visão, auxiliar na navegação em todos os sentidos e suas formas, ampliar órgãos humanos, de animais, insetos, micro e nano-organismos, plantas e objetos em geral.

5 APLICAÇÕES de realidade aumentada nos SIG. *INFO Portugal*, 30 out. 2020. Disponível em: https://shre.ink/egGU. Acesso em: 23 jun. 2025.

Sustentabilidade e ética

As preocupações ambientais e éticas estão moldando a forma como o software é desenvolvido, com uma ênfase crescente em práticas de programação responsável e sustentável.

Na sustentabilidade, destacam-se os seguintes aspectos:

- **Energia**: desenvolvimento de software que consuma menos recursos computacionais, uso de provedores de acesso que utilizem energia renovável e extensão da vida útil do hardware.

- **Ambiente e obsolescência**: redução do nível de obsolescência do software, de forma que este continue em operação por mais tempo e evite descartes prematuros.

Na ética, acentuam-se aspectos como:

- privacidade e segurança;
- inclusão e acessibilidade;
- responsabilidade social;
- transparência algorítmica.

E-business

Conhecido também como software empresarial, é o de maior área de aplicação. O e-business por vezes incorpora o e-commerce e sistemas de software que processam folhas de pagamentos, contas a pagar/receber, controle de estoques, transportes e demais funções empresariais.

Os tipos mais comuns de sistemas de informação estão na categoria do e-business (negócios eletrônicos). São eles:

- **ERP**: responsável por integrar diversos processos empresariais, é embasado na arquitetura servidor/cliente ou em uma infraestrutura em nuvem. Pode ser aplicado em manufatura, varejo, saúde, finanças, serviços e educação.

- **CRM**: envolve ferramentas usadas para gerenciar as interações de uma empresa com seus clientes. O CRM é integrado ao ERP e uma das aplicações de maior abrangência que fazem uso do CRM está no call center com soluções VoIP (telefonia por IP), headsets, IVR (Interactive Voice Response – Resposta Interativa por Voz) e ACD (Automatic Call Distribution – Distribuição Automática de Chamadas).

- **SCM (Supply Chain Management – Gerenciamento da Cadeia de Suprimentos)**: específico para o controle de processos logísticos, tem ferramentas como scanner de código de barras, RFID e dispositivos IoT; gera inventários, responde ao processamento do pedido, monitora o roteamento do transporte nos armazéns e nos centros de distribuição.

Devido à ligação dos servidores locais com outros servidores de diferentes plataformas e ambientes operacionais distintos, a infraestrutura de TI se baseia na implementação de sistemas distribuídos, intérpretes de objetos e dados entre diferentes ambientes operacionais. Sua infraestrutura pode também estar embasada na nuvem.

Essas tecnologias processam grandes volumes de dados, algo às vezes necessário para integrar outra tecnologia do e-business, que é o Big Data. Esse dispositivo trata de grandes volumes de dados, difíceis de serem analisados e processados, usando métodos tradicionais de processamento de dados. Eles podem ser originados de diversas fontes, incluindo redes sociais, transações financeiras, logística, sensores da IoT e outras mais.

A integração do Big Data com o BI ocorre do seguinte modo: o Big Data responde pela captação dos dados, enquanto o BI faz o tratamento desses dados para gerar relatórios e dashboards para visualização e análise.

1.4 Fábrica de software

Inspirada nas linhas de produção industrial, a fábrica de software é um modelo de produção organizado e eficiente para o desenvolvimento de software. Utilizam-se processos padronizados, metodologias, ferramentas automatizadas e equipes especializadas.

Nesse contexto, são adotadas várias abordagens para mitigar problemas associados à crise do software, o que estudaremos a seguir.

1.4.1 Crise do software

O primeiro autor a utilizar o termo "crise do software" foi Dijkstra (1972).

> Nos primórdios da computação o programador competente tinha uma mente voltada para quebra-cabeças e gostava muito de truques inteligentes e acreditava-se que a programação nada mais era do que otimizar a eficiência do processo computacional. [...] Mas, em vez de nos encontrarmos no estado de felicidade eterna de todos os problemas de programação resolvidos, nos encontramos até o pescoço na crise do software! (Dijkstra, 1972, p. 2).

Pressman (2002) dizia que o software era uma crise no horizonte. O autor define crise como um ponto decisivo no curso de algo. O termo se refere a um conjunto de problemas que são encontrados no desenvolvimento de software de computador.

Os problemas considerados naquela época são os mesmos da atualidade: o rápido crescimento de demanda por software, com toda a complexidade em seu desenvolvimento, e a inexistência da engenharia de software. Não havia técnicas estabelecidas para desenvolver sistemas que funcionassem adequadamente ou que pudessem ser validadas.

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 acompanha a demanda por seus serviços;
- a qualidade de software às vezes não é adequada.

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

- **Falta de dados**: não são dedicados tempo e recursos suficientes para coletar dados sobre o andamento dos projetos. Sem indicadores sólidos de desempenho, não podemos avaliar com precisão a eficácia de novas metodologias, ferramentas ou práticas de gestão.

- **Insatisfação dos stakeholders**: a insatisfação dos stakeholders com o projeto "concluído" ocorre frequentemente. A comunicação entre os stakeholders e a equipe de gestão de projetos muitas vezes é inadequada.

- **Qualidade do projeto**: a qualidade do software entregue frequentemente é questionável. Apenas agora começam a surgir conceitos quantitativos sólidos de confiabilidade e garantia de qualidade na gestão do projeto de software.

1.4.2 Metodologias ágeis em resposta à crise do software

Para solucionar essa crise, foram criadas as metodologias ágeis, que têm um impacto significativo na melhoria da gestão de projetos de software, como os acentuados a seguir:

- Ciclos curtos de desenvolvimento, permitindo a coleta regular de dados do projeto, feedbacks contínuos e ajustes rápidos nas práticas de desenvolvimento do software.

- A comunicação entre cliente e desenvolvedor é contínua e transparente; são praticadas revisões regulares, de forma que os requisitos sejam claros e ajustados de acordo com a necessidade.

- A integração contínua permite testes e diagnósticos antecipados, melhorando a qualidade do software; com entregas frequentes de versões incrementais, promovem-se resultados melhores e alinhados às expectativas dos stakeholders.

- As metodologias ágeis alinhadas a um planejamento iterativo e incremental permitem mudanças rápidas e adaptativas das prioridades e dos requisitos.

- As metodologias ágeis incentivam equipes à autodisciplina e ao empoderadamento, o que aumenta a motivação e o compromisso dos stakeholders. As histórias de cada sprint permitem que a equipe desenvolvedora identifique melhorias e aprendizagem contínua.

2 ENGENHARIA DE REQUISITOS

> Entender os requisitos de um problema está entre as tarefas mais difíceis enfrentadas por um engenheiro de software. Quando se começa a pensar sobre isso, a engenharia de requisitos não parece 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? E mesmo que clientes e usuários finais sejam explícitos quanto às necessidades, essas vão se modificar ao longo do projeto (Pressman, 2016, p. 131).

2.1 Conceito de processo da engenharia de requisitos

O processo de engenharia de requisitos é composto por um conjunto estruturado de atividades para identificar, documentar e gerenciar os requisitos de um software, de forma a garantir que o produto final atenda às necessidades e expectativas dos usuários e dos stakeholders.

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

- **Estudo da viabilidade do sistema**: embora essa etapa não esteja inclusa no projeto do software, é obrigatória para iniciar o projeto.
- **Elicitação**: engloba a identificação das entidades envolvidas no projeto, a abstração de conhecimentos do negócio, as necessidades ou alguma situação-problema que será convertida em software com base nos requisitos do usuário e RNF.
- **Análise**: é a interpretação do negócio junto ao cliente com revisões sucessivas por meio de modelos do negócio para elaborar o escopo do software e medir a necessidade do negócio em relação à capacidade e aos recursos para que os profissionais desenvolvam o software. Nessa fase, inicia-se a elaboração dos requisitos do software.
- **Especificação e modelagem**: corresponde a uma descrição detalhada dos requisitos funcionais e do sistema, ao lado da modelagem do software, para a interpretação dos desenvolvedores.
- **Validação**: verifica-se se os requisitos documentados estão de acordo com as necessidades do cliente.
- **Gestão de requisitos**: monitoramento dos requisitos ao longo do ciclo de vida do desenvolvimento do software, de forma a garantir que os requisitos estão sendo executados corretamente.

A figura 6 mostra o fluxo de atividades do processo da engenharia de requisitos do software. Observe que o desenvolvimento do processo de engenharia de requisitos só ocorre após a validação deles. Assim, caso eles não sejam validados, ocorrerá uma iteração, o que significa que todo o processo será revisado, incluindo novas abordagens para o levantamento dos requisitos.

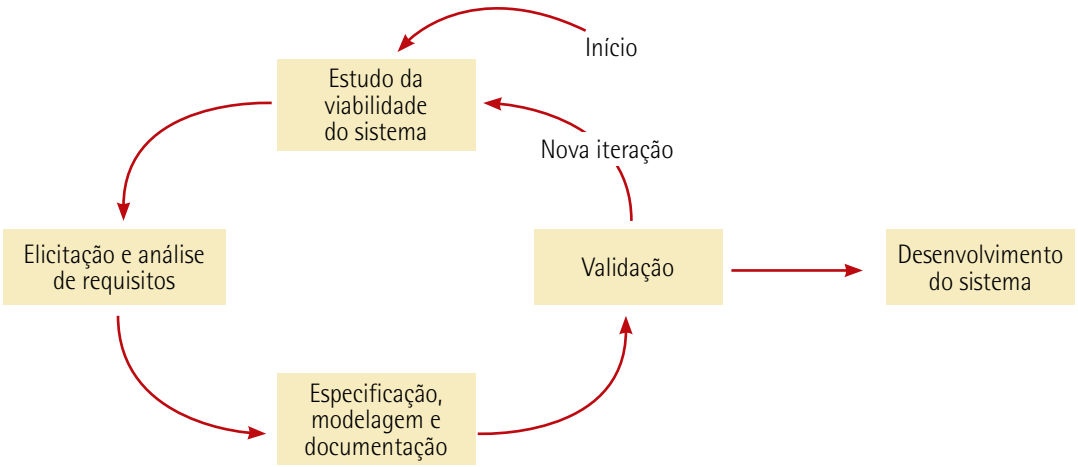


Figura 6 – Modelo do processo da engenharia de requisitos do software/sistema

2.2 Estudo de viabilidade do sistema

Na identificação (ou concepção) das necessidades do software, o projeto começa quando uma necessidade de negócio é identificada ou um mercado ou serviço novo é descoberto.

Os desenvolvedores fazem uma série de perguntas para estabelecer um entendimento básico do problema e para que o projeto do software tenha uma boa probabilidade de sucesso. Mesmo antes da alocação de recursos e do projeto propriamente dito, deve-se analisar a viabilidade do software.

Para todo o software novo, o processo de engenharia de requisitos deve começar com o estudo de viabilidade, antes mesmo da obtenção e da análise de requisitos. Os resultados do estudo de viabilidade devem apresentar um relatório que recomenda se compensa ou não realizar o processo de engenharia de requisitos e o processo de desenvolvimento do sistema.

O estudo de viabilidade busca alinhar as necessidades do negócio com a capacidade para o desenvolvimento. É um estudo breve, direcionado e que se destina a responder algumas perguntas, como:

- O sistema proposto contribui para os objetivos gerais da organização?
- O sistema pode ser integrado com os outros já em operação? Qual é o impacto das mudanças no ambiente operacional do usuário?
- O sistema pode ser implementado com a utilização de tecnologia atual dentro das restrições de custo e prazo?
- O que o cliente quer está dentro do orçamento e do prazo para entrega do sistema?

2.3 Processos de requisitos do software

No ciclo de vida do desenvolvimento, o processo inicial se dá pela elicitação. É a fase de concepção do projeto, que inclui as atividades de coletar e descobrir informações sobre o negócio junto aos stakeholders (partes interessadas). O objetivo é entender perfis de usuários, necessidades, expectativas e restrições do software. A tarefa básica na elicitação é comunicar-se com os usuários e clientes para determinar quais são os requisitos.

O analista deve ter habilidade e sutileza para extrair informações durante uma conversa e também especificações de uso. Para que haja uma boa compreensão das partes interessadas, em reuniões, o analista deve apresentar especificações e modelos de seu ponto de vista para validar o conhecimento obtido sobre o negócio.

Técnicas comuns empregadas na elicitação:

- Organizar reuniões, questionários, brainstorm, entrevistas, grupos focais (workshops) e/ou pesquisas e, consequentemente, a criação da lista de requisitos.

- Prototipação e UC (Use Cases – Casos de Uso). O protótipo é recomendado para avaliar a interface do IU (usuário), os problemas de comunicação com outros produtos e a possibilidade de atendimento aos requisitos de desempenho. Nesse contexto, os UC são úteis para identificar funcionalidades do software.

Para análise de IHC, a figura 7 mostra o layout (diagramação ou wireframe) de uma tela para uma determinada página web, que é um protótipo de tela que considera: áreas reservadas para textos e para imagens, botões, barras de rolagem e destaque.

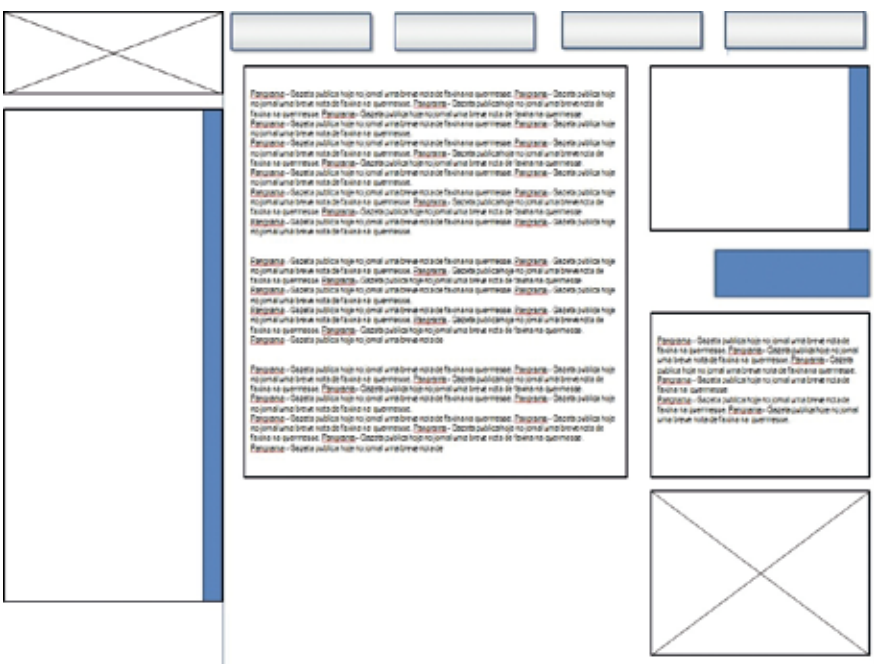


Figura 7 – Protótipo de tela de software (diagramação, layout ou wireframe)

Para entender as operações do software e convertê-las em funcionalidades, a figura 8 mostra um diagrama de UC, que permite identificar as entidades (usuários) envolvidas e as operações do software. As operações são acentuadas pelos UC, podendo representar uma ou mais funcionalidades.

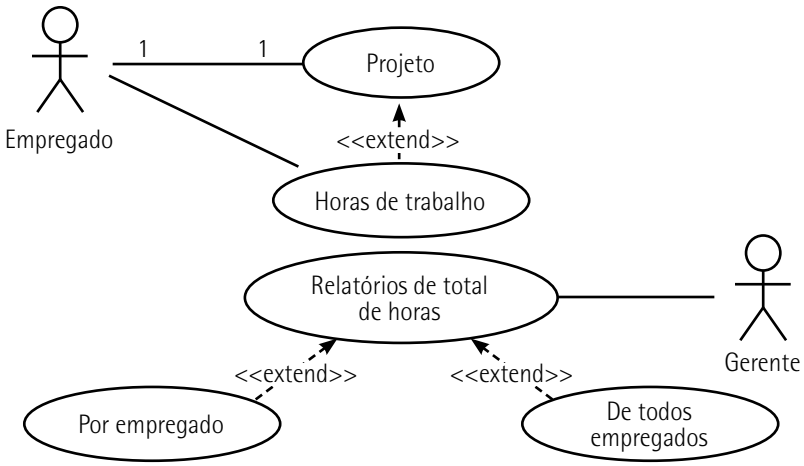


Figura 8 – Diagrama de UC na gestão de projetos

BPM (Business Process Management – Modelagem do Processo de Negócio)

Similar a um fluxograma ou diagrama de atividades da UML, é uma ferramenta útil para que o cliente, os usuários e os analistas interpretem em uma única linguagem visual como o negócio funciona.

Adotada em 2005 pela OMG (Object Management Group), a BPM é uma notação gráfica que descreve a lógica dos passos de um processo de negócio. É um padrão internacional que permite modelar o processo de uma maneira unificada e padronizada. Em 1997, o mesmo grupo adotou a UML (Unified Modeling Language – Linguagem de Modelagem Unificada) como padrão. Seu objetivo é dar suporte ao gerenciamento, fornecendo uma notação intuitiva e capaz de representar semânticas de processos complexos.

A figura 9 ilustra a modelagem do processo de negócio na gestão de estoque, destacando a sequência de operações no recebimento de material em um determinado almoxarifado. Nessa análise, é possível visualizar o comportamento do software quando em operação pelo usuário.

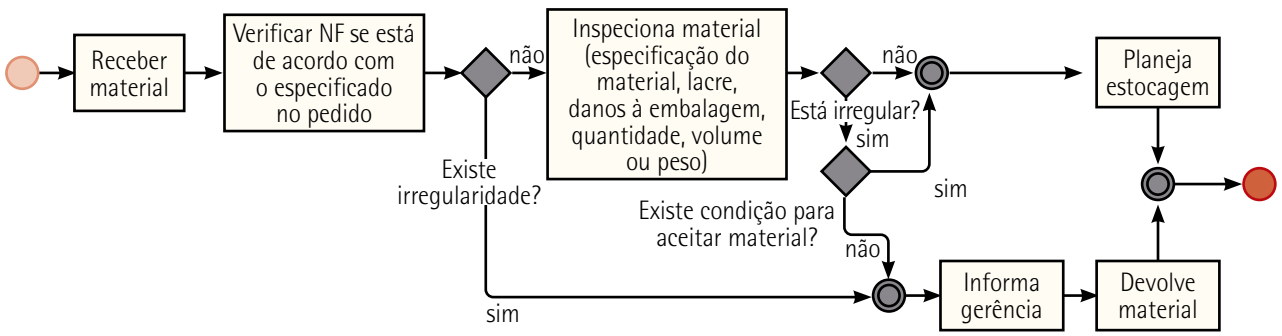


Figura 9 – Modelagem do processo de negócio na gestão de estoque

Saiba mais

Crie modelos de processos de negócios para identificar oportunidades de melhoria dos processos e aumentar a eficiência e a agilidade em toda a organização. Conheça mais sobre o Bizagi e o Bizagi Modeler em:

MODELAGEM de processos de negócios com Bizagi. *Bizagi*, 2024. Disponível em: https://shre.ink/eghM. Acesso em: 23 jun. 2025.

Em resumo, um documento com a especificação e a modelagem dos produtos que serão produzidos é o resultado do trabalho da engenharia de requisitos.

Em sistemas de software, o termo especificação serve como fundamento para as atividades de engenharia de software subsequentes. Assim, descrevem-se a função, o desempenho do sistema e as restrições que acompanharam seu desenvolvimento.

A modelagem parte de um modelo gráfico, um modelo matemático formal, como: UC, protótipos, diagramas ou qualquer combinação desses elementos.

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

Os quatro principais grupos de requisitos para elaboração do contrato do software serão estudados a seguir.

RU (Requisitos do Usuário)

São declarações em linguagem natural, formulários e diagramas simples sobre as funções que o software deve fornecer e as restrições sobre as quais deve operar.

Os requisitos do usuário são obtidos na elicitação do projeto de software. Esse documento pode ser elaborado pelo próprio cliente ou pelo analista do processo de negócio alinhado ao cliente e ao representante dos usuários.

Nessa fase, modelos de negócios e UC são produzidos com especificações que permitem determinar os demais requisitos. Isso incluem os requisitos de interface com usuários e outros sistemas, normas e leis, aos quais o software deve ser submetido e regras de acesso, tais como níveis de permissões para diretores, gerentes e funcionários.

Lembrete

A prática da elicitação para obter requisitos do usuário é uma ação constante, envolvendo desde a concepção do negócio até adaptações que atendam às necessidades específicas do projeto, que ocorrem durante todo o ciclo de desenvolvimento, na implantação e no suporte ao usuário. Também é importante, com base na coleta de insights dos usuários, criar histórias de usuários. Para criar uma linguagem comum, construa modelos visuais do negócio, protótipos de tela e de UC.

RNF

Determinam a qualidade e as propriedades que um software deve ter, sendo o diferencial que faz com que uma aplicação seja boa e atrativa. Uma forma de memorizar o que são RNF é saber que eles são tudo aquilo que o cliente não pede e que, se houver problema, ele vai reclamar.

Sommerville (2003) mostra na figura 10 os principais tipos de RNF, formados pelos grupos: requisitos do produto, requisitos organizacionais e requisitos externos.

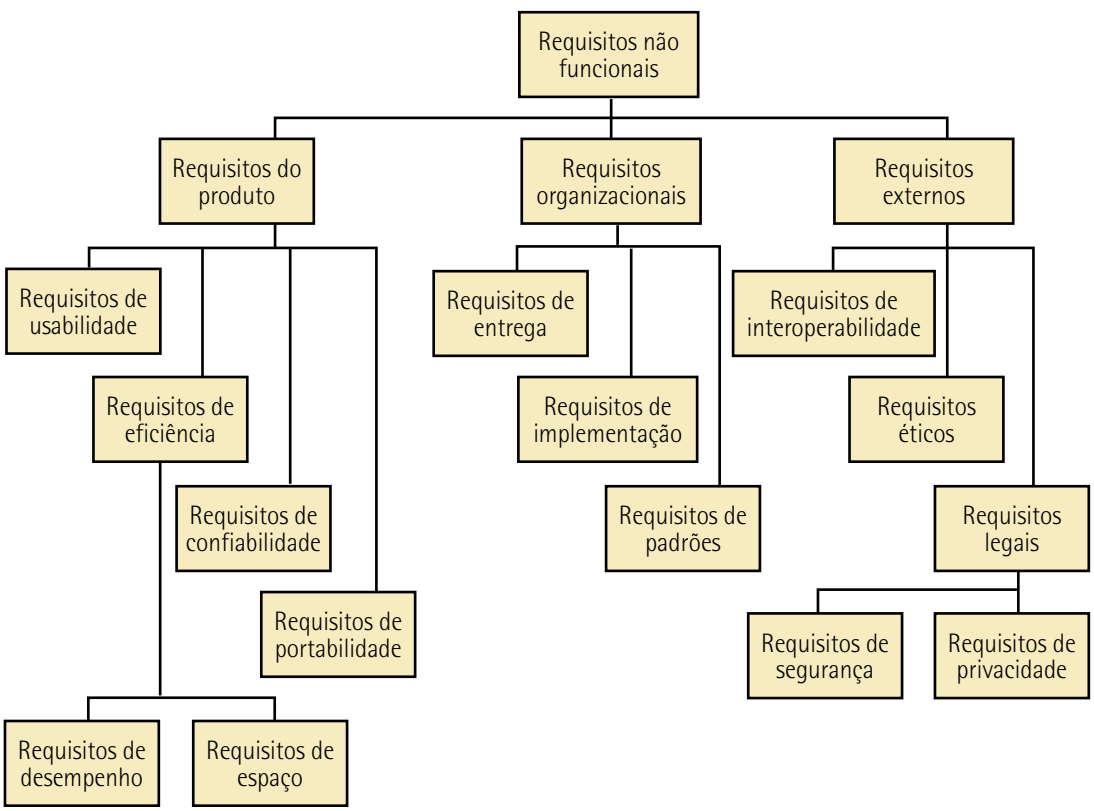


Figura 10 – Tipos de RNF

Adaptada de: Sommerville (2003).

No modelo de Sommerville (2003), cada grupo de requisitos é específico, considerando características (ou atributos) associadas ao grupo. Acompanhe os conceitos a seguir para cada grupo de RNF, como observado na figura 10.

O primeiro grupo a ser acentuado é o de requisitos do produto. Seu objetivo é atender clientes e usuários em suas necessidades e expectativas em relação ao software. É composto pelos requisitos:

- **Usabilidade**: se refere à IU, atributo que estudaremos em pormenores a seguir.
- **Eficiência**: é fazer o correto da melhor forma possível, considerando o período de execução da tarefa e o valor econômico associado. As principais medidas são o desempenho e o espaço para processamento e armazenamento.
- **Confiabilidade**: é atender à expectativa de apresentar um determinado resultado de acordo com o esperado no requisito.
- **Portabilidade**: é a capacidade de escalar o software em outros ambientes operacionais com base no desempenho e no espaço de armazenamento.

Logicamente, podemos associar outras características no grupo de requisitos do produto, tais como:

- **Funcionalidade**: característica específica do software, representa o conjunto de uma ou mais funções com aspectos operacionais e técnicas similares.
- **Manutenibilidade**: durante o ciclo de vida do software, ocorrem diversas mudanças no software a pedido do usuário, seja pela implementação de algum componente no ambiente operacional, seja por alguma falha do software. A característica manutenibilidade se preocupa com a facilidade a responder às mudanças de forma rápida.

O segundo grupo é o de requisitos organizacionais. Eles estão associados à gestão do atendimento aos requisitos conforme as restrições, os padrões e as normas internas e externas fixadas para desenvolver o software. O objetivo é alinhar o desenvolvimento do software com a estratégia de serviços e sua construção para garantir a alocação dos recursos para efetivar o projeto. Nesse contexto, envolve a disponibilidade dos recursos necessários para a entrega do software, sua implementação e os procedimentos para atender aos padrões organizacionais e legais.

Por fim, o terceiro grupo é o de requisitos externos, que envolve o atendimento a regulamentações governamentais, normas, padrões da indústria de software, expectativas dos stakeholders, competitividade etc.

É composto pelos seguintes elementos:

- **Interoperabilidade**: é a capacidade do software em efetuar operações em ambientes integrados a outros sistemas de software ou sistemas distribuídos.

- **Ética**: conjunto de princípios morais e diretrizes que orientam o comportamento dos profissionais participantes do projeto para criar uma boa comunicação, responsabilidade, confiabillidade e transparência.

- **Aspectos legais**: englobam leis e regulamentações políticas, internas ou externas, que orientam o que é permitido e exigido no desenvolvimento, no uso e na distribuição do software. Os principais requisitos dos aspectos legais, incluindo segurança e privacidade, são:

  – **Segurança**: é associada a implementações de normas de segurança, responsabilidades por violações de segurança, conformidade com as leis, divulgação de incidentes, contratos, treinamento e auditorias.

  – **Privacidade**: determina como as empresas devem coletar, armazenar e tratar os dados pessoais dos usuários, garantindo sua privacidade e sua segurança. No Brasil, temos a LGPD; na Europa, GDPR.

  – **Propriedade intelectual**: do projeto, da codificação, por exemplo, no que se refere a direitos autorais, patentes e marcas registradas.

  – **Blockchain**: tecnologia que permite a criação de um registro digital seguro. Em relação à segurança dos dados, suas principais características são integridade, distribuição, descentralização e criptografia; quanto à privacidade, anonimato e controle de acesso.

Exemplo de aplicação

Estudo de caso: situação de falha no recebimento de comprovantes.

Situação-problema: em qualquer pagamento, é comum o usuário receber um comprovante da operação. Contudo, ao pagar uma conta online, o sistema emite uma notificação: "o processo de pagamento está em andamento", é nesse instante que acontece a falha. O sistema fica travado nessa mensagem e, por essa razão, o usuário fica sem saber se houve ou não o pagamento.

Esse é um caso típico de segurança e confiabilidade do software, atributos da qualidade e que devem constar na lista de RNF. Para garanti-los, deverá ser implementada uma função, que, por meio do protocolo de comunicação, a mensagem desejada pela estação está associada a um determinado número de tentativas ou timeout (tempo de espera). Se o problema ocorrer na estação, uma mensagem de garantia seria "verifique em seu extrato se houve a operação". Caso o servidor não receba uma mensagem de "comprovante recebido" da estação por tentativas, timeout ou cancelamento da operação por parte do usuário, medidas administrativas devem ser adotadas ou ocorrer até o cancelamento da operação.

RF

Trata-se de especificações detalhadas, extraídas dos requisitos do usuário, com uma abordagem completa e consistente da funcionalidade e dos serviços que são esperados que o sistema forneça.

A atividade de elaboração dos RF consiste em extrair dos UC as funções necessárias que irão operar no sistema. O objetivo é preparar um material detalhado para que ele possa ser passado para a equipe de codificação e testes.

Para ter clareza a respeito de objetivos e funções a serem implementados, sugere-se a construção de um quadro com a identificação da funcionalidade ou da função, por meio de um código (label) próprio, acompanhada da especificação da função para ser codificada, como é mostrado no quadro 1.

Quadro 1 – Especificação de RF

RF	Especificação
RF01	CRUD (Create, Read, Update e Delete) para a gestão de usuários do software. Inserir botão para chamada de menu: cadastro – deverá exibir no menu de cadastros: funcionário e cliente/usuário
RF01.1	Adicionar novo usuário: criação do registro para funcionário ou cliente/usuário
RF01.2	Ler usuário: a função deve permitir a busca do usuário por algum termo próprio do usuário ou listar os usuários por funcionários ou cliente/usuário

Lembrete

Os requisitos organizacionais estão associados à gestão do atendimento, conforme as restrições, os padrões e as normas internas e externas fixadas para desenvolver o software.

RS (Requisitos do Sistema)

Trata-se de especificações detalhadas, extraídas dos requisitos do usuário, com uma abordagem completa e consistente de todos os componentes que serão integrados na construção do sistema na qual o software irá operar.

Os RS correspondem à infraestrutura de TI e os seus principais componentes são: software, hardware, pessoas, banco de dados e rede de computadores. Além da especificação, esses componentes podem ser apresentados em vários modelos, que mostram objetos, fluxo de dados e troca de mensagens. Eles servem como base para o contrato destinado à implementação do software.

De acordo com Booch (2002) e Nunes (2010), na modelagem dos RS são utilizados basicamente os diagramas de componentes e o diagrama de implantação da UML.

Os componentes podem ser divididos em dois grupos:

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

- **Visão dinâmica da arquitetura**: acentua 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 RF, pode-se montar um quadro com a identificação do requisito por meio de um código e sua respectiva especificação.

Quadro 2

RS	Especificação
RS01	Computadores de clientes (estações): cinco computadores desktop padrão PC com médio desempenho e 30 notebooks padrão PC de desempenho padrão
RS01.1	Sistema operacional do computador de cliente: 35 licenças Windows 11, compatíveis com RS01
RS02	Rede de computadores com uma taxa de transferência de 100 Mbits/s por usuário

Validação dos requisitos

Ela ocorre formalmente. Nesse documento, os produtos de trabalho resultantes da engenharia de requisitos são avaliados e aprovados pelo cliente e pelo desenvolvedor.

A atividade de validação é a última fase do processo da engenharia de requisitos, responsável por autorizar o desenvolvimento do software. O objetivo é aprimorar, incluir mudanças propostas e aprovar (aceite do cliente) o início do projeto e o desenvolvimento do software.

O cliente valida os requisitos como compromisso das informações fornecidas e reconhecimento dos elementos que permitem a construção do sistema, bem como custos e prazos acordados com o desenvolvedor.

Na validação, os profissionais se comprometem a desenvolver o sistema no custo e no prazo determinados. Durante o desenvolvimento, ocorrem algumas ou muitas mudanças nos requisitos, o que resulta em nova V&V dos requisitos, incluindo custo e prazo para novas implementações.

2.4 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.

Na engenharia de requisitos, é importante incluir em RNF e RF o projeto da IU, que envolve a UX, com base na abordagem aos usuários e na prototipagem.

Nesse contexto, destacam-se o RNF, a acessibilidade e a usabilidade. O design de IU deve ser intuitivo e fácil de usar, sendo uma das características da ergonomia cognitiva. São considerados os formatos das informações a serem apresentadas, a forma pela qual os comandos são solicitados e como o usuário interage com o software.

Por sua vez, a engenharia cognitiva, proposta por Norman em 1986 (*apud* Souza *et al.*, 1999), é uma das abordagens de grande influência no design centrado no usuário. Inicialmente, o designer desenvolve um modelo mental do sistema, denominado modelo de design, fundamentado nos modelos de usuário e na tarefa, conforme ilustrado na figura 11.

Então, o usuário interage com essa imagem do sistema e cria seu modelo mental da aplicação, chamado de modelo do usuário. Ele permite ao usuário formular suas intenções e objetivos em termos de comandos e funções do sistema.

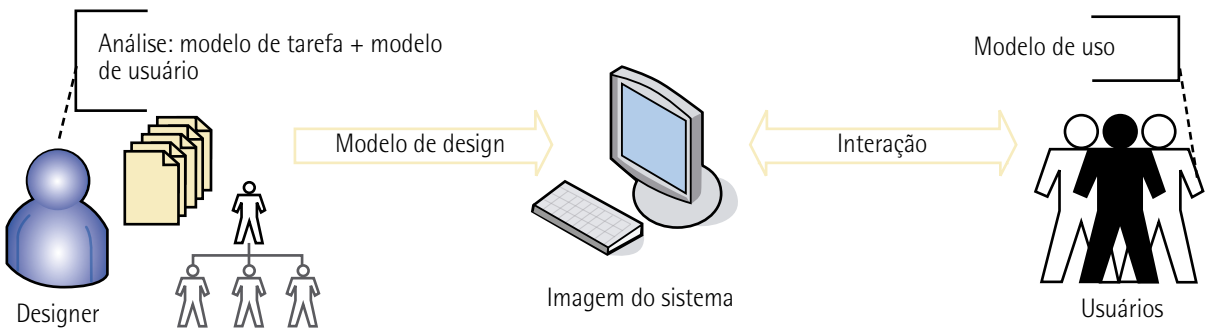


Figura 11 – Processo de design: modelo de interação da engenharia cognitiva

As considerações referentes à usabilidade estão ditadas na NBR ISO 9241-11 (ABNT, 2021). Essa norma ajuda os benefícios a medir a usabilidade em termos de desempenho e satisfação do usuário, conforme a extensão dos objetivos pretendidos de uso, os recursos gastos e a aceitação do usuário no uso do produto.

A figura 12 mostra os componentes da NBR ISO 9241-11 e como se relacionam, ilustrando o design do software, o resultado pretendido e o resultado para atingir os objetivos de usabilidade.

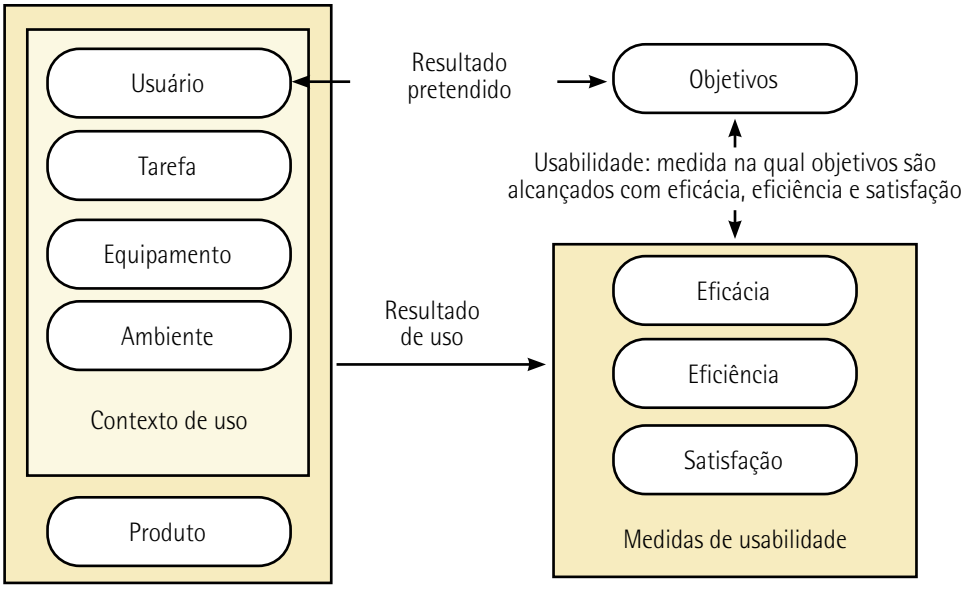


Figura 12 – Estrutura da ISO 9241-11

Adaptada de: ABNT (2021).

Os aspectos de acessibilidade a serem considerados na ergonomia cognitiva são as características ditadas na NBR ISO 9241-210 (ABNT, 2024). Essa norma fornece orientações ergonômicas que garantem acesso a todos os usuários de sistemas interativos considerando suas necessidades específicas.

Saiba mais

Para conhecer melhor a usabilidade na IHC, acesse:

ABNT. *NBR ISO 9241-11*: ergonomia da interação humano-sistema. Parte 11: usabilidade: definições e conceitos. 2. ed. Rio de Janeiro, 2021.

Conheça também os aspectos da acessibilidade na IHC:

ISO. *9241-20:2021*. Ergonomics of human-system interaction. Part 20: an ergonomic approach to accessibility within the ISO 9241 series. Genebra (Suíça), 2021b. Disponível em: https://shre.ink/eblf. Acesso em: 23 jun. 2025.

2.5 Métodos, ferramentas e técnicas (MF&T): elicitação e modelagem ágil (AM)

O mapa mental é uma ferramenta versátil e poderosa que pode ser aplicada em diversas áreas e contextos. É muito útil nos processos iniciais do projeto de software de elicitação e análise. Ele ajuda a representar ideias, conceitos ou tarefas; estruturando-os de uma maneira que facilita a compreensão e a recordação, sendo vital para brainstorming e planejamento.

Segundo Ambysoft (s.d.), a AM é uma metodologia pautada na prática para modelagem e documentação efetiva de sistemas baseados em software. É um método pautado em alguns princípios fundamentais que, apresentados em um mapa mental, permitem visualizar a organização das informações.

No processo de elicitação ou análise com mapa mental, para obter os requisitos, alguns dos princípios da AM deverão ser atendidos.

Os princípios fundamentais da AM são:

- **Assuma a simplicidade**
  - Com um mapa mental, é possível visualizar os requisitos.
- **Abrace a mudança**
  - Habilitar o próximo esforço é seu objetivo secundário.
- **Mudança incremental**
  - Ao usar uma ferramenta para construir um mapa mental, as mudanças e os incrementos podem ser alterados de forma simples.
- **Maximize o ROI (Return on Investment – Retorno sobre Investimento) das partes interessadas**
  - Quando você visualiza um processo em vez de ler inúmeras páginas, o contexto do processo tem maior clareza.
- **Modelo com um propósito**
  - Os objetivos ficam evidentes.

- **Vários modelos**

  – Pode-se criar vários modelos com mapas mentais, tais como: requisitos, mapas de navegação, protótipos, análise de infraestruturas e outros.

- **Trabalho de qualidade e feedback rápido**

  – O software funcional é seu objetivo principal. Assim, o mapa mental cria uma linguagem comum na equipe de projeto; o modelo é visível e permite refinar o processo de forma rápida antes mesmo de codificar o software.

- **Viagem leve**

  – O mapa mental está no formato eletrônico e disponível em várias plataformas, são preservados apenas os modelos realmente úteis.

Para construir o modelo de mapa mental mostrado nas figuras 13 e 14, foi utilizada a ferramenta FreeMind, um software gratuito de mapeamento mental escrito em Java.

A figura 13 ilustra o mapa mental para coleta de requisitos de IHC. Começa-se com uma ideia central ou apenas uma parte dessa ideia e, a partir dela, ramifica-se em subtemas e detalhes relacionados.

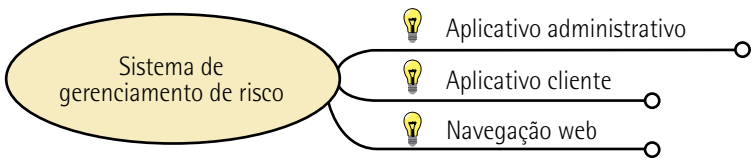


Figura 13 – Levantamento de requisitos para IHC – nível 1. Imagem criada por meio da ferramenta FreeMind

O mapa mental tem um bloco principal chamado de nó-pai; a partir dele, temos os nós-filhos, como é mostrado na figura 14, que também ilustra a tela principal do FreeMind e suas funcionalidades.

O nó-pai pode representar um sistema, um processo, uma funcionalidade, uma atividade ou uma tarefa. Enfim, é o objeto em estudo ou análise. Os nós-filhos são as derivações do objeto em estudo, que podem representar fases, metas, subprocessos, atividades ou funções. Ao observar as terminações das linhas, pode ser visto um ponto vazado, que significa que existem extensões dessas especificações.

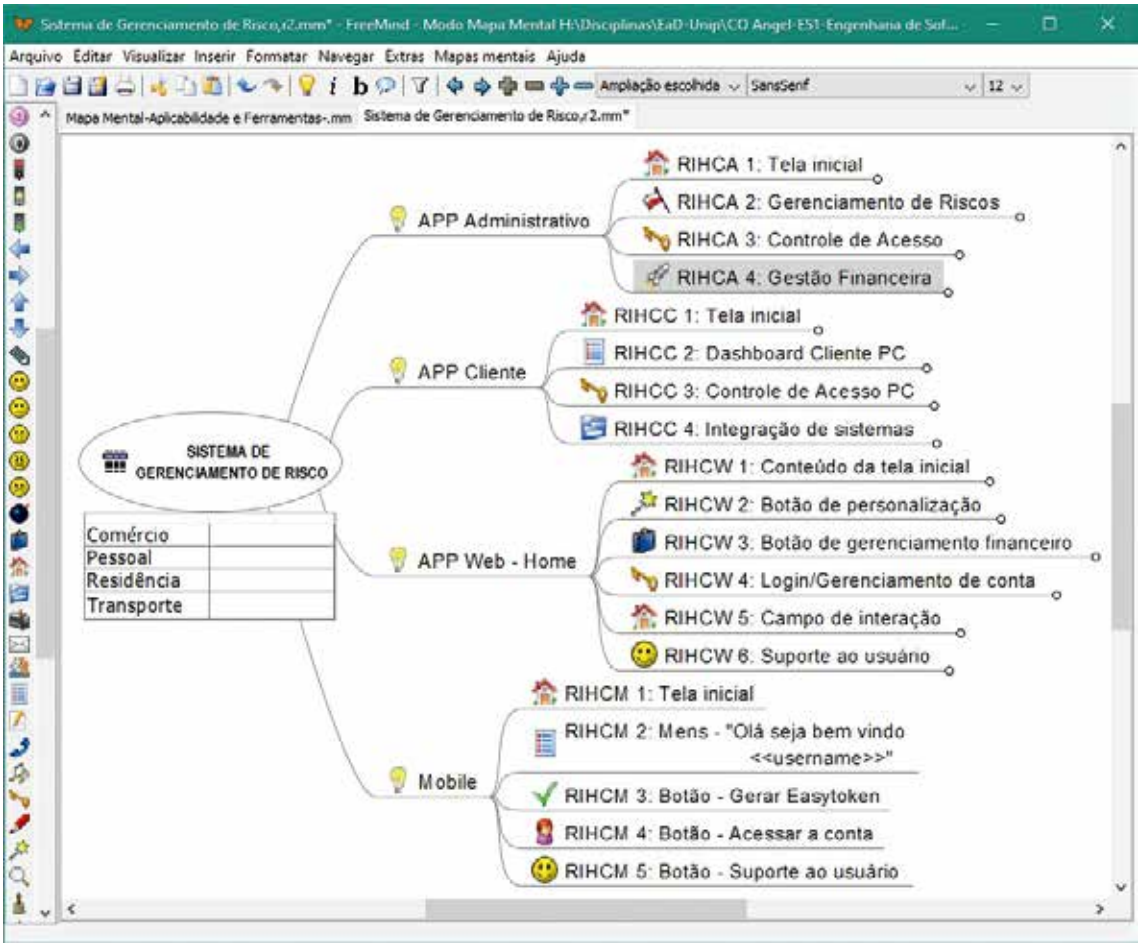


Figura 14 – Levantamento de requisitos para IHC – nível 2. Imagem criada por meio da ferramenta FreeMind

Resumo

Nesta unidade, abordaram-se as fundamentações da TI, a base para a infraestrutura da engenharia de software e de sistemas. Foram exploradas as características do software, incluindo sua vida útil, reusabilidade e componentização, além da demanda crescente e da indústria de software. Também foram analisados os processos de produção e manutenção do software, destacando os desafios enfrentados e as estratégias de melhoria adotadas.

A engenharia de software direciona o conhecimento técnico do profissional, fazendo com que ele compreenda o que constitui um software de alta qualidade e o monitoramento contínuo ao longo de todo o ciclo de vida do desenvolvimento, desde a concepção e implementação até a evolução, incluindo as mudanças inevitáveis que ocorrem em todo o ciclo de desenvolvimento, de forma a assegurar uma manutenção contínua. Nessa situação, é vital haver um gerenciamento de configuração de software que acompanhe, rastreie e controle todas as atividades do desenvolvimento, dando suporte e ajustes necessários.

Ao longo do ciclo de desenvolvimento do software, discutimos alguns dos controles que monitoram a evolução do software. Trata-se do versionamento, que envolve controle de versões, releases e infraestrutura.

Vimos a aplicabilidade do software, que é ampla e diversificada, como mostram as tecnologias e as tendências da engenharia de software. Foram descritas várias aplicações do software, que vão desde simples aplicativos, sistemas de software do tipo e-business e e-commerce, análise de dados, IA e outros.

O aprendizado gera capacidades para conceber o software, por meio da engenharia de requisitos, não importando seu tamanho, complexidade ou nível exigido de qualidade. Nesse contexto, foram abordados requisitos como RU, RNF, RF, RS e IHC.

Por fim, sugeriu-se o uso de uma ferramenta de mapa mental para a elicitação e análise, seguindo os princípios da AM.

Exercícios

Questão 1. Considerando a evolução histórica da engenharia de software e as características intrínsecas do software em contraste com o hardware, avalie as afirmativas.

I – O software, diferentemente do hardware, não sofre desgaste físico, mas deteriorações devido a mudanças em seu ciclo de vida.

II – A curva de falhas do hardware apresenta uma fase de "mortalidade infantil", seguida por estabilidade e, posteriormente, desgaste, enquanto a curva do software, após a estabilidade, entra em deterioração devido a modificações sucessivas.

III – A reutilização de código é uma prática que prejudica a qualidade do software, pois aumenta a complexidade e a probabilidade de erros, reduzindo a produtividade da equipe de desenvolvimento.

IV – A dualidade entre software e hardware significa que um sistema computacional só é viável se ambos evoluírem de forma independente, sem necessidade de adaptações mútuas ao longo do tempo.

É correto o que se afirma em

A) I, apenas.

B) I e II, apenas.

C) II e IV, apenas.

D) II, III e IV apenas.

E) I, II, III e IV.

Resposta correta: alternativa B.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: o software não se desgasta fisicamente como o hardware, mas se deteriora com mudanças, como atualizações e manutenções, que podem introduzir novos defeitos.

II – Afirmativa correta.

Justificativa: as curvas de falhas ilustram que o hardware sofre desgaste físico, enquanto o software enfrenta deterioração por modificações.

III – Afirmativa incorreta.

Justificativa: a reutilização de código melhora a qualidade e a produtividade, reduzindo erros e acelerando o desenvolvimento do projeto.

IV – Afirmativa incorreta.

Justificativa: software e hardware evoluem de forma interdependente, exigindo adaptações mútuas.

Questão 2. (UFSM 2022, adaptada) Em relação à engenharia de requisitos de software, avalie as afirmativas.

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

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

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

É correto o que se afirma em

A) I, apenas.

B) II, apenas.

C) I e II, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: alternativa C.

Análise das afirmativas

I – Afirmativa correta.

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

III – Afirmativa incorreta.

Justificativa: o requisito "o sistema deve fazer o cancelamento de consultas" descreve uma função específica que o software deve realizar. Da forma como foi expresso, ele é classificado como um requisito de usuário, já que foi escrito em linguagem natural. Como ele está diretamente relacionado à funcionalidade do sistema, dá origem a um RF.

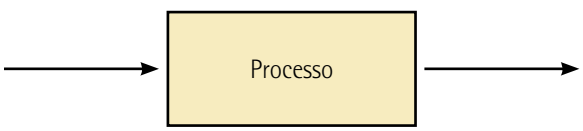
Unidade II

3 PROCESSOS DE SOFTWARE

> O processo de software constitui a base para o controle do gerenciamento de projetos de software e estabelece o contexto no qual são aplicados métodos técnicos, são produzidos artefatos (modelos, documentos, dados, relatórios, formulários etc.), são estabelecidos marcos, a qualidade é garantida e as mudanças são geridas de forma apropriada (Pressman, 2021, p. 15).

O processo é um conjunto de atividades, métodos, práticas e transformações empregadas no desenvolvimento e na manutenção de software. Trata-se de um guia para a equipe de desenvolvimento do software, garantindo que todos os aspectos importantes sejam explorados de maneira sistemática e controlada.

De acordo com os princípios estabelecidos em sua sexta edição, *PMBOK: guia do conhecimento em gerenciamento de projetos* (PMI, 2017), um processo de gerenciamento de projetos deve ser claramente delimitado, definindo seus pontos de iniciação e conclusão. A figura 15 ilustra essas fronteiras, representando os artefatos de entrada (inputs) e os de saída (outputs). Em projetos, esses pontos críticos são operacionalizados como marcos (milestones), elementos de referência essenciais para o monitoramento do progresso, permitindo a mensuração de desempenho, a avaliação da eficácia, a projeção de melhorias incrementais e a identificação de tendências evolutivas.



Input (entradas)
* Objetivos
* Recursos necessários
* Procedimentos
* Documentos
* Custo/cronograma
* Métodos/ferramentas

Output (saídas)
* Produto
* Componente
* Serviços

Figura 15 – Modelo básico do processo

O modelo de processo de software fornece estabilidade, controle e organização para as atividades; caso contrário, elas se tornariam caóticas.

> [..] existem muitos tipos diferentes de sistemas de software, e não há um método universal de engenharia de software que seja aplicável a todos eles. Consequentemente, não há um processo de software universalmente aplicável. O processo utilizado em diferentes empresas depende do tipo de software que está sendo desenvolvido, dos requisitos do cliente de software e das habilidades das pessoas que escrevem o software (Sommerville, 2016, p. 44).

O processo ágil é uma abordagem flexível e colaborativa para desenvolver software com o foco em entregas contínuas, agregando valor ao cliente. É formado pelo conjunto de diversas outras metodologias ágeis, fundamentadas na prática para modelagem efetiva e sistemas baseados em software, e elas podem ser aplicadas por profissionais no dia a dia.

Observação

PMBOK é a sigla de Project Management Body of Knowledge (Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos).

3.1 Agilidade e decomposição do processo

A agilidade no desenvolvimento de software mede a capacidade de fornecer feedbacks rápidos às mudanças no software que ocorrem ao longo de seu ciclo de vida.

Na engenharia de software, os processos são tratados de duas formas: desenvolvimento prescritivo e desenvolvimento ágil. Comumente, esses modelos são integrados para construir software. A ideia é ter um arcabouço, não só de cada processo, mas também de uma diversidade de processos que podem ser aplicados em diversas situações ou outros modelos de desenvolvimento.

O desenvolvimento prescritivo (baseado em planos) é caracterizado por seguir uma abordagem mais estruturada e sequencial, com etapas bem definidas e especificadas antes do início do projeto. Geralmente, pauta-se no modelo de processo Cascata, um clássico no desenvolvimento de software.

Observe a figura 16. São consideradas três etapas: engenharia de requisitos, especificação de requisitos e projeto e implantação. Quando ocorre uma solicitação de mudança, deve-se percorrer todo o ciclo com novas ou revisões das especificações.

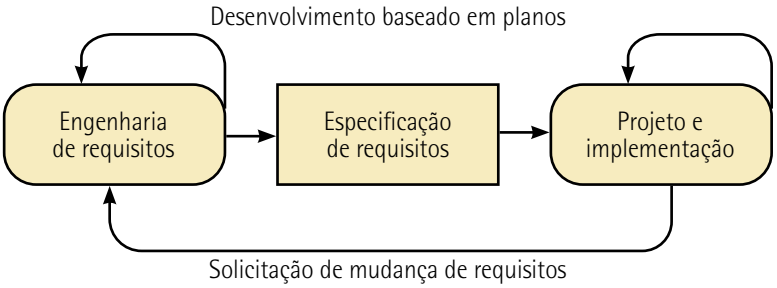


Figura 16 – Modelo prescritivo

Adaptada de: Sommerville (2016).

Esse modelo deve ser aplicado para atender toda a organização em uma visão global. Tanto o software como o processo de desenvolvimento de software é orientado por estruturas organizacionais.

O desenvolvimento prescritivo enfatiza a previsibilidade e o planejamento detalhado, enquanto o desenvolvimento ágil valoriza a flexibilidade, a colaboração e o feedback constante. Ele pode ser mais adequado para projetos estáveis e bem definidos, enquanto o modelo ágil se destaca em cenários dinâmicos e sujeitos a mudanças constantes.

O desenvolvimento ágil é uma abordagem iterativa e incremental, com baixo índice de especificação, que prioriza a adaptação a mudanças e a entrega de valor contínuo ao cliente.

Quando se utiliza o processo descritivo, basicamente, há um único modelo aplicado em todo o desenvolvimento do software, podendo inclusive conter metodologias ágeis em seu contexto. O processo ágil pode conter apenas uma das metodologias ágeis ou o conjunto dessas, conforme as necessidades do software.

Observe a figura 17. Enquanto o modelo prescritivo é formado por três etapas do desenvolvimento, em uma visão na logística de serviços, as metodologias ágeis buscam reduzir ou até zerar a etapa de especificação de requisitos.

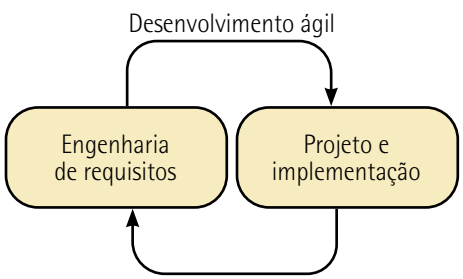


Figura 17 – Modelo de processo ágil

Adaptada de: Sommerville (2016).

A escolha de um ou mais modelos de desenvolvimento ou qualquer outro método determina os recursos alocados para construir o software, o que envolve custos e prazos.

Em um contexto geral, essa escolha depende do tamanho, da complexidade, da qualidade exigida e da tecnologia a ser implementada. Envolve também a escolha de pessoal capacitado, parceiros e uma boa comunicação.

Normalmente, um processo prescritivo é aplicado em projetos de grande escala, nos quais os requisitos são claros e estáveis.

Por sua vez, o processo ágil é aplicado em projetos pequenos, médios e em startups; denotando certo grau de incerteza. Uma das características fundamentais das metodologias ágeis é que os requisitos podem variar de acordo com novas mudanças requeridas. Mesmo que não estejam tão claros e sejam poucos estáveis, eles evoluem com o tempo por meio de feedbacks rápidos e contínuos.

Em ambos os casos, será necessário decompor os processos do projeto para avaliar o escopo e os recursos necessários para desenvolver o software.

Lembrete

O processo é um conjunto de atividades, métodos, práticas e transformações empregados no desenvolvimento e na manutenção de software.

Decompor o processo em uma sequência lógica de práticas para o desenvolvimento ou a evolução de sistemas de software engloba as atividades de análise, especificação, modelagem, implementação, testes, implantação, suporte e manutenção. Por exemplo: o levantamento de requisitos de uma funcionalidade específica.

O bom processo é caracterizado pelo emprego de dispositivos específicos de escolha do arcabouço do processo, registro e interação das ferramentas de trabalho, pessoas envolvidas no processo e métodos aplicáveis.

Observação

Funcionalidade: conjunto formado por uma ou várias funções que empregam recursos e lógicas similares, tecnologias comuns e que capacitam o software a prover funções que atendam às necessidades externas e internas do software.

Por exemplo: a funcionalidade CRUD (Create, Read, Update e Delete – Criar, Ler, Atualizar e Apagar) descreve quatro funções básicas a serem aplicadas na gestão de arquivos e operações em banco de dados.

De acordo com o escopo do projeto, o processo é decomposto, basicamente, em atividades e tarefas. Os processos podem ser subdivididos em subprocessos, a depender do escopo do projeto, e em conjunto com os procedimentos de serviços a serem seguidos. A figura 18 mostra como é a estrutura genérica de um processo e os elementos que a compõem.

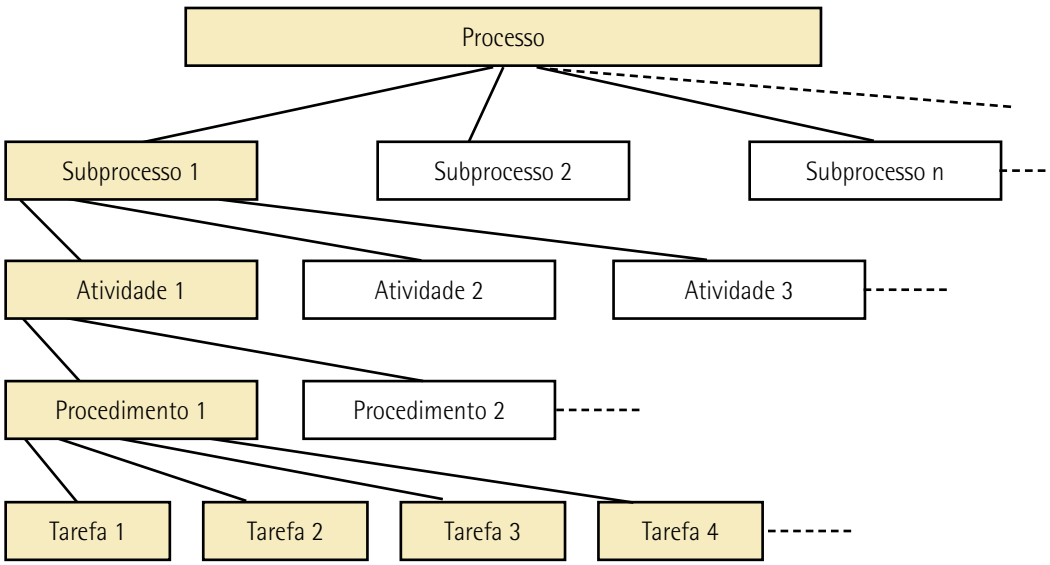


Figura 18 – Estrutura genérica do processo

Observe cada um desses elementos a seguir.

- **Subprocesso (ou meta)**: é uma parte distinta de um processo maior, funcionando como se fosse uma célula de trabalho. Assim, quando se integra a outros subprocessos, contribui para o processo geral. Em cada subprocesso, nos pontos de início e fim, define-se um marco de referência, útil para o monitoramento, visando assegurar que os planos, os artefatos e as atividades de software estejam alinhados com os requisitos fixados e com o período de entrega.

- **Atividade**: é uma ação ou conjunto de ações como parte do processo ou projeto; refere-se a uma função específica ou a uma unidade de trabalho.

- **Procedimento**: é a forma pela qual se deve agir, fazer ou cumprir para que uma atividade seja realizada. Os procedimentos normalmente são caracterizados por meio de uma lista que apresenta passo a passo as tarefas a serem executadas para fazer algo de forma lógica.

- **Tarefas**: são os trabalhos que devem ser executados na atividade; representam a empreitada de serviços, ou seja, a quantidade de trabalhos realizados para efetivar a atividade.

Exemplo de aplicação

Caso: processo de implementação de uma funcionalidade.

A decomposição do processo de implementação de uma funcionalidade, apresentado na figura 19, mostra a parte referente ao subprocesso software, que corresponde a um dos itens da infraestrutura de TI, formados pelos elementos: software, hardware, pessoas, banco de dados e rede de computadores. O subprocesso "pessoas" não aparece porque nosso foco é o ambiente computacional.

Observe que o modelo mostra apenas o desenvolvimento da atividade "codificação e testes unitários". No caso, as outras atividades também devem ser decompostas porque fazem parte do subprocesso de software. Cada subprocesso também é decomposto. Esse modelo serve de guia para o processo de desenvolvimento, traz mais clareza e permite maior flexibilidade no acompanhamento do processo, podendo ser formado por guias de cartões (Kanban).

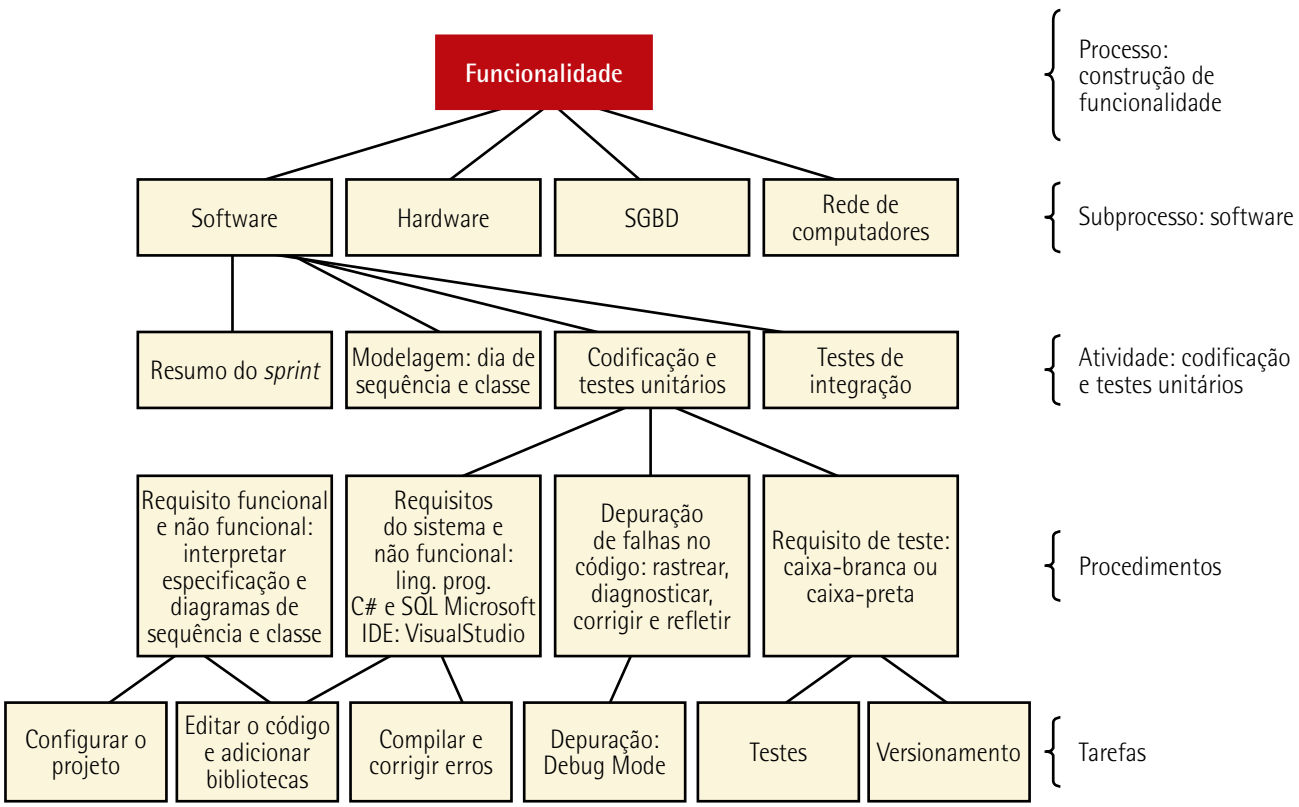


Figura 19 – Modelo para decompor o processo de implementação de funcionalidade.
Imagem criada por meio da ferramenta de diagramação draw.io

Saiba mais

O modelo apresentado na figura 19 foi construído com o draw.io. Ele é próprio para construção de arquiteturas, diagramas, protótipos, topologias e diversos outros modelos gráficos.

O draw.io tem uma coleção de funcionalidades apropriadas para a diagramação. Assim, temos diagramação, modelos e arquiteturas em um só aplicativo!

Acesse o link a seguir para aprender a usar o draw.io:

Disponível em: https://shre.ink/egu4. Acesso em: 23 jun. 2025.

3.2 PSP (Personal Software Process – Processo de Software Pessoal)

Trata-se de uma metodologia criada pelo SEI (Software Engeneering Institute). Específica para o engenheiro de software, visa melhorar a engenharia individual de software, proporcionando uma metodologia disciplinada para planejar, medir e analisar o trabalho pessoal, tentando aumentar a qualidade e a produtividade. O PSP é focado no desenvolvimento de sistemas de software por engenheiros em nível pessoal (Humphrey, 1995).

Parte integrante do CMMI (Capability Maturity Model Integration – Modelo Integrado de Capacidade e Maturidade), o PSP tem o foco na melhoria organizacional.

Desde a criação do modelo PSP, sua estrutura permanece a mesma, porém várias pequenas alterações foram feitas para torná-lo mais aplicável, eficiente e adaptável às necessidades modernas, tais como práticas como DevOps e automação de testes (Humphrey, 1995).

A proposta do PSP é interagir com as práticas organizacionais da qualidade (como ISO 9001 e ISO 9126) e modelos de qualidade, como CMMI e SPICE (Software Process Improvement & Capability Determination – Melhoria de Processos de Software e Determinação de Capacidades). O objetivo é fazer com que os processos pessoais sejam conhecidos, controlados e melhorados. Assim, o PSP:

- contribui para o cumprimento da norma ISO 9001 ao fornecer um framework de práticas individuais que são consistentes e documentáveis;
- promove a qualidade individual, o que indireta e diretamente contribui para a melhoria nas características da norma ISO 9126;

- é uma extensão pessoal que se alinha ao CMMI, pois promove a disciplina individual necessária para que as organizações alcancem níveis mais elevados de maturidade de processo;

- complementa o SPICE ao incentivar uma abordagem disciplinada e sistemática para o trabalho individual, fornecendo os dados e as métricas pessoais que podem ser usados para avaliações de capacidade de processo.

Segundo Hayes (1997), o framework do PSP, ilustrado na figura 20, apresenta sete níveis do processo e cada um se baseia no nível anterior, adicionando algumas etapas do processo a ele.

> Cada nível se baseia nas capacidades desenvolvidas e nos dados históricos coletados no anterior nível. Os engenheiros aprendem a usar o PSP escrevendo dez programas, um ou dois em cada um dos sete níveis e prepara cinco relatórios escritos. Os engenheiros podem usar qualquer método de projeto ou linguagem de programação na qual eles são fluentes (Hayes, 1997, p. 6, tradução nossa).

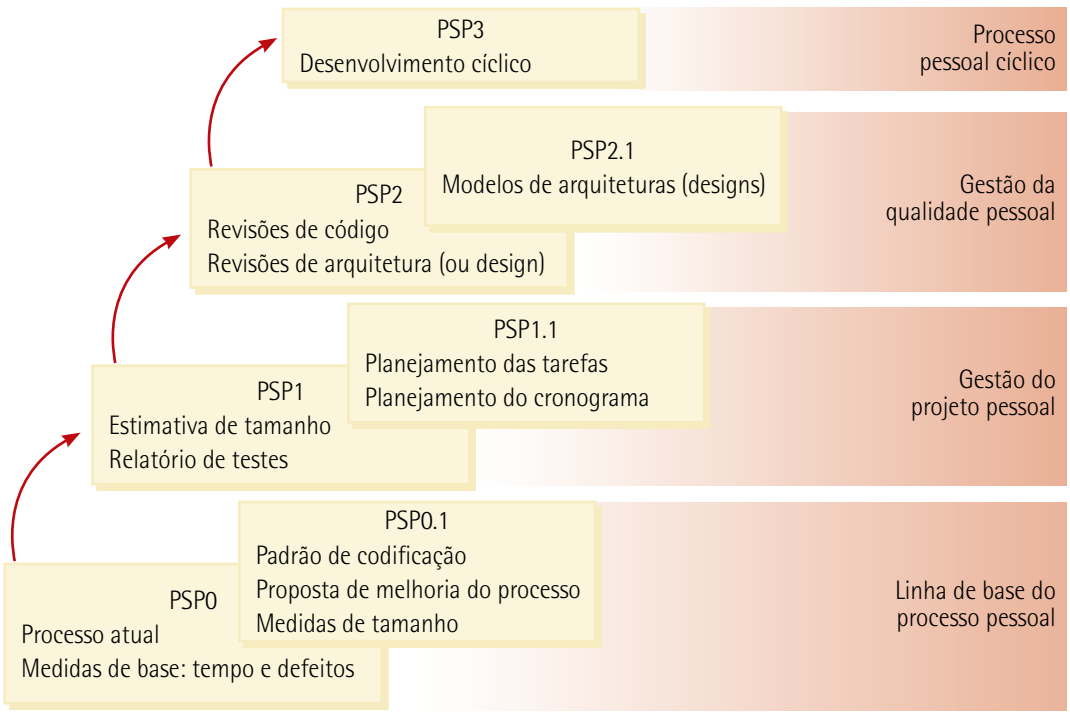


Figura 20 – Níveis do processo PSP

Adaptada de: Hayes (1997).

O PSP orienta o planejamento e o desenvolvimento dos componentes do software e tem como principais objetivos:

- **Melhoria da qualidade do software**: práticas rigorosas de revisão e inspeção reduzem o número de defeitos no software desde as primeiras fases do desenvolvimento.

- **Precisão nas estimativas de tempo e custo**: melhora a estimativa de prazo e esforço para o desenvolvimento do software.

- **Aumento da produtividade**: um bom planejamento faz com que o índice de manutenção seja reduzido, de forma que o projeto pronto seja concluído em menos tempo, elevando a produtividade.

- **Disciplina e organização**: ajuda os desenvolvedores a seguir processos definidos e coleta de medidas de seus trabalhos, melhorando a distribuição de serviços pela equipe de desenvolvimento.

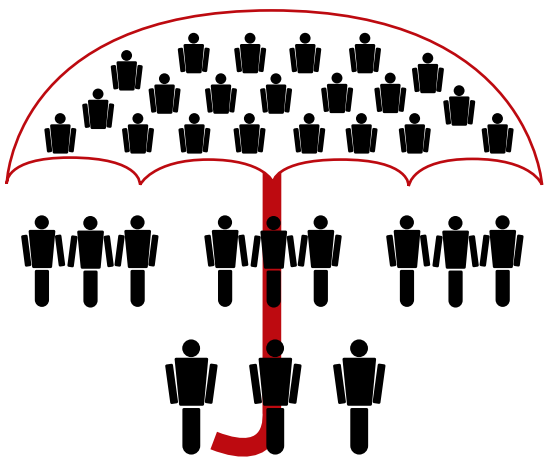
- **Análise de desempenho**: cria um comprometimento pessoal com a qualidade e a melhoria contínua do processo, fornecendo métricas para que os engenheiros avaliem e aprimorem continuamente seu desempenho com dados reais.

3.3 TSP (TEAM SOFTWARE PROCESS – PROCESSO DE SOFTWARE DA EQUIPE)

Trata-se de uma metodologia criada para auxiliar equipes de desenvolvimento de software a produzir produtos de alta qualidade de maneira eficiente e previsível. São práticas disciplinares que melhoram a produtividade e a capacidade na entrega do software dentro do prazo e do orçamento.

O TSP foi desenvolvido por Watts Humphrey (criador do CMMI e do PSP), com foco na equipe de trabalho. Nele, o profissional não trabalha sozinho no desenvolvimento de software.

Memorize e entenda essas três siglas na ordem em que ocorrem, conforme ilustrado na figura 21:



CMMI – melhora a capacidade da organização; foco na gestão

TSP – melhora o desempenho da equipe; foco na equipe e no produto

PSP – melhora as habilidades individuais e a disciplina; foco pessoal

Figura 21 – Métodos de melhoria de processos

Adaptada de: Humphrey (2000).

> A versão inicial do CMM, v 1.0, foi revisada e utilizada pela comunidade de software durante 1991 e 1992 (Paulk, 1993, p. 3).
>
> A partir do ano 2000, dois profissionais do CPqD (André Vilas-Boas e José Marcos Gonçalves) realizaram traduções informais do SW-CMM v 1.1 e do CMMI v 1.1, no âmbito dos projetos do Programa Brasileiro da Qualidade e Produtividade em Software, da SEPIN/MCT (Secretaria de Política de Informática – Ministério de Ciência e Tecnologia) (CMMI, 2006, p. 17).

O TSP complementa o PSP, ajudando cada membro da equipe a desenvolver e aprimorar suas habilidades pessoais dentro do contexto de trabalho em equipe, sendo que o treinamento prévio no PSP é essencial.

O TSP e o PSP podem servir como solução para pequenas organizações de software que se consideram muito pequenas para enfrentar as complexidades do CMMI.

O TSP fornece um framework para equipes de software planejar e gerenciar seus próprios processos de trabalho, acompanha o progresso e gerencia as tarefas do dia a dia.

Processos bem definidos e métricas claras fornecem à equipe um aumento significativo da qualidade e da previsibilidade de seus projetos; também integram práticas de melhoria contínua, ajudando as equipes a identificar e corrigir problemas ao longo do ciclo de vida de desenvolvimento do software.

De acordo com Pressman (2016), os objetivos para o TSP são:

- Criar equipes autodirigidas que planejem e acompanhem seu próprio trabalho, estabeleçam metas e sejam proprietárias de seus processos e planos. As equipes podem ser puras ou equipes de produto integradas (IPTs – Integrated Product Teams), com cerca de 3 a 20 engenheiros.
- Mostrar aos gerentes como treinar e motivar suas equipes e como ajudá-las a manter alto desempenho.
- Acelerar o aperfeiçoamento dos processos de software, tornando o comportamento CMM nível 5 algo normal e esperado.
- Fornecer orientação para melhorias a organizações com elevado grau de maturidade.
- Facilitar o ensino universitário de habilidades de trabalho em equipe de nível industrial.

Observação

CMM nível 5 é conhecido como o nível de otimização, o nível mais alto de maturidade descrito no modelo.

O TSP oferece um conjunto de elementos fundamentais para ajudar as equipes de desenvolvimento a entregar software de alta qualidade de forma previsível e eficiente. Observe esses aspectos no quadro a seguir.

Quadro 3

Elementos fundamentais	
Papéis e responsabilidades	Os papéis são definidos para cada membro da equipe, podendo adquirir as seguintes funções como papéis: responsável pela interação com o cliente, responsável pelo design, responsável pela implementação, gestor de testes, gestor de planejamento, gestor de processos, gestor de qualidade, responsável pelo suporte e líder de equipe
Planejamento detalhado	Envolve a criação de planos, que incluem cronograma, alocação de recursos e marcos de referência
Ciclos de desenvolvimento	Trata-se da divisão das tarefas em ciclos curtos e iterativos para o desenvolvimento de determinadas funcionalidades
Garantia da qualidade	Engloba a prática disciplinada de controle de qualidade, revisões de código, inspeções e testes que garantem que o software atenda aos padrões de qualidade
Métricas e monitoramento	Métricas de desempenho aplicáveis para a coleta e a análise contínua de dados são usadas para monitorar o progresso do projeto e tomar decisões informadas
Comunicação e coordenação	São empregadas ferramentas e práticas que favorecem uma comunicação eficaz e a coordenação entre os membros da equipe
Gerenciamento de riscos	Envolve a identificação e a mitigação de riscos potenciais que podem afetar o desenvolvimento do projeto
Incidentes e defeitos	O registro e o acompanhamento de defeitos e incidentes encontrados ao longo do ciclo de desenvolvimento têm o objetivo de prever e resolver problemas de forma rápida
Revisão e melhoria contínua (iteração)	A reflexão dos ciclos de desenvolvimento ajuda a identificar áreas de melhoria e implementar mudanças no processo
Documentação	Trata-se do registro detalhado do projeto, incluindo planos, métricas, resultados de testes e evidências de defeitos

Lembrete

O TSP é uma metodologia criada para auxiliar equipes de desenvolvimento de software a produzir produtos de alta qualidade de maneira eficiente e previsível.

3.4 Modelos de processos de software

Pressman (2021) apresenta a metodologia do processo de software, acentuada na figura 22. Esse esquema mostra que cada atividade metodológica é composta por um conjunto de ações de engenharia de software.

Cada ação (ou procedimento) é formada por um conjunto de tarefas que devem ser completadas, artefatos que serão produzidos, fatores de garantia da qualidade e marcos do projeto que acompanham o progresso do desenvolvimento.

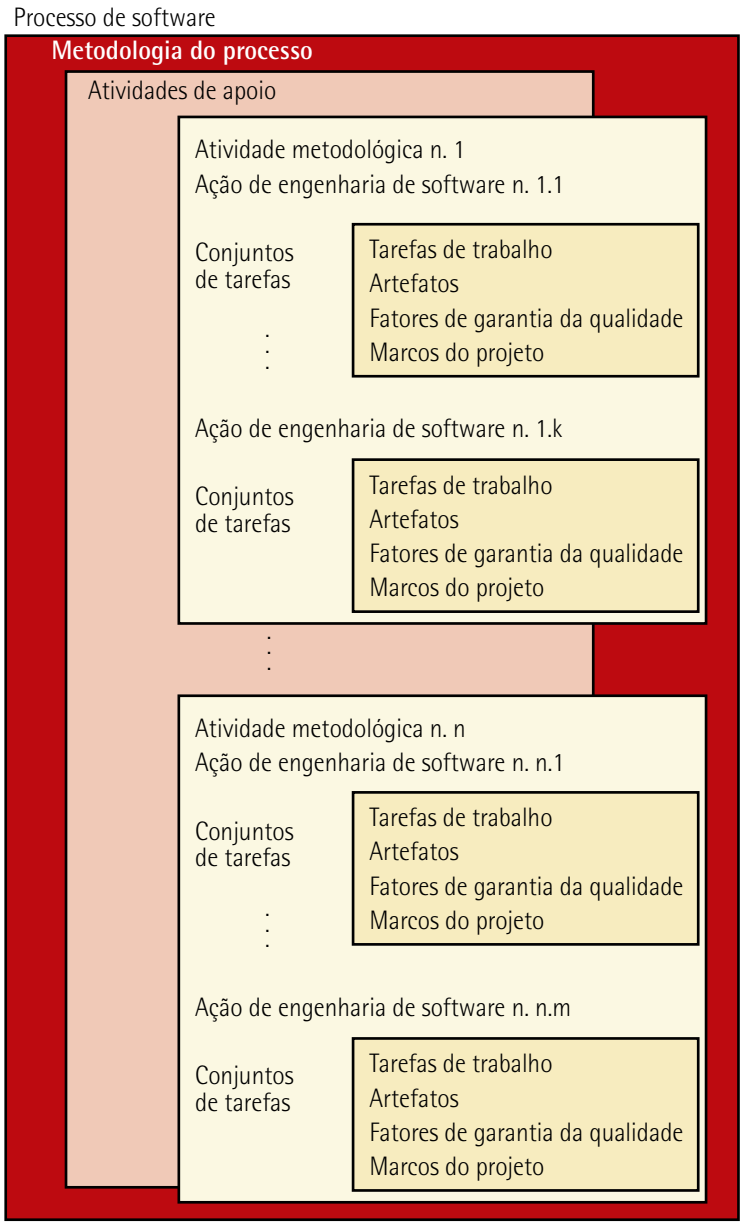


Figura 22 – Metodologia do processo de software

Adaptada de: Pressman (2021).

Esse modelo básico de metodologia do processo de software permite que as organizações se orientem por um modelo de processo, que serve de base para estruturar a organização do desenvolvimento.

> Uma metodologia genérica do processo para a engenharia de software estabelece cinco atividades metodológicas: comunicação, planejamento, modelagem, construção e entrega. Além disso, um conjunto de atividades de apoio que é aplicado ao longo do processo, como o acompanhamento e o controle do projeto, a administração de riscos a garantia da qualidade, o gerenciamento das configurações, as revisões técnicas, entre outras (Pressman, 2021, p. 18).

Os modelos de processos de software englobam um conjunto de atividades, métodos, práticas e transformações empregadas no desenvolvimento e na manutenção de software. Eles fornecem estabilidade, controle e organização para as atividades, que, sem controle, tornam-se bastante caóticas.

Para o projeto ou partes específicas dele, a adoção de um modelo de processo de software permite ter uma visão geral do sistema como um processo orientado por uma estrutura organizacional.

Os modelos de processo de software são mostrados a seguir:

- **Modelos de processos**: PSP e TSP.
- **Modelos de processos tradicionais**: Cascata, Balbúrdia, Prototipagem, Incremental, RAD (Rapid Application Development – Desenvolvimento Rápido de Aplicação) e Espiral.
- **Processo unificado**: RUP, Praxis e Iconix.

3.4.1 Cascata (Waterfall Model ou Sequencial Linear)

> O modelo Cascata, algumas vezes chamado ciclo de vida clássico, sugere uma abordagem sequencial e sistemática para o desenvolvimento de software, começando com a especificação dos requisitos do cliente, avançando pelas fases de planejamento, modelagem, construção e entrega, e culminando no suporte contínuo do software concluído (Pressman, 2016, p. 64).

Ilustrado na figura 23, ele é considerado o modelo clássico do ciclo de vida de desenvolvimento do software, pois provavelmente foi o primeiro a ser apresentado.

Como observado na figura 23, o processo só termina após a última atividade, a entrega feita ao cliente, só depois o software passa por revisões que possibilitam alguma mudança.

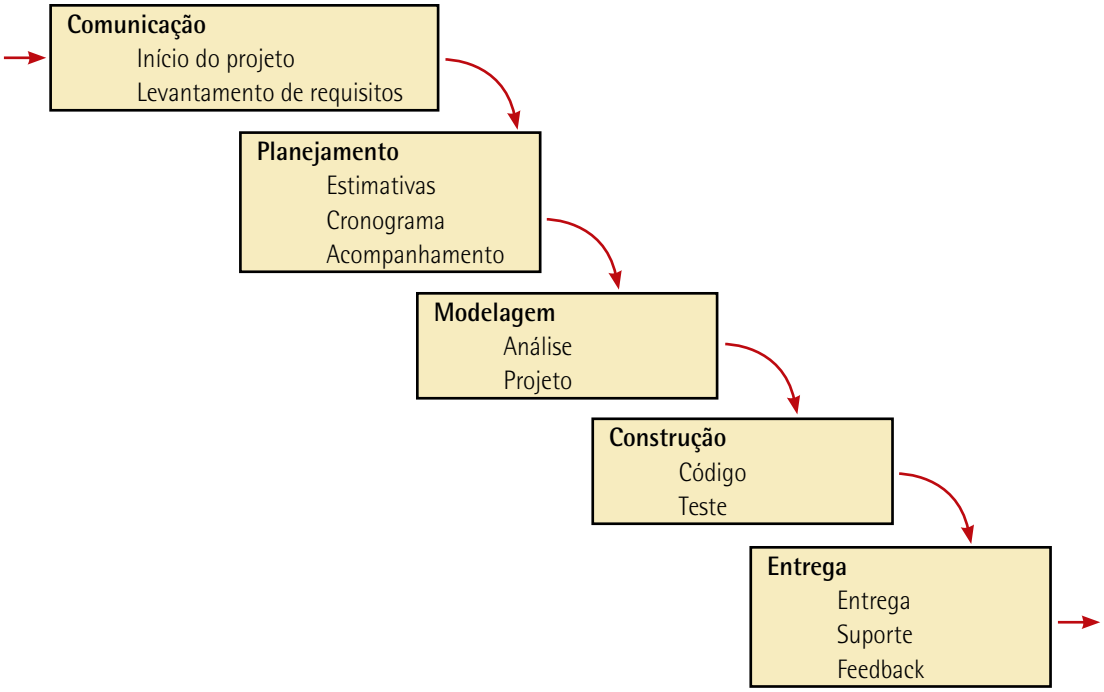


Figura 23 – Cascata

Adaptada de: Pressman (2021).

São descritos a seguir os conceitos sobre as atividades do Cascata:

- **Comunicação**: é a fase de elicitação, quando se inicia o projeto. Nela, é feito o levantamento de requisitos, determinando as funcionalidades, as restrições e os objetivos do sistema.

- **Planejamento**: nessa atividade, os requisitos são agrupados para definir o escopo do sistema, determinar a arquitetura modelada, estimar prazos e custos. São produzidos artefatos de cronogramas, análises de caminho crítico e seleção de recursos a serem disponibilizados. Com os artefatos construídos, é feito o acompanhamento de cada etapa do projeto.

- **Modelagem**: toda modelagem é acompanhada de sua especificação, além do processo de modelagem do negócio, que está na atividade "Comunicação". Os modelos a serem criados envolvem três arquiteturas: modelagem da informação – diagrama de classes e de pacotes; modelagem da lógica de processamento – diagramas de UC, sequência, estado e outros; e modelagem da infraestrutura de TI – diagrama de componentes e de implantação.

- **Construção**: refere-se à codificação e aos testes a serem realizados do componente de software produzido. A verificação do requisito ocorre pelo atendimento das especificações, pela inspeção de código e pelos testes unitários e de integração. As unidades são integradas e testadas como um sistema completo.

- **Entrega**: é a implantação do software no ambiente do cliente. Com o suporte necessário, feedbacks são colhidos de acordo com as observações dos usuários, tais como erros que não foram identificados anteriormente, melhoria na interface, aumento de funções para o sistema e descoberta de novos requisitos.

Embora o Cascata seja frequentemente utilizado como base para outros paradigmas de desenvolvimento, apresenta vantagens específicas em cenários que envolvem sistemas estruturados, caracterizados por requisitos bem definidos, ambientes operacionais restritos e de alta capacidade, além de aplicações que demandam a integração de múltiplos componentes em soluções complexas. Sua utilização é mais apropriada para problemas complexos ou de difícil resolução, quando o planejamento detalhado e sequencial facilita a coordenação das etapas.

Deve-se acentuar que o Cascata apresenta limitações consideráveis, especialmente devido à sua inflexibilidade na segmentação do projeto em fases distintas e sequenciais. Na prática, raramente é possível capturar todos os RS de maneira completa e precisa no início do projeto. Além disso, problemas identificados em etapas anteriores só são percebidos após a entrega do produto final, resultando em retrabalhos substanciais ou, em casos extremos, na inviabilidade do projeto. Outro ponto crítico é que ele adota uma perspectiva de alto nível que não reflete com precisão o processo real de desenvolvimento de software. Alterações nos requisitos ou no escopo durante o desenvolvimento são significativamente desestimuladas, tornando-o inadequado para projetos que usam paradigmas mais dinâmicos, como o desenvolvimento orientado a objetos.

3.4.2 Balbúrdia (codifica/corrige ou codifica/remenda)

Ilustrado na figura 24, é uma abordagem de desenvolvimento de software caracterizada pela ausência de planejamento formal, estrutura organizada ou definições claras dos requisitos.

Nesse modelo, o desenvolvimento é conduzido de forma improvisada e iterativa, com foco em resolver problemas imediatos ou entregar funcionalidades rapidamente, sem considerar uma visão de longo prazo ou práticas consolidadas de engenharia de software. Existe um ciclo contínuo de mudanças não planejadas que distancia cada vez mais o software de seu estado de maturidade e estabilidade.

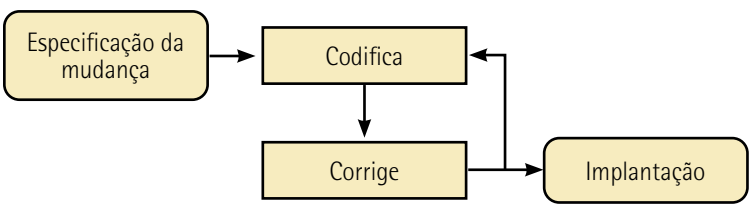


Figura 24 – Balbúrdia

Logicamente, o Balbúrdia não é um modelo a ser seguido para desenvolver software, porém inúmeros profissionais o seguem, mesmo não sabendo disso. Ele é caracterizado pela administração de crises, informalidade, loop de gestão codifica/corrige, em que o planejamento, os requisitos do software, a modelagem e a documentação são caóticos ou até mesmo inexistentes.

3.4.3 Prototipagem

> Protótipos são produtos em menor escala, modelos 2D e 3D gerados em computador ou simulações. Os protótipos permitem que as partes interessadas façam experiências com um modelo do seu produto final, em vez de somente discutirem representações abstratas dos requisitos. Os protótipos suportam o conceito de elaboração progressiva em ciclos iterativos de criação de modelos, experimentação de usuário, geração de feedbacks e revisão do protótipo. Quando ciclos de feedback suficientes forem realizados, os requisitos obtidos a partir do protótipo estarão completos o suficiente para passar para a fase de design ou de construção (PMI, 2017, p. 147).

A prototipagem tem como objetivo principal simular o comportamento e a aparência do sistema final. O projeto passa por várias investigações para garantir que o desenvolvedor, o usuário e o cliente cheguem a um consenso sobre o que é necessário e o que deve ser proposto.

Antes da entrega definitiva, são desenvolvidos protótipos que servem como representações iniciais ou intermediárias do sistema. Eles permitem identificar e analisar potenciais problemas, além de facilitar a comunicação e o alinhamento entre desenvolvedores e clientes, promovendo um entendimento mais claro dos requisitos e das expectativas do projeto.

O protótipo de software é um mecanismo para identificar requisitos de IHC, RF, práticas de UX em mapas de navegação interativos, mapa de estilos para CSS e IDE (Integrated Development Environment – Ambiente de Desenvolvimento Integrado).

As atividades da prototipagem podem ser determinadas a partir do paradigma de prototipagem, conforme ilustrado na figura 25:

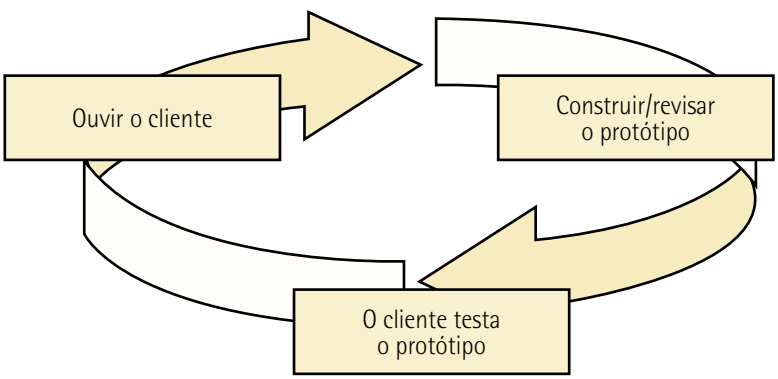


Figura 25 – O paradigma de prototipagem

Adaptada de: Pressman (2011).

Em uma análise do paradigma de prototipagem, o ciclo de atividades da prototipagem tem os seguintes estágios:

1) Começa com um conjunto simples de requisitos fornecido pelos clientes e usuários para estabelecer objetivos do protótipo. Nesse instante, é preciso ter o bom senso de selecionar apenas o que é relevante como requisito.

2) O protótipo é construído, podendo ser apenas um esboço ou algum modelo gráfico desenhado com uma ferramenta de software.

3) A revisão permite que clientes e usuários façam testes e experimentações, assim, eles decidem o que querem. Depois, os requisitos são revisados, alterados, detalhados, documentados e o sistema passa a ser codificado.

4) O estágio final do processo, como descreve Sommerville (2016), é a avaliação do protótipo. Durante esse estágio, provisões devem ser feitas para o treinamento do usuário e os objetivos do protótipo devem ser usados para derivar um plano de avaliação. Os usuários necessitam de um tempo para se sentirem confortáveis com um sistema novo e se situarem em um padrão normal de uso. Ao usar o sistema normalmente, eles descobrem erros e omissões de requisitos.

As atividades da prototipagem podem ser classificadas de acordo com a fase de prototipagem do software:

- planejamento de definição dos requisitos;
- construção de esboço (croqui);
- wireframe;
- protótipos de diversos tipos com baixo ou alto grau de detalhes;
- testes de usabilidade e feedback;
- interação;
- refinamento;
- protótipo final.

Devido à falta de familiaridade dos usuários com aspectos técnicos avançados, esse método de desenvolvimento é particularmente eficaz, permitindo uma participação significativa dos usuários no processo de interação com a solução proposta.

Entre as vantagens da prototipagem, destacam-se:

- **Ciclo de vida eficiente**: permite ao desenvolvedor criar um modelo representativo do software a ser construído.

- **Adequação a objetivos definidos**: é apropriada quando o cliente definiu os objetivos gerais do software, mas ainda não especificou ou identificou detalhes dos requisitos de entrada, processamento e saída.

- **Identificação antecipada de requisitos**: facilita a análise conjunta entre desenvolvedor, cliente e usuário; mitigando riscos e incertezas durante o desenvolvimento. O crucial é definir as regras do jogo logo no começo.

- **Aceleração do desenvolvimento**: proporciona uma visão mais tangível do software em desenvolvimento para o usuário, permitindo a visualização antecipada de telas e relatórios resultantes.

- **Envolvimento direto do usuário**: transformando-o em um coautor do projeto, o que aumenta a precisão das especificações e melhora a usabilidade do produto final.

Como desvantagens, acentuam-se:

- **Percepção do cliente sobre a totalidade do software**: quando o cliente não está ciente de que o software em teste é apenas um protótipo, não considera a qualidade global ou a manutenibilidade necessária durante a operação do software, isso pode levar a expectativas mal alinhadas. O simples fato de o usuário clicar em um botão do protótipo pode fazê-lo questionar, por exemplo, que o botão não está "chamando" o algoritmo da função. O algoritmo pode ainda nem existir, isto é, o que está sendo apresentado é apenas um layout de tela.

- **Expectativas do usuário**: a prototipagem pode criar a falsa impressão de que todas as sugestões dos usuários podem ser facilmente implementadas, independentemente da fase do desenvolvimento. Além disso, os usuários podem não entender o motivo da demora na entrega final após visualizar uma versão demo, levando a frustrações e pressões para que o protótipo seja utilizado como produto final.

Um dos exemplos de protótipos de tela pode ser visto na figura 26. Ela mostra o resultado da prática de diagramação (layout ou wireframe) de tela, que é um protótipo que mostra espaços reservados para os signos de imagem: caixas de texto, caixas de imagem, botões, barras de rolagem, enfim, todos os elementos necessários de IHC.

Em portais e diversos sites, é comum ter um ou mais modelos de telas diagramados. Por vezes, esses espaços são vendidos, reservados por empresas para publicidade, propaganda, notícias e outras aplicações.

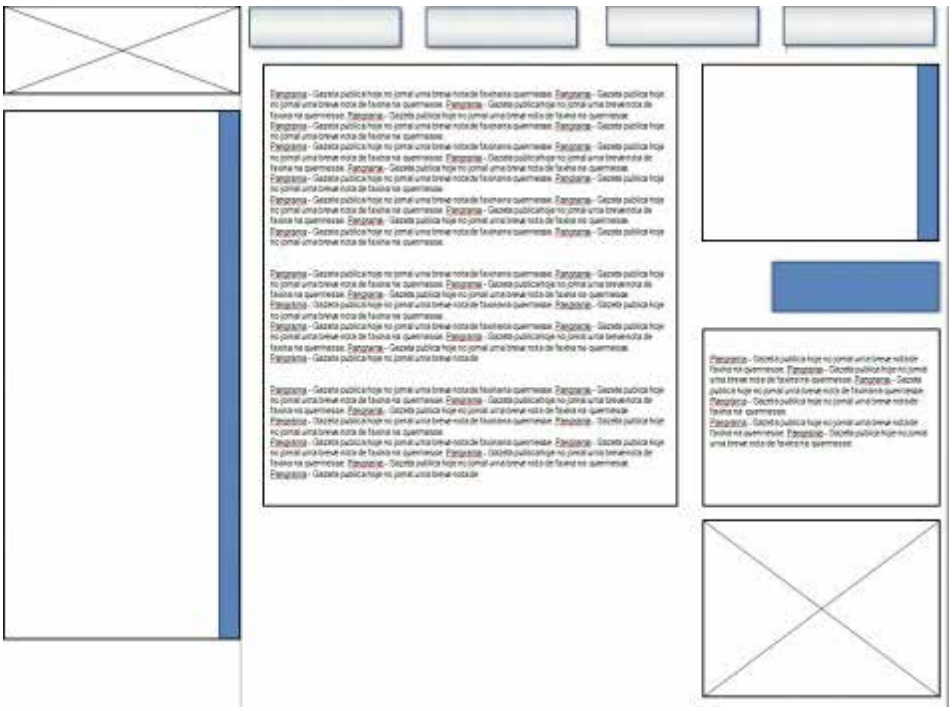


Figura 26 – Diagramação de tela web

Saiba mais

A engenharia de usabilidade se refere a conceitos e técnicas para análise, síntese e avaliação de usabilidade em sistemas (Preece, 2002 *apud* Martins, 2011).

A usabilidade a ser avaliada no projeto diz respeito à relação do software com as tarefas dos usuários e com o objetivo de confirmar o nível em que os requisitos da organização e dos usuários foram alcançados, fornecendo também informações para o refinamento do projeto.

Para entender melhor como elaborar um teste de usabilidade, consulte o site de Neil Patel.

Disponível em: https://shre.ink/egup. Acesso em: 23 jun. 2025.

Figura 27 – Design de IU

Disponível em: https://shre.ink/xEeE. Acesso em: 23 jun. 2025.

3.4.4 Incremental

É projetado para atender as exigências empresariais e usuais de prazos acelerados e alta qualidade. Ilustrado na figura 28, esse modelo operacionaliza a liberação do sistema em partes incrementais, cada uma delas entregando funcionalidades utilizáveis enquanto o desenvolvimento contínuo prossegue em paralelo.

Esse modelo incorpora elementos sequenciais do Cascata de forma evolucionária. Ele permite a entrega de incrementos iterativos com base em marcos de entrega, aprovação e validação, assegurando que o usuário utilize partes do sistema em desenvolvimento desde cedo, minimizando o tempo de espera total e adaptando-se às necessidades emergentes ao longo do ciclo de desenvolvimento.

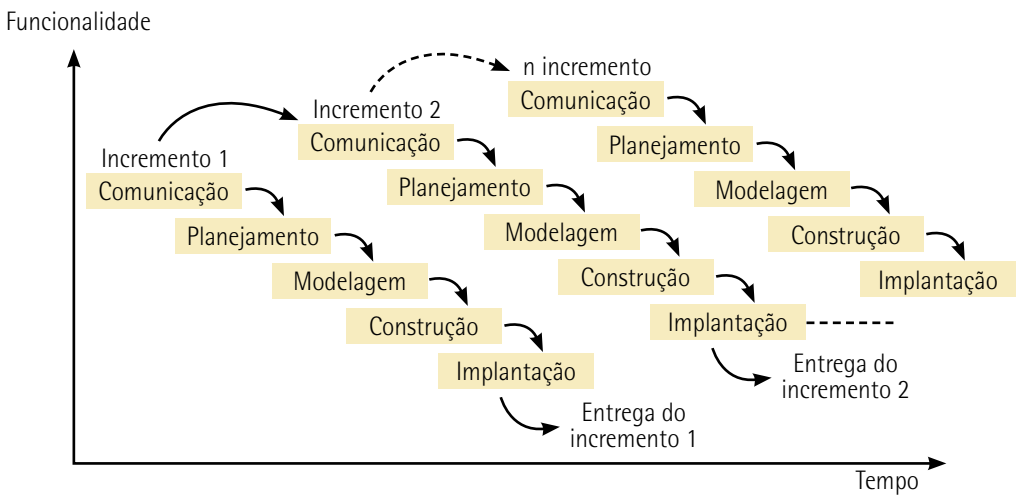


Figura 28 – Modelo de processo incremental

Adaptada de: Pressman (2007).

O processo incremental é caracterizado por sua natureza iterativa, com o objetivo de apresentar um produto operacional com cada incremento. Esse método é fundamentado no conceito de versões, quando iterações sequenciais são utilizadas para aprimorar continuamente o software.

Observação

Iteração é uma estratégia de planejamento que envolve retrabalho no processo de desenvolvimento, revisões de prazos, identificação e correção de falhas, erros e melhorias tecnológicas. Esse ciclo contínuo visa melhorar o sistema e distribuir as tarefas de forma eficiente.

A cada iteração, novas versões são geradas, registrando as mudanças implementadas no software até a entrega do release, que é a versão finalizada para os usuários. As iterações do processo são necessárias para a melhoria contínua do processo de software.

Nesse modelo, o sistema é especificado na documentação dos requisitos e decomposto em subsistemas funcionais. As versões são inicialmente definidas a partir de um pequeno subsistema funcional, com novas funcionalidades sendo adicionadas a cada versão seguinte. O software alcança sua funcionalidade total gradualmente por meio desses incrementos sucessivos.

3.4.5 RAD

Outros modelos de desenvolvimento apresentam grande utilidade, muitos dos quais resultam da combinação de conceitos de diferentes metodologias, como é o RAD. Trata-se de uma abordagem mista, com características genéricas, sendo amplamente utilizada no desenvolvimento de software.

O RAD permite que uma equipe de desenvolvimento crie um sistema totalmente funcional em um período de tempo reduzido. Conforme mencionado por Pressman (2002), o modelo RAD é um processo de software incremental que enfatiza um ciclo de desenvolvimento curto.

É uma abordagem que visa à entrega rápida de software, frequentemente utilizando ferramentas especializadas de programação de bancos de dados e apoio ao desenvolvimento, como geradores de interfaces e relatórios (Sommerville, 2011).

O RAD adota uma estratégia baseada na construção modular e orientada a componentes reutilizáveis. Ele permite acelerar o desenvolvimento de sistemas, reduzindo significativamente o tempo necessário para a entrega de soluções de complexidade moderada.

Conforme ilustrado na figura 29, é possível concluir o desenvolvimento completo de um sistema nesse modelo em um intervalo de 60 a 90 dias, dependendo da complexidade dos requisitos e da integração eficiente dos componentes.

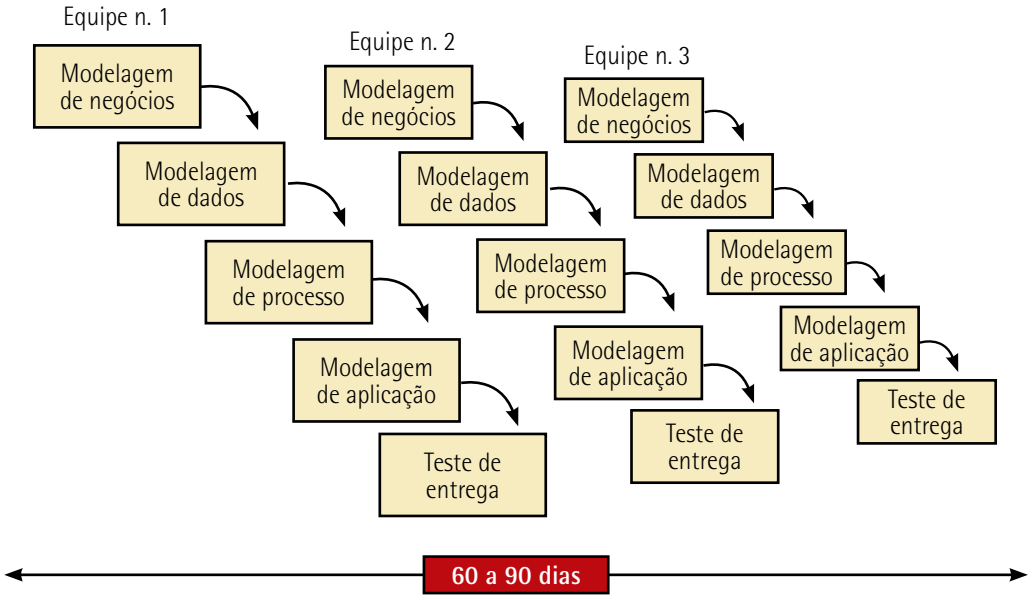


Figura 29 – Modelo RAD

Adaptada de: Pressman (2007).

As principais vantagens desse modelo são:

- **Ciclo de desenvolvimento curto**: segue uma abordagem sequencial linear com ciclos de desenvolvimento significativamente reduzidos, permitindo entregas em intervalos curtos, geralmente entre 60 e 90 dias.

- **Modularidade e componentização**: a construção baseada em componentes reutilizáveis permite modularizar o sistema, facilitando o gerenciamento do projeto e a identificação precoce de desvios no escopo.

- **Requisitos claramente delimitados**: os requisitos iniciais devem ser suficientes para viabilizar o escopo do projeto, otimizando o foco da equipe.

- **Escalabilidade para sistemas grandes**: o modelo é ideal para aplicações de sistemas de informação extensos e modularizados, possibilitando que funções ou módulos principais sejam atribuídos a equipes independentes e que depois integrarão os componentes ao sistema completo.

- **Visibilidade incremental**: a entrega contínua de incrementos oferece maior visibilidade ao cliente, que pode acompanhar e validar partes do sistema ao longo do desenvolvimento.

Entre suas principais desvantagens, acentuam-se:

- **Dependência de modularidade**: o sucesso do RAD depende da adequada modularização do sistema. Caso a segmentação dos componentes seja ineficiente, o desenvolvimento pode enfrentar dificuldades significativas.

- **Inadequação para riscos técnicos elevados**: o modelo pode não ser apropriado quando o projeto envolve tecnologias emergentes ou pouco dominadas pela equipe, aumentando os riscos técnicos.

- **Necessidade de recursos humanos intensivos**: a abordagem exige equipes dedicadas para cada módulo ou funcionalidade, o que pode ser inviável em projetos com limitações de pessoal.

- **Comprometimento intenso**: tanto desenvolvedores quanto clientes devem estar altamente comprometidos com as atividades de ritmo acelerado ("fogo-rápido") para garantir que os prazos curtos sejam cumpridos.

- **Limitações em aplicações específicas**: nem todos os tipos de sistemas ou aplicações se adaptam bem ao modelo RAD. Projetos que requerem alta integração ou sincronização entre componentes podem enfrentar dificuldades com essa abordagem.

- **Dependência de sincronismo e desempenho**: quando o desempenho do sistema está fortemente vinculado ao sincronismo das interfaces entre componentes, o RAD pode não ser eficiente, comprometendo o resultado final.

3.4.6 Espiral

Conforme ilustrado na figura 30, é uma abordagem evolucionária que integra a natureza iterativa da prototipagem com a estrutura sequencial do Cascata. O desenvolvimento do software ocorre por meio de múltiplas versões incrementais, evoluindo progressivamente a partir de ciclos iterativos.

Em cada iteração da espiral, são realizadas atividades estruturais previamente definidas no arcabouço, abrangendo desde o planejamento e a análise de riscos até a implementação e a avaliação.

Ao final de cada ciclo, uma versão parcial ou completa do sistema é entregue (release), com novas funcionalidades ou melhorias incorporadas. Após múltiplas iterações, o software atinge sua totalidade funcional, contemplando todos os requisitos estabelecidos.

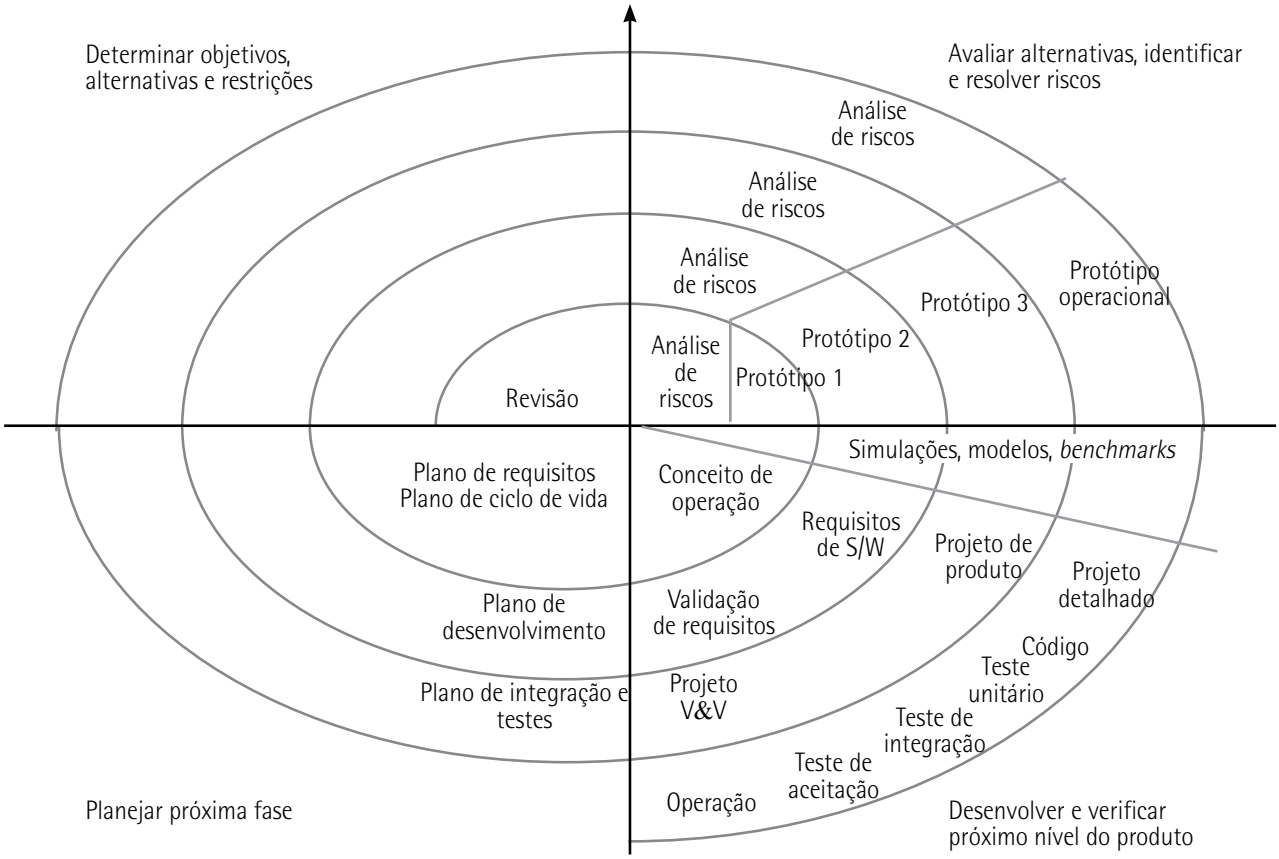


Figura 30 – Modelo de processo espiral

Fonte: Sommerville (2016, p. 33).

O modelo Espiral é adequado para sistemas complexos, projetos de software de grande porte e aqueles que demandam um alto nível de interação com os usuários. Sua abordagem iterativa permite identificar e resolver problemas progressivamente, garantindo maior alinhamento com os requisitos e os objetivos do sistema.

O modelo é dividido em quatro etapas principais, que se repetem ao longo de cada ciclo da espiral:

- **Ativação**: definem-se os objetivos específicos, identificam-se as restrições para o processo e é preparado um plano de gerenciamento detalhado. Identificam-se também os riscos sem analisá-los profundamente (foco da próxima fase).

- **Análise de riscos e prototipagem**: com base nos riscos identificados na fase anterior, são realizadas análises detalhadas e tomadas providências para amenizar esses riscos. Criam-se várias versões de protótipos para apoiar essa fase.

- **Desenvolvimento**: fundamentado pelas fases anteriores, escolhe-se o modelo mais adequado para o desenvolvimento do software. A experiência profissional e a vivência do desenvolvedor em outros sistemas são estratégicas para essa fase. Dependendo da complexidade ou do tamanho do sistema, por vezes, é necessária a presença de um consultor especialista.

- **Planejamento**: o projeto é revisto nessa fase e é tomada uma decisão de realizar um novo ciclo, na espiral ou não. Se continuar com o aperfeiçoamento do sistema, é traçado um plano para a próxima fase do projeto. Um diferencial nesse modelo comparado com outros é a explícita consideração de riscos dentro do projeto como um todo. Para tanto, criou-se uma fase específica de análise de riscos nesse modelo.

4 PLANEJAMENTO DO PROCESSO DE SOFTWARE

Refere-se à definição detalhada das atividades, metodologias e práticas a serem adotadas ao longo do ciclo de vida do desenvolvimento de software, com o objetivo de garantir a entrega de um produto de alta qualidade, dentro dos requisitos de tempo e custo estipulados.

Cada modelo de processo representa o processo em uma perspectiva particular, por exemplo, o Incremental é acompanhado do versionamento, que acompanha a evolução ou as mudanças do software por meio de registros de entrega, como uma funcionalidade. No caso Incremental, são apresentadas as atividades sequenciais desenvolvidas ao longo de seu ciclo; no entanto, não mostra a definição dos papéis e das responsabilidades das pessoas envolvidas no projeto.

De acordo com Sommerville (2016), um modelo de processo de software é uma representação simplificada de suas atividades, que acompanham todo o SDLC (Software Development Life Cycle – Ciclo de Vida de Desenvolvimento de Software).

O planejamento relacionado ao processo de desenvolvimento acompanha cada etapa do SDLC, o que inclui abordagens técnicas na escolha dos seguintes aspectos:

- modelos de processo de desenvolvimento;
- definição de metodologias e frameworks de como as atividades serão executadas;
- planejamento de iterações e releases;
- gestão da qualidade e testes;
- gestão e alocação de recursos;
- gestão do risco;
- definição das ferramentas que automatizam cada uma das atividades ou tarefas a serem executadas.

4.1 SDLC

Essencialmente, ele é adaptado para uma estrutura organizacional específica, ilustrada na figura 31. Esta detalha as diferentes fases da organização, que devem ser alinhadas a um ciclo de vida, e as inter-relações entre elas, que afeta diretamente como cada fase do ciclo é planejada, executada e monitorada.

O alinhamento do SDLC com estruturas organizacionais envolve as atividades, as funções (papéis) das pessoas, as responsabilidades e os fluxos de trabalho dentro da organização do processo de desenvolvimento.

A estrutura influencia o grau de autoridade e de responsabilidades, que podem ser centralizadas ou distribuídas, podendo ter uma estrutura horizontal ou operar com equipes de projeto auto-organizáveis, como é o caso das metodologias ágeis.

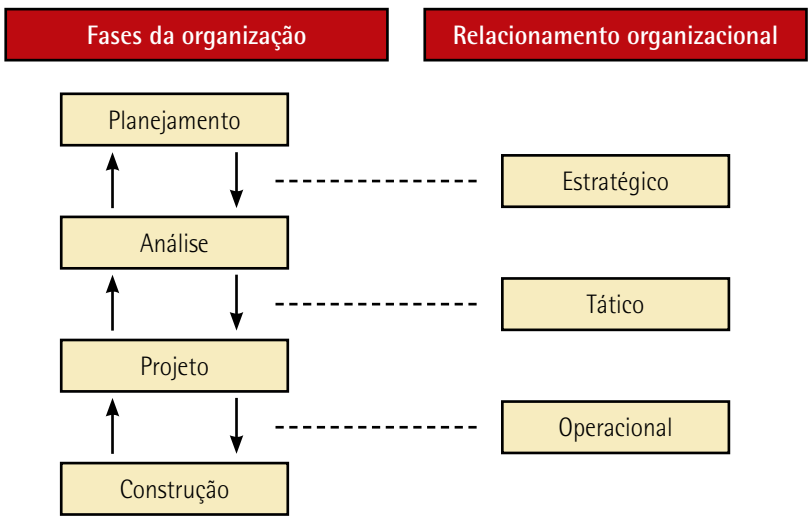


Figura 31 – Estrutura para o desenvolvimento de software com suas fases e formas de relacionamento

O ciclo de vida do processo de software, acompanhado por um cronograma, envolve planejar, monitorar e controlar o tempo dedicado a cada fase do ciclo de vida.

O cronograma é uma ferramenta prática para gerenciar o tempo e as entregas do projeto de software, enquanto o ciclo de vida do processo define as etapas estruturadas pelas quais o desenvolvimento do software passa. A integração de ambos visa garantir que o software seja entregue no tempo previsto, atendendo aos requisitos e padrões de qualidade.

No planejamento do processo de software, cada fase da estrutura organizacional é acompanhada de seus processos e cada um deverá ser executado com base em métodos, ferramentas e técnicas, que fazem parte das tarefas da equipe de desenvolvimento. O quadro 4 exemplifica algumas das escolhas de MF&T.

Quadro 4

Fase da organizacão	Especificação
Planejamento	É a fase em que são levantados os requisitos do negócio e os requisitos não funcionais, com base nas arguições sobre os negócios, as necessidades de clientes e usuários, e as restrições do software Também são acordados custos e prazos, efetivados após a análise
Análise	Ocorre após o levantamento de requisitos do negócio e consiste em coletar dados, definir modelos de processo de software, especificar requisitos funcionais e do sistema, determinar as atividades do desenvolvimento do software, medir o escopo do sistema e avaliar a relação necessidade/capacidade (ou custo/benefício) São elaborados cronogramas, medição de capacidade da equipe de desenvolvimento, análises de caminho crítico, modelos do processo de negócios, UC, diagrama de atividades e diagramas de componentes e implantação
Projeto	Após a viabilidade do projeto (ou design), usa-se a engenharia de software para escolher procedimentos de serviços, metodologias e técnicas apropriadas para o controle e a operacionalização do sistema que atendam aos requisitos. Definem-se as interfaces, a estrutura da informação e as topologias Criam-se diagramas de entidades e relacionamento dos dados com base em protótipos, modelos de testes (unitário de integração), classes, diagramas de sequência, diagramas de estado e outros, normalmente desenvolvidos em UML O desenvolvimento do software ocorre na sequência, quando são definidos frameworks para codificação, linguagens de programação e SGBD associados, formas de testar e diagnosticar o código
Construção	A construção está associada à implementação, quando ocorrem processos como codificação, testes, implantação do software, V&V, entrega, manutenção, treinamento e suporte técnico aos usuários Nota: a fase de desenvolvimento normalmente realiza 90% do software e os outros 10% são desenvolvidos após a execução. O fato é que os 90% previsíveis ocupam a metade do tempo total para a finalização do software, ou seja, a outra metade é ocupada na implementação de código devido a mudanças requeridas. Novos problemas não previstos surgem: correções, mudança de requisitos, melhorias no software e mudanças no ambiente operacional

Observação

Quais são as melhores técnicas e métodos da engenharia de software?

Embora todos os projetos de software devam ser desenvolvidos e gerenciados profissionalmente, diferentes técnicas são apropriadas para diferentes tipos de sistemas. Por exemplo, os jogos devem sempre ser elaborados usando uma série de protótipos, enquanto sistemas de controle críticos de segurança requerem que uma especificação completa e analisável seja desenvolvida. Não existem métodos e técnicas que sirvam para tudo (Sommerville, 2016).

4.2 Aspectos humanos da engenharia de software

Software é criado por pessoas, usado por pessoas e dá suporte à interação entre pessoas. Assim, características, comportamentos e cooperações humanas são vitais no desenvolvimento prático de software (Pressman, 2021).

São vários os profissionais que escrevem programas de software. Pessoas envolvidas na área de negócios criam programas para planilhas na automatização dos processos operacionais; cientistas e engenheiros elaboram software para processamento e análise de dados experimentais; e há pessoas que usam a programação por interesse ou para aperfeiçoar seus conhecimentos.

Contudo, a maior parte do desenvolvimento de software é feita por profissionais com a função de solucionar problemas específicos de negócios, construir algoritmos para cálculos básicos e científicos, IHC, interfaces entre dispositivos ou computadores do software, protocolos de comunicação, software para web etc.

O engenheiro de software é quem precisa dominar uma diversidade de processos, métodos, ferramentas e técnicas da engenharia de software, desenvolver habilidades para entender problemas ou necessidades para projetar soluções com qualidade.

De acordo com Pressman (2021), existem sete características pessoais que demonstram competência do engenheiro de software:

- **Responsabilidade individual**: determinação em cumprir promessas para gerência, membros da equipe e stakeholders.
- **Consciência aguçada**: ter um estado de percepção elevada, sensibilidade ao ambiente, capaz de se adaptar a emoções, mudanças, comportamentos e necessidades da equipe.
- **Extremamente honesto**: ser transparente, realista e sincero de forma construtiva para conduzir problemas e falhas.
- **Resiliência sob pressão**: a engenharia de software está sempre à beira do caos. O engenheiro de software é capaz de suportar a pressão de modo que seu desempenho não seja prejudicado (Pressman, 2021).
- **Lealdade**: à equipe e ao projeto, tendo boa vontade para colaborar e compartilhar conhecimentos, além de evitar conflitos e não sabotar o trabalho de outros. O profissional age com lealdade ao cliente e às expectativas, sempre com foco no usuário, pois seus objetivos envolvem garantir a qualidade do software entregue ao cliente e ser honesto sobre o progresso, os desafios e os possíveis atrasos no projeto.
- **Atenção aos detalhes**: não significa ser perfeccionista. O engenheiro de software deve estar atento a prazos, requisitos, desempenho, custo e qualidade estabelecidos para o projeto e o software.

- **Pragmático**: reconhece que a engenharia de software não é uma religião na qual devem ser seguidas regras dogmáticas, mas sim uma disciplina que pode ser adaptada conforme as circunstâncias (Pressman, 2021).

A estrutura da equipe e os fatores sociais governam o sucesso. A comunicação, a colaboração e a coordenação do grupo são tão importantes quanto as habilidades dos membros individuais da equipe (Pressman, 2021).

4.3 Análise de recursos para os processos do projeto

A análise de recursos para os processos do projeto é uma atividade que envolve a gerência administrativa e analistas. Essa análise envolve a identificação, a avaliação e a alocação dos recursos necessários para garantir que os processos do projeto sejam executados de maneira eficiente, eficaz e dentro das restrições, como prazo, custo e qualidade.

O PMBOK (PMI, 2021) orienta que na gerência de projetos pode haver áreas de sobreposições que ele chama de "princípios de gerenciamento de projetos" e "princípios do gerenciamento geral", conforme ilustrado na figura 32.

Praticado pela gerência administrativa, o princípio de gerenciamento de projetos inclui processos para identificar, adquirir e gerenciar os recursos necessários para a conclusão bem-sucedida do projeto, bem como comprar ou adquirir produtos, serviços ou resultados externos à equipe e subcontratações.

Por sua vez, o princípio do gerenciamento geral, praticado pela gerência da TI, refere-se ao projeto do software.

Observe com atenção. O princípio do gerenciamento geral é aplicado na produção do negócio, que pode estar ligado a diversas áreas da indústria em geral, incluindo produção de software, comércio, medicina, instituições etc. Já o princípio de gerenciamento de projetos corresponde especificamente à área administrativa ligada a fornecedores, contratos, parcerias e transporte.

Os padrões de qualidade, processos, métodos e ferramentas são diferentes entre as áreas administrativas e a de TI. No entanto, essas gerências têm algumas práticas em comum, conforme ilustrado a seguir.

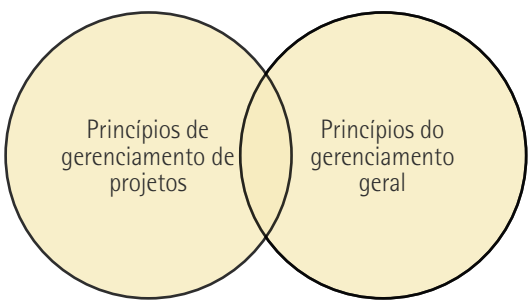


Figura 32 – Sobreposição dos princípios do gerenciamento de projetos e geral

Os objetivos da análise de recursos são:

- **Garantir as metas do projeto**: avaliar se os recursos disponíveis atendem às necessidades para atingir os objetivos do projeto.

- **Minimizar riscos**: identificar lacunas ou pontos críticos nos recursos que possam comprometer prazos, custos ou qualidade do produto.

- **Aumentar a eficiência**: alocar recursos de forma otimizada para evitar desperdícios e sobrecarga da equipe.

4.3.1 Recursos do projeto e do software

Na projeção dos recursos para atender os requisitos do software, três perfis de analistas são essenciais:

- **Analista de negócios (ou autor do processo)**: responsável por identificar as necessidades do negócio, sejam elas de geração, correção ou melhoria; visando atender objetivos ou necessidades específicas. Ele utiliza uma linguagem natural e diagramas simples para comunicar claramente o que é exigido e o que precisa ser feito.

- **Analista de processos**: interpreta a ideia do negócio, avaliando seus riscos e suas regras, com o objetivo de determinar as atividades e as tarefas necessárias para processar o negócio. Ele busca padronizar a nomenclatura e especificar, por meio de textos concisos e diagramas detalhados, a ordenação das atividades e das tarefas que compõem o processo do negócio.

- **Analista de sistemas**: converte as atividades em componentes específicos que integram o processo. Cada componente representa uma parte do software suportado pelo processamento de dados. A interpretação das tarefas e do workflow (fluxo de trabalho) leva à criação das funções que irão operacionalizar o software, a lógica de processamento (programa de computador) e os recursos computacionais necessários para executar essas funções.

Na prática da engenharia de software, o analista de sistemas especifica cada função através de descrições detalhadas e modelos gráficos (modelagem), determinando as funcionalidades que serão interpretadas pelos especialistas em TI.

Com base nos requisitos definidos pelos analistas de negócios, do processo e de sistemas, para desenvolver o software serão necessárias duas gerências: a de projeto e a de TI. Vamos estudá-las a seguir.

Gerência do projeto

Promove o projeto em todos os aspectos, o que significa apoiar, defender e divulgá-lo na organização tanto interna como externamente. Essas ações ajudam a criar um ambiente positivo e de suporte ao redor do projeto, aumentando as chances de sucesso e de adesão tanto dos stakeholders quanto da equipe de desenvolvimento.

Um stakeholder é qualquer pessoa, grupo ou organização que tem interesse ou é afetado pelo resultado de um projeto, produto ou negócio; envolve as partes interessadas que podem influenciar ou serem influenciadas pelo que está sendo desenvolvido ou implementado.

O gerente de projeto:

- define os limites e os objetivos do projeto;
- mantém uma comunicação clara e contínua com stakeholders e a equipe de desenvolvimento;
- desenvolve, analisa e entende os processos envolvidos conforme os requisitos;
- aloca tarefas e responsabilidades de acordo com as habilidades e competências;
- estabelece uma estrutura organizada para administrar e monitorar o progresso da equipe;
- estima e controla os custos do projeto de acordo com o orçamento.

Os principais recursos sob a gerência do projeto incluem:

- **Recursos humanos**: os recursos humanos envolvidos em um projeto variam de acordo com o tamanho, a complexidade e os objetivos do projeto. Esses setores atuam no suporte, na coordenação e na governança, garantindo que o projeto seja conduzido de forma eficiente e alinhada aos objetivos organizacionais. Incluem todos os envolvidos e interessados, os coordenadores (que auxiliam na comunicação entre equipes), os analistas etc. Acentua-se, por fim, que os recursos consumidos pela gerência de TI também estão sob a gerência do projeto.

- **Recursos materiais**: referem-se a localização, transporte, mobília e instalações para o ambiente de uso e ambiente de desenvolvimento do software. O objetivo é criar um ambiente de trabalho eficiente e confortável, que suporte as atividades do desenvolvimento e os testes de software. A gerência de TI é responsável por fornecer a lista de recursos tangíveis consumidos pela equipe de desenvolvimento.

- **Recursos tecnológicos**: envolvem os recursos intangíveis fornecidos pela gerência de TI, como ferramentas de software, licenças, plataformas de desenvolvimento e infraestrutura de TI. Esses recursos são essenciais para o desenvolvimento de software, garantindo que a equipe tenha as tecnologias necessárias para atender às expectativas dos usuários em termos de funcionalidade e desempenho.

- **Recursos financeiros**: fornecem o financiamento necessário para cobrir todos os custos associados ao desenvolvimento do software, incluindo salários, ferramentas, infraestrutura e outros recursos. A gestão eficiente dos recursos financeiros é crucial para garantir que o projeto seja concluído dentro do orçamento e atenda às expectativas dos stakeholders e usuários finais.

Gerência da TI

Ela envolve a administração dos recursos tecnológicos necessários para construir ou produzir software.

Criar ou produzir software pode estar associado ao seu desenvolvimento por completo: módulo, componente ou subproduto dele. O gerenciamento da TI corresponde às fases do ciclo de vida do desenvolvimento de software, que incluem áreas específicas da aplicação.

No ambiente computacional informatizado, o desenvolvimento do software abrange a administração da infraestrutura de TI, que suporta o desenvolvimento de sistemas em uma sequência ordenada que acompanha esse ciclo. Os principais recursos tecnológicos necessários para desenvolver software incluem:

- **Recursos de software**: referem-se a ferramentas, plataformas e soluções digitais fáceis para desenvolvimento, gestão, implantação e manutenção de sistemas e aplicativos. Esses recursos são classificados como intangíveis e desempenham um papel essencial em todas as etapas do ciclo de vida do desenvolvimento de software. Os principais recursos de software são:

  – Ferramentas de desenvolvimento, a exemplo de IDEs como Visual Studio, Eclipse, Netbeans e PyCharm. São ferramentas que oferecem funcionalidades como edição de código, depuração e execução. O engenheiro de software tem ao seu dispor inúmeros outros dispositivos que dão apoio ao desenvolvimento: aparelhos de modelagem e arquitetura, ferramentas CASE (Computer-Aided Software Engineering – Engenharia Assistida por Computador), ferramentas de gerenciamento de projetos e muitas outras.

  – Editores de código como VS Code, Sublime Text ou Notepad++.

  – Compiladores e interpretadores para linguagens de programação, o software traduz o código-fonte para o código correspondente ao da máquina, por exemplo: GCC (GNU Compiler Collection – Coleção de Compiladores GNU) e JDK (Java Development Kit – Kit de Desenvolvimento Java).

  – Licenças de software ou utilitários para manutenção do sistema no ambiente do desenvolvedor, tais como sistemas operacionais, aplicativos e gerenciadores de banco de dados. O software livre que estiver útil no desenvolvimento deve compor essa lista.

- **Recursos de hardware**: são recursos tangíveis que garantem o desempenho necessário para executar o software com bom desempenho, oferece suporte para o desenvolvimento de sistemas escaláveis e seguros e que possibilitam conexões e operações em ambientes coorporativos. Os principais recursos de hardware são:

  – Computadores servidores e estações de trabalho usadas por desenvolvedores, analistas, programadores, gerenciadores do banco de dados e testadores. São considerados como estações de trabalho: desktops, notebooks e laptops.

- Dispositivos de armazenamento para o ambiente de desenvolvimento e para infraestrutura de serviços, bem como dispositivos de backups.

- Periféricos, componentes como teclado, mouse, monitores e impressoras.

- Acessórios, dispositivos de entrada como webcam, tablets ou scanner.

- Infraestrutura de rede, tais como placas de rede, roteadores, modem, switches, pontos de acesso, cabos, firewalls e dispositivos de segurança.

- Infraestrutura física e de suporte como racks, ar-condicionado, nobreak, câmeras de segurança, sensores de movimento e dispositivos inteligentes.

- **Recursos humanos (peopleware)**: são reservados aos especialistas que trabalham no desenvolvimento do software. Esses profissionais têm diferentes responsabilidades e especializações. Os principais peopleware são:

  - Gestores e líderes, como o gerente do projeto de software, que coordena todas as atividades do projeto, gerencia recursos, cronogramas e riscos e mantém a comunicação com os stakeholders. Também são citados o product owner, que representa os interesses dos stakeholders e dos usuários, e o Scrum master, que facilita o processo ágil, removendo obstáculos e garantindo que a equipe siga as práticas Scrum.

  - Programadores: escrevem o código, implementam funcionalidades e resolvem problemas técnicos. Acentua-se que o front-end tem como foco a IU. Já a UX usa tecnologias como HTML, CSS e JavaScript. Por sua vez, os back-end trabalham na lógica do servidor, dos bancos de dados e da integração de sistemas e utilizam tecnologias como Java, Python, Ruby e outras.

  - IU e UX: são os especialistas em usabilidade. Eles analisam o uso do software e fazem recomendações de melhorias. O designer de UI/UX cria visuais atraentes e intuitivos para melhorar a experiência.

  - Equipe de suporte e manutenção.

  - Outros profissionais que atuam no desenvolvimento de software, como o engenheiro de software, responsável pela criação de soluções tecnológicas que atendem a necessidades complexas e, ao mesmo tempo, escaláveis e sustentáveis, acompanhando todo o ciclo de vida do software. Citam-se ainda engenheiros de DevOps, responsáveis por integrar desenvolvimento e operações para automatizar processos de construção, teste e implantação; e os especialistas em segurança, que garantem que o software seja seguro contra ameaças e vulnerabilidades.

- **Recursos de dados**: envolvem qualquer tipo de informação ou dados que podem ser usados para análises, tomada de decisões ou alimentar sistemas de software. Permitem manipular a informação de forma eficiente e garantem a entrega de soluções que atendam às demandas do negócio e dos usuários. Os principais recursos de dados são:

  – **Dados coletados diretamente da fonte**: dados estruturados como tabelas, dados não estruturados como imagens, vídeos, documentos e demais extensões de arquivos.

  – **Base de dados (databases)**: conjuntos organizados de dados armazenados e acessados eletronicamente. Incluem bancos de dados relacionais (SQL): MySQL, Microsoft SQL, PostgreSQL e Oracle; não relacionais (NoSQL): MongoBD e Cassandra; em nuvem: Azure Cosmos DB e AWS RDS.

  – **APIs (Application Programming Interfaces – Interface de Programação de Aplicações)**: englobam um repositório de rotinas e ferramentas para construir o software, permitindo acesso e interação com dados de outros sistemas.

  – **Streams de dados**: são fluxos contínuos de dados em tempo real, como logs de servidor, dados de sensores, redes sociais e plataformas de vídeo.

  – **Data Warehouse e Big Data**: repositórios de grande volume de dados e múltiplas fontes que fornecem como resultados relatórios analíticos para tomada de decisões.

  – **Segurança e proteção de dados**: como criptografia, controle de acesso, backups, monitoramento e auditoria.

- **Recursos da rede de computadores**: são serviços e componentes associados que permitem uma conectividade com bom desempenho, segura e escalabilidade. O investimento em recursos de rede traz um bom funcionamento de qualquer infraestrutura de tecnologia da informação, e os principais deles são:

  – **Recursos físicos**: componentes tangíveis, responsáveis pela conectividade e pela infraestrutura de rede. Eles devem compor a lista dos recursos de hardware e suas configurações fazem parte dos recursos da rede de computadores. Os principais componentes são: roteadores, switches, access points (pontos de acesso de dispositivos Wi-Fi em redes sem fio), modems, servidores de rede, periféricos associados, cabos e conexões.

  – **Infraestrutura de comunicação**: meios físicos e lógicos que suportam a troca de dados como topologias de rede, mapas de endereçamento, backbones e conectividade de nuvem.

– **Recursos lógicos**: protocolos de rede como TCP (Transmission Control Protocol – Protocolo de Controle de Transmissão), HTTP/HTTPS, FTP, DNS e SMTP/IMAP/POP3. Citam-se também sistemas operacionais de rede (SOR) como Windows Server, UNIX e Linux (as licenças dos sistemas operacionais fazem parte dos recursos de software).

– **Armazenamento de rede (network storage)**: dispositivos ou sistemas de armazenamento acessíveis através da rede, como NAS (Network Attached Storage – Armazenamento Conectado à Rede) e SAN (Storage Area Network – Rede de Área de Armazenamento).

– **Recursos de segurança e controle**: ferramentas e práticas para proteger a integridade e a confidencialidade da rede, como firewall, VPN, criptografia de dados, antivírus, IPS (Intrusion Prevention System – Sistema de Prevenção de Intrusão), controle de acesso e monitoramento da rede.

4.4 Métodos, ferramentas e técnicas: modelagem da infraestrutura de TI

No projeto de software, basicamente são desenvolvidas três arquiteturas: lógica de processamento, informação e infraestrutura de TI. Elas foram abordadas nas atividades de modelagem, comentadas quando estudamos o Cascata.

As arquiteturas da lógica de processamento, da informação e da infraestrutura de TI, assim como diversos modelos derivados, compõem um repertório de alternativas de componentes para construir sistemas de software que ficam armazenados em um repositório e disponíveis para os desenvolvedores.

Observação

O repositório do desenvolvimento de sistemas de software é um local de armazenamento de "peças" de software para sua construção.

O repositório do desenvolvimento de sistemas (figura 33) não apenas armazena dados e informações, mas é útil para a gestão e o compartilhamento de vários elementos do projeto de software, como componentes, módulos, objetos, projetos, modelos, ferramentas de software, documentos ou quaisquer outros artefatos produzidos para implementar ou construir sistemas de software.

Normalmente, as ferramentas de desenvolvimento da indústria de software de qualquer tamanho trabalham com seu repertório de alternativas para o projeto de software. Elas ficam armazenadas em repositórios, que podem estar alocados na estação de trabalho, como é o caso da aplicação Astah Community, ou alocados em servidor ou nuvem, a exemplo do RUP.

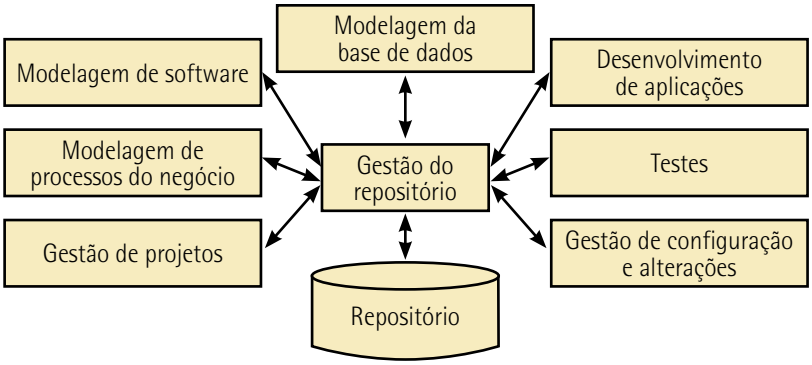


Figura 33 – Arquitetura de ferramentas de software com repositório para desenvolvimento de sistemas de software

A reusabilidade de componentes de software é um princípio da engenharia de software, uma métrica da qualidade usada para avaliar o quanto um programa ou parte dele pode ser usada em outras aplicações. Tem como foco a criação de módulos de código que podem ser utilizados em diferentes aplicações, sem a necessidade de retrabalho significativo. A figura 34 mostra o conteúdo de um repositório com um repertório de alternativas de componentes de software.

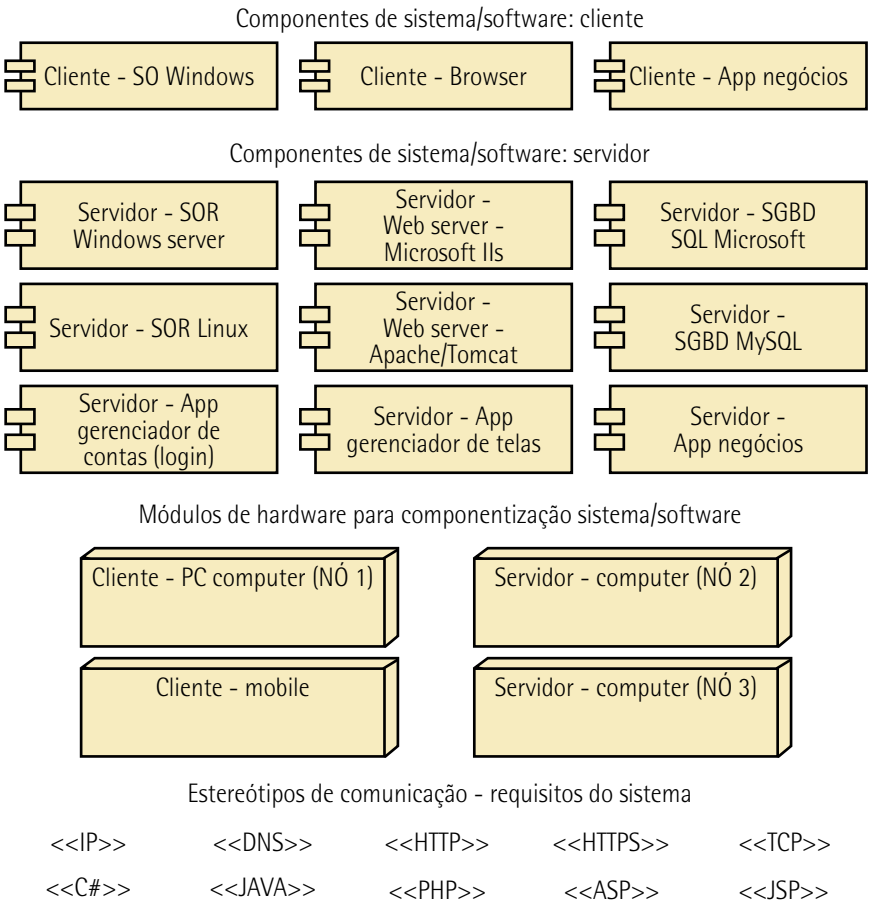


Figura 34 – Repertório de alternativas de componentes e módulos para a construção de uma infraestrutura de TI para diversos sistemas

O componente de software é uma unidade modular e independente de código externo, ou seja, tem endereçamento próprio e os dados são consumidos no componente, o que significa que é possível fazer pequenas mudanças nele sem afetar a integridade do sistema. O componente realiza uma função específica dentro de um sistema maior. Assim, esses componentes podem ser desenvolvidos de maneira isolada e, depois, integrados a outros para construir aplicações.

Conforme a figura 35, no padrão UML, o bloco componente pode ter duas formas, sendo representado por uma caixa e dois tabs:

- na caixa são descritos o nome do componente e do(s) estereótipo(s), que classificam o componente;
- os tabs são as interfaces de entrada/saídas responsáveis pela ligação com outros componentes.

Ainda na figura 35, vemos os blocos de implantação, que representam os módulos, em que estão encapsulados diversos componentes. O bloco de implantação (UML) é apresentado como uma caixa em 3D com deslocamento da aba para cima e à direita. Um módulo pode estar conectado a outros módulos ou componentes. Logo mais abaixo da figura 35 estão os estereótipos, usados como um mecanismo de extensão, que permite adicionar novos elementos ou propriedades aos elementos da UML. Eles personalizam o elemento UML (apresentado no formato <<nome do estereótipo>>) para se ajustar às necessidades dos requisitos.

A modelagem da infraestrutura de TI ajuda a alinhar a infraestrutura de TI com os objetivos e as necessidades da organização, garantindo que os recursos tecnológicos sejam implementados. Ela permite otimizar recursos de hardware e software e melhora a segurança com a identificação de pontos críticos, trazendo uma visão escalável e resiliente da arquitetura que garante a continuidade das operações, mesmo diante de falhas ou incidentes, permitindo a implementação de novas tecnologias.

Já a modelagem da arquitetura do software é construída a partir dos requisitos, principalmente os RS. Sua construção ocorre pelo processo de componentização do software.

A componentização é uma abordagem de design e desenvolvimento na qual o sistema ou aplicação é dividido em componentes independentes e reutilizáveis. Cada componente é uma parte modular do sistema com uma função específica.

Na UML, um nó em um diagrama de implantação representa um recurso físico como dispositivo de hardware, servidor ou estação de trabalho, ambiente de execução de software como uma máquina virtual ou um contêiner, ou um recurso lógico que pode executar artefatos (software, bibliotecas ou arquivos).

A modelagem da infraestrutura de TI pode ser elaborada com o software Astah Community ou simplesmente Astah, ilustrado na figura 35. Trata-se de uma ferramenta de modelagem e diagramação usada por desenvolvedores de software, engenheiros e analistas de negócios. Criada pela Change Vision, Inc., a Astah é conhecida por oferecer soluções que facilitam a visualização e o design de sistemas complexos.

Potencializando a excelência em engenharia com a Astah

As ferramentas de modelagem da Astah permitem que você visualize a essência de suas ideias e designs de software. Crie diagramas de forma rápida e sem esforço que criem um entendimento claro entre as equipes.

Crie UML, diagramas ER, diagramas de fluxo de dados, fluxogramas, mapas mentais e muito mais no software de modelagem mais poderoso para todos, desde estudantes até equipes corporativas (Changevision, 2006).

Observe a seguir algumas das principais características do Astah:

- **Modelagem UML**: tem uma ampla gama de ferramentas para criar diagramas UML, incluindo de classes, de UC, de sequência, de atividades etc.
- **Diagramas ER e DFD**: além dos diagramas UML, o Astah permite a criação de diagramas de entidade-relacionamento (ER) e de fluxo de dados (DFD), úteis para modelar dados e processos de negócios.
- **Interface intuitiva**: com recursos de arrastar e soltar que facilitam a rápida criação de diagramas.
- **Colaboração em equipe**: há ferramentas que permitem que equipes trabalhem juntas em projetos, compartilhando e mesclando diagramas facilmente.
- **Exportação e importação**: os diagramas criados no Astah podem ser exportados em vários formatos, como imagens (JPG, PNG, EMF, SVG), documentos RTF, HTML, relatório de definição de entidade e XMI.
- **Versatilidade**: o software é usado em diversas indústrias, como TI, desenvolvimento de software e engenharia de sistemas, para criar representações visuais claras e padronizadas de sistemas complexos.

Exemplo de aplicação

A arquitetura da infraestrutura de TI apresentada na figura 35 tem quatro nós: CLIENTE 1 – Estação PC, CLIENTE 2 – Estação Mobile, SERVIDOR 1 – SOR e SERVIDOR 2 – APPs, SGBD. Suas ligações estão em rede e os meios de comunicação estão assinalados pelos estereótipos de protocolos <<TCP>>, <<IP>>, <<HTTP>> e <<DNS>>.

Analisando a arquitetura, vemos que é uma arquitetura servidor/cliente. CLIENTE 1 é um computador de trabalho padrão (desktop ou notebook), CLIENTE 2 usada é para acesso remoto de usuários, SERVIDOR 1 é o servidor que faz frente ao usuário e o SERVIDOR 2 é onde estão localizados os serviços das aplicações e dos bancos de dados.

O Astah tem um repositório que permite compartilhar os artefatos de software produzidos com a equipe de desenvolvimento. Vários componentes vão desde SOR, apps, SGBD e aplicações de servidores, como tratamento de tela, contas do usuário e relatórios.

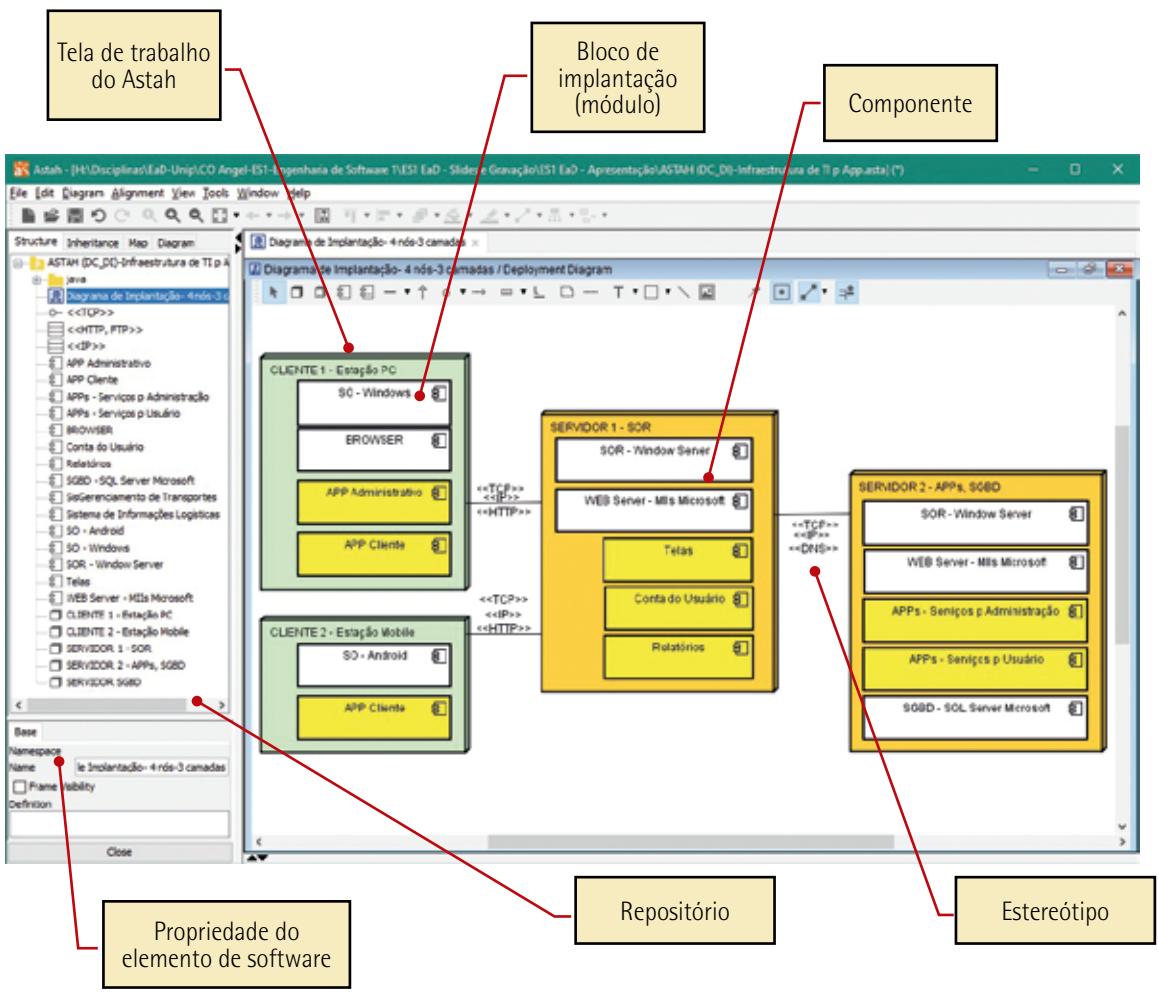


Figura 35 – Ferramenta de desenvolvimento Astah Community com a modelagem da arquitetura de infraestrutura de TI

Saiba mais

Explore mais a componentização com a IBM:

IBM. *Rational software architect standard edition*. [s.d.]. Disponível em: https://shre.ink/egum. Acesso em: 23 jun. 2025.

Explore suas ideias de design de software e coloque isso em prática. Faça download do software Astah e pratique suas funcionalidades.

Disponível em: https://shre.ink/egKy. Acesso em: 23 jun. 2025.

Consulte também:

ASTAH. *Powering engineering excellence with Astah*. [s.d.]. Disponível em: https://shre.ink/egKD. Acesso em: 23 jun. 2025.

Resumo

Abordamos nesta unidade o processo de software, determinando que todo o processo é composto por um conjunto de atividades e tarefas que, organizadas, conduzem a uma prática ou estratégia que melhoram a produtividade.

Vimos que é essencial que todo desenvolvedor ou equipe tenha sob domínio um arcabouço do processo com os recursos necessários para o projeto e a construção do produto, algo que contemple a agilidade no desenvolvimento de software e as iterações do processo para obter capacidade de respostas rápidas às mudanças que ocorrem ao longo do ciclo de vida do desenvolvimento. Nesse contexto, foram destacados os modelos prescritivos, o desenvolvimento ágil e a subdivisão em subprocessos, atividades, procedimentos e tarefas. Os principais processos foram: PSP, TSP, Cascata, Balbúrdia, Prototipagem, Incremental, RAD e Espiral.

Em seguida, detalhamos o planejamento do processo de software relacionando-o ao SDLC. Enfatizamos a estrutura organizacional como regra para produzir um bom software com métodos, ferramentas e técnicas específicas. Vimos a importância de montar um cronograma para planejar, monitorar e controlar os recursos, pois a inteligência organizacional é responsável pelo sucesso do software.

O planejamento do processo de software compõe a integração da estrutura organizacional alinhada com os recursos para os processos do projeto subdivididos nos gerenciamentos de projeto. Dessa forma, na projeção dos recursos são conceituados os perfis profissionais da análise do negócio, do processo e do sistema, avaliando-se os recursos sob a gerência do projeto e sob a gerência da TI.

Por fim, acentuamos os MF&T aplicados à modelagem da infraestrutura de TI, com destaque para o repositório do sistema, a reusabilidade, o componente, o módulo e os estereótipos.

Exercícios

Questão 1. (Fadesp 2024, adaptada) Sobre o modelo de desenvolvimento de software Espiral, avalie as afirmativas.

I – Uma das características mais marcantes do Espiral é a sua ênfase na identificação, análise e mitigação de riscos.

II – Segue a abordagem de passos sistemáticos do Cascata, incorporando-os a uma estrutura iterativa.

III – É uma abordagem realista para o desenvolvimento de sistemas de software de grande porte.

É correto o que se afirma em

A) I, apenas.

B) II, apenas.

C) I e II, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: alternativa E.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: uma das principais características do Espiral é justamente o gerenciamento de riscos, que ocorre em cada iteração do processo. Ele identifica, analisa e mitiga riscos sistematicamente.

II – Afirmativa correta.

Justificativa: o Espiral combina elementos do Cascata (abordagem linear) com iterações (ciclos repetitivos). Cada ciclo passa por fases como planejamento, análise de riscos, desenvolvimento e avaliação, incorporando aspectos sequenciais e evolutivos.

III – Afirmativa correta.

Justificativa: o Espiral é especialmente adequado para sistemas complexos e de grande porte, pois permite ajustes contínuos, mitigação de riscos e evolução incremental, sendo mais conveniente e realista do que modelos puramente lineares, como o Cascata.

Questão 2. (FGV 2024, adaptada) A analista Luísa foi designada como responsável pelo acompanhamento de todo o ciclo de vida de um software.

Sobre o ciclo de vida do produto, é correto afirmar que

A) Finaliza com a entrega do produto.

B) É similar ao ciclo de vida de um projeto qualquer.

C) Apresenta escopo bem definido antecipadamente.

D) O seu sucesso é medido exclusivamente por prazos e orçamento.

E) É similar ao ciclo de vida de programas de longa duração.

Resposta correta: alternativa E.

Análise das alternativas

A) Alternativa incorreta.

Justificativa: o ciclo de vida de um software vai além da entrega inicial, incluindo manutenção, atualizações e eventual descontinuação.

B) Alternativa incorreta.

Justificativa: projetos simples e de requisitos estáveis têm início, meio e fim definidos; enquanto produtos de software têm um ciclo contínuo e com evolução constante.

C) Alternativa incorreta.

Justificativa: principalmente em metodologias ágeis, o escopo pode evoluir durante o desenvolvimento.

D) Alternativa incorreta.

Justificativa: embora prazos e orçamento sejam importantes, o sucesso também depende da qualidade, da satisfação do usuário e de outros fatores.

E) Alternativa correta.

Justificativa: assim como programas de longa duração, produtos de software têm um ciclo contínuo com fases de desenvolvimento, operação e manutenção. Isso é evidente em softwares que, após serem lançados, precisam de atualizações constantes e de suporte.

Unidade III

5 FUSÃO DO PRODUTO E DO PROCESSO DE SOFTWARE

A fusão do software com o processo de desenvolvimento para gerar artefatos ou sistemas de software ajuda as empresas desenvolvedoras a consolidar suas operações, reduzindo os custos e aumentando a produtividade.

Essa fusão representa a integração estreita entre a definição do produto final e as atividades necessárias para desenvolvê-lo. É uma abordagem holística que assegura que os requisitos do software e seus processos de desenvolvimento sejam considerados simultaneamente.

O software é o resultado final do desenvolvimento, incluindo elementos como código, interfaces do usuário e do hardware, documentação, funcionalidades e desempenho esperado.

O processo de software abrange as camadas da engenharia de software, como qualidade, processos, métodos, ferramentas e técnicas utilizadas para criá-lo. Inclui desde a engenharia de requisitos, arquiteturas, codificação, testes unitários e de integração até a manutenção contínua.

A prática das metodologias ágeis e as práticas DevOps promovem a colaboração entre equipes de desenvolvimento e operações. São utilizadas ferramentas de integração e entregas contínuas fundamentais para automatizar e monitorar o progresso do projeto, de forma que o produto final atenda aos padrões de qualidade desejados.

De acordo com Sommerville (2011), as quatro atividades básicas do processo – especificação, desenvolvimento, validação e evolução – são organizadas por processos que diferem conforme o produto a ser gerado e os métodos aplicados pela equipe de desenvolvimento. Por exemplo: ao utilizar o modelo de processo Cascata, as atividades são organizadas de forma sequencial; já no modelo Incremental, as iterações, revisões sucessivas do processo, permitem mudanças nas fases do ciclo de desenvolvimento.

As atividades dependem do tipo de software a ser elaborado, dos desenvolvedores e da estrutura organizacional. Por exemplo: na metodologia ágil Scrum, as especificações e as tarefas são escritas em cartões (Kanban).

As principais características de um bom processo são:

- planejamento adequado com objetivos e cronograma claros;
- gestão eficaz de requisitos;

- modelagem e arquiteturas com base em padrões do tipo UML;
- iterações que permitem ciclos curtos e melhorias contínuas;
- garantia da qualidade com revisões, testes e diagnósticos;
- documentação detalhada e atualizada;
- gestão de mudanças para se adaptar às necessidades do cliente, às mudanças no ambiente operacional e a possíveis falhas no projeto;
- riscos identificáveis e mitigados;
- comunicação e colaboração entre as partes envolvidas no projeto;
- ferramentas para integração de uso repetitivo para automatizar os processos e os métodos.

O bom processo deve ser bem definido, configurável, adaptável e repetível, com nomenclatura e métricas padronizadas entre clientes e desenvolvedores.

5.1 Domínio da análise do engenheiro de software

Já tratamos das características pessoais do engenheiro de software, agora acentuaremos seu domínio de análise, que se refere ao conjunto de áreas de conhecimento e aspectos técnicos, organizacionais e de negócio que esse profissional deve considerar para desenvolver soluções eficazes alinhadas às necessidades dos usuários, da equipe e da organização.

Essa análise envolve a compreensão detalhada do problema ou das necessidades a serem implementadas, o conhecimento de normas e padrões da qualidade do software, processos organizacionais, de desenvolvimento e tecnologias envolvidas, ferramentas, infraestrutura de TI e comunicação com clientes e usuários.

A engenharia de software é uma tecnologia em camadas. Como ilustra a figura 36, qualquer abordagem (inclusive engenharia de software) deve estar fundamentada em um comprometimento organizacional com a qualidade (Pressman, 2021).

A visão da tecnologia em camadas fornece uma base sólida de conhecimentos diversos, que mostra a dimensão de uma boa parte das habilidades profissionais necessárias ao engenheiro de software.

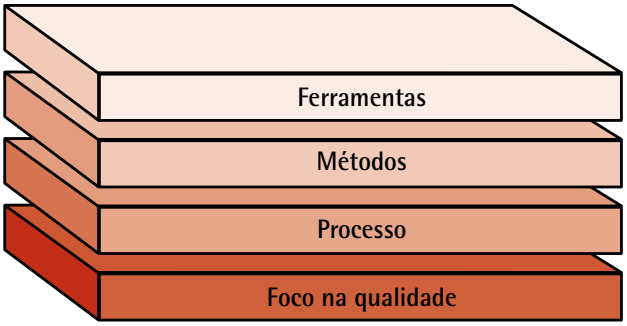


Figura 36 – Camadas da engenharia de software

Adaptada de: Pressman (2021).

Lembrete

O software é o resultado final do desenvolvimento, incluindo elementos como código, interfaces do usuário e do hardware, documentação, funcionalidades e desempenho esperado.

5.1.1 Foco na qualidade

A qualidade estabelece regras, padrões, normas, procedimentos e atividades que orientam o desenvolvimento de software de alta qualidade, levando à cultura de melhoria contínua dos processos e a abordagens cada vez mais eficientes da engenharia de software.

Ela assegura que o produto final atenda à expectativa dos usuários. No entanto, se o usuário não estiver satisfeito, toda a excelência técnica torna-se irrelevante e nada mais realmente importa.

Satisfação do usuário = produto adequado
+ máxima qualidade
+ entrega dentro do orçamento e do cronograma

Figura 37

Observe a seguir alguns exemplos de normas.

- ISO/IEC 25010 (SQuaRE – Systems and Software Quality Requirements and Evaluation – Requisitos de Qualidade e Avaliação de Sistemas e Software):
  - Define os modelos de qualidade para sistemas e software, incluindo características de qualidade e subcaracterísticas como funcionalidade, desempenho, segurança e manutenibilidade.
  - Substitui a antiga norma ISO/IEC 9126 e a ISO/IEC 14598.

- **ISO 12207 – Processos do ciclo de vida do software**:

  – Padroniza os processos envolvidos no ciclo de vida do software, desde a concepção até a manutenção.

- **ISO 15504 – SPICE**:

  – Define um modelo para avaliação e melhoria de processos de software, permitindo medir a maturidade dos processos organizacionais.

  – Serve de base para modelos como o CMMI.

- **ISO 9001 – Sistema de gestão da qualidade**:

  – Foco na padronização e melhoria contínua. A ISO 9001 é referência na garantia de gestão da qualidade em projeto, instalação, desenvolvimento, produção, arquitetura e serviço.

Agora observe alguns modelos.

- **CMMI**: os níveis de maturidade acentuados a seguir representam diferentes estágios de evolução dos processos de uma organização. Esses níveis ajudam a identificar a capacidade de uma organização em desenvolver processos eficazes e eficientes.

  – Modelo de maturidade que avalia e melhora processos de desenvolvimento de software.

  – Usado por empresas para garantir eficiência e previsibilidade no desenvolvimento.

  – Tem cinco níveis de maturidade:

    - **Inicial (initial)**: processos não padronizados.

    - **Gerenciado (managed)**: processos controlados.

    - **Definido (defined)**: processos padronizados.

    - **Quantitativamente gerenciado (quantitatively managed)**: controle por métricas.

    - **Em otimização (optimizing)**: melhoria contínua.

- **MPS.BR (Melhoria do Processo de Software Brasileiro)**

  – Desenvolvido em 2003 pela Softex (Associação para Promoção da Excelência do Software Brasileiro), com o apoio do MCTI (Ministério da Ciência, Tecnologia e Inovações).

- Elaborado para atender micro, pequenas e médias empresas, fazendo uso das principais abordagens internacionais voltadas para a definição, a avaliação e a melhoria dos processos de software.

- Compatível com o CMMI, ele tem sete níveis de maturidade alinhados aos cinco níveis do CMMI.

- Os sete níveis de maturidade no modelo MPS.BR são:

  - G (parcialmente gerenciado).

  - F (gerenciado).

  - E (parcialmente definido).

  - D (largamente definido).

  - C (definido).

  - B (gerenciado quantitativamente).

  - A (em otimização).

- **ITIL (Information Technology Infrastructure Library – Biblioteca de Infraestrutura de Tecnologia da Informação)**: é o conjunto de melhores práticas para gerenciamento de serviços de TI, incluindo suporte e manutenção de software.

- **COBIT (Control Objectives for Information and Related Technologies – Objetivos de Controle para Informação e Tecnologias Relacionadas)**: trata-se de modelo de governança de TI que ajuda empresas a alinhar software e tecnologia aos objetivos estratégicos do negócio.

5.1.2 Processo

Define a estrutura fundamental para a concepção e o desenvolvimento de software orientando-se por padrões e normas da qualidade estabelecida. O processo assegura a integração das camadas tecnológicas, permitindo um desenvolvimento de software racional e oportuno.

O processo estabelece uma metodologia a ser seguida para a entrega eficaz de tecnologia, constituindo a base para o controle e o gerenciamento do projeto. Ele pode ser identificado quando o analista especifica uma série organizada de eventos e atividades, acompanhada de documentação e modelos que orientam a obtenção de resultados específicos, seja na prestação de um serviço, seja na construção de um produto.

Um framework do ciclo de desenvolvimento de software é composto basicamente pelos processos:

- **Elicitação (início ou concepção)**: compreende o gerenciamento das relações com o cliente, a comunicação e a interpretação do negócio com o objetivo de levantar os requisitos do software. A intenção é identificar as entidades e as operações e compreender os objetivos dos stakeholders (conjunto dos interessados).

- **Análise**: após a interpretação do negócio, elabora-se um modelo conceitual do plano de projeto de software, que abrange a especificação detalhada do software a ser desenvolvido. Esse plano contempla a definição das atividades e tarefas técnicas, a estruturação do cronograma, a gestão de riscos, além da alocação de recursos e a estimativa de custos envolvidos no processo. Faz-se nessa etapa a relação entre as necessidades funcionais do cliente e a capacidade da equipe ou empresa desenvolvedora para construir o software.

- **Especificação e modelagem**: a especificação define com clareza e precisão as características, os requisitos e as condições de algo. Normalmente, ela é acompanhada de um modelo gráfico com a modelagem, que expressa a especificação. Essa ação envolve detalhar as funcionalidades, os comportamentos e as restrições de um sistema ou componente, fornecendo uma descrição completa que orienta a sua construção, implementação e verificação.

- **Construção**: é o processo de codificação (geração do código), depuração (exame e limpeza dos dados, do código e da interface), testes unitários e diagnósticos.

- **Implementação**: é a integração do código construído ao software, realizando-se nessa etapa testes de integração e diagnósticos.

- **Implantação**: corresponde à fase de entrega do release do software ao cliente, que, por sua vez, fornece feedbacks de avaliação para validar e aceitar o software.

- **Validação**: é o processo que envolve a garantia da qualidade ao cliente. Esse acordo entre cliente e desenvolvedor garante que o software produzido atende às necessidades e expectativas do cliente ou do usuário final. É a confirmação de que o produto faz o que deve fazer de maneira correta e adequada para seu propósito.

Esses frameworks são mostrados nos modelos de processos:

- **PSP**: criado pelo SEI, é específico para engenheiros de software.

- **TSP**: também criado pelo SEI, seu foco é a equipe de trabalho.

- **Cascata**: modelo clássico do ciclo de vida de desenvolvimento do software.

- **Prototipagem**: modelo iterativo com atividades para montagem de protótipos.

- **Incremental**: modelo iterativo desenvolvido com o conceito de versões.

- **RAD**: usando frameworks de linguagens de programação, visa à entrega rápida de sistemas de software, adotando uma estratégia baseada na construção modular e orientada a componentes reutilizáveis.

- **Espiral**: modelo evolucionário que combina a natureza iterativa da prototipagem com os aspectos sistemáticos do Cascata.

- **RUP**: processo proprietário de engenharia de software criado pela Rational Software Corporation e atualmente representado pela IBM.

5.1.3 Métodos

É o conjunto estruturado de procedimentos, regras e técnicas utilizados para guiar o desenvolvimento do software para alcançar uma meta, um fim ou um conhecimento. Um método assegura que as etapas do ciclo de vida do software sejam seguidas de maneira sistemática e eficiente.

O método determina os passos a serem seguidos na construção do software ou parte dele. Envolve um amplo conjunto de tarefas, desde gestão de equipes, análise de requisitos, projeto e construção de programas, até os testes, a entrega, o suporte e a manutenção.

Os métodos são princípios fundamentais que regem cada área da tecnologia, englobando atividades de modelagem e outras técnicas descritivas que garantem a precisão e a eficiência no desenvolvimento de soluções tecnológicas.

Serão acentuados alguns exemplos de métodos.

Metodologias ágeis

Trata-se de uma coleção de metodologias baseada na prática para modelagem efetiva de sistemas de software e podem ser aplicadas por profissionais no dia a dia.

Em contraponto aos modelos de processos estruturados, as metodologias ágeis são específicas para cada atividade do ciclo de desenvolvimento e, basicamente, são aplicadas em projetos orientados a objetos, podendo ser integradas em determinadas atividades dos modelos de processos estruturados.

A metodologia ágil que mais se destaca é o Scrum, por ser flexível e adaptável, com entregas incrementais, foco no cliente e melhoria contínua. Ele pode ser integrado a outras metodologias ágeis, como XP, Kanban, FDD e demais metodologias ágeis que se fizerem necessárias no desenvolvimento do software.

MR (Matriz de Responsabilidades)

É um método de gestão para definir papéis e responsabilidades de cada membro da equipe em relação às atividades e tarefas do projeto, com o objetivo de deixar claro quem está envolvido e qual função deve desempenhar.

A MR traz melhoria nas comunicações e no controle das atividades, podendo se associar a técnicas de cronogramas para distribuir com detalhes a tarefa de cada membro da equipe e técnicas de construção de diagramas de rede de tarefas (ou rede Kanban), a exemplo de PERT (Program Evaluation and Review Technique – Técnica de Revisão e Controle de Programas) e CPM (Critical Path Method – Método do Caminho Crítico).

A MR é orientada pelo PMBOK na gestão de stakeholders, tema que será tratado mais adiante.

PERT

Preocupa-se com o fator tempo, com o objetivo de planejar e controlar o projeto. Os prazos são tratados de forma probabilística, indicando que, inicialmente, os períodos são estimados com base em experiências anteriores.

CPM

Preocupa-se com o fator custo, em que são aplicadas técnicas de planejamento e controle para realização das tarefas por meio de cálculos determinísticos, o que significa que, pela CMP ou rede de cartões (Kanban), é possível ajustar os períodos das tarefas em um período total mais curto.

Outros métodos

Amplamente constantes na engenharia de software, pode-se citar os métodos de:

- engenharia de requisitos;
- modelagem e design;
- implementação e programação;
- testes de software;
- manutenção e evolução de software.

Lembrete

O método determina os passos a serem seguidos na construção do software ou parte dele. Envolve um amplo conjunto de tarefas, desde gestão de equipes, análise de requisitos, projeto e construção de programas, até os testes, a entrega, o suporte e a manutenção.

5.1.4 Ferramentas

Trata-se de recursos tecnológicos que dão apoio automatizado para os processos e os métodos em todas as fases do desenvolvimento de software.

As ferramentas de software agilizam as tarefas, aumentam a produtividade, a comunicação e a colaboração entre as equipes. Entre elas, pode-se citar ferramentas para gestão de projetos, modelagem, IDE, controle de versões, inspeção da qualidade do código, testes, documentação etc.

Citaremos a seguir alguns exemplos de ferramentas de software.

Gestão de projetos

Na análise de customização dos serviços, o software GanttProject, fornecido gratuitamente, tem várias funcionalidades compatíveis com o Project, da Microsoft, que é pago.

Com o GanttProject é possível obter cronogramas, gráfico de análise do caminho crítico da rede de tarefas e diagrama de recursos (ou pessoas), entre outras aplicações. Esses gráficos e análises são os resultados da automação dos métodos PERT/CPM, MR e Kanban.

O levantamento desses dados é feito por um especialista em gestão de projetos de software, sendo obtidos por meio da análise de dados referente aos métodos utilizados e requisitos do software.

A tabela 1 mostra a aplicação dos métodos PERT/CPM, como técnica de revisão de controle da programação da sprint, referente à metodologia ágil Scrum.

A tabela PERT combina as tarefas da sprint (e backlogs) a serem realizadas em uma ordem lógica, considerando o tempo esperado (te) em duração de dias para realizar a tarefa e a relação com as tarefas anteriores.

Tabela 1 – Tabela PERT da lista de tarefas do Scrum

Item	Tarefa	Tarefas anteriores	Duração (dias)
0	Planejamento da sprint (sprint planning)	–	5
1	Desenvolvimento do incremento (sprint execution)	0	10
2	Plano de reuniões diárias (daily scrum)	0	1
3	Revisão da sprint (sprint review)	0, 1, 2	3
4	Retrospectiva da sprint (sprint retrospective)	3	1
5	Refinamento do backlog (backlog refinement)	4	2
6	Plano para entrega incremental	5	2
7	Documentação técnica	4, 5, 6	2
8	Plano para suporte técnico	6, 7	1

A figura 38 ilustra a implementação dos dados da tabela PERT com a utilização do GanttProject: como resultado, é obtido o cronograma do projeto. Na parte esquerda da figura tem-se a lista de tarefas, equivalente ao "input dos dados" da tabela PERT. Na parte direita, acentua-se o cronograma gerado automaticamente.

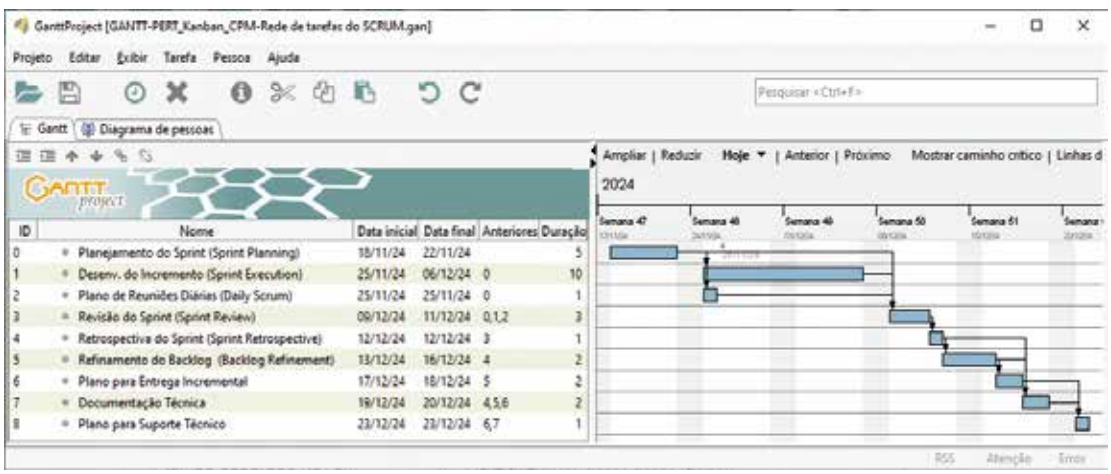


Figura 38 – Tela do software GanttProject com o cronograma gerado

Por sua vez, a figura 39 mostra o gráfico PERT como resultado obtido pelo GanttProject, referente aos dados da tabela PERT. Esse gráfico apresenta um Kanban. Com ele, pode-se fazer o CPM entre as tarefas, possibilitando a identificação de melhorias para reduzir os períodos necessários à execução de cada tarefa.

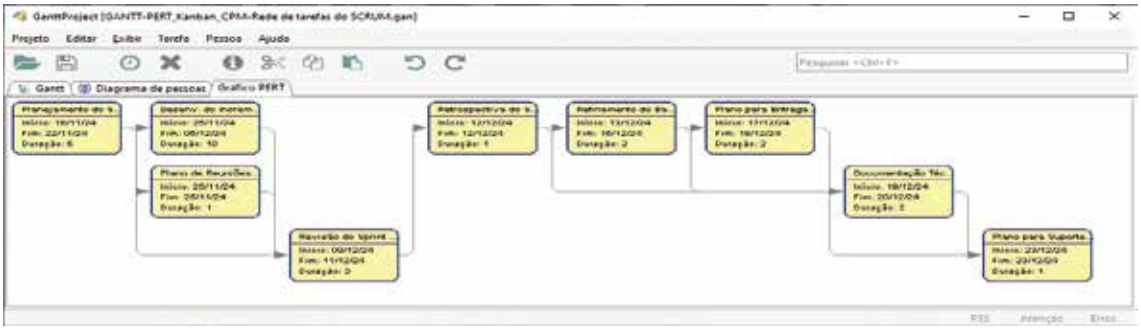


Figura 39 – Tela do software GanttProject com o cronograma gerado

O mercado de software disponibiliza uma ampla variedade de ferramentas para a gestão de projetos. No quadro 5 estão descritas algumas das principais ferramentas de gestão de projetos.

Quadro 5

Ferramenta	Especificação
GanttProject	Ferramenta de gestão de projetos, de código aberto para pequenas e médias empresas, bem como para equipes ágeis. Permite criar gráficos de Gantt, gerenciamento de recursos e cálculos de custos de projetos
Git	Sistema de controle de versão distribuído, é utilizado para rastrear alterações no código-fonte. Plataformas como GitHub e GitLab fornecem interfaces adicionais e funcionalidades colaborativas
Jira	Desenvolvido pela Atlassian, é uma ferramenta para a gestão de projetos ágeis. Suporta Scrum, Kanban e métodos híbridos, permitindo o rastreamento de tarefas, bugs e histórias de usuário
Microsoft Project	Ferramenta poderosa para planejamento, agendamento e controle de projetos. Suporta gráficos de Gantt e outros recursos avançados de gestão de projetos
Microsoft Teams	Solução para comunicação e colaboração em tempo real, com funcionalidades de chat, videochamadas e integração com as ferramentas do Office 365
Trello	Desenvolvido pela Atlassian, usa um sistema de quadros e cartões para gerenciar tarefas e projetos. É altamente visual e fácil de usar, ideal para equipes que preferem uma abordagem simplificada

Com essas ferramentas, é possível criar modelos gráficos detalhados do modelo de negócio, da aplicação, dos dados e da infraestrutura de TI, os quais acompanham as especificações de requisitos do software. Essas ferramentas facilitam a visualização e o design de sistemas complexos, permitindo uma melhor compreensão e comunicação entre os membros da equipe de desenvolvimento.

Como exemplo, a figura 40 mostra o uso da ferramenta de modelagem Visio para a construção de um diagrama de sequência pelo padrão UML. O diagrama de sequência representa a interação entre os componentes do sistema ao longo do tempo, especificando claramente como são realizados os processos e as operações.

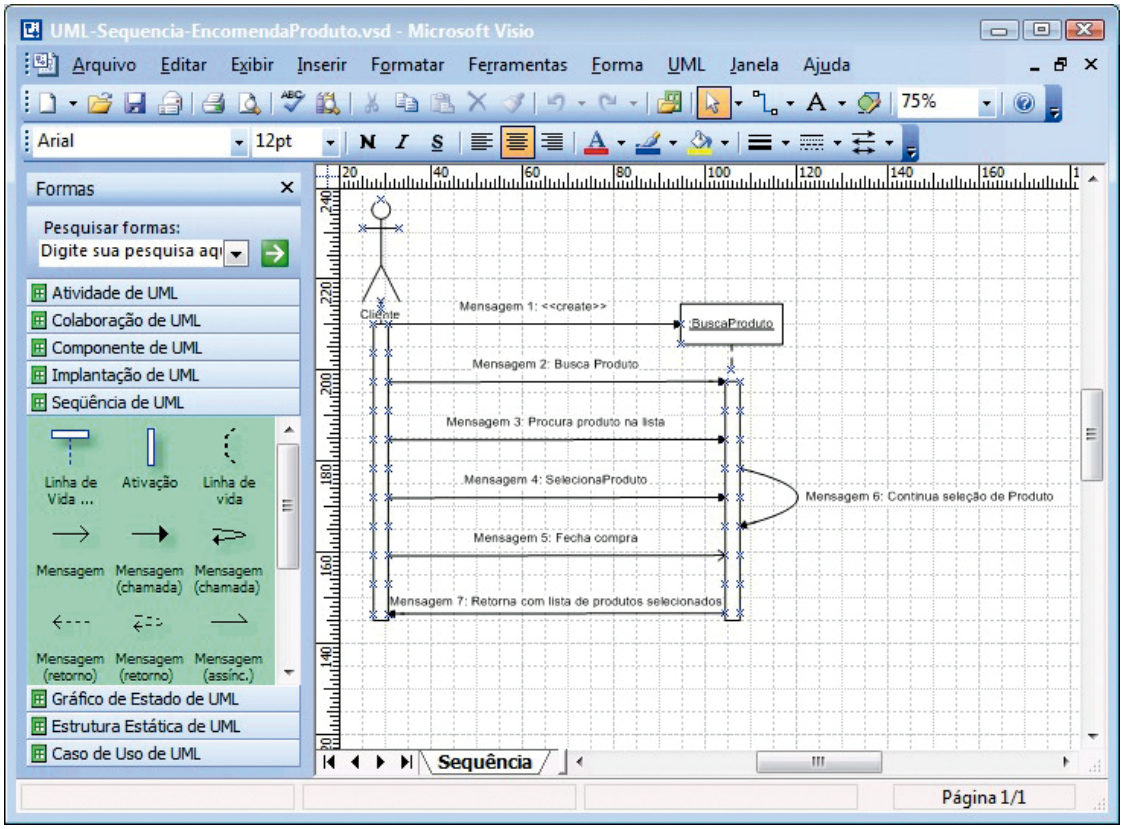


Figura 40 – Modelagem de software com diagrama de sequências elaborado pela ferramenta Visio, da Microsoft

O mercado de software disponibiliza uma gama de aplicações pagas e gratuitas como ferramentas de software para modelagem e o quadro 6 acentua algumas das principais.

Quadro 6

Ferramenta	Especificação
Astah	Permite criar diagramas UML de UC, atividades, classes, sequência, componentes, implantação, entre outros. Tem recursos de exportação/importação de arquivos ou programas e integração com ferramentas de desenvolvimento
Bizagi	Utilizada na BPM, com suporte para criação de diagramas BPMN e automatização de processos empresariais
Draw.io	É uma ferramenta de modelagem e diagramação. Permite criar uma variedade de diagramas, desde fluxogramas, diagramas UML, design de telas de software, mapas mentais, diagramas em rede etc.
FreeMind	É uma ferramenta de modelagem mental que permite aos usuários criar mapas mentais para organizar, visualizar informações, criar mapas de navegação em software, entre outros. O mapa mental é utilizado em abordagens para levantamento de requisitos e recursos, brainstorm, planejamento e organização de ideias
MySQL Workbench	É uma ferramenta de design de banco de dados que permite criar e visualizar esquemas de banco de dados, realizar engenharia direta e reversa e gerenciar bases de dados MySQL
Microsoft Visio	É uma ferramenta de modelagem e diagramação. Com gráficos vetoriais, foi lançada pela primeira vez em 1992 pela Visio Corporation e a Microsoft adquiriu a empresa em 2000. É usada para criar uma variedade de diagramas, incluindo fluxogramas, organogramas, planos de piso, diagramas de rede, diagramas UML e mapas mentais

IDE

É um framework de programação que opera no domínio da aplicação, fornecendo uma solução completa para uma "família de problemas" específicos de uma determinada funcionalidade. Essencialmente, é composto por um conjunto de classes e interfaces, decomposto em quadros com seus respectivos códigos.

Os frameworks acentuados permitem a reutilização de componentes de software, padronizam a arquitetura do sistema e promovem a consistência no desenvolvimento.

A figura 41 traz o exemplo de uso do framework de programação NetBeans, que está sendo usado para o desenvolvimento de classes. Observe as caixas de comandos na tela: são caixas de acesso a bibliotecas e ferramentas integradas para agilizar o processo de codificação e garantir a integridade do código.

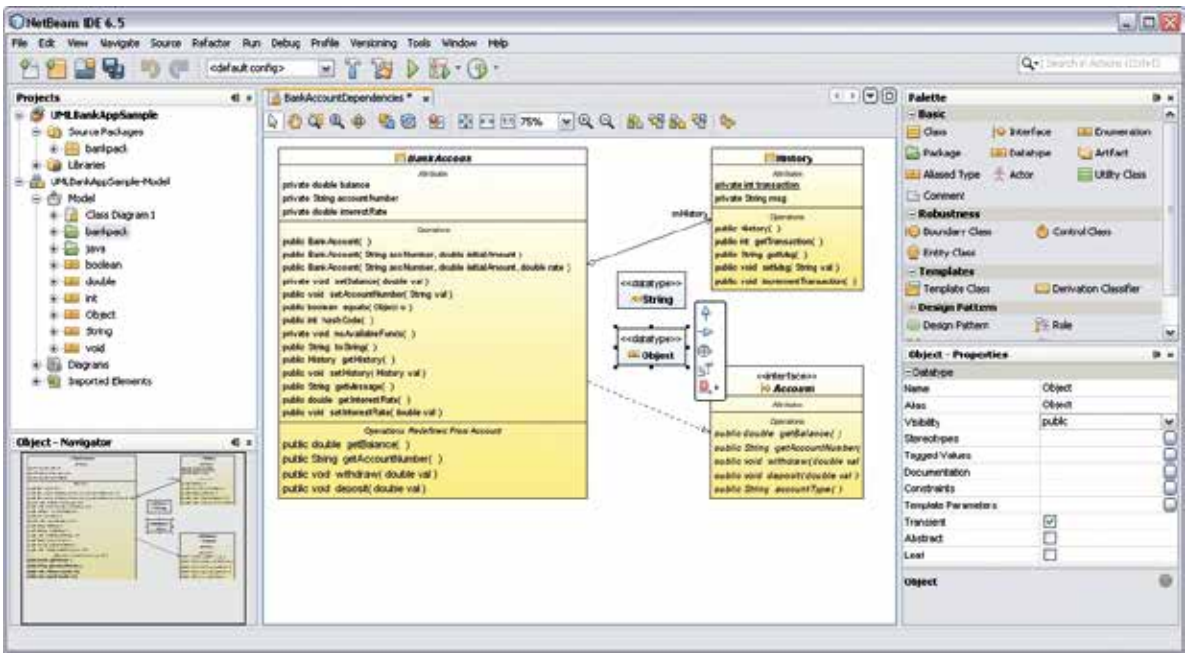


Figura 41 – Ambiente de trabalho na codificação e modelagem de software com o framework de programação NetBeans

Existem diversas ferramentas IDE para o desenvolvimento de software e o quadro 7 destaca algumas delas:

Quadro 7 – Ferramentas IDE

Ferramenta	Especificação
ASP.NET Core (C#)	De código aberto, é uma ferramenta para a construção de aplicações web modernas baseadas em.NET
Django (Python)	Promove o desenvolvimento rápido e o design limpo e pragmático. É ideal para construir aplicações web robustas
Eclipse	Desenvolvido pela Eclipse Foundation, é um dos IDEs mais populares e versáteis disponíveis atualmente, operando com diversas linguagens de programação em ambiente multiplataforma
Electron (JavaScript, HTML, CSS)	Permite a criação de aplicações desktop multiplataforma usando tecnologias web
NetBeans	Plataforma de desenvolvimento de software modular, foi inicialmente desenvolvida pela Sun Microsystems e hoje é mantida pela Apache Software Foundation
Qt (C++)	Ferramenta robusta e completa para o desenvolvimento de aplicações desktop e móveis com suporte multiplataforma
React Native (JavaScript)	Ferramenta para desenvolvimento de aplicativos móveis nativos usando React, permite a criação de apps para iOS e Android a partir de um único código-fonte
Xamarin (C#)	Utiliza o.NET para construir aplicativos móveis para Android e iOS com uma única base de código compartilhada

5.2 JAD (Joint Application Development – Desenvolvimento de Aplicação Conjunta)

De acordo com Fournie (1994), o JAD é uma metodologia criada pela IBM em 1977 e tem por finalidade reunir uma equipe de usuários, stakeholders e desenvolvedores para alcançar acordos baseados na compreensão das operações do processo.

O JAD é uma metodologia colaborativa de ciclo rápido utilizada no desenvolvimento de software para acelerar o processo de comunicação, por meio de reuniões estruturadas denominadas sessões JAD. Essas sessões são lideradas por um facilitador, que orienta a discussão e garante que as partes interessadas contribuam com suas perspectivas para determinar requisitos, design de software e resolução de problemas.

> A JAD reúne usuários finais e especialistas em sistemas de informação em uma sessão interativa para discutir o projeto do sistema. Se adequadamente preparadas e mediadas, as sessões JAD podem acelerar significativamente a fase de projeto e envolver intensamente os usuários (Laudon; Laudon, 2014, p. 416).

As sessões JAD, conforme mostra a figura 42, incluem representantes das diversas áreas do conhecimento, com o objetivo principal de reunir as partes interessadas no projeto para definir requisitos de software e especificações.

Figura 42 – Sessão JAD. Imagem gerada por IA por meio da OPENAI

Disponível em: https://shre.ink/eBO8. Acesso em: 23 jun. 2025.

Vamos acentuar a seguir os perfis dos principais participantes.

Facilitador (ou líder da sessão)

Conduz a sessão JAD. É responsável pelas interações entre os participantes, contribuindo para que os objetivos sejam alcançados.

O líder deve garantir que o processo revisional prossiga conforme o planejado, controlar as interações entre os participantes e arbitrar quando necessário.

Esse profissional deve dominar o relacionamento interpessoal e ser experiente na condução efetiva de reuniões. É respeitado tecnicamente, independente e tem envolvimento no projeto.

Executivo patrocinador (autor do produto, analista de negócios ou analista de processos)

É a representação da gerência de mais alto nível, pertencente às fileiras dos usuários, estando muito comprometido com as atividades do processo de negócio.

Esses profissionais fornecem as diretrizes sobre as principais metas e objetivos do negócio. Muitas vezes, eles são o próprio cliente patrocinador do projeto.

Gerência funcional (ou representante do usuário)

Trata-se das pessoas peritas na matéria em discussão nas sessões JAD, são aquelas que precisam do software. Elas devem ser escolhidas pela sua capacitação profissional em descrever as práticas atuais e operações do negócio que serão implementadas no software a ser desenvolvido.

Gerente de projeto

Auxilia na coordenação da sessão JAD com o objetivo de alinhar os interesses estratégicos da organização com os recursos necessários para produzir o software.

Esse profissional atua na gestão da equipe de desenvolvimento e stakeholders com a preocupação de controlar a comunicação, os custos e o tempo do projeto. Elabora cronogramas, planilhas de custo e distribuição dos recursos de uma forma geral.

Gerente de projeto da infraestrutura de TI

Atua diretamente com a equipe de desenvolvimento, avaliando a capacitação do pessoal de desenvolvimento para produzir o software e os recursos de infraestrutura.

Esse profissional atua na gestão da equipe específica do desenvolvimento. Elabora arquiteturas, distribuição dos recursos de TI e faz análise de Kanban para melhoria da produtividade.

Analistas de sistemas (e/ou desenvolvedores)

São aqueles que trabalham no desenvolvimento do software, que levam conhecimentos técnicos e específicos de como os assuntos de negócios serão implementados no software.

Esses profissionais documentam os requisitos e ajudam na interpretação das necessidades dos usuários em especificações técnicas, analisam a viabilidade técnica das soluções propostas e identificam possíveis riscos.

Durante as sessões JAD, os analistas (com o secretário ou não) elaboram protótipos do software para que usuários e clientes possam ver o que se pretende ser mostrado nele.

Secretário (escriba ou documentador)

Registra apenas as informações relevantes das discussões e das decisões tomadas durante a sessão. Para tal, usa ferramentas de software para organizar, documentar e apontar tendências dos assuntos pertinentes na discussão. O secretário JAD é muitas vezes citado como um especialista no uso de ferramentas CASE e de prototipação.

5.3 Gerenciamento da equipe de desenvolvimento

A gestão de equipes de desenvolvimento de software assegura que todas as partes interessadas estejam devidamente informadas e engajadas em uma comunicação eficaz e melhor alocação de recursos.

Para promover um ambiente colaborativo e adaptável que atenda às necessidades de stakeholders e equipe do projeto, a metodologia sugerida baseia-se na MR do PMBOK. Também conhecida como MR RACI, ajuda a definir claramente papéis e responsabilidades de cada participante da equipe de

projeto. Atende a todo e qualquer projeto que necessite alinhar os recursos humanos com as tarefas a serem realizadas, como é o caso da formação de equipes de desenvolvimento nos modelos de processos tradicionais e metodologias ágeis.

As principais aplicações da MR na engenharia de software envolvem:

- **Definição clara de papéis e responsabilidades**: com ela, todos os membros da equipe de stakeholders sabem exatamente suas funções e responsabilidades no projeto, o que melhora a comunicação e evita sobreposições e redundâncias de tarefas.
- **Gestão de recursos**: a MR ajuda na alocação e no controle eficiente dos recursos, garantindo que o indivíduo trabalhe especificamente na sua tarefa e saiba também as responsabilidades dos demais companheiros de equipe que trabalham na atividade, facilitando o relacionamento das partes e a hierarquia de decisão na realização da atividade.
- **Integração com os métodos PERT/CPM**: são métodos de gerenciamento de projetos aplicados ao desenvolvimento e à produção em diversos ramos da indústria, neste caso, a indústria de software.
- **Integração com metodologias ágeis**: nos ambientes de desenvolvimento ágil como XP, Scrum, FDD e outras, as responsabilidades podem mudar de sprint para sprint.

A transparência proporcionada pela integração da MR com metodologias ágeis é útil em projetos de software, pois, neles, as mudanças são frequentes e a colaboração contínua é essencial, promovendo uma melhor gestão dos recursos e dos prazos. Essa integração permite que a equipe tenha uma estrutura flexível e adaptável de forma rápida, melhorando a produtividade de software com Scrum, por exemplo.

5.3.1 Análise de dados e construção da MR

Na construção da MR neste livro-texto, são considerados stakeholders todos os interessados no software com a seguinte hierarquia de decisão: cliente, representante do usuário, autor do processo, gerente de projeto, gerente do sistema, programador, DBA (Data Base Administrator – Administrador do Banco de Dados).

Observação

O conceito dos termos stakeholders e desenvolvedores é usado com bom senso. Na aplicação do método MR, vale a própria definição de stakeholder: qualquer indivíduo, grupo ou organização que tem interesse direto ou indireto em um projeto, produto ou serviço e pode afetar ou ser afetado pelo resultado.

Dessa forma, nos exemplos a seguir serão considerados todos os participantes do projeto, ou seja, estão inclusos representantes, com perfil administrativo, outros interessados no projeto e desenvolvedores.

Uma matriz é uma tabela de números, símbolos ou expressões, organizados em um quadro retangular que comporta linhas e colunas, em que cada elemento (ou célula) da matriz é identificado por dois índices: um para a linha e outro para a coluna, conforme o quadro 8.

> Matriz de responsabilidades (MR). Uma tabela que mostra os recursos do projeto alocados a cada pacote de trabalho. A matriz RACI é uma maneira comum de mostrar às partes interessadas quem são responsáveis, as pessoas que devem prestar contas, ser consultadas ou informadas, associadas às atividades, decisões e entregas do projeto (PMI, 2021, p. 240).

O quadro 8 mostra uma MR para a gestão de equipe de desenvolvimento. De acordo com os participantes e as atividades, essa matriz se refere à fase do processo de concepção do software.

Quadro 8 – Distribuição de colunas e linhas de uma MR para a gestão da equipe de desenvolvimento

ID	Atividade	Stakeholder				
		Cliente	Gerente de projeto	Gerente de sistemas	Engenheiro de software	Analista de sistemas
1	Elicitação					
2	Análise de requisitos					
3	Modelagem do negócio					

Veja onde são distribuídos os atributos RACI no quadro 9. Observe que é atribuído um R apenas para cada atividade e que normalmente o atributo R é acompanhado pelo atributo C, que pode ser distribuído o quanto você achar necessário.

Quadro 9 – MR RACI para a gestão da equipe de desenvolvimento no processo de concepção do software

ID	Atividade	Stakeholder				
		Cliente	Gerente de projeto	Gerente de sistemas	Engenheiro de software	Analista de sistemas
1	Elicitação	C	R	C	I	–
2	Análise de requisitos	C	A	R	C	I
3	Modelagem do negócio	A	C	C	–	R

A distribuição dos atributos RACI não é atribuída ao acaso. A identificação das responsabilidades é vital para o controle do recurso humano no processo organizacional.

Para determinar os critérios de responsabilidades, algumas questões devem ser observadas quanto aos integrantes da equipe de desenvolvimento:

- Qual é o organograma que será aplicado para os stakeholders?
- Quem é o responsável mais cobrado por um determinado resultado: do processo, da atividade ou da tarefa?
- Quem influencia mais no processo e tem maior poder de decisão?
- Quem é o autor do processo?
- Quem faz a maior parte do trabalho ou quais são os integrantes que são mais cobrados no processo?

A distribuição dos atributos deve se orientar de acordo com o grau de envolvimento do integrante da equipe na atividade. Na MR RACI, a definição e a função para cada atributo do processo na montagem da matriz são acentuadas a seguir:

- **R (Responsável)**: é o integrante responsável pelo processo, atividade ou tarefa. Ele alinha a equipe, cria procedimentos de serviços e dispõe os recursos necessários para a execução da atividade. É quem responde pelo sucesso de execução da atividade. Para todas as atividades, sem exceção, deve existir obrigatoriamente apenas um R.
- **A (Aprova)**: é o integrante responsável pela validação (ou aceite) dos resultados da atividade, que normalmente é o cliente (interno ou externo), o autor da atividade, a diretoria ou a gerência. Deve ser usado apenas um atributo A para a atividade, porém não existe obrigatoriedade para aplicá-lo em todas as atividades, porque nem todas elas necessitam de aprovação.
- **C (Consultado)**: envolve os auxiliares diretos que trabalham na atividade junto ao responsável (R). Os integrantes que dão suporte podem representar um determinado membro da equipe, terceiros contratados para trabalhar na atividade ou consultores. Para cada R é sugerido que seja acompanhado por pelo menos um C.
- **I (Informado)**: engloba os convidados a integrar a equipe para a realização de uma determinada atividade. Os informados atuam na atividade de forma indireta e podem representar vários perfis da equipe. Eles podem ter o perfil de consultores que têm conhecimentos específicos sobre uma determinada atividade ou interessados na atividade por alguma razão, por exemplo, avaliar o custo da atividade quando esta envolver as operações de verificação de integração dos componentes do software.

Exemplo de aplicação

Caso: gestão de equipe de desenvolvimento Scrum com MR RACI.

Antes de iniciar as tarefas da metodologia ágil Scrum, deve-se fazer o alinhamento das tarefas com os serviços a serem desenvolvidos pela(s) equipe(s) Scrum. Isso é feito uma única vez e vale para a(s) equipe(s) que faz(em) parte do desenvolvimento do software em questão.

A distribuição das tarefas está relacionada às responsabilidades de cada participante da equipe e, com a MR RACI, se define o papel do participante no projeto.

Tarefas do Scrum

Criação do backlog do produto (product backlog), uma lista priorizada dos itens e dos requisitos do produto que é criada e mantida pelo product owner.

- **Planejamento da sprint (sprint planning)**: é a seleção dos itens do backlog do produto que definem o objetivo da sprint, a qual é realizada pelo product owner com a equipe de desenvolvimento.
- **Execução da sprint (sprint execution)**: é a execução do backlog do produto pela equipe de desenvolvimento.
- **Reuniões diárias (daily stand-ups)**: são reuniões rápidas (15 minutos) que ocorrem com a participação de cada membro da equipe.
- **Revisão da sprint (sprint review)**: é a revisão do trabalho concluído com a demonstração dos incrementos do produto. Ela ocorre ao término da sprint com a equipe de desenvolvimento, o product owner e as partes interessadas.
- **Retrospectiva da sprint (sprint retrospective)**: é a análise da sprint para verificar se tudo está funcionando bem. Para tal, a equipe realiza testes, diagnósticos e sugestões de melhoria no produto.
- **Incremento do produto (product increment)**: ao término do trabalho, a sprint é integrada ao software, implantada e condicionada à validação do cliente. Nessa fase, participam todos da equipe e os interessados no produto.
- **Stakeholders**: são considerados aqui os participantes que formam a estrutura do desenvolvimento do software com o Scrum.
- **Cliente**: não tem um papel definido no Scrum, porém participa do desenvolvimento fornecendo feedbacks do produto.
- **Mentor ágil (agile coach)**: orienta e treina a equipe quanto a princípios e práticas ágeis.

- **Proprietário do produto (product owner)**: é um analista de sistema que prioriza os backlogs mais importantes da sprint.

- **Mestre do Scrum (Scrum master)**: faz o papel na equipe do gerente do sistema, garantindo que a equipe siga as práticas do Scrum. Ele facilita o processo e remove impedimentos que possam atrapalhar a evolução do desenvolvimento.

- **Desenvolvedor (developer)**: é responsável pela codificação, pela implementação e pelos testes dos requisitos, de forma a garantir que o produto esteja funcionando de acordo com o esperado.

Construção da MR RACI: distribuição dos atributos RACI na MR

Apesar de considerarmos um modelo básico e estático de sete tarefas (linhas) e cinco stakeholders (colunas), a distribuição dos atributos da matriz é dinâmica. Não existe especificamente uma distribuição estática dos atributos, esses parâmetros são variáveis, porque tudo depende do projeto, do custo, do prazo, do interesse na troca de conhecimento e na melhoria da comunicação.

O analista responsável pela construção da matriz deve ter um bom senso crítico para a definição da quantidade de tarefas, de stakeholders e de distribuição dos atributos. O objetivo é ter um workflow eficiente.

A matriz apresentada no quadro 10 é um modelo aplicado a um caso específico e que pode ser usado em projetos similares.

Quadro 10 – MR RACI do Guia PMBOK para gestão de equipe Scrum

ID	Tarefa	Stakeholders				
		Cliente	Agile coach	Product owner	Scrum master	Desenvolvedor
1	Criação do backlog do produto	A	–	R	C	–
2	Planejamento da sprint	I	I	R	C	C
3	Execução da sprint	–	C	C	A	R
4	Reuniões diárias	–	I	C	R	C
5	Revisão da sprint	A	C	R	C	C
6	Retrospectiva da sprint	–	–	I	R	C
7	Incremento do produto	I	I	A	R	C

A aplicação da matriz não para por aí! Com o uso de uma ferramenta de software que construa gráficos de Gantt, é possível integrar na matriz os métodos PERT/CPM para criar cronogramas, diagrama de Kanban, efetuar análises de caminho crítico na rede de cartões e análise na distribuição de períodos e custos para cada tarefa ou horas acumuladas de serviços pelo stakeholder, além da customização completa do desenvolvimento do software.

Observação

Uma boa estratégia para a gestão de equipes de desenvolvimento de software é o comprometimento para atender a demanda sem sobrecarga de serviços.

6 PROCESSO UNIFICADO

Trata-se de um framework de desenvolvimento de software que oferece uma abordagem iterativa e incremental. Ele enfatiza a importância da arquitetura, do gerenciamento de riscos e da qualidade do software.

O processo unificado tem como base o RUP, a implementação comercial mais conhecida. Entre as suas diversas ramificações, destacam-se:

- **Praxis**: é um framework para desenvolvimento de software que se baseia nos princípios do processo unificado, mas com foco em sistemas embarcados e de tempo real. Ele observa a importância da modelagem formal e da verificação rigorosa para garantir a confiabilidade e a segurança de sistemas críticos.

- **Iconix**: é uma abordagem de desenvolvimento de software ágil que simplifica o processo unificado, tornando-o mais leve e adaptável a projetos menores e equipes com menos experiência. O foco do Iconix é se concentrar na modelagem visual e na geração automática de código, para agilizar o desenvolvimento, mantendo os princípios essenciais do processo unificado.

- **AUP (Agile Unified Process – Processo Unificado Ágil)**: mantém as principais fases do RUP (concepção, elaboração, construção e transição), porém com iterações mais curtas e flexíveis. É uma versão simplificada e adaptada para metodologias ágeis com ênfase em colaboração, feedbacks rápidos e entrega contínua de software funcional. Também envolve o RUP-Scrum Hybrid.

- **EUP (Enterprise Unified Process – Processo Unificado Empresarial)**: é uma extensão do RUP que visa tratar de todo o ciclo de vida do software, não se limitando apenas ao desenvolvimento, incorporando as fases de produção e desativação do software.

- **OpenUP (Open Unified Process – Processo Unificado Aberto)**: leve e flexível, é projetado para ser adaptado a diferentes tipos de projetos e equipes. O OpenUP busca simplicidade, colaboração e entrega de valor ao cliente.

6.1 RUP: fases e workflow

Trata-se de um processo de engenharia de software. Originalmente desenvolvido pela Rational Software Corporation, agora está integrado à IBM, como IRUP (IBM Rational Unified Process).

O RUP oferece uma estrutura disciplinada para a produção e a atribuição de software de alta qualidade, com foco em práticas recomendadas e em uma abordagem iterativa e incremental. É distribuído online, utilizando tecnologia web, disponibilizando upgrades regulares de software pela Rational Software. A figura 43 mostra o ambiente operacional de desenvolvimento via web.

Ele é configurado conforme as necessidades de cada organização. É documentado, desenhado, desenvolvido, distribuído e mantido como uma ferramenta de software usando a UML.

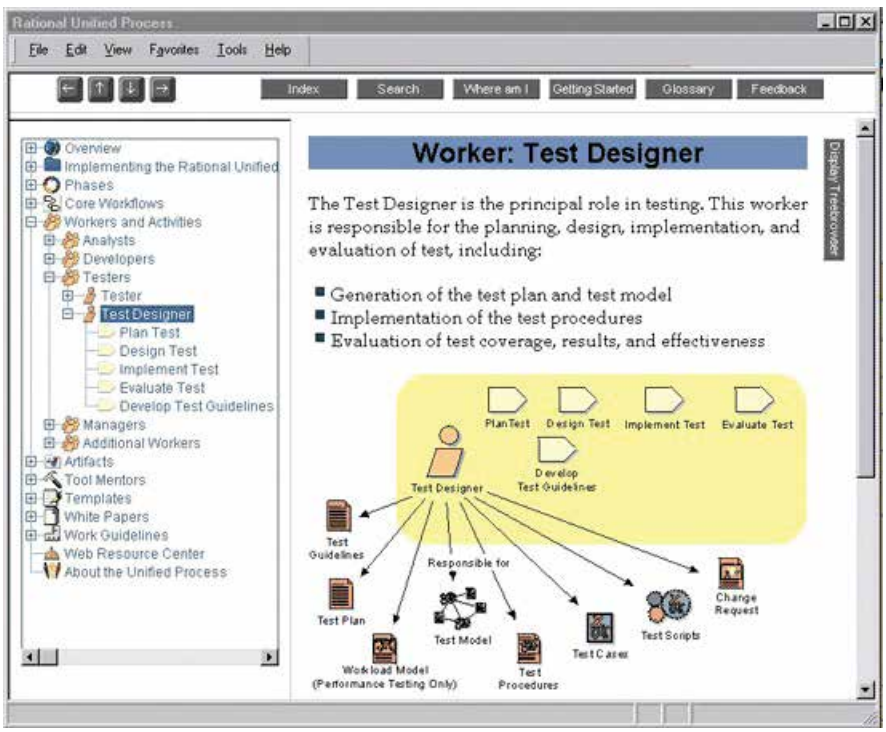


Figura 43 – Plataforma de trabalho RUP pela web

Fonte: Kruchten (2000, p. 19).

O RUP oferece uma abordagem baseada em disciplinas para atribuir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Sua meta é garantir que a produção de software de alta qualidade atenda às necessidades dos usuários dentro do cronograma e com orçamentos previsíveis. Ele captura 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 gestão e o controle de mudanças de software.

Vamos acentuar a seguir as principais características do RUP.

Desenvolvimento iterativo e incremental

No desenvolvimento de um software de média à alta complexidade, não é possível definir um problema e uma solução eficiente em uma única etapa.

Os requisitos mudam com frequência, devido a vários fatores como:

- restrições nas arquiteturas da lógica de programação, da informação e da infraestrutura de TI;
- mudanças nas necessidades primárias do cliente e mudanças ocorridas devido a um refinamento do problema inicialmente levantado.

Cada iteração resulta em um incremento do software com novas funcionalidades e refinamentos, correspondendo a um novo release. As iterações permitem reduzir o risco do projeto e o acompanhamento do progresso.

Modelagem orientada a objetos (OO)

Usa a UML para modelagem OO do software, o que facilita a compreensão e a comunicação entre os membros da equipe e outras partes interessadas.

A linguagem UML se transformou em um padrão para as indústrias ao representar projetos e é usada amplamente pelo RUP (Kruchten, 2000).

Ao representar o projeto utilizando construções gráficas, o RUP expressa de forma eficiente uma visão geral da solução, o que permite que clientes sem nenhuma afinidade com a área de desenvolvimento de sistemas possam facilmente compreender o projeto e adquirir mais conhecimentos sobre a evolução do desenvolvimento do projeto.

Arquitetura baseada em componentes

O RUP é baseado na arquitetura de componentes desde o início do projeto. Ela é desenvolvida e validada nas fases iniciais, servindo como pilar para o desenvolvimento subsequente.

Uma arquitetura baseada em componentes viabiliza um sistema extensível, promovendo a reutilização de "peças" do software. Promove a definição inicial de uma arquitetura de software robusta, que facilita a parametrização do desenvolvimento, a reutilização e a manutenção.

O RUP suporta essa sistemática de construção de sistema, concentrando-se em uma arquitetura executável nas primeiras fases do projeto (Kruchten, 2000).

Centrado em UC

O modelo RUP é conduzido por UC, observe o framework da figura 44. Os UC capturam os RS e dirigem o processo de desenvolvimento.

O RUP descreve como documentar as funcionalidades, as restrições do sistema, as restrições do projeto e os requisitos através do UC. Isso ocorre desde a análise dos requisitos até testes do sistema finalizado para implantação.

A partir do UC são criadas diversas visões sustentadas por outros diagramas da UML, que compõem uma determinada funcionalidade, por exemplo: visão da lógica, do processo, do desenvolvimento e a visão da implementação. Isso ajuda a garantir que o software atenda às necessidades dos usuários.

Os UC e os cenários são exemplos de artefatos do processo que se mostram altamente eficazes para documentar os requisitos funcionais (Kruchten, 2000).

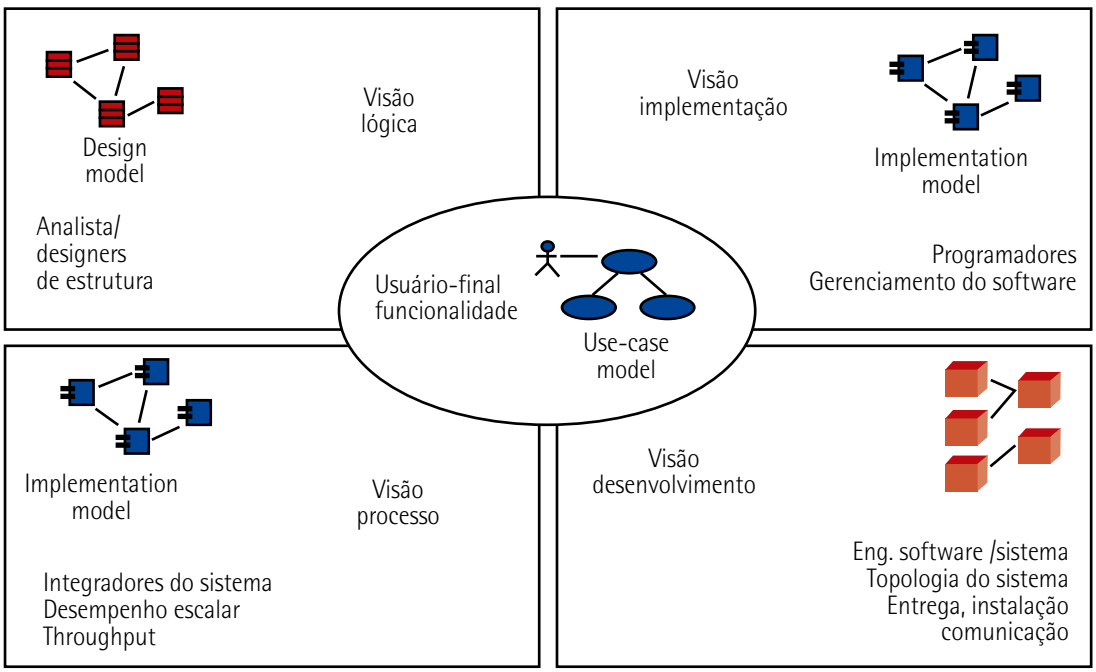


Figura 44 – Construção de funcionalidade a partir do UC, de acordo com o RUP

Adaptada de: Kruchten (2000).

Gerenciamento de riscos

Identifica e gerencia riscos ao longo do ciclo de vida do desenvolvimento do projeto, permitindo que a equipe tome medidas proativas para mitigá-los.

Fases bem definidas e disciplinas estruturadas

O framework do RUP divide o processo em fases, formando a estrutura dinâmica, que acompanha todo o ciclo de desenvolvimento, bem como a estrutura estática com as disciplinas (workflows). Elas variam com a evolução do ciclo de desenvolvimento, proporcionando uma abordagem estruturada e organizada.

Documentação

Enfatiza a criação e a manutenção de documentação abrangente e detalhada ao longo do desenvolvimento, facilitando a manutenção e a evolução do sistema.

Customizável

Pode ser adaptado para atender às necessidades específicas de diferentes projetos e organizações, oferecendo flexibilidade na implementação.

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 no 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 mudanças não comprometam inteiramente o sucesso do projeto (Kruchten, 2000).

6.1.1 Framework do RUP

Ilustrado na figura 45, apresenta um processo bidimensional, mostrando o ciclo de vida do desenvolvimento de software praticado no RUP.

O framework é dividido em fases e cada ciclo resulta em uma nova geração do produto ou parte dele. Cada fase é dividida por iterações definidas para cada projeto.

O RUP apresenta um modelo gráfico com dois eixos: o horizontal tem quatro fases; o vertical, seis disciplinas de trabalho e três disciplinas gerenciais.

Na dimensão horizontal estão as fases do ciclo de vida do desenvolvimento, designadas como estrutura dinâmica, que representa as mudanças ao longo da linha de tempo que ocorrem no processo de desenvolvimento.

Na dimensão vertical estão as disciplinas de trabalho (ou workflow), denominadas como estrutura estática, expressando as mudanças no esforço despendido no desenvolvimento.

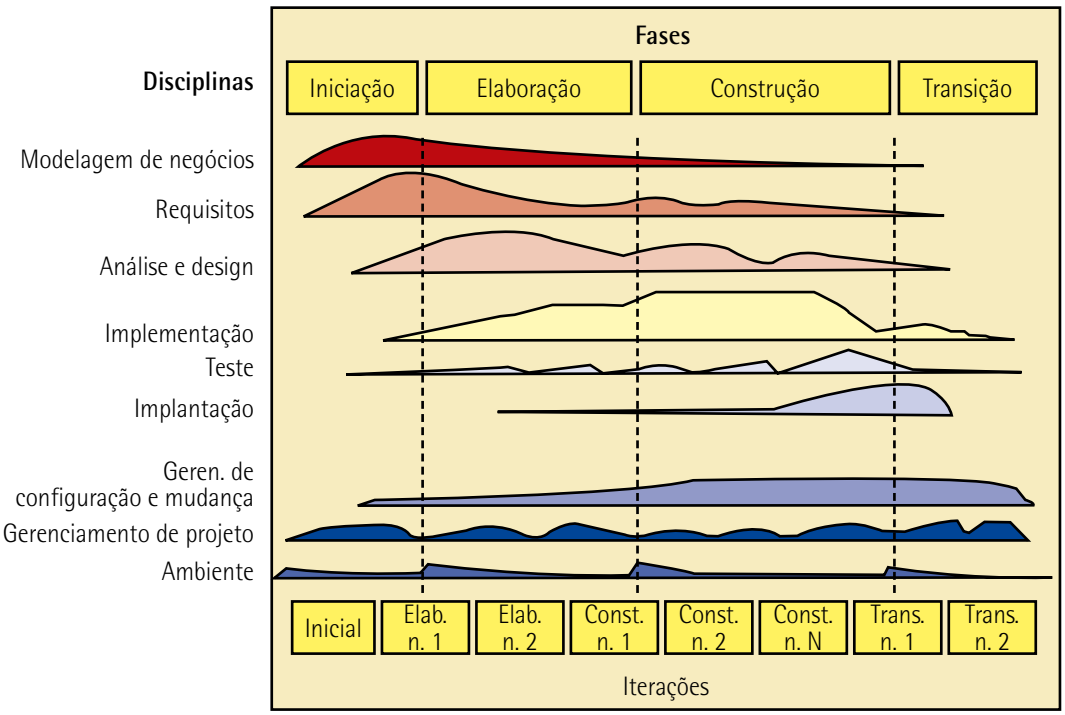


Figura 45 – Framework do RUP com suas fases e disciplinas (workflows)

Adaptada de: Kruchten (2000).

6.1.2 Estrutura dinâmica

Acompanhada de diversas iterações, ela envolve as quatro fases do RUP ao longo do ciclo de vida de desenvolvimento do software. Ao término de cada uma das fases, faz-se uma breve retrospectiva do processo e prepara-se a próxima etapa, verificando se os objetivos da fase atual foram concluídos. Após completar as quatro fases, uma versão release do software estará completa. Para cada nova versão, o software passa por todas as fases novamente (desenvolvimento evolucionário).

Acentuaremos a seguir as características das fases do RUP que acompanham o ciclo de desenvolvimento:

- **Fase 1: Iniciação/Concepção (Inception/Conception)**

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

  – Elaborar um UC crítico ao sistema (atores e requisitos).

– Criar o business case do sistema, que deve incluir os critérios de sucesso, a avaliação de riscos, a estimativa de recursos e o cronograma do ciclo de desenvolvimento.

– Ao término da fase, deverá haver a concordância de stakeholders e desenvolvedores, a compreensão dos requisitos e a credibilidade nas estimativas. O projeto pode ser cancelado ou fortemente reformulado se falhar nesse aspecto.

- **Fase 2: Elaboração (Elaboration)**

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

  – Ao término da fase, deve-se obter estabilidade da arquitetura, demonstrar que o projeto é viável (tempo e custo) e que ele pode ser cancelado ou reformulado se falhar nesse aspecto.

- **Fase 3: Construção (Construction)**

  – Minimizar custos por meio da utilização ótima 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, por meio de um manual e de uma descrição da versão corrente.

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

  – Ao término da fase, deve-se verificar se a disponibilização pode ser adiada se os critérios não forem cumpridos.

- **Fase 4: Transição/Implantação (Transition) – atividades de entrega do software**

  – Realizar testes de integração para validar o novo sistema.

  – Treinar usuários e pessoas responsáveis pela manutenção.

  – Iniciar as tarefas de marketing e distribuição.

  – Ao término da fase, deve-se obter a satisfação dos clientes e os gastos devem ser aceitáveis.

6.1.3 Estrutura estática

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

- **Papéis**

  – Descrevem o comportamento e as 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.

  – Exemplos de papéis incluem especificador de UC, arquiteto de softwares, projetista de interface e gerente de projetos.

- **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 e revisão.

Quadro 11 – Modelo para a distribuição de papéis e respectivas atividades

Papel	Atividade
Gerente de projetos	Planejamento das iterações
Gerente de sistemas	Especificação e modelagem
Analista de sistemas	Determina UC e atores
Analista de testes de desempenho	Executa testes de desempenho

- **Artefatos**

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

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

  – Exemplos incluem modelos (UC, projetos), documentos (plano de negócios, plano de projeto, conjunto de artefatos, de componentes e outros) e código-fonte.

- **Workflows**

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

  – A modelagem pode ser expressa em forma de diagramas de atividades, sequência e colaboração.

  – Como mostra o quadro 12, o workflow do RUP é apresentado em dois grupos, compostos por seis disciplinas de trabalho e três gerenciais.

Quadro 12 – Workflow do RUP

Disciplinas de trabalho	Especificação
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 UC de negócio e modelo de objetos de negócio
Requisitos (requirements)	Possibilita um melhor entendimento dos RS partindo de um acordo com o cliente e com objetivo de oferecer uma orientação aos desenvolvedores. São produzidos um modelo de UC 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
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

6.2 Extensões do RUP

6.2.1 Praxis

De acordo com Paula Filho (2015), o modelo Praxis foi desenvolvido pela equipe de Barry Boehm, fundamentando-se no modelo de ciclo de vida Espiral, com as mesmas raízes no RUP, porém com estrutura diferente das disciplinas organizacionais.

Trata-se de uma metodologia que combina teoria e prática de maneira dinâmica, muitas vezes associada a metodologias ágeis.

Igualmente ao RUP, o Praxis é dirigido por UC, centrado na arquitetura, iterativo e incremental. A modelagem do Praxis é feita com uso do EPF (Eclipse Process Framework), que facilita a adaptação, a personalização, a extensão e a evolução. Herda a arquitetura RUP, complementada por um metamodelo da parte estática do RUP e pelo perfil exclusivo do Praxis.

Ele também se conecta aos métodos ágeis, como Scrum, Kanban e XP; enfatizando a aplicação prática das ideias teóricas em um ciclo dinâmico, promovendo aprimoramento contínuo entre prática e teoria.

Observe o framework do método Praxis na figura 46. Ele é uma abordagem integrada que une diferentes aspectos da gestão de projetos, programas e portfólios; permitindo que organizações e indivíduos desenvolvam maturidade, ampliem o conhecimento e aprimorem competências de maneira contínua e alinhada. Os quatro pilares são: métodos, competências, enciclopédia e comunidade.

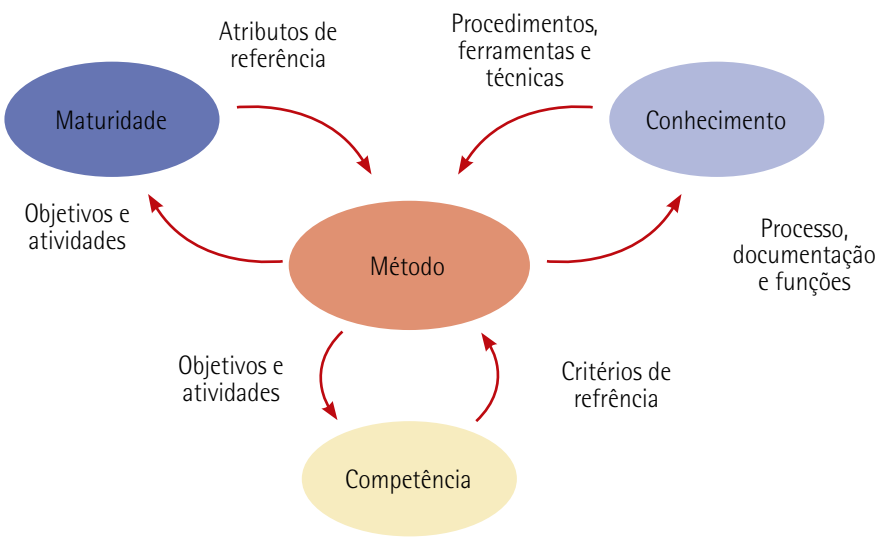


Figura 46 – Framework do método Praxis

Saiba mais

Para entender melhor essa metodologia, consulte:

PRAXIS. *Visão geral*. 2023. Disponível em: https://shre.ink/egKn. Acesso em: 23 jun. 2025.

6.2.2 Iconix

O processo Iconix é uma metodologia proprietária de desenvolvimento de software baseada na UML. O processo é orientado a caso de uso e usa diagramas baseados em UML para definir quatro marcos. A principal característica do processo é um conceito chamado modelagem de robustez, baseado no trabalho inicial de Ivar Jacobson, que ajuda a preencher a lacuna entre análise e design (Iconix, s.d.).

No desenvolvimento de software, a metodologia Iconix antecede tanto o RUP, a XP e as práticas de desenvolvimento ágil.

Os modelos comportamentais apresentados na figura 47 são construídos com a ferramenta Enterprise Architect. Ela compila os modelos a partir de diagramas específicos como máquina de estado, sequência, atividades e outros modelos da UML para gerar código na linguagem descritiva do software. As linguagens suportadas incluem C(OO), C++, C#, Java, VB.Net, VHDL, Verilog e SystemC.

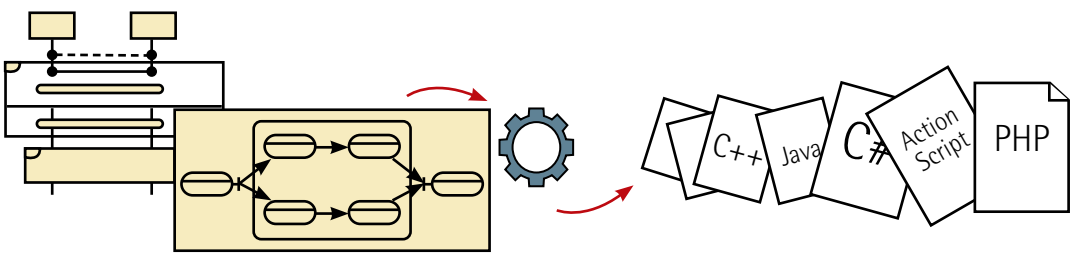


Figura 47 – Modelos comportamentais baseados em diagramas da UML são gerados pelo Enterprise Architect na automatização da codificação

Os quatro marcos principais do Iconix que ajudam a estruturar o processo de desenvolvimento são:

- **Modelagem do domínio**: inicialmente, a partir do levantamento de requisitos, é criado o modelo de domínio para identificar os conceitos-chave do sistema, utilizando diagramas UML, como diagramas de classes, para representar os objetos e suas relações.

- **Revisão de requisitos**: são extraídos UC para especificar os RF do sistema. Esse marco é importante para que o design seja alinhado às expectativas e às necessidades do cliente.

- **Revisão preliminar do design**: dos UC são extraídos diagramas de sequência e de colaboração, convertendo os UC em interações entre objetos, conectando requisitos ao design técnico do software.

- **Revisão crítica do design**: é a validação dos diagramas de design que garante que o modelo esteja completo e que o sistema pode ser implementado com base nos artefatos criados.

6.2.3 EUP

De acordo com Ambler (2023), o EUP estende processos iterativos/ágeis, como DAD (Disciplined Agile Delivery), XP ou Scrum para atender às necessidades reais das organizações de TI.

Saiba mais

O EUP é uma extensão do RUP que visa tratar de todo o ciclo de vida do software, não se limitando apenas ao desenvolvimento, incorporando no ciclo de vida as fases Production (Produção) e Retirement (Desativação), como pode ser observado no gráfico exposto na indicação a seguir:

ENTERPRISE Unified Process (EUP): strategies for enterprise Agile. *Ambsoft Inc*, [s.d.]. Disponível em: https://shre.ink/xEpP. Acesso em: 23 jun. 2025.

6.2.4 OpenUP

É um processo unificado de código aberto, leve e flexível, projetado para ser adaptado a diferentes tipos de projetos e equipes; busca simplicidade, colaboração e entrega de valor ao cliente.

O OpenUP é uma metodologia de desenvolvimento de software que faz parte do EPF. Criado como uma versão mais leve e ágil do RUP, tem basicamente o mesmo ciclo de vida estruturado do RUP, com as quatro fases principais: concepção, elaboração, construção e transição; mantendo suas características essenciais, mas simplificando e reduzindo a complexidade. É ideal para equipes pequenas, podendo ser adaptado para diferentes contextos e necessidades, enfatizando a agilidade e a colaboração entre desenvolvedores e stakeholders.

Existem dois objetivos do EPF:

- Fornecer uma estrutura extensível e ferramentas exemplares para engenharia de processos de software para criação de métodos e processos, gerenciamento de bibliotecas, configuração e publicação do processo.

- Fornecer conteúdo de processo exemplar e extensível para uma variedade de processos de desenvolvimento e gerenciamento de software que suportem desenvolvimento iterativo, ágil e incremental, sendo aplicável a um amplo conjunto de plataformas e aplicativos de desenvolvimento (Eclipse..., s.d.).

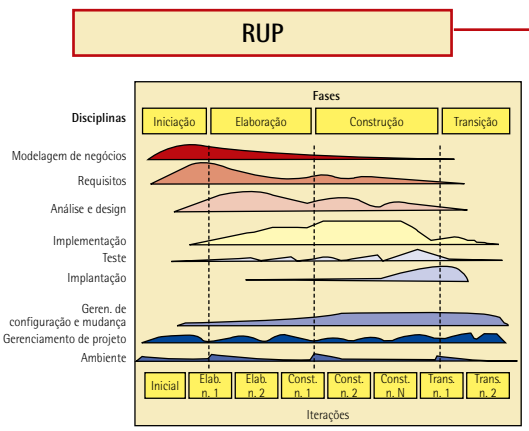
6.3 Métodos, ferramentas e técnicas: definição de papéis e responsabilidades para atividades do RUP

No contexto do RUP, vimos que a matriz MR RACI é uma excelente ferramenta para organizar e comunicar as atribuições individuais e coletivas das equipes envolvidas.

A definição de papéis e responsabilidades com MR RACI garante clareza e eficácia no gerenciamento de projetos de software, permitindo categorizar as responsabilidades de forma estruturada e padronizada, associando cada atividade do RUP às principais funções, como analista de negócios, gerente de projeto, arquiteto de software, gerente do sistema, analista dos sistemas, desenvolvedor e demais stakeholders.

A abordagem da matriz MR RACI, conforme as diretrizes do RUP, assegura que cada fase do ciclo de vida do desenvolvimento esteja alinhada às funções definidas, aumenta a colaboração entre os membros da equipe, reduz ambiguidades e promove um fluxo de trabalho mais eficiente.

O modelo apresentado na figura 48 exemplifica essa aplicação. Considerando as fases e o workflow (disciplinas) do RUP para a montagem da MR RACI e as dimensões do RUP, nessa aplicação serão ponderadas as disciplinas do RUP como as atividades do processo a serem desempenhadas. As responsabilidades atribuídas a stakeholders e desenvolvedores serão acentuadas nas fases do ciclo de vida do RUP.



MR - RACI

ATIVIDADE / PESSOAL	Representante do Usuário	Gerente de Sistemas	Analista de Sistemas	Programador Sênior	Programador Pleno	DBA
Determinação de Casos de Uso	R	S	S	S	----	----
Seleção de Funcionalidades	A	R	S	S	P	
Componentização	----	A	R	S	----	S
Diagrama de Classes	A	S	R	P	P	S
Diagrama de Sequência	----	P	A	R	S	S
Codificação e Testes	----	P	P	A	R	S
Verificação e Validação (V&V)	A	R	S	P	----	----

Figura 48 – Modelo de aplicação da MR RACI no processo unificado

Acompanhe a montagem da matriz observando a figura 45. Para a construção da MR RACI, observe o quadro 13 e suas especificações.

Quadro 13 – Matriz de distribuição de papéis e responsabilidades para o processo unificado

ID	Disciplinas	Stakeholders e desenvolvedores								
		CL	US	AN	GP	GS	AS	PR	TE	DB
1	Modelagem de negócios	A	C	R	C	C	I	–	–	I
2	Requisitos	C	C	A	C	R	C	I	–	I
3	Análise e design	C	C	C	A	R	C	I	I	I
4	Implementação	I	I	C	I	C	C	R	C	C
5	Teste	I	C	I	I	A	C	C	R	C
6	Implantação	C	A	C	R	C	C	C	C	C
7	Gerenciamento de configuração e mudança	–	–	–	C	R	C	C	C	C
8	Gerenciamento de projeto	A	C	C	R	C	C	C	C	C
9	Ambiente	C	C	C	C	A	R	C	I	I
Fases		Iniciação		Elaboração		Construção			Transição	

Atributos da matriz RACI: R – Responsável; A – quem Aprova; C – Consultado (suporte); I – Informado (participa)

Cliente (CL), Usuário (US), Analista de Negócios (AN), Gerente de Projeto (GP), Gerente de Sistema (GS), Analista de Sistemas (AS), Programação (PR), Tester (TE) e Administrador do Banco de Dados (DB)

Análise dos dados da MR RACI

Uma parte da análise é atribuir uma pontuação para os atributos de acordo com a hierarquia de decisão. Por exemplo: R = 5 pontos; A = 4 pontos; C = 3 pontos; e I = 2 pontos. Essa pontuação pode ser convertida posteriormente em valores de tempo ou custo.

Observe a fase de Construção na figura 45:

- **Acumulado em GS**: R = 20 pontos; A = 8 pontos; C = 12 pontos; I = 0 pontos. Total = 40 pontos.
- **Acumulado em AS**: R = 5 pontos; A = 0 ponto; C = 21 pontos; I = 2 pontos. Total = 28 pontos.
- **Acumulado em PR**: R = 5 pontos; A = 0 ponto; C = 15 pontos; I = 4 pontos. Total = 24 pontos.
- O total de pontos acumulados na fase de construção é de 92.

Agora observe a fase de Elaboração na figura 45:

- Acumulado em AN: R = 5 pontos.; A = 4 pontos; C = 15 pontos; I = 2 pontos. Total = 26 pontos.
- Acumulado em GP: R = 10 pontos; A = 4 pontos; C = 12 pontos; I = 4 pontos. Total = 30 pontos.
- O total de pontos acumulados na fase de Construção é de 56.

Observe a seguir a análise dos dados:

- A fase de Construção (92 pontos) exige um maior esforço de serviços do que a fase de Elaboração (56 pontos).

- O Gerente de Sistema (GS = 40 pontos) é o profissional mais exigido nas atividades do projeto de software.

- Apenas para a estimativa de cálculos, se considerar o ponto como equivalente a 1 hora de serviço ou o custo de R$ 100,00/h, temos que a atividade de Construção levará 92 H.h (Homens.hora) a um custo de R$ 9.200,00.

Observação

O mesmo método de análise pode ser feito também por disciplina. Por exemplo: a disciplina Implementação obtém: R = 5 pontos; A = 0 ponto; C = 15 pontos; I = 6 pontos. Total = 26 pontos. Ou seja, a atividade de Implementação leva 26 H.h a um custo de R$ 2.600,00.

Resumo

Nesta unidade, traçamos um panorama do alinhamento entre técnicas, processos e ferramentas para promover um desenvolvimento de software estruturado e eficiente.

A fusão do produto e do processo aborda como alinhar o desenvolvimento do produto ao processo. Essa integração efetiva permite que o software atenda aos requisitos técnicos e às necessidades do cliente.

Vimos também que o domínio da análise do engenheiro de software compreende aspectos como: foco na qualidade, enfatizando práticas que asseguram a qualidade durante todas as etapas do ciclo de vida do software; e o processo com base nas atividades a serem desenvolvidas, determinando estruturas bem definidas que guiam as etapas do desenvolvimento baseando-se em abordagens iterativas e incrementais.

Em uma abordagem colaborativa, o método JAD envolve stakeholders e desenvolvedores em workshops estruturados com o objetivo de definir requisitos e soluções, promovendo maior alinhamento entre as partes.

Estudamos o gerenciamento da equipe de desenvolvimento com o emprego do método MR do PMBOK, usando os atributos RACI. Estes, ao serem integrados aos métodos PERT/CPM, auxiliam no planejamento e no monitoramento de tempos e custos do projeto, mantendo a equipe alinhada com os objetivos do projeto. A MR RACI é flexível o suficiente para adaptar de forma rápida as atividades das metodologias ágeis, aumentando a produtividade de software.

Em seguida, acentuamos o RUP e suas estruturas, como fases e workflow, segmentando o ciclo de vida em concepção, elaboração, construção e transição. A progressão temporal é determinada pela estrutura dinâmica e a estrutura estática organiza os artefatos de software a serem produzidos em disciplinas e atividades, mantendo um alinhamento consistente entre as tarefas a serem desempenhadas.

Nesse contexto, foram apresentadas algumas extensões do RUP, como o Praxis, que integra práticas ágeis ao RUP, e o Iconix, que tem como foco o design orientado a objetos. Vimos também que o OpenUP é uma metodologia do EPF que apresenta uma versão simplificada e ágil para equipes menores e com princípios colaborativos.

Finalmente, destacaram-se métodos, ferramentas e técnicas com o uso da MR RACI para definir papéis claros em atividades do RUP, promovendo eficiência no desenvolvimento.

Exercícios

Questão 1. (Cesgranrio 2024, adaptada) Na gestão de projetos, é recomendável que haja um processo para identificar e analisar regularmente suas partes interessadas (stakeholders), documentando informações relevantes sobre interesses, envolvimento, interdependências, influência e impacto potencial no sucesso do projeto.

As partes interessadas de um projeto

A) Incluem os membros da equipe do projeto, excetuando os profissionais de empresas terceirizadas ou fornecedores.

B) São as pessoas ou os grupos que fornecem recursos e suporte para o projeto, sendo responsabilizados pelo seu sucesso.

C) São as pessoas ou os grupos que necessariamente estão envolvidos com aspectos financeiros do projeto.

D) São as pessoas ou os grupos afetados ou que podem afetar o projeto apenas de forma positiva.

E) Podem ser um indivíduo, um grupo ou uma organização que possa afetar, ser afetado ou sentir-se afetado por uma decisão, uma atividade ou um resultado de um projeto, um programa ou um portfólio.

Resposta correta: alternativa E.

Análise das alternativas

A) Alternativa incorreta.

Justificativa: as partes interessadas incluem todos os envolvidos, inclusive terceirizados e fornecedores, que, muitas vezes, são essenciais para o sucesso do projeto.

B) Alternativa incorreta.

Justificativa: embora stakeholders de fato forneçam recursos e suporte, a definição é muito restrita para diversos tipos de projetos, como os de software, nos quais há muitos outros tipos de interessados além dos financiadores.

C) Alternativa incorreta.

Justificativa: stakeholders em projetos de software vão muito além dos aspectos financeiros, incluindo usuários finais, equipe de desenvolvimento, órgãos reguladores etc.

D) Alternativa incorreta.

Justificativa: stakeholders podem afetar o projeto tanto positiva quanto negativamente. Por exemplo, em projetos de software, usuários resistentes a mudanças podem afetar o projeto negativamente.

E) Alternativa correta.

Justificativa: o texto da alternativa traz a definição mais completa e alinhada com os princípios de gestão de projetos de software, abrangendo diversos tipos possíveis de stakeholders.

Questão 2. (Cespe-Cebraspe 2024, adaptada) Determinada disciplina ou fase, pertencente à dimensão estática do RUP, visa fornecer uma estrutura para gerenciar risco, bem como equilibrar objetivos concorrentes, a fim de superar restrições para entregar com sucesso um produto que atenda às necessidades dos clientes. Trata-se da

A) Disciplina implementação.

B) Disciplina gerenciamento de projetos.

C) Fase construção.

D) Fase transição.

E) Fase elaboração.

Resposta correta: alternativa B.

Análise da questão

No RUP, existem dois conceitos fundamentais para organizar o desenvolvimento de software: fases e disciplinas.

As fases representam a linha do tempo do projeto e dividem o ciclo de vida em etapas sequenciais com objetivos específicos. Cada fase termina com um marco que avalia se os critérios para avançar foram atendidos.

As disciplinas são áreas de atividade que ocorrem ao longo de todo o projeto, independentemente da fase. Elas representam as melhores práticas a serem aplicadas continuamente.

No RUP, o gerenciamento de projetos é uma das disciplinas centrais. Trata-se de uma disciplina que planeja e controla o projeto, gerencia riscos, balanceia requisitos concorrentes e garante que o produto entregue atenda às expectativas, considerando as restrições impostas.

Unidade IV

7 METODOLOGIAS ÁGEIS I

As metodologias ágeis são abordagens de desenvolvimento de software que promovem flexibilidade, colaboração e adaptação rápida às mudanças. Elas foram criadas como uma alternativa aos modelos tradicionais, como o Cascata e o RUP.

Embora os modelos tradicionais sejam úteis em determinados contextos, a crescente complexidade dos projetos e a necessidade de maior flexibilidade impulsionaram a busca por alternativas. As metodologias ágeis surgiram como uma resposta a essas limitações e desafios enfrentados pelas abordagens tradicionais de desenvolvimento de software.

Agora trataremos das seguintes metodologias: Manifesto Ágil, XP, Scrum e FDD.

7.1 Manifesto para Desenvolvimento Ágil de Software

Antes da criação das metodologias ágeis, a indústria de software enfrentava desafios como atrasos, custos elevados e baixa qualidade nos projetos. O Manifesto para Desenvolvimento Ágil de Software, publicado nos dias 11 a 13 de fevereiro de 2001, por Kent Beck e mais 16 notáveis desenvolvedores, foi precedido por uma série de eventos e reflexões que surgiram da necessidade de melhorar os processos de desenvolvimento de software.

Kent Beck e seus colegas se reuniram para defender as seguintes regras:

> Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Por meio desse trabalho, passamos a valorizar:
>
> Indivíduos e interações mais que processos e ferramentas.
>
> Software em funcionamento mais do que documentação abrangente.
>
> Colaboração com o cliente mais do que negociação de contratos.
>
> Responder a mudanças mais que seguir um plano.
>
> Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda (Beck *et al.*, 2001).

Saiba mais

Explore o que o Copilot IA da Microsoft tem a dizer sobre o Manifesto Ágil.

Um manifesto é uma declaração pública que expressa os princípios, os valores, as intenções ou os objetivos de um grupo ou indivíduo. Geralmente, ele serve como um documento oficial que define a visão e a direção de algo, seja um movimento, uma organização ou uma ideia. Manifestos costumam ser usados para inspirar mudanças, engajar pessoas e comunicar crenças fundamentais. Um exemplo famoso inclui o Manifesto Ágil (2001), que define os princípios e os valores para o desenvolvimento ágil de software.

Compreenda melhor o exposto em:

BECK, K. *et al.* Manifesto para Desenvolvimento Ágil de Software. *Agile Manifesto*, 2001. Disponível em: https://shre.ink/ebdl. Acesso em: 23 jun. 2025.

Observe a figura 49. O desenvolvimento baseado em planos (desenvolvimento prescritivo) pauta-se em uma abordagem estruturada e sequencial, com etapas planejadas e documentadas antes do início do projeto. Essa metodologia usa modelos clássicos de processo, como o Cascata, que organiza o trabalho em fases distintas, incluindo levantamento de requisitos, design, codificação, testes e manutenção.

Contudo, o desenvolvimento prescritivo é valorizado pela sua capacidade de prever resultados e reduzir riscos em projetos bem definidos. É apoiado em técnicas como a modelagem estruturada e a criação de artefatos detalhados, garantindo clareza e rastreabilidade ao longo de todo o ciclo de vida do software.

Embora eficiente para sistemas com requisitos estáveis, essa abordagem enfrenta desafios em cenários mais dinâmicos, nos quais a inovação tecnológica e a evolução dos requisitos demandam maior flexibilidade. Por esse motivo, ela tem sido complementada por metodologias ágeis e iterativas que permitem maior adaptabilidade sem comprometer os princípios fundamentais de planejamento e controle da engenharia de software.

O desenvolvimento ágil é uma abordagem iterativa e incremental, com baixo índice de especificação, que prioriza a adaptação a mudanças e a entrega de valor contínuo ao cliente. Observando a figura 49, do ponto de vista da logística de serviços, vemos que a atividade omitida é a "especificação de requisitos", ou seja, foi uma grande mudança nas estratégias de serviços. O desenvolvimento ágil integrou essa fase em seus processos, aproximando mais o cliente do desenvolvimento e tornando claras e exatas as especificações, com histórias sobre o negócio, as atividades de reuniões breves e dinâmicas e as situações do desenvolvimento em tempo real. Assim, promoveu mais eficiência na comunicação entre stakeholders e desenvolvedores.

Enfim, enquanto o modelo prescritivo enfatiza a previsibilidade e o planejamento detalhado com documentação excessiva e que nem sempre são usadas, o ágil valoriza flexibilidade, colaboração e feedback constante.

O desenvolvimento prescritivo pode ser mais adequado para projetos estáveis e bem definidos, enquanto o ágil se destaca em cenários dinâmicos com interação entre indivíduos sobre processos rígidos, software funcional e sujeitos a mudanças constantes.

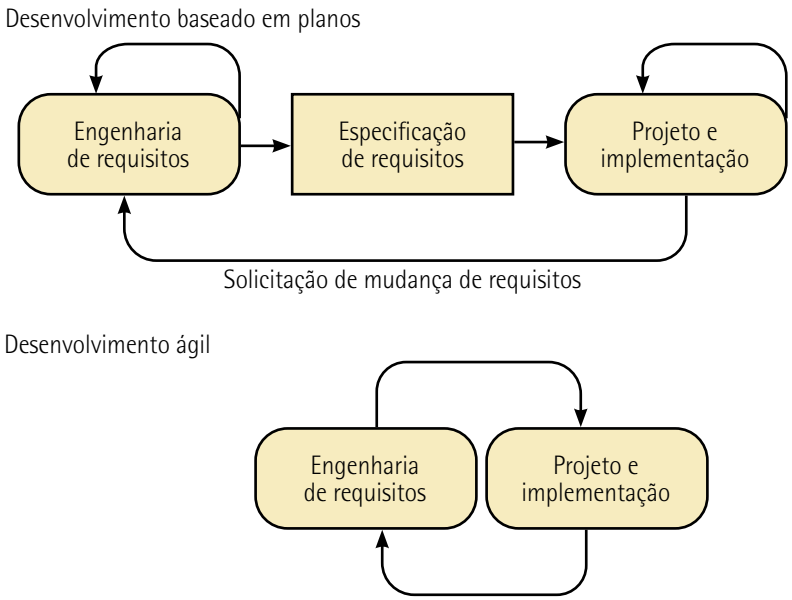


Figura 49 – Especificação dirigida com desenvolvimento ágil

Adaptada de: Sommerville (2016).

7.1.1 Princípios das metodologias ágeis

Métodos ágeis representam um conjunto de abordagens estruturadas pautadas na prática para modelagem efetiva de sistemas baseados em software, fundamentando-se em princípios de flexibilidade, colaboração e entrega incremental. É uma filosofia na qual muitas metodologias se encaixam.

Dessa forma, o desenvolvimento ágil não é uma metodologia única, mas um paradigma no qual diferentes frameworks, como XP, Scrum e FDD se encaixam; proporcionando um conjunto de diretrizes adaptáveis às necessidades específicas de cada projeto e equipe.

Guiados por valores estabelecidos no Manifesto Ágil, os métodos ágeis empregam práticas como integração contínua, desenvolvimento iterativo, refatoração de código e testes automatizados para garantir qualidade e eficiência no ciclo de vida do desenvolvimento.

As metodologias ágeis empregam uma coleção de práticas, guiadas pelos princípios apresentados no quadro 14, e valores que podem ser aplicados por profissionais de software no dia a dia.

Quadro 14 – Princípios dos métodos ágeis

Princípios	Descrição
Envolvimento do cliente	Os clientes devem estar intimamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos RS e avaliar iterações
Entrega incremental	O software é desenvolvido em incrementos com o cliente, especificando os requisitos incluídos em cada um
Pessoas, não processos	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Membros da equipe devem desenvolver suas próprias maneiras de trabalhar, sem processos prescritivos
Aceitar as mudanças	Deve-se ter em mente que os RS vão mudar. Por isso, projete o sistema de maneira a acomodar essas mudanças
Manter a simplicidade	Tenha foco na simplicidade, tanto do software a ser desenvolvido quanto do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema

Adaptado de: Sommerville (2011).

Lembrete

As metodologias ágeis são abordagens de desenvolvimento de software que promovem flexibilidade, colaboração e adaptação rápida às mudanças.

7.2 XP

A metodologia ágil XP tem foco na melhoria contínua da qualidade do código e na capacidade de respostas às mudanças de requisitos. Ela é ideal para projetos dinâmicos, nos quais os requisitos mudam com frequência, promovendo um ambiente colaborativo e eficiente no desenvolvimento de software.

> As ideias subjacentes aos métodos ágeis foram desenvolvidas mais ou menos ao mesmo tempo por várias pessoas diferentes na década de 1990. No entanto, talvez a abordagem mais significativa para mudar a cultura de desenvolvimento de software foi o desenvolvimento da Extreme Programming (XP) (Sommerville, 2016, p. 77).

Criada por Kent Beck em 1998, a XP elevou a níveis "extremos" boas práticas reconhecidas, a exemplo do desenvolvimento iterativo. Ela enfatiza valores como comunicação, simplicidade, feedback, coragem e respeito. Por exemplo, várias versões novas de um sistema podem ser desenvolvidas por diferentes programadores, integradas e testadas em um dia (Sommerville, 2016).

A figura 50 ilustra o ciclo de desenvolvimento do processo da XP para produzir um incremento do sistema. São quatro as atividades-chave da XP a serem desenvolvidas: planejamento, projeto, codificação e teste.

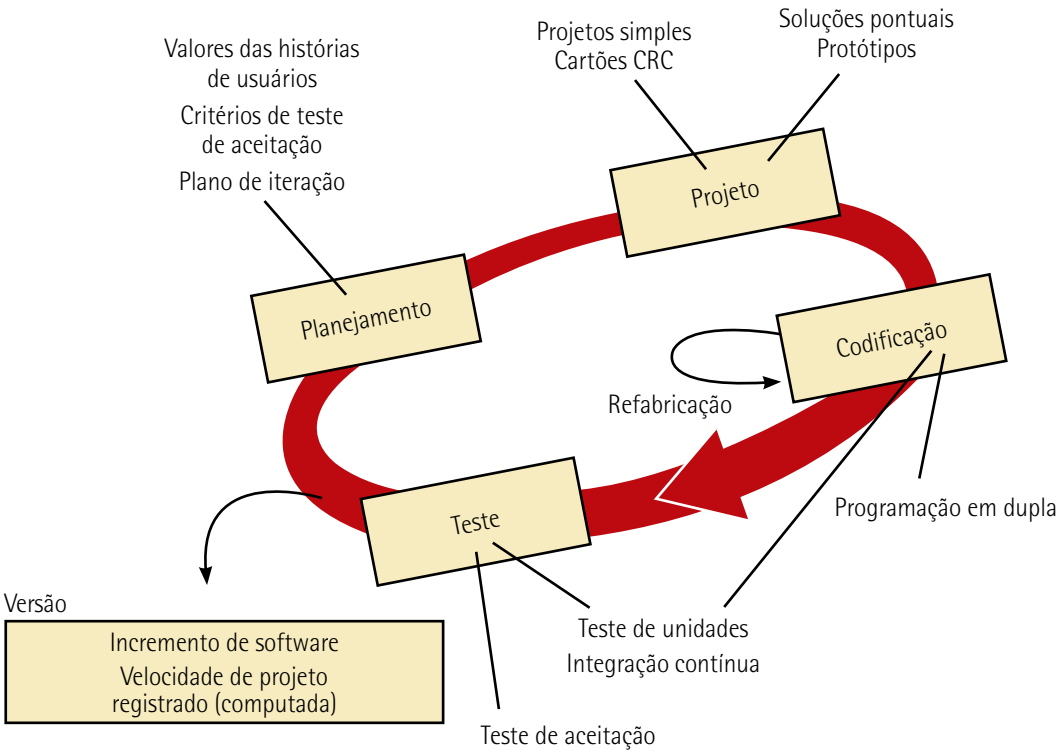


Figura 50 – O processo da XP

Adaptada de: Pressman (2021).

Planejamento

O planejamento do projeto em metodologias ágeis prioriza a eficiência e a adaptabilidade, evitando processos complexos ou demorados. Conforme o projeto avança, o planejamento incremental e iterativo permite entregas frequentes e implementação das mudanças necessárias durante todo o ciclo de desenvolvimento.

Os valores das histórias de usuários são baseados em descrições simples e claras de funcionalidades desejadas, escritas na perspectiva do usuário. Cada história é priorizada com base no valor agregado ao negócio.

Os critérios de aceitação são formulados em contratos entre stakeholders e desenvolvedores para assegurar que os requisitos estejam claros.

A XP considera o conhecimento técnico da equipe, permitindo que certos detalhes operacionais sejam omitidos, confiando na experiência dos desenvolvedores para interpretar e implementar soluções alinhadas aos objetivos do projeto. A simplicidade dessa ação evita a execução de funcionalidades desnecessárias.

O plano de iteração consiste em organizar as histórias selecionadas dos usuários para serem implementadas. A equipe trabalha para estimar o esforço necessário para cada história, garantindo que a iteração seja realista e alcançável dentro do prazo definido, normalmente de uma a duas semanas. Nesse item, é essencial a colaboração entre stakeholders e desenvolvedores para ajustes necessários.

A fase de planejamento é conhecida como jogo do planejamento (planning game), que se inicia com a atividade "ouvir", na qual são coletados e analisados os RF e os RNF. Essa fase inclui critérios da qualidade e expectativas do software, garantindo que a equipe compreenda as demandas do cliente, a definição das funcionalidades prioritárias e tenha foco na entrega de valor contínua.

Projeto

O design de software na XP segue o princípio KIS (Keep it Simple – Preservar a Simplicidade), ou seja, prioriza soluções simples e eficazes, evitando complexidades desnecessárias.

O projeto não antecipa funcionalidades além do necessário, as mudanças e as novas necessidades são tratadas de forma iterativa ao longo do desenvolvimento. Implementações de funcionalidades baseadas em suposições futuras são desencorajadas para que o sistema permaneça enxuto e adaptável.

Nas ações são usados cartões CRC (Classe-Responsabilidade-Colaborador) para sessões de design. Esses cartões identificam e organizam classes orientadas a objetos, suas responsabilidades e colaboradores.

Em apoio ao gerenciamento do fluxo de trabalho, a XP faz uso do Kanban, que é um sistema de gestão visual que permite acompanhar o progresso das tarefas. Ele identifica gargalos e garantias de um desenvolvimento contínuo e eficiente.

Ponderando desafios técnicos ou incertezas no design, a XP recomenda a prática da prototipagem de partes do sistema antes de sua execução definitiva, possibilitando uma validação rápida de soluções antes de sua implementação ao código principal.

Codificação

Durante as iterações na XP, o código é continuamente aprimorado por meio da implementação incremental e da realização de diversos testes de unidade. A cada ciclo iterativo, o código-fonte é revisado de forma criteriosa, permitindo a identificação de oportunidades de refatoração, a correção de falhas e a incorporação de novas funcionalidades, promovendo um desenvolvimento incremental, sustentável e com alta qualidade técnica.

Um dos pilares da XP é a prática da programação em par (pair programming), que consiste na colaboração simultânea de dois desenvolvedores em uma mesma tarefa de codificação: enquanto um assume o papel de driver, escrevendo efetivamente o código, o outro atua como observador, com foco em aspectos estratégicos como detecção de erros, aderência a padrões de projeto, clareza do código e cobertura de testes automatizados que são escritos antes da implementação do código TDD (Test-Driven Development – Desenvolvimento Orientado por Testes). Os papéis se alternam frequentemente, promovendo colaboração intensa, disseminação do conhecimento entre a equipe e aumento na qualidade do software produzido. Essa prática reflete o princípio de que "duas mentes pensam melhor do que uma" e é sustentada por estudos que demonstram ganhos em produtividade e redução de defeitos em ambientes colaborativos.

A prática de refabricação se aproxima do conceito de reengenharia de software, que envolve analisar, redesenhar e reconstruir partes significativas do sistema, geralmente para adaptar o software a novas tecnologias, arquiteturas ou requisitos que a versão atual não suporta bem. Essa prática inclui a refatoração (reestruturação) do código-fonte com o objetivo de melhorar sua legibilidade, seu desempenho e sua manutenção sem alterar a funcionalidade. Essa abordagem garante que o projeto evolua de forma sustentável, mantendo a qualidade do software ao longo de seu ciclo de vida.

Por sua vez, a prática de codificação envolve feedback rápido para que a equipe responda às mudanças requeridas de clientes e usuários de forma eficiente, bem como manter um ritmo sustentável, ou seja, trabalho sem horas extras excessivas, evitando a exaustão da equipe.

Exemplo de aplicação

O foco da prática de refatoração envolve reduzir a complexibilidade, eliminar redundâncias de código, melhorar a legibilidade e a manutenibilidade, aumentando a coesão e reduzindo o acoplamento. Veja um exemplo simples no quadro 15, que mostra a refatoração em JavaScript:

Quadro 15 – Refatoração do código-fonte

Código antes da refatoração	Código depois da refatoração

javascript
function calcularDesconto(preco, tipo) {
if (tipo === 'vip') {
return preco * 0.8;
} else if (tipo === 'comum') {
return preco * 0.9;
} else {
return preco;
}
}
	javascript
	function calcularDesconto(preco, tipo) {
	const descontos = {
	vip: 0.8,
	comum: 0.9,
	};
	return preco * (descontos[tipo] || 1);
	}

O comportamento é o mesmo, porém o código ficou mais limpo, reutilizável e fácil de manter	

Testes

O código é frequentemente integrado e testado para evitar grandes conflitos de versão. Todos os desenvolvedores podem requisitar mudanças em qualquer parte do código para promover colaboração e flexibilidade.

A atividade de testes é uma prática contínua e integrada ao ciclo de desenvolvimento. São realizados testes unitários, que validam o comportamento isolado de métodos e funções; testes de integração, que verificam a ligação entre os componentes ou módulos do software; e testes de aceitação, que asseguram que o software atenda aos requisitos do negócio determinados pelos stakeholders.

Observação

Na XP, os testes automatizados são escritos preferencialmente na fase de codificação, antes da implementação do código funcional, como defendido na prática do TDD, o que favorece o design orientado a testes, aumentando a confiabilidade da base de código. Em sua essência, os testes correspondem ao processo de apoio V&V, de forma a assegurar a estabilidade do software e viabilizar o processo de manutenção contínua com baixo risco.

Na identificação de um erro ou falha, é comum reproduzir um novo teste antes de corrigi-lo, garantindo que o erro ou falha não volte a ocorrer (regressão). São produzidos e registrados novos casos de testes para que possam ser reutilizados.

Os testes de aceitação, realizados com o envolvimento do cliente, ocorrem em ciclos curtos e são acompanhados por métricas (como pontuação ou cobertura de critérios) que ajudam a avaliar o progresso e a conformidade das entregas com os requisitos esperados.

7.2.1 Kanban

Surgiu no Japão na década de 1940 como parte do sistema de produção da Toyota, desenvolvido por Taiichi Ohno, um dos engenheiros responsáveis pelo revolucionário TPS (Toyota Production System – Sistema Toyota de Produção).

Na língua japonesa, Kanban significa "cartão". Na manufatura, ele era um método de controle de estoque e fluxo de produção que utilizava cartões físicos para sinalizar a necessidade de reposição de materiais ou peças. Isso permitia produzir apenas o necessário, no tempo certo (Just-In-Time – JIT), promovendo a filosofia de evitar desperdícios de recursos.

Na engenharia de software, Kanban é um método ágil de gerenciamento de trabalho (workflow), que visa melhorar a eficiência do fluxo de desenvolvimento e promover a entrega contínua de valor. Ele foi adaptado para a engenharia de software como uma ferramenta visual e evolutiva para gerenciamento de processos, especialmente em ambientes que demandam alta flexibilidade e melhoria contínua.

Esse método pode ser aplicado em qualquer ramo de tarefas da engenharia de software. Como exemplo, a figura 51 mostra uma aplicação usando o processo da XP com base no Kanban, fornecendo como resultado um Kanban, que pode inclusive ser aplicado em métodos de análise de caminho crítico no caso de uma rede mais complexa.

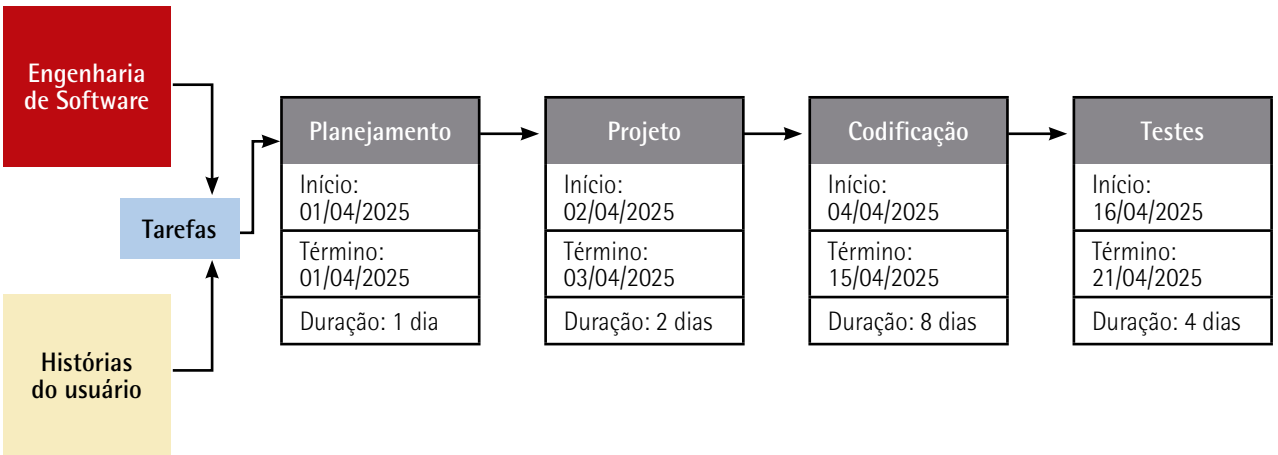


Figura 51 – Método Kanban aplicado à XP

7.3 Scrum

> Scrum (o nome provém de uma atividade que ocorre durante a partida de rugby) é um método de desenvolvimento ágil de software bastante popular concebido por Jeff Sutherland e sua equipe de desenvolvimento no início dos anos 1990 (Pressman, 2021, p. 126).

O Scrum está estruturado nos valores e nos princípios do Manifesto Ágil, buscando proporcionar uma abordagem iterativa e incremental para o desenvolvimento de software.

Essa metodologia é projetada para atender às necessidades de projetos nos quais os requisitos são frequentemente instáveis ou desconhecidos, o que é típico em ambientes dinâmicos e imprevisíveis. Ao invés de um planejamento detalhado desde o início, o Scrum adota ciclos curtos e repetitivos, conhecidos como sprints, que permitem ajustes constantes e a entrega de valor incremental ao cliente.

O Scrum se destaca como um framework que combina práticas orientadas a objetos e princípios ágeis, promove a colaboração intensa entre desenvolvedores, stakeholders e o product owner (dono do produto), garantindo maior transparência e adaptabilidade ao longo do ciclo de desenvolvimento.

Saiba mais

Elaborado por seus criadores, Ken Schwaber e Jeff Sutherland, o Scrum guide é considerado a principal referência para a aplicação do Scrum. Esse guia define regras, papéis, eventos e artefatos do framework. Sua versão mais recente foi atualizada em novembro de 2020, reforçando os princípios do Scrum como um framework adaptável para diferentes contextos.

Você pode obtê-lo gratuitamente no site oficial:

Disponível em: https://shre.ink/egv8. Acesso em: 23 jun. 2025.

Observe a seguir as características da metodologia Scrum:

- **Equipes pequenas e multifuncionais**: ênfase na formação de equipes com competências diversas, com maior integração e que trabalham de forma autônoma e colaborativa, ideal para contextos complexos. Assim, obtém-se uma rápida solução de problemas.

- **Requisitos emergentes e pouco definidos**: os requisitos são mutáveis e críticos do negócio. No Scrum, o conhecimento é compartilhado de forma estratégica para enriquecer o trabalho em equipe e acelerar resultados. O foco da equipe é a capacidade de absorver mudanças e adaptar-se às condições do projeto de forma eficiente, promovendo uma cultura de colaboração contínua, na qual o progresso é medido continuamente com o mínimo de riscos.

Observação

Na dinâmica Scrum do rúgbi, os jogadores trabalham em conjunto, formando uma unidade coesa para mover a bola pelo campo. Essa metáfora é utilizada para descrever o espírito colaborativo e a proximidade das equipes na metodologia de desenvolvimento Scrum.

- **Sprints curtas e entregas frequentes**: no framework, o Scrum é um subconjunto do product backlog, que representa um ciclo de trabalho determinado regularmente para ter uma duração de 1 a 4 semanas. Assim, a equipe se concentra para realizar um conjunto de tarefas ou funcionalidades. O ciclo iterativo reduz riscos e oferece visibilidade contínua do progresso, possibilitando decisões informadas em tempo hábil. Nesse período curto de tempo, os desenvolvedores devem realizar uma série de atividades e a cada nova sprint, o profissional adquire novas experiências com base na etapa anterior, levando a melhorias em cada sprint subsequente. O número de sprints necessárias no product backlog varia conforme o tamanho e a complexidade do artefato a ser construído.

- **Product backlog**: é uma lista priorizada de requisitos ou características do artefato que agrega valor ao software, contendo funcionalidades, melhorias, correções de bugs, requisitos e tarefas que a equipe pode trabalhar. É gerenciado pelo product owner, que prioriza os itens com base no valor para o negócio e nas necessidades dos stakeholders.

O Scrum promove reuniões diárias de 15 minutos e todos respondem às questões:

- O que realizou hoje?
- Está tendo alguma dificuldade?
- O que irá fazer amanhã?

Observação

No esporte e em muitas outras atividades ocorrem as seguintes etapas: no início empregam-se muita energia e velocidade para conseguir uma boa posição; durante a competição, administra-se todo o desempenho; no fim, aplicam-se toda atenção, energia, potência e velocidade até a exaustão. O objetivo é conquistar a melhor posição e, se possível, a vitória. Essa finalização é chamada de sprint.

Observe os seguintes papéis desempenhados:

- **Agile coach**: orienta várias equipes Scrum, coordenando as interações entre as equipes e promovendo práticas ágeis.
- **Product owner**: responsável por maximizar o valor do produto e gerenciar o backlog. Prioriza as funcionalidades, mas não gerencia diretamente a equipe.
- **Scrum master**: é o facilitador das tarefas a serem realizadas pela equipe, removendo obstáculos que possam impedir seu progresso, promovendo a colaboração e a comunicação na realização dos objetivos da sprint.
- **Equipe de desenvolvimento**: conjunto de profissionais que trabalham na criação do produto, nas práticas de especificação, na modelagem, na programação, na análise de dados etc. Pode inclusive incorporar outras metodologias ágeis para atividades específicas, como XP, FDD e DevOps.

Observe como funciona o Scrum no framework ilustrado na figura 52. O processo ágil é organizado em ciclos definidos por breves períodos, acompanhados de certa disciplina em compasso com a sprint. No framework vemos um conjunto de ações de desenvolvimento do processo de software, aplicados de forma sequencial, com diversas iterações para alinhamento do conhecimento e acompanhamento do processo. No final da sprint ocorre a entrega do software, por incremento, e uma nova funcionalidade é demonstrada.

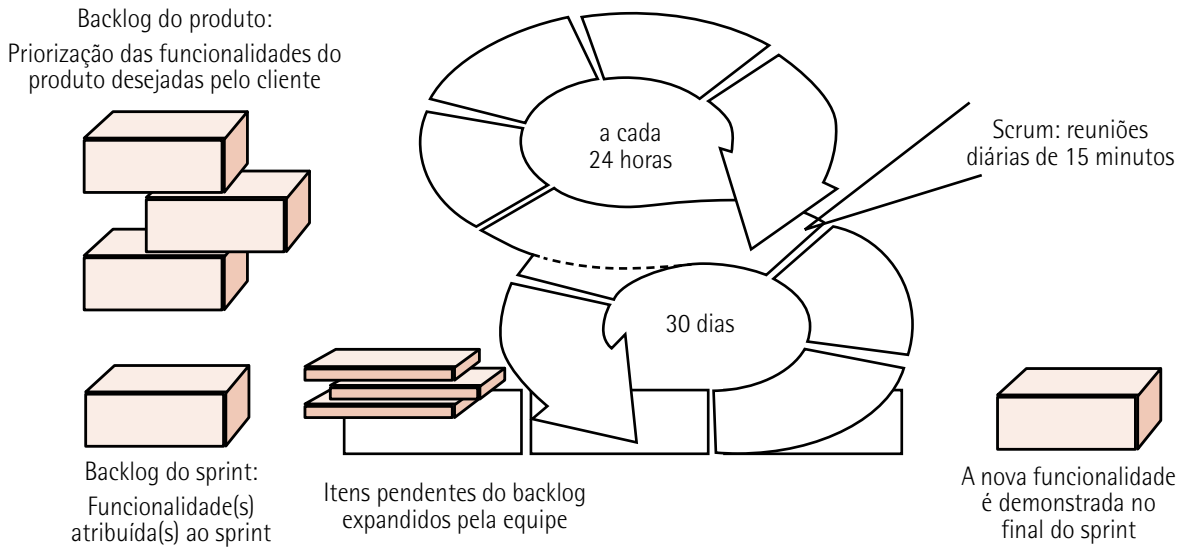


Figura 52 – Fluxo do processo Scrum

Adaptada de: Pressman (2021).

Acentuaremos a seguir as atividades estruturais da metodologia Scrum:

- **Planejamento**
  - **Sprint planning**: reunião inicial de cada sprint na qual o time define o objetivo, as tarefas prioritárias e os critérios de aceitação para o ciclo.
- **Execução**
  - **Daily Scrum**: reuniões rápidas diárias para sincronização, identificação de impedimentos e ajuste de planos, promovendo transparência.
  - **Desenvolvimento de incrementos**: o time trabalha para construir funcionalidades ou soluções utilizáveis, baseando-se no backlog priorizado.
- **Monitoramento e controle**
  - **Backlog refinement**: sessões para ajustar, detalhar ou repriorizar itens do backlog, garantindo alinhamento com as necessidades do produto.
  - **Sprint review**: demonstração de incrementos concluídos para stakeholders, permitindo feedback e validação das entregas.
- **Reflexão e melhoria**
  - **Sprint retrospective**: reunião de fechamento para refletir sobre o desempenho do time, identificar melhorias e implementar ações para sprints futuros.

Saiba mais

Explore mais a metodologia Scrum em:

Disponível em: https://shre.ink/egvn. Acesso em: 23 jun. 2025.

Um quadro Kanban pode ser integrado à metodologia Scrum, como é mostrado na figura 53. Com esse método, o ciclo de desenvolvimento do Scrum é feito a partir das etapas que se alinham ao Kanban.

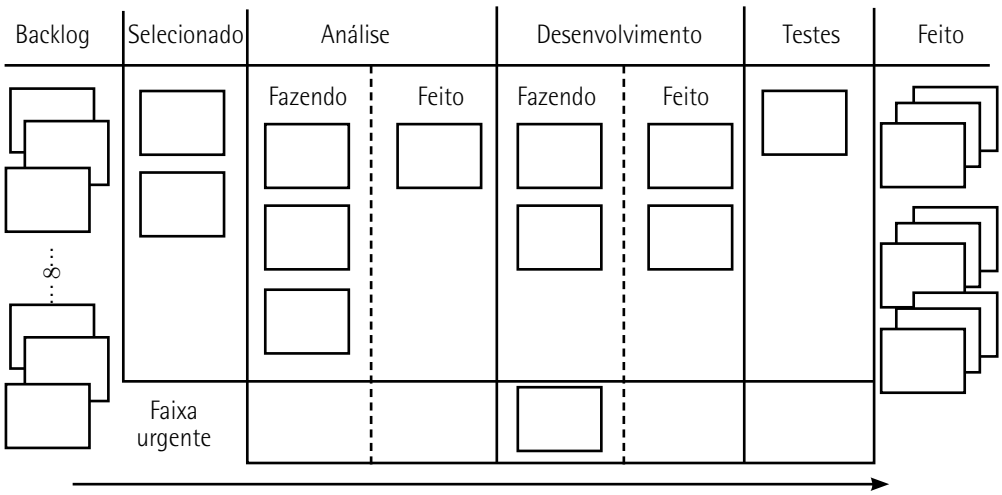


Figura 53 – Quadro Kanban

Adaptada de: Pressman (2021).

7.4 FDD

Feature Driven Development (FDD) é uma metodologia ágil que se destaca pela sua abordagem estruturada e orientada a objetos. O FDD é apropriado para equipes que desejam aprimorar a colaboração e aumentar a eficiência na entrega de projetos complexos.

Isso porque esta metodologia ágil enfatiza a importância de se dividir o desenvolvimento em características palpáveis, permitindo um progresso mensurável e contínuo.

Ou seja, ao entendê-la, os desenvolvedores e gestores de projeto podem descobrir novas maneiras de enfrentar desafios, otimizar workflows e alcançar objetivos com maior clareza e previsibilidade (CTC, 2024).

O FDD tem como foco a entrega incremental de funcionalidades visíveis e utilizáveis, sendo adaptado a projetos que exigem alto nível de adaptabilidade tecnológica.

Com o uso do FDD, os stakeholders têm acesso a entregas rápidas e relatórios de progresso estruturados em uma linguagem acessível, facilitando o alinhamento entre as expectativas de negócio e o desenvolvimento técnico.

Os gerentes de projeto se beneficiam de uma visibilidade abrangente e precisa do acompanhamento do projeto, com métricas objetivas que permitem tomadas de decisão ágeis e baseadas em dados.

Para os engenheiros de software, o modelo favorece ciclos curtos de desenvolvimento, possibilitando o início de novas funcionalidades em poucos dias e promovendo uma participação ativa em todas as etapas do processo, desde a análise de requisitos até o design e, consequentemente, a implementação de código.

No contexto do FDD, o termo central utilizado para controle e organização do projeto é chamado de característica (feature), uma unidade funcional de software que representa um comportamento do sistema que agrega valor perceptível para o usuário final.

Uma característica é uma funcionalidade granular, normalmente de pequena escala, que deve ser definida de forma colaborativa com o cliente ou as partes interessadas. Essa definição promove alinhamento entre os requisitos de negócio e a arquitetura técnica do sistema, permitindo que os engenheiros de software traduzam necessidades em implementações concretas.

A robustez da metodologia reside em sua combinação de práticas tradicionais de engenharia de software, como modelagem OO, com aspectos ágeis, como adaptação a mudanças e foco na colaboração contínua. Essa fusão permite um fluxo de trabalho adaptável e eficiente, garantindo alinhamento técnico e estratégico.

Observe a seguir os principais aspectos das características:

- As características são modeladas como unidades funcionais independentes ou pequenos blocos de funcionalidades, cuja definição considera o tamanho, o escopo e a complexidade da aplicação a ser desenvolvida. Isso permite maior rastreabilidade, mensuração de progresso e controle sobre o ciclo de desenvolvimento.

- Estruturalmente, essas características são agrupadas segundo critérios hierárquicos ou prioridades estratégicas do negócio, o que viabiliza a gestão eficiente do backlog técnico e o alinhamento com os objetivos organizacionais.

- A equipe de desenvolvimento trabalha com metas iterativas, comumente em ciclos quinzenais (timeboxes), com foco na entrega incremental de novas características. Essa abordagem impulsiona a entrega contínua de valor e promove práticas ágeis de engenharia, como integração contínua, revisão técnica frequente e validação incremental.

Observe agora o framework do FDD ilustrado na figura 54. A partir dos requisitos de software, iniciam-se a concepção e o planejamento, com o foco em criar um plano para a feature. A partir daí, em ciclos iterativos de desenvolvimento, é realizada a construção da feature, que faz parte do pacote de trabalho, gerando como resultado o software e os modelos de objetos, que podem ser reusados.

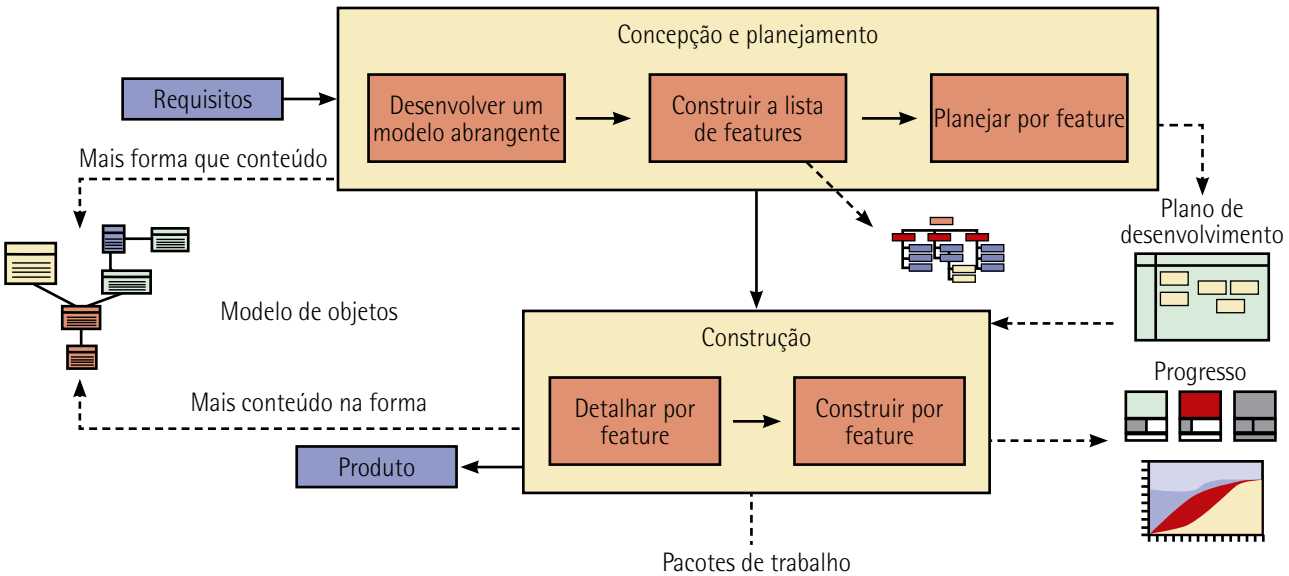


Figura 54 – Engenharia de software com FDD

O FDD é a metodologia ágil de desenvolvimento que mais enfatiza diretrizes e técnicas de gestão de projetos. O projeto é claro para todos os envolvidos e essa metodologia pode ser integrada com outras metodologias ágeis, como o Scrum. O FDD é composto por cinco processos principais (listados a seguir), que organizam e guiam o ciclo de desenvolvimento desde a modelagem inicial até a implementação de funcionalidades específicas. Ao final, a funcionalidade está pronta para entrega e avaliada.

- **Desenvolver um modelo geral**: criação de um modelo conceitual abrangente para entender os RS e criar uma visão de alto nível. Utilizam-se técnicas como modelagem de domínio, diagramas de classes e mapas conceituais. O objetivo é estabelecer a estrutura e os principais componentes do sistema antes de detalhar as funcionalidades.

- **Construir a lista de características**: com base no modelo desenvolvido, é criada uma lista hierárquica de funcionalidades pequenas, claras e orientadas ao negócio. Cada característica representa algo que o sistema deve fazer e é escrita em linguagem voltada ao cliente, como: "relatório de clientes". A lista é organizada em grupos chamados de grandes funcionalidades (major feature sets) e conjuntos de funcionalidades (feature sets).

- **Planejar por característica**: é estruturar o trabalho a partir de prioridades. Nesse processo, é feito o planejamento da implementação com base nas características identificadas. Atribuem-se responsabilidades como analistas de sistemas, programadores-chefe e outros e define-se a sequência de desenvolvimento. As funcionalidades são priorizadas com base em valor de negócio, dependências e complexidade.

- **Projetar por característica**: engloba a criação de designs específicos, quando cada funcionalidade é projetada detalhadamente antes da codificação. O programador-chefe seleciona a equipe que trabalhará na característica. É feita uma revisão de design envolvendo a equipe técnica, resultando em diagramas, documentos técnicos e decisões de arquitetura.

- **Construir por característica**: é o processo de codificação, teste e integração da característica ao sistema. Envolve implementação de código, testes unitários, integração contínua e revisão de código.

8 METODOLOGIAS ÁGEIS II

Neste tópico vamos trazer uma visão geral sobre algumas outras aplicações das metodologias ágeis, como DSDM, Crystal, AM e DevOps.

8.1 DSDM

É uma abordagem iterativa e incremental de CBSE (Component-Based Software Engineering – Engenharia de Software Baseada em Componentes) para o projeto de sistemas de software.

Criado no Reino Unido na década de 1990, o DSDM destacou-se por sua flexibilidade e seu foco na entrega rápida de soluções de alta qualidade, mesmo em cenários com prazos e orçamentos limitados. Ele é amplamente utilizado em projetos com escopos dinâmicos, permitindo adaptações rápidas às mudanças e garantindo que os objetivos sejam alcançados de forma prática e eficiente.

O DSDM surgiu como uma extensão do RAD. Com cronogramas e custos limitados, é aplicado em projetos de sistemas de software com foco em especificação, integração de seus componentes e testes para verificar se o sistema atende aos requisitos especificados.

Lembrete

Com diversas opções de metodologias ágeis, vale lembrar que a melhor aplicação de uma metodologia ágil é aquela que atende especificamente aos requisitos de negócio.

Saiba mais

O DSDM é particularmente útil em situações de componentização do software e que ajudam a manter o projeto no caminho certo, com limitações claras de tempo e orçamento.

Acesse o conteúdo indicado a seguir e conheça mais sobre essa metodologia ágil.

DSDM: tudo que você precisa saber sobre essa metodologia ágil. *FlowUp*, 13 fev. 2020. Disponível em: https://shre.ink/ebpk. Acesso em: 23 jun. 2025.

Os princípios fundamentais do DSDM são:

- **foco na necessidade do negócio**: priorizar entregas que agreguem valor ao cliente;
- **entrega dentro do prazo**: garantir que os projetos sejam concluídos de forma eficiente;
- **colaboração**: promover o trabalho conjunto entre equipes e stakeholders;
- **qualidade inegociável**: assegurar que os padrões de qualidade sejam mantidos;
- **desenvolvimento incremental e iterativo**: melhorar continuamente as entregas com base em feedbacks.

O framework do DSDM apresentado na ilustração da figura 55 é composto por oito fases principais, organizadas de forma cíclica e contínua.

- **Viabilidade (feasibility) – início do ciclo**: avalia se o projeto é tecnicamente viável e se faz sentido do ponto de vista do negócio. As ações se baseiam em fazer uma análise técnica preliminar, estimando inicialmente custos, tempo, recursos disponíveis e análise de riscos. Essa etapa inicial se refere à concepção do projeto, e somente depois dela se inicia o projeto.
- **Pré-projeto (pre-project)**: garante que apenas projetos com uma base sólida e justificativa de negócios sejam iniciados. Seu objetivo é confirmar que existe uma justificativa clara para o projeto. Consiste nas práticas de análise inicial de viabilidade, definição de escopo em alto nível e identificação de stakeholders.
- **Fundamentos (foundations)**: essa fase se preocupa em fixar uma base firme para o projeto antes de iniciar o desenvolvimento iterativo. As principais atividades são: definir a arquitetura, fazer a modelagem do processo de negócio, incluindo critérios de aceitação e plano de entrega.

- **Exploração (exploration)**: nessa fase são desenvolvidos incrementos de solução que atendam aos requisitos priorizados. As principais atividades estão no desenvolvimento colaborativo com usuários, prototipagem rápida, iterações frequentes e feedbacks constantes.

- **Desenvolvimento iterativo (iterative development)**: é o núcleo do DSDM. Essa fase busca garantir que a solução esteja tecnicamente robusta e pronta para produção. São realizados refino técnico, testes mais profundos e integração com sistemas de software existentes.

- **Implantação (deployment)**: é a entrega da solução para uso real, garantindo que esteja pronta e validada pelos usuários. A fase de implantação consiste em treinamento, migração de dados, planejamento da transição e entrega do incremento.

- **Pós-projeto (post-project)**: avalia os resultados do projeto para assegurar uma manutenção contínua da solução. Realizam-se revisões das lições aprendidas, avaliação de benefícios e plano de suporte.

- **Viabilidade (feasibility) – fim do ciclo**: é quando o produto é construído incrementalmente com base em ciclos curtos. Nos ciclos de planejamento, ocorrem o desenvolvimento e revisões contínuas com foco na entrega de valor.

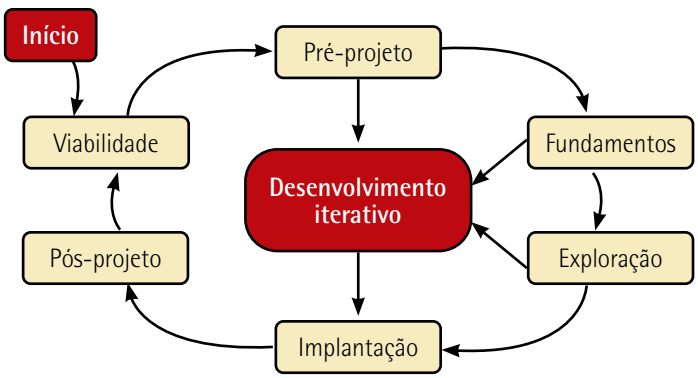


Figura 55 – Framework do DSDM

8.2 Crystal

De acordo com Pressman (2011), a família Crystal de metodologias ágeis foi concebida por Alistair Cockburn em 2005 com contribuições de Jim Highsmith em 2002. No contexto da busca por métodos mais flexíveis, humanos e adaptáveis no desenvolvimento de software, eles criaram a família Crystal.

A nomenclatura faz uma analogia direta com os cristais geológicos como Crystal Clear, Crystal Yellow e Crystal Orange, que se diferenciam pela cor, forma e dureza. Cada metodologia da família é representada por uma cor refletindo sua adequação ao nível de criticidade do projeto e ao número de pessoas envolvidas na equipe. Quanto maior o risco do sistema, como risco de morte, mais rigorosa será a metodologia correspondente.

O Crystal é fundamentado em princípios fortemente centrados nas relações humanas, comunicação e simplicidade. Ponderando que todo projeto tem necessidades, convenções e uma metodologia diferente, o Crystal destaca-se por valorizar os princípios do Manifesto Ágil: pessoas e interações mais do que processos e ferramentas, iterações frequentes com entregas operacionais, reflexão e melhoria contínua.

O método Crystal Clear, por exemplo, é direcionado a projetos com baixo risco e com equipes que variam entre seis e oito desenvolvedores. Suas principais práticas são:

- **Entrega frequente (frequent delivery)**: refere-se à entrega contínua de versões executáveis do sistema em intervalos curtos, que variam entre um a três meses, permitindo obter feedback constante do cliente, reduzindo erros acumulados e mantendo o produto alinhado às necessidades do negócio.

- **Melhoria reflexiva (reflective improvement)**: a equipe realiza pausas regulares para refletir sobre o processo de desenvolvimento, identificando pontos fracos e melhoria contínua. Promove um ciclo de aprendizado organizacional, elevando a maturidade da equipe e a qualidade do produto.

- **Segurança pessoal (personal safety)**: garantias de que cada membro da equipe possa expressar suas opiniões, preocupações e ideias sem medo de retaliação. Assim, cria-se um ambiente psicológico seguro, que estimula a criatividade, a colaboração e a resolução de problemas de forma aberta e honesta.

- **Acesso fácil a usuários especialistas (easy access to expert users)**: assegura que desenvolvedores tenham acesso direto e frequente a usuários finais ou especialistas no domínio do sistema. Esse acesso facilita o esclarecimento de dúvidas, o refinamento de requisitos e a validação de decisões em tempo real.

- **Testes automatizados (automated tests)**: são incorporados testes automatizados como parte integrante do processo de desenvolvimento, de forma a garantir qualidade contínua do código, redução de regressões e permissões de refatoramento seguro, dando suporte ao desenvolvimento incremental e integração contínua.

- **Comunicação osmótica (osmotic communication)**: envolve a comunicação informal e constante que ocorre naturalmente quando a equipe trabalha fisicamente próxima, compartilhando o mesmo ambiente, favorecendo a troca rápida de informações, as decisões eficientes e a redução da necessidade de reuniões formais.

- **Foco (focus)**: as condições de trabalho devem favorecer a concentração e a produtividade, evitando interrupções desnecessárias, tarefas paralelas excessivas e ambientes ruidosos.

Saiba mais

Conheça mais a metodologia ágil Crystal, suas categorias e como ela pode ser aplicada na empresa em:

ENTENDA o que é a metodologia Crystal e saiba como ela pode ser aplicada no dia a dia das empresas. *Pontotel*, 5 out. 2023. Disponível em: https://shre.ink/ebDo. Acesso em: 23 jun. 2025.

8.3 AM

> Modelagem Ágil (AM) consiste em uma metodologia prática, voltada para a modelagem e documentação de sistemas baseados em software. Simplificando, Modelagem Ágil consiste em um conjunto de valores, princípios e práticas voltados para a modelagem do software que podem ser aplicados a um projeto de desenvolvimento de software de forma leve e eficiente. Os modelos ágeis são mais eficientes do que os tradicionais pelo fato de serem simplesmente bons, pois não têm a obrigação de ser perfeitos (Pressman, 2016, p. 81).

A AM oferece uma abordagem estruturada, mas flexível, para representar e documentar software em projetos de desenvolvimento. Esse framework enfatiza a colaboração, a adaptabilidade e o uso de artefatos de modelagem que são "suficientemente bons" para atingir os objetivos do projeto ao invés de buscar uma perfeição detalhista, que pode se tornar obsoleta ou onerosa.

Ela apoia a criação de modelos que têm papel essencial na comunicação entre as equipes de desenvolvimento, stakeholders e designers; garantindo alinhamento técnico e funcional. Com uma abordagem iterativa e incremental, os modelos evoluem com o software, permitindo ajustes rápidos para atender às mudanças de requisitos e demandas do projeto.

A expressão "viajar leve" é central na filosofia AM. Ela orienta que apenas os modelos com valor prático e estratégico sejam mantidos a longo prazo, enquanto modelos temporários ou descartáveis são criados e eliminados conforme sua utilidade decresce. Essa abordagem apoia a engenharia de software moderna ao priorizar agilidade, redução de desperdícios e foco na entrega de valor contínuo.

O site oficial da Agile Modeling (AM, 2020) destaca alguns conceitos importantes:

> **Modelo**: um modelo é uma abstração que expressa aspectos importantes de uma coisa ou conceitos. Os modelos podem ser visuais (diagramas), não visuais (descrições de texto) ou executáveis (código de trabalho ou equivalente). Às vezes, os modelos são chamados de mapas ou roteiros dentro da comunidade ágil.

Modelo ágil: Os modelos ágeis podem ser algo tão simples quanto adesivos em uma parede, esboços em um quadro branco, diagramas capturados digitalmente por meio de uma ferramenta de desenho ou modelos detalhados capturados usando uma ferramenta de engenharia de software baseada em modelo (MBSE – model-based software engineering).

Modelagem: modelagem é o ato de criar um modelo. A modelagem às vezes é chamada de mapeamento.

AM: é realizada de forma colaborativa e evolutiva.

Documento: um documento é uma representação persistente de uma coisa ou conceito. Um documento é um modelo, mas nem todos os modelos são documentos (a maioria dos modelos não é persistente).

Documento ágil: os documentos ágeis podem ser algo tão simples quanto notas pontuais, texto detalhado, testes executáveis ou um ou mais modelos ágeis.

Existe uma variedade de práticas ágeis e, de acordo com AM (2020), as práticas de design ágil vão desde práticas de arquitetura de alto nível até práticas de programação de baixo nível como é mostrado na figura 56. Cada uma dessas práticas é importante e cada uma é necessária para que sua equipe seja eficaz no design ágil. Em uma sequência de eventos, as práticas ágeis fazem a seguinte abordagem:

Arquitetura	**Previsão de arquitetura**: modelagem leve no início do projeto para identificar e pensar em problemas críticos no nível da arquitetura
	Modelagem da iteração: modelagem leve por alguns minutos no início de uma iteração/sprint para ajudar a identificar a estratégia da equipe para essa iteração. É a parte do esforço de planejamento da iteração/sprint
	Tempestade de modelos: modelagem leve por alguns minutos no tempo certo (JIT, Just-In-Time) para pensar no aspecto de uma solução
	Modelo de TFD (Test-First Design – primeiro-teste): escrever um único teste, antes de escrever o código de produção, suficiente para atender o teste
Programação	**Refatoração**: fazer pequenas alterações em uma parte da solução que melhorem a qualidade sem alterar a semântica dessa parte
	Integração contínua: compilar, testar e validar automaticamente os componentes da solução sempre que um desses componentes for alterado

Figura 56 – Prática de design ágil

Saiba mais

Para obter mais informações sobre o AM, acesse:

Disponível em: https://shre.ink/egCC. Acesso em: 23 jun. 2025.

8.4 DevOps

> O DevOps foi criado por Patrick DeBois para combinar Desenvolvimento (Development) e Operações (Operations). O DevOps tenta aplicar os princípios do desenvolvimento ágil e do enxuto a toda a cadeia logística de software (Pressman, 2021, p. 51).

A figura 57 mostra o fluxo de trabalho típico do DevOps, um paradigma amplamente adotado na engenharia de software para integrar equipes de desenvolvimento e operações em um ciclo contínuo de entrega de software.

Essa abordagem promove automação, colaboração e integração ao longo de diversas etapas essenciais, como planejamento, desenvolvimento, testes, integração contínua, entrega contínua, monitoramento e feedback. Elas são interligadas em um ciclo iterativo que permite ajustes rápidos e eficiência ao longo do desenvolvimento.

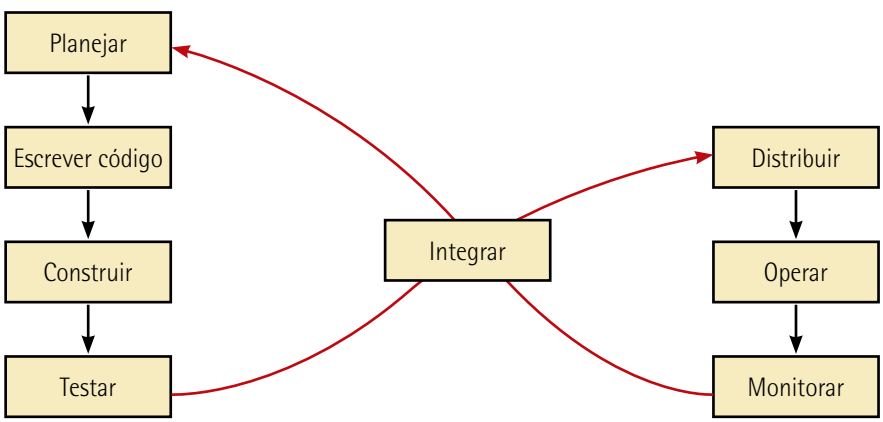


Figura 57 – Framework do DevOps

Adaptada de: Pressman (2021).

Do ponto de vista tecnológico, o DevOps faz uso de ferramentas especializadas, como Jenkins, Docker e Kubernetes, para automatizar processos críticos, desde a integração de código até o provisionamento e o monitoramento de infraestrutura. Elas são frequentemente utilizadas em conjunto: Jenkins para automação, Docker para conteinerização e Kubernetes para orquestração. Esse trio cria um fluxo de trabalho DevOps moderno, eficiente e resiliente.

Jenkins

É uma ferramenta de automação open-source focada em CI (Continuous Integration – Integração Contínua) e CD (Continuous Delivery – Entrega Contínua (CD). Ela permite que equipes automatizem tarefas como construção, teste e implantação de software, reduzindo o trabalho manual e acelerando os ciclos de desenvolvimento.

Entre seus recursos, destacam-se: suporte a pipelines declarativos, integração com repositórios como Git e uma ampla biblioteca de plugins que permite personalizar e estender sua funcionalidade.

Exemplo de uso: sempre que um desenvolvedor realiza uma atualização no código Jenkins pode automaticamente compilar o código, executar testes e, se aprovado, implantar a aplicação no ambiente de staging ou produção.

Docker

É uma plataforma de conteinerização que permite empacotar aplicações e suas dependências em contêineres leves e portáteis. Ele garante que o software seja executado de forma consistente em diferentes ambientes do desenvolvimento, testes e produção, independentemente do sistema operacional ou da infraestrutura subjacente. Como benefícios, citam-se: redução de conflitos de dependências, maior escalabilidade e facilidade de migração de aplicações.

Exemplo de uso: uma equipe pode criar uma imagem Docker para sua aplicação web, garantindo que ela será executada da mesma maneira em um laptop de desenvolvimento ou em um cluster na nuvem.

Kubernetes (ou K8s)

É um sistema de orquestração de contêineres projetado para gerenciar grandes volumes de contêineres em ambientes distribuídos. O K8s automatiza tarefas complexas, como balanceamento de carga, escalonamento automático, gerenciamento de clusters e implantação contínua.

Tal dispositivo promove o controle por escalonamento automático (autoscaling), permitindo que uma funcionalidade de software se ajuste dinamicamente à infraestrutura de software, ambientes de computação em nuvem e a recursos disponíveis de acordo com a demanda do sistema. Também permite a verificação contínua do status e do funcionamento correto dos componentes da aplicação com reinício automático de contêineres defeituosos.

Exemplo de uso: uma organização pode usar Kubernetes para implantar e gerenciar microsserviços em um cluster de servidores na nuvem, garantindo alta disponibilidade e recuperação automática em caso de falhas.

O DevOps tem como objetivo otimizar a entrega de valor ao cliente por meio de ciclos curtos e iterativos, garantindo flexibilidade para atender às mudanças de requisitos e necessidades do mercado, bem como às metodologias Scrum e XP, que também enfatizam a colaboração, a comunicação eficaz entre equipes multifuncionais e o aprendizado contínuo.

Exemplo de aplicação

Caso: monitoramento e atualização constante no e-commerce.

Uma plataforma de e-commerce precisa ser constantemente atualizada para incluir novos recursos, corrigir bugs e lidar com picos de tráfego durante promoções sazonais.

Etapas do workflow

1. CI: desenvolvedores trabalham em diferentes partes do código e enviam as alterações para um repositório compartilhado, como o Git. Com a ajuda de ferramentas como Jenkins ou GitHub Actions, o código é automaticamente integrado, testado e construído; garantindo que novas implementações não comprometam o sistema existente.

2. CD: após a passagem do código nos testes automáticos, o código é automaticamente empacotado em contêineres Docker e implantado em um ambiente de pré-produção para validação final.

3. IaC (Infrastructure as Code – Infraestrutura como Código): a infraestrutura de hospedagem da plataforma (servidores, balanceadores de carga e bancos de dados) é gerenciada por meio de scripts usando ferramentas como Terraform ou AWS CloudFormation, permitindo replicar o ambiente em minutos.

4. Monitoramento contínuo: uma vez em produção, ferramentas como Prometheus e Grafana monitoram o desempenho da aplicação, verificando métricas como latência, uso de CPU e comportamento de banco de dados. São configurados alertas para identificar problemas em tempo real, como quedas no desempenho ou falhas nos serviços.

5. Autoscaling e resiliência: durante grandes eventos do comércio como Black Friday e períodos de fim de ano, em que há uma demanda maior no e-commerce, o Kubernetes gerencia o escalonamento automático de pods (componentes de implantação) para lidar com o aumento do tráfego. Caso ocorra uma falha no pod, ele é automaticamente reiniciado para evitar interrupções no serviço.

8.5 Métodos, ferramentas e técnicas: aplicativos para operações com metodologias ágeis

Normalmente, em metodologias ágeis, o gerenciamento de projetos e organização de tarefas é baseado em sistema visual de quadros, listas e cartões; acompanhado de regras que simplificam processos repetitivos, como mover cartões automaticamente ao serem concluídos.

Observação

Cada backlog pode ser refinado e dividido em tarefas menores durante o planejamento da sprint, garantindo que a equipe Scrum mantenha foco e transparência.

Cada sprint pode ser organizada em um quadro específico, permitindo que a equipe acompanhe o progresso de forma colaborativa.

Nessa categoria de serviços, exploraremos uma ferramenta de software que atenda a essas necessidades de serviços.

Existem várias ferramentas, a exemplo do Trello, uma ferramenta visual e intuitiva que facilita o gerenciamento de projetos e a organização de tarefas, sendo altamente eficiente quando aplicado à metodologia Scrum.

Os quadros do Trello representam o projeto, as listas são usadas para organizar o fluxo das tarefas e os cartões representam as histórias ou tarefas individuais e que podem conter informações detalhadas.

Os principais recursos do Trello são:

- **Quadros**: representam projetos ou áreas de trabalho.
- **Listas**: organizam tarefas em diferentes etapas, como "a fazer", "em progresso" e "concluído".
- **Cartões**: contêm tarefas específicas, que podem incluir descrições, verificações (checklists), prazos, anexos e comentários.
- **Automação**: com o recurso Butler, é possível automatizar fluxos de trabalho repetitivos.
- **Integrações**: pode-se conectar a outras ferramentas, como Slack, Google Drive e Jira.
- **Colaboração**: permite que equipes trabalhem juntas em tempo real, com notificações e atualizações instantâneas.

A figura 58 acentua os quadros Trello, templates e outras funções. São diversas as aplicações na gestão de tarefas e equipes, para os mais diversos processos e métodos da engenharia de software. Os templates são modelos úteis para diversos tipos de negócios, como design, engenharia, gestão de projetos e produtividade.

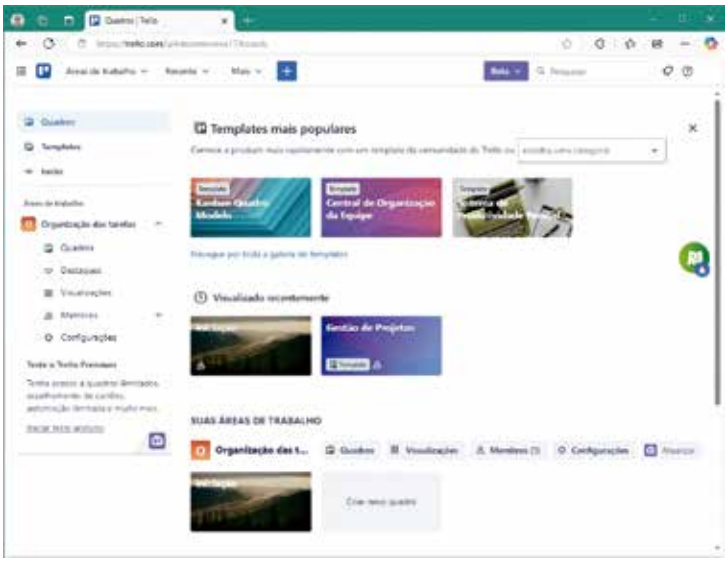


Figura 58 – Quadros Trello

Disponível em: https://shre.ink/ebkn. Acesso em: 23 jun. 2025.

Saiba mais

Para conhecer melhor o assunto tratado, acesse o site oficial da Trello:

Disponível em: https://shre.ink/egCe. Acesso em: 23 jun. 2025.

Exemplo de aplicação

Caso: Scrum, organização de backlogs e sprints em Kanban com uso da ferramenta Trello.

Aplicação: implementação em software IoT utilizando a metodologia Scrum.

A seguir foram selecionados alguns backlogs para a implementação de um software IoT. Use o Trello e acompanhe as operações a seguir:

1. Conexão de dispositivos

- implementar APIs para conexão entre dispositivos IoT;
- configurar protocolos de comunicação (MQTT ou HTTP).

2. Coleta e armazenamento de dados

- projetar bancos de dados para dados coletados;
- otimizar armazenamento em nuvem ou local;
- codificar e implementar app.

3. UX e UI

- melhorar interface para usuários administrarem dispositivos;
- testar usabilidade com clientes reais.

No início, escolha um template para um negócio específico e monte o quadro de cartões. O modelo de cartões dos backlogs apresentados na figura 59 foi criado com o nome "Modelo Scrum – Kanban". Como entrada de dados, essa é a primeira etapa para a distribuição de períodos e responsabilidades.

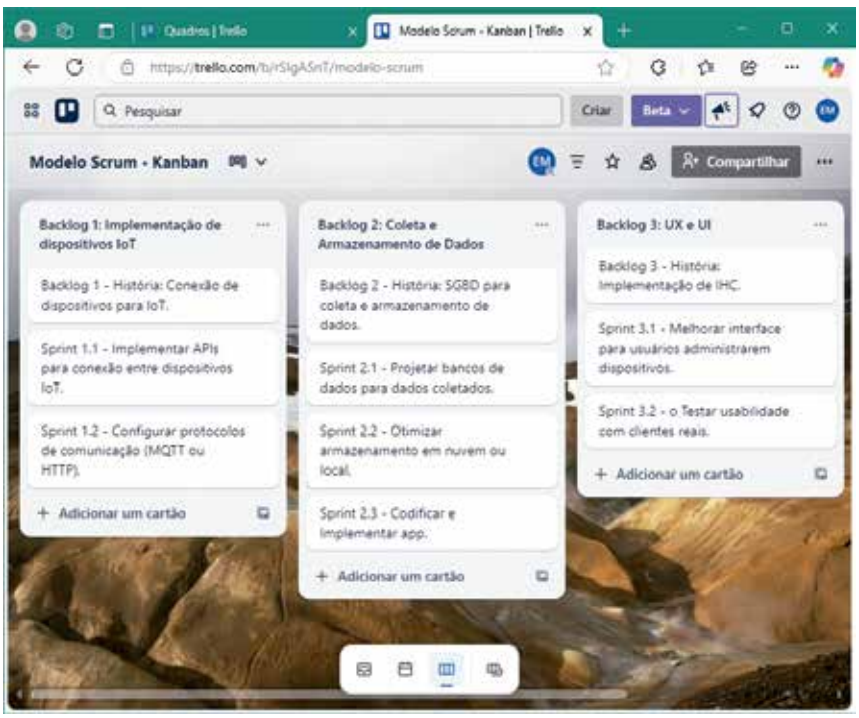


Figura 59 – Cartões "Modelo Scrum – Kanban"

Após a operação inicial, o desenvolvedor pode configurar a sua forma com diversas funções mostradas na figura 60.

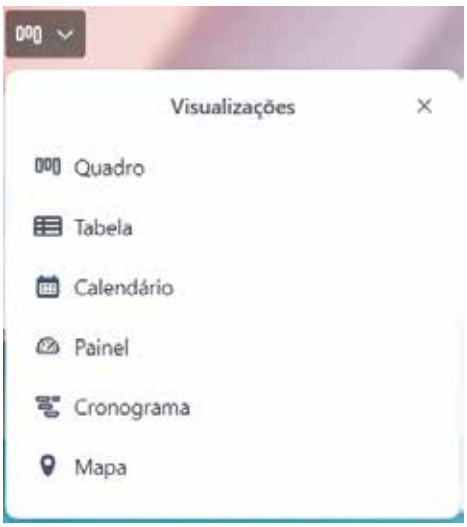


Figura 60 – Cartões "Modelo Scrum – Kanban"

Com o template "Kanban Quadro Modelo", é possível associar o backlog com seu respectivo status e sequência pelos cartões: "Backlog", "Design", "A Fazer", "Em andamento", "Revisão de código", "Fase de teste" e "Concluído" e outros que queira criar. Veja como fica a tela de operação "Tabela", mostrada na figura 61.

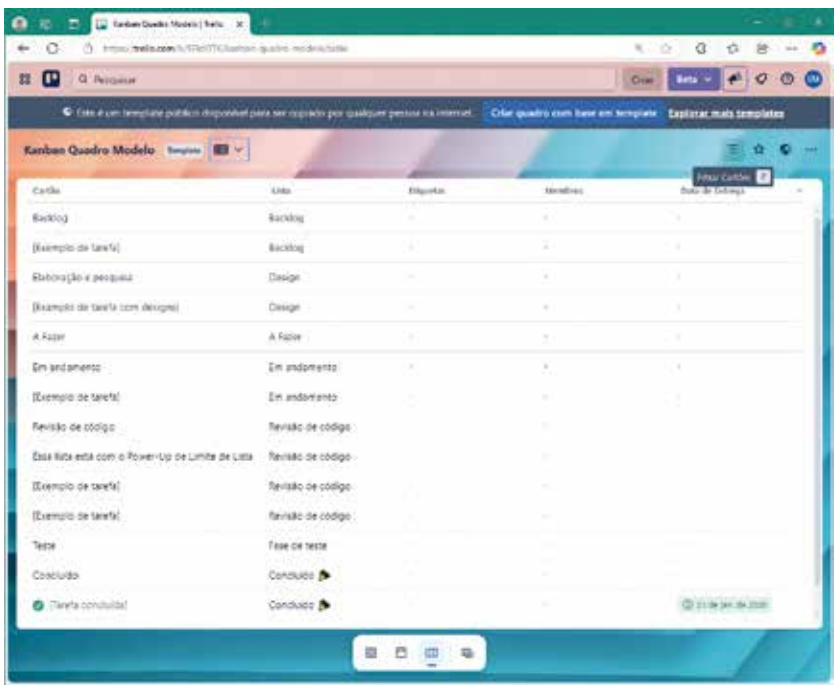


Figura 61 – Trello: Template "Kanban Quadro Modelo", função Tabela

Na figura 62 vemos a função Painel, que mostra os níveis de eficácia das tarefas em andamento naquele instante.

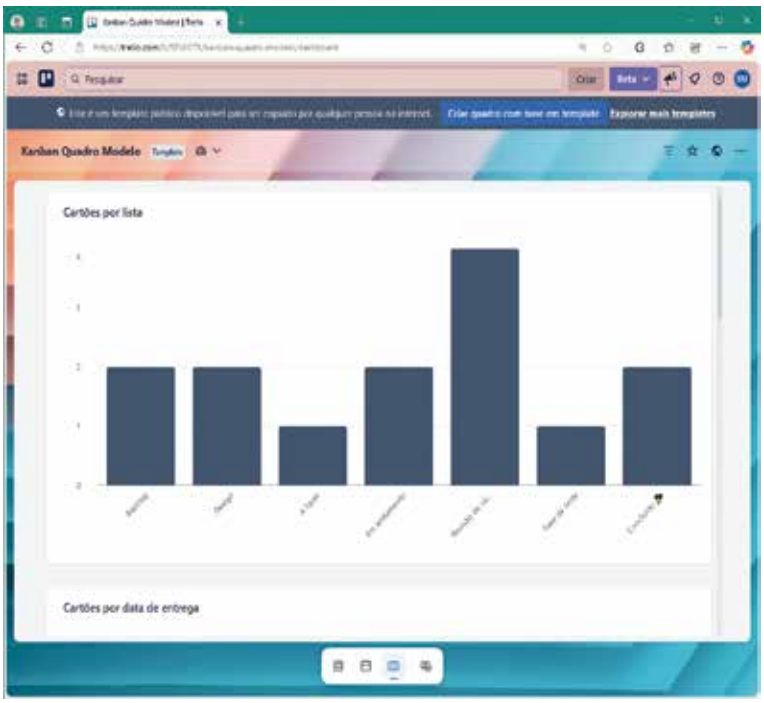


Figura 62 – Trello: Template "Kanban Quadro Modelo", função Painel

Enfim, o uso da ferramenta Trello, aplicada com a metodologia Scrum, promove transparência, agilidade e organização; tornando o gerenciamento de projetos mais visual e acessível para todos os membros da equipe. Facilita a comunicação entre equipes por meio de quadros, listas e cartões intuitivos, personalizados para diferentes metodologias. Como se adapta a necessidades específicas, integra-se com outras plataformas, com recursos de automação de tarefas repetitivas, otimizando o tempo e aumentando a produtividade de software.

Resumo

Esta unidade aborda os fundamentos, as práticas e as ferramentas relacionadas às metodologias ágeis de desenvolvimento de software, dividindo-se em duas partes principais, as metodologias ágeis I e II.

Inicialmente, destacamos o Manifesto Ágil, que estabelece valores e princípios orientados no desenvolvimento ágil de software como a valorização da colaboração com o cliente, a entrega contínua de software funcional e a rápida resposta às mudanças. Em seguida, apresentamos as vantagens e as desvantagens entre os modelos estruturados, como RUP e os modelos ágeis, com destaque para o Scrum.

Na sequência, foram acentuadas metodologias ágeis específicas que se destacam no mercado entre as dezenas existentes, como XP, cujo foco envolve técnicas rigorosas, como programação em par, integração contínua e testes automatizados; e Kanban, um método visual de gerenciamento de tarefas por meio de cartões para melhorar o fluxo de trabalho contínuo.

Vimos que o baixo uso de algumas metodologias ágeis ocorre porque elas são metodologias mais específicas, porém de grande uso e aplicabilidade, como é o caso do FDD, um processo orientado por funcionalidades com ênfase em planejamento incremental e design baseado em modelos.

A segunda parte (metodologias ágeis II) explora outras metodologias ágeis específicas e completas para diversos processos e atividades do desenvolvimento de software e que podem ser aplicadas isoladamente ou integradas a outros modelos de processos estruturados ou metodologias ágeis.

Nesse contexto, acentuamos o DSDM, uma extensão do modelo de processo RAD. É um framework iterativo e incremental para entrega rápida de soluções de alta qualidade com orçamentos limitados. Também citamos a família Crystal de metodologias ágeis, fortemente centrada nas relações humanas, comunicação e simplicidade, elementos considerados ajustáveis conforme o tamanho e a criticidade do projeto.

Prosseguindo nossos estudos, elencamos as ferramentas e as técnicas de AM, um conjunto de práticas que promove a simplicidade e a comunicação sob o princípio de "viajar leve", que orienta apenas manter os modelos com valor prático e estratégico.

As metodologias ágeis trouxeram eficiência na forma de produzir software e a cada dia surgem técnicas e frameworks que prometem ainda mais melhorar essa eficiência, como é o caso do DevOps, que integra desenvolvimento e operações para automatizar processos e acelerar entregas com qualidade.

Por fim, estudamos a ferramenta Trello, com recursos de criar quadros, listas, cartões, automação, integrações e colaboração. Ela é amplamente utilizada nas metodologias ágeis Scrum e Kanban; tornou-se popular devido à sua interface intuitiva e flexível, apropriada para equipes menores ou projetos menos complexos.

Exercícios

Questão 1. (UFU-MG 2023, adaptada) Metodologias de desenvolvimento de software chamadas de ágeis são baseadas em desenvolvimento iterativo, no qual requisitos e soluções evoluem pela colaboração entre equipes auto-organizadas. Essas metodologias encorajam frequentes inspeção e adaptação, alinhamento entre o desenvolvimento e os objetivos dos clientes e um conjunto de boas práticas que permita entregas rápidas e de qualidade.

Considerando as metodologias ágeis de desenvolvimento de software, avalie as afirmativas.

I – O Scrum adota uma abordagem empírica, pois entende que o problema pode não ser totalmente compreendido nem estar totalmente definido na análise e que os requisitos podem mudar com o passar do tempo. Essa abordagem mantém o foco em maximizar a habilidade da equipe em responder de forma ágil aos desafios emergentes.

II – Um dos valores do método XP é a busca pela simplicidade, de modo que o software deve ser mantido o mais simples possível pelo maior tempo possível.

III – O TDD é uma prática que envolve pequenas iterações, assim, novos casos de teste são escritos considerando uma melhoria ou uma nova funcionalidade. O código necessário é implementado para atender a esse teste.

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 E.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: o Scrum é uma metodologia ágil que enfatiza a adaptabilidade da equipe de desenvolvimento às mudanças. Ele é aplicado a equipes que têm o foco em construir e integrar softwares com requisitos pouco estáveis, desconhecidos ou que mudam com frequência. Ao adotarmos esse método, devemos reconhecer que os requisitos podem evoluir ao longo do tempo e que a compreensão do problema pode ser aprimorada à medida que o projeto avança.

II – Afirmativa correta.

Justificativa: no método XP, são incentivadas a colaboração estreita entre clientes e desenvolvedores e a produção de documentações simplificadas. São projetadas apenas as necessidades imediatas, com projetos simples e de fácil implementação de código.

III – Afirmativa correta.

Justificativa: o TDD consiste na conversão de requisitos de software em testes, o que ocorre antes de o projeto ser concluído. No TDD, casos de teste são escritos antes da implementação do código. O código é desenvolvido para atender a esses testes, promovendo o desenvolvimento iterativo e a melhoria contínua do software.

Questão 2. (FGV 2024, adaptada) O Time de Soluções Inovadoras (TISI) de uma organização está utilizando práticas do Kanban no processo de desenvolvimento de soluções de software.

Com o uso do Kanban, o TISI visa

A) Utilizar uma abordagem de mudança evolutiva, baseada em feedbacks, para o processo de desenvolvimento de soluções de software.

B) Manter um workflow padrão para o processo de inovação.

C) Gerenciar e melhorar o desempenho dos indivíduos do time.

D) Estabelecer uma metodologia de desenvolvimento de soluções de software de modo iterativo.

E) Aplicar um processo para definição de incrementos para desenvolvimento de soluções de software.

Resposta correta: alternativa A.

Análise das alternativas

A) Alternativa correta.

Justificativa: Kanban é um método ágil de gestão visual de trabalho que ajuda a otimizar fluxos, reduzir desperdícios e melhorar continuamente seus processos. Nesse método, é utilizado um quadro Kanban, que é dividido em colunas que representam os estágios do fluxo de trabalho. Cada tarefa é indicada por um cartão que avança entre as colunas. A essência do Kanban inclui a mudança evolutiva (não disruptiva), a melhoria contínua baseada em feedback, a adaptação gradual do processo e o foco na otimização do fluxo de trabalho.

B) Alternativa incorreta.

Justificativa: o Kanban tem o objetivo de visualizar e melhorar o fluxo de trabalho, não necessariamente padronizá-lo.

C) Alternativa incorreta.

Justificativa: o Kanban foca no fluxo de trabalho do sistema como um todo, seu objetivo principal não é a avaliação individual.

D) Alternativa incorreta.

Justificativa: o Kanban não estabelece iterações fixas como no Scrum. O fluxo no Kanban é contínuo, não iterativo.

E) Alternativa incorreta.

Justificativa: o método Kanban não trabalha com incrementos predefinidos. A descrição do texto da alternativa aproxima-se mais do Scrum que do Kanban.

REFERÊNCIAS

ABNT. *NBR ISO 9241-11*: ergonomia da interação humano-sistema. Parte 11: usabilidade: definições e conceitos. 2. ed. Rio de Janeiro, 2021.

ABNT. *NBR ISO 9241-210*: ergonomia da interação humano-sistema. Parte 210: projeto centrado no ser humano para sistemas interativos. 2. ed. Rio de Janeiro, 2024.

ABNT. *NBR ISO/IEC 9241-11*: requisitos ergonômicos para trabalho de escritórios com computadores. Rio de Janeiro, 2002.

AMBLER, J. *et al.* Including digital sequence data in the Nagoya Protocol can promote data sharing. Trends in biotechnology, v. 39, n. 2, p. 116-125, 2021.

AMBYSOFT. *Effective practices for software solution delivery*. [s.d.]. Disponível em: https://shre.ink/xEWy. Acesso em: 23 jun. 2025.

5 APLICAÇÕES de realidade aumentada nos SIG. *INFO Portugal*, 30 out. 2020. Disponível em: https://shre.ink/egGU. Acesso em: 23 jun. 2025.

ASTAH. *Powering engineering excellence with Astah*. [s.d.]. Disponível em: https://shre.ink/egKD. Acesso em: 23 jun. 2025.

BASTOS, A. Os tipos de metodologias ágeis mais usados pelas empresas. *Alura para Empresas*, 3 maio 2023. Disponível em: https://shre.ink/ebdg. Acesso em: 23 jun. 2025.

BECK, K. *et al.* Manifesto para Desenvolvimento Ágil de Software. *Agile Manifesto*, 2001. Disponível em: https://shre.ink/ebdl. Acesso em: 23 jun. 2025.

BOOCH, G. *UML*: guia do usuário. São Paulo: Campus, 2002.

CAMARGO, R. Extreme Programming: quais principais regras e valores? *Robson Camargo Projetos e Negócios*, 3 out. 2019. Disponível em: https://shre.ink/ebWL. Acesso em: 23 jun. 2025.

CAPUTO, K.; HEFNER, R. *Strategies for transitioning from SW-CMM to CMMI*. Flórida: DELTA Business Solutions, 2004. Disponível em: https://shre.ink/ebs9. Acesso em: 23 jun. 2025.

CMMI. *CMMI(R) para desenvolvimento*: versão 1.2. Pittsburgh: Carnegie Mellon – Software Enginnering Institute, 2006.

COSTA, O. W. D. *JAD*: Joint Application Design. Rio de Janeiro: IBPI Press, 1994.

DELUCA, J. A importância da engenharia de software: FDD Feature-Driven Development. *Engenheiros do Software*, 7 out. 2008. Disponível em: https://shre.ink/ebBp. Acesso em: 23 jun. 2025.

DIJKSTRA, E. W. The humble programmer [1972 ACM Turing Award Lecture]. *Communications of the ACM*, v. 15, n. 10, p. 859-866, 1972.

DSDM: tudo que você precisa saber sobre essa metodologia ágil. *FlowUp*, 13 fev. 2020. Disponível em: https://shre.ink/ebpk. Acesso em: 23 jun. 2025.

ECLIPSE process framework project. *Eclipse Foundation*, [s.d.]. Disponível em: https://shre.ink/ebp8. Acesso em: 23 jun. 2025.

ENTENDA o que é a metodologia Crystal e saiba como ela pode ser aplicada no dia a dia das empresas. *Pontotel*, 5 out. 2023. Disponível em: https://shre.ink/ebDo. Acesso em: 23 jun. 2025.

ENTENDENDO o Feature Driven Development (FDD) no desenvolvimento ágil. *CTC*, 22 abr. 2024. Disponível em: https://shre.ink/ebBz. Acesso em: 23 jun. 2025.

ENTERPRISE Unified Process (EUP): strategies for enterprise Agile. *Ambsoft Inc*, [s.d.]. Disponível em: https://shre.ink/xEpP. Acesso em: 23 jun. 2025.

FOURNIER, R. *Desenvolvimento e manutenção de sistemas estruturados*. São Paulo: Makron Books, 1994.

FOWLER, M. *UML essencial*: um breve guia para a linguagem padrão de modelagem de objetos. 2. ed. Porto Alegre: Bookman, 2000.

GUEDES, G. T. A. *UML 2*: guia prático. São Paulo: Novatec Editora, 2007.

HAYES, W.; OVER, J. W. The personal software process (PSP): an empirical study of the impact of PSP on individual engineers. Pensilvânia (EUA): CMU/SEI, 1997. Disponível em: https://shre.ink/ebf3. Acesso em: 23 jun. 2025.

HUMPHREY, W. S. *A discipline for software engineering*. Massachusetts (EUA): Addison-Wesley, 1995.

HUMPHREY, W. S. *The Team Software Process (TSP)*. Pensilvânia (EUA): CMU/SEI, 2000.

IBM. *Rational software architect standard edition*. [s.d.]. Disponível em: https://shre.ink/egum. Acesso em: 23 jun. 2025.

ICONIX. *Sparx Systems*, [s.d.]. Disponível em: https://shre.ink/egDa. Acesso em: 23 jun. 2025.

ISO. *9241-11:2018*. Ergonomics of human-system interaction. Part 11: usability: definitions and concepts. Genebra (Suíça), 2021a. Disponível em: https://shre.ink/eblE. Acesso em: 23 jun. 2025.

ISO. *9241-20:2021*. Ergonomics of human-system interaction. Part 20: an ergonomic approach to accessibility within the ISO 9241 series. Genebra (Suíça), 2021b. Disponível em: https://shre.ink/eblf. Acesso em: 23 jun. 2025.

KRUCHTEN, P. *The rational unified process*: an introduction. 2. ed. Massachusetts (EUA): Addison-Wesley, 2000.

LARMAN, C. *Utilizando UML e padrões*: uma introdução à análise e ao projeto orientados a objetos e ao processo unificado. 2. ed. Porto Alegre: Bookman, 2007.

LAUDON, K. C.; LAUDON, J. P. *Sistemas de informação gerenciais*. 11. ed. São Paulo: Bookman, 2014.

LAUDON, K. C.; LAUDON, J. P. *Sistemas de informação gerencial*: administrando a empresa digital. 17. ed. São Paulo: Bookman, 2023.

MARTINS, J. V. *Desenvolvimento de protótipos de interfaces humano-computador para uma funcionalidade do Moodle para convergência digital*. 2011. Trabalho de Conclusão de Curso (Sistemas de Informação) – Universidade Federal de Santa Catarina, Santa Catarina, 2011.

A MISSÃO da Modelagem Ágil. Agile Modeling, 2020. Disponível em: https://shre.ink/xLBh. Acesso em: 4 jul. 2025.

MODELAGEM de processos de negócios com Bizagi. *Bizagi*, 2024. Disponível em: https://shre.ink/eghM. Acesso em: 23 jun. 2025.

NUNES, M. *Fundamental de UML*. 2. ed. Lisboa: Editora de Informática, 2010.

PAKISTAN INSTITUTE OF ENGINEERING AND APPLIED SCIENCES (PIEAS). *Software development methodologies*: computer science. Islamabad (Paquistão): Pieas, 2017. Disponível em: https://shre.ink/eb88. Acesso em: 23 jun. 2025.

PATEL, N. Teste de usabilidade: o que é e como fazer passo a passo. *NEILPATEL*, 2018. Disponível em: https://shre.ink/ebr2. Acesso em: 23 jun. 2025.

PAULA FILHO, W. P. *Engenharia de software*: fundamentos, métodos e padrões. 3. ed. Rio de Janeiro: LTC, 2015.

PAULK, M. C. *et al. The capability maturity model for software*: version 1.1. Pittsburgh (EUA): Software Engineering Institute Customer Relations, 1993.

PFLEEGER, S. L. *Engenharia de software*. 2. ed. São Paulo: Prentice Hall, 2004.

PRAXIS. *Visão geral*. 2023. Disponível em: https://shre.ink/egKn. Acesso em: 23 jun. 2025.

PRESSMAN, R. S. *Engenharia de software*. 5. ed. Rio de Janeiro: McGraw-Hill, 2002.

PRESSMAN, R. S. *Engenharia de software*. 6. ed. Rio de Janeiro: McGraw-Hill, 2007.

PRESSMAN, R. S. *Engenharia de software*: uma abordagem profissional. 7. ed. Rio de Janeiro: McGraw-Hill, 2011.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software*: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software*: uma abordagem profissional. 9. ed. Porto Alegre: AMGH, 2021.

PROJECT MANAGEMENT INSTITUTE (PMI). *PMBOK*: um guia do conhecimento em gerenciamento de projetos. 4. ed. Atlanta (EUA): PMI, 2010.

PROJECT MANAGEMENT INSTITUTE (PMI). *PMBOK*: guia do conhecimento em gerenciamento de projetos. 6. ed. Atlanta (EUA): PMI, 2017.

PROJECT MANAGEMENT INSTITUTE (PMI). *PMBOK*: guia do conhecimento em gerenciamento de projetos. 7. ed. Atlanta (EUA): PMI, 2021.

ROBASKI, J. E.; RAMOS, E. FDD (Feature Driven Development). *Medium*, 25 ago. 2014. Disponível em: https://shre.ink/xEeQ. Acesso em: 23 jun. 2025.

SOMMERVILLE, I. *Engenharia de software*. 8. ed. São Paulo: Pearson Prentice Hall, 2003.

SOMMERVILLE, I. *Engenharia de software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

SOMMERVILLE, I. *Software engineering*. 10. ed. Edimburgo (Escócia): Pearson Education Limited, 2016.

SOUZA, C. S. *et al. Projeto de interfaces de usuário*: perspectivas cognitivas e semióticas. Rio de Janeiro: PUC, 1999.

STAIR, R. M.; REYNOLDS, G. W. *Princípios de sistemas de informação*: uma abordagem gerencial. São Paulo: Pioneira Thomson Learning, 2006.

Informações:
www.sepi.unip.br ou 0800 010 9000