O processo de desenvolvimento de *software*

Primeiramente, é necessário compreender o que é processo e como se dá sua estrutura dentro de uma organização ou dentro de um projeto de desenvolvimento de *software*.

Processo

Processo é um conjunto de atividades relacionadas e voltadas ao alcance de resultados. A palavra “processo” é oriunda do latim “processu” e significa “o ato de proceder, de ir adiante, execução sequencial” e “maneira pela qual se realiza uma operação, tarefa ou rotina segundo determinadas normas ou técnicas”.

Em uma organização, existem vários processos. De acordo com a Norma ISO 9000:2000, processo é um conjunto de atividades inter-relacionadas ou interativas que transformam insumos (entradas) em produtos (saídas). Exemplos de processos: processo de manufatura, processo produtivo, processo de corte, processo de RH, processo de embarque, processo de editoração, processo comercial etc.

Os processos estão vinculados às atividades dos colaboradores da organização, conforme ilustrado na Figura 2.1, e são compostos de rotinas de trabalho (“instruções de trabalho”). Um sistema de folha de pagamento, um planejamento de controle e produção, tudo isso está relacionado a processos organizacionais. Em um processo, temos: ‘entrada’ de informações ou de matérias-primas; ‘processamento’ de cálculos ou ‘transformação’ das informações ou das matérias-primas; ‘saída’ de informações, de resultados ou de produtos acabados; ‘recursos’ de pessoal ou financeiro; e ‘regras e padrões’, como políticas, instruções de trabalho, normas, regulamentações etc.

A atividade diz respeito a um conjunto de tarefas que ocorrem dentro de um processo ou subprocesso. Já a tarefa é definida como toda atividade executada por uma pessoa dentro da organização em que trabalha. A tarefa pode ser considerada a menor unidade dentro da divisão do trabalho, isto é, representa a menor divisão do processo da organização. Além disso, ela pode chegar a uma divisão menor ainda, como rotinas,

procedimentos ou instruções de trabalho, representando tempos e movimentos. Esses movimentos são denominados de movimentos elementares ou *therbligs*.

1. Planejar
2. Escolher
3. Pegar
4. Transportar
5. Posicionar
6. Separar
7. Utilizar
8. Inspecionar

A análise e o estudo desses procedimentos, com relação ao tempo e aos seus movimentos, podem elevar a eficiência do operário (correta utilização dos recursos disponíveis).

Da mesma forma, dentro de um processo de produção de *software*, ocorre a aplicação de uma abordagem industrial enxuta. Segundo Santos *et al*. (2021, p. 65),

[...] a industrialização do processo de desenvolvimento de software foi inspirada na indústria automotiva que utiliza a metodologia de linha de produtos para a produção de veículos em larga escala. Com o passar dos anos, a engenharia de software observou as vantagens de aplicar esta metodologia, com suas devidas adaptações, poderia proporcionar a criação de novos produtos de software em curto período de tempo.

Outro conceito importante é o de macroprocesso, que é um conjunto de processos que podem ser representados por unidades e áreas funcionais. O macroprocesso também pode ser um conjunto de processos definidos por uma organização, como, por exemplo, o processo de vendas de um curso de tecnologia, o processo de vendas de caminhões, o processo de vendas pelo *site*, o processo de atendimento de chamados etc.

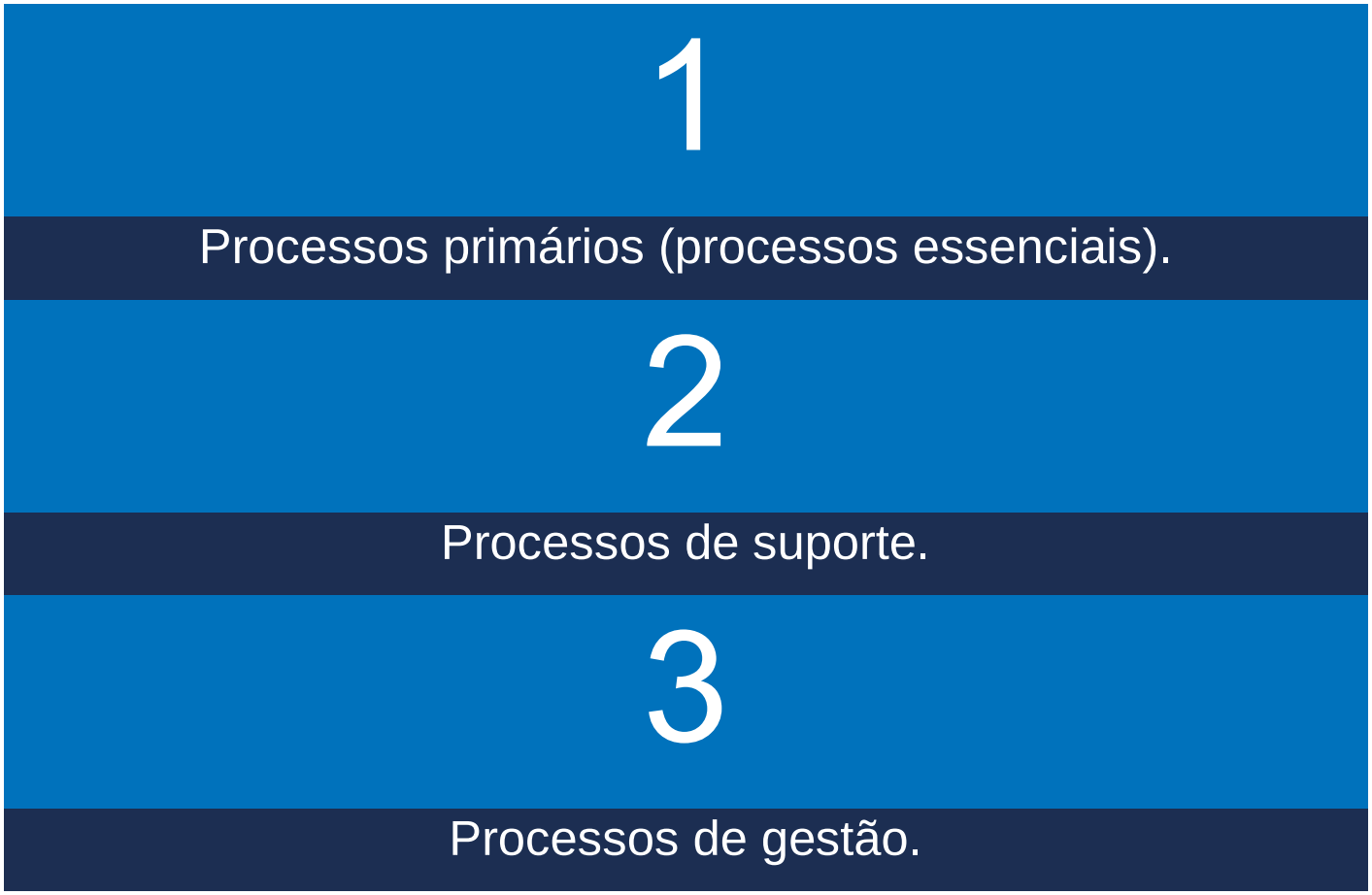
As representações das subdivisões, conforme a Figura 2.3, são um critério de detalhamento das atividades da empresa e dependem de um

controle maior dos processos. São, geralmente, realizadas por uma unidade (pessoa ou departamento) para produzir um resultado específico. Como comentado anteriormente, os projetos se utilizam de tarefas, portanto, de atividades, subprocessos e processos, de diversas áreas interconectadas em uma organização.

Portanto, nas organizações, os processos e macroprocessos são representações gráficas e prescritivas, representando um conjunto de tarefas e rotinas de suas áreas funcionais. A seguir, vejamos os tipos e atributos de um processo.

Tipos de processos

Há uma nomenclatura específica para os tipos de processos, e existem três tipos diferentes de processos de negócios.



Processos primários (processos essenciais): são processos ponta a ponta, multifuncionais, que direcionam entregas de valores aos clientes. Eles são chamados por alguns autores de “processos de negócios” ou “processos de núcleo”, pois representam processos ligados diretamente à cadeia de valor, *core business*, atividades essenciais ao negócio da organização. A constituição de cada etapa somada, medida pela sua contribuição, agrega valores para a criação ou entrega do produto.

Exemplos de processos de negócios da organização: vendas, financeiro, produção, logística, distribuição etc.

Processos de suporte: são representações estruturadas que apoiam os processos primários, por meio da gestão de recursos e/ou da infraestrutura solicitada pelos processos primários. Os processos de suporte orientam, planejam e fornecem os recursos para os processos de negócios. Alguns exemplos de processos de suporte são os serviços de tecnologia da informação e gestão de recursos humanos.

Processos de gestão: são processos para medir, monitorar e controlar atividades de negócios. Alguns autores chamam de “processos reguladores” ou de “processos de controle”. São conjuntos de políticas, regulamentações, regras e procedimentos que regem um processo de negócio ou de suporte. Esses processos têm como objetivo principal a garantia de que a organização opere com eficiência e eficácia.

Atributos de um processo

Um processo possui atributos que devem ser elencados no momento de sua identificação, como clientes, fornecedores, requisitos de entrada e saída, eventos de entrada, metas, métricas, indicadores, nível de maturidade, lista de riscos, documentações e interfaces de conformidade.

Exemplos de atributos.

a. Cliente: lista de clientes do processo.
b. Fornecedor: lista de fornecedores do processo.
c. Requisitos de entrada e saída: lista de todos os requisitos de entrada e saída.
d. Data do processo: data de identificação do processo atual.
e. Aprovação do processo: responsável pela aprovação do processo atual.
f. Metas: lista que identifica as metas do processo.
g. Métricas: métricas relacionadas ao processo.
h. Indicadores de desempenho: indicadores que medem o desempenho do processo.
i. Indicadores de resultado (qualidade): indicadores que medem o resultado ou a qualidade do processo.
j. Nível de maturidade: nível de capacidade do processo.

k. Lista de riscos: riscos relacionados diretamente ou indiretamente ao processo.
l. Documentos: lista de todos os documentos que fazem parte da documentação do processo.

Eventos de negócios

Também é importante compreender o que são os eventos de negócios. São estímulos internos ou externos que ocorrem nas organizações e que ativam os processos nas áreas de negócios, produzindo uma resposta. São exemplos de eventos de negócios externos.

Agora que compreendemos como é constituída a estrutura de um processo, vejamos alguns detalhes sobre o processo de *software*, uma estrutura de tarefas e rotinas que têm o objetivo de realizar a produção de um *software*.

Processos de *software*

Um processo de *software* é constituído de fases ou etapas lineares e/ou iterativas, e estas podem ser tradicionais e/ou ágeis. Essas fases (ou estágios) definem uma produção de *software*. Nesse sentido, Santos *et al.* (2021, p. 68) comentam que:

[...] uma linha de produção de software é definida como um conjunto de funcionalidades comuns e compartilhadas, com um sistema-base (core) contendo os principais módulos que podem ser específicos a um determinado segmento, com base nos requisitos do cliente.

As fases tradicionais de desenvolvimento do ciclo de vida do *software* são quatro, conforme ilustra a figura a seguir.

Especificação de *software* (requisitos);

Projeto e implementação (modelagem e codificação);

Validação do *software* (testes);

Evolução do *software* (manutenção).

As quatro fases (ou estágios) de um processo de *software* tradicional seriam: (1) análise e definição de requisitos ou especificações de *software*, etapa para o entendimento dos serviços requisitados pelo

sistema; (2) projeto e implementação, em que é realizada a conversão da especificação em um sistema executável, desde a modelagem (projeto) até a codificação do *software* (implementação); (3) validação do *software* por meio de testes, com a intenção de apresentar um *software* funcional e adequado às especificações do cliente; e (4) evolução do *software*, que se refere às manutenções para a sobrevivência do *software*.

Fase de especificações do software

O ciclo de vida do *software* inicia com a análise de requisitos. Essa fase também é chamada de especificação de *software* ou "engenharia de requisitos". O objetivo é especificar quais funcionalidades (ou serviços) o sistema proposto fornecerá.

A fase de especificações possui atividades do processo de engenharia de requisitos, tendo como resultado final a documentação de requisitos. As atividades são: (1) estudo de viabilidade; (2) elicitação e análise de requisitos; (3) especificação de requisitos; e (4) validação de requisitos.

Estudo de viabilidade

Primeira atividade do processo de engenharia de requisitos, serve para verificar se o sistema proposto é viável, a partir do ponto de vista orçamentário. Se for aplicado um conceito de gestão de projetos, essa etapa será analisada na fase de abertura e de planejamento do projeto, em que é realizado o levantamento das necessidades, compreendendo, ainda, o planejamento do escopo, o tempo e o custo do projeto, com base no orçamento proposto e selecionado pelo portfólio de projetos. Alguns autores também chamam essa etapa de "fluxo de trabalho de levantamento de necessidades", em que são identificadas as necessidades do cliente, além da verificação do prazo, da confiabilidade e do custo.

Elicitação e análise de requisitos

A segunda atividade do processo de engenharia de requisitos compreende a elicitação e a análise de requisitos. A atividade não é isolada, e algumas técnicas para a elicitação de requisitos são: *brainstorming*, entrevista, observação, análise de interfaces, prototipagem, *workshop*, pesquisa e questionário.

Especificação de requisitos

Atividade de tradução das informações coletadas durante a análise, em um documento que define um conjunto de requisitos. Há dois tipos de requisitos: (1) requisitos do usuário (funcionalidades desejadas); e (2) requisitos do sistema (funcionalidades atendidas). Nessa etapa, verifica-se se os requisitos atendem ao que o cliente deseja (requisitos do cliente).

Validação de requisitos

É um trabalho de garantia de qualidade na etapa de engenharia de requisitos, de modo a assegurar que todos os requisitos especificados estejam alinhados com os requisitos do negócio, ou seja, o objetivo é satisfazer todas as necessidades de negócio das partes interessadas, no escopo do projeto.

Portanto, a fase de especificações, como o próprio nome diz, especifica quais funcionalidades (requisitos de sistemas) ou serviços o sistema proposto fornecerá e se estão de acordo com os requisitos desejados pelo cliente.

Fase de projeto e implementação de desenvolvimento de software

Após a etapa de elicitação e análise de requisitos, é necessário utilizar ferramentas de modelagem para uma melhor compreensão dos requisitos voltados ao codificador de programa. Por exemplo, a utilização do diagrama “caso de uso” (UML – Linguagem de Modelagem Unificada) e das devidas validações, como mostra a Figura 2.8.

A fase de projeto de desenvolvimento de *software* é a ligação entre a codificação e seus requisitos, visando estabelecer uma arquitetura de *software*. A fase de projeto pode ser constituída de duas atividades principais: (1) projeto de arquitetura; e (2) projeto detalhado. Na atividade de projeto de arquitetura, são desenvolvidos os diagramas de implementação.

a. Projeto de arquitetura: define as relações entre os principais elementos estruturais de *software*, os estilos arquiteturais e os padrões de projeto, em que é possível identificar a estrutura geral do sistema e seus principais componentes. Abrange os elementos de *software*, suas propriedades (atributos) visíveis e seus relacionamentos.

b. Projeto detalhado: é o detalhamento do projeto de *software*, composto do:

(1) projeto de interface, em que são definidas as interfaces entre os componentes do sistema, e (2) projeto de componentes, blocos que compõem a arquitetura do software e se comunicam entre si com outros sistemas e entidades. Exemplos de componentes: executáveis, bibliotecas, tabelas, documentos etc. Já os tipos de componentes seriam: base de dados, ficheiro ou fonte de dados, documentos, biblioteca estática ou dinâmica e componente executável em um nó. A fase de detalhamento do projeto também é constituída de: (3) projeto de banco de dados, que projeta as estruturas de dados do sistema e como eles devem ser representados em um banco de dados. Um projeto de banco de dados envolve: (1) análise de requisitos, no caso, essa atividade é contemplada na fase de especificação do software; e (2) projeto lógico, composto por diagramas de modelo de dados conceitual que mostram os dados e seus relacionamentos, por meio do diagrama DER (diagrama "entidade relacionamento") ou UML (diagramas de classe).

Modelos de processos de desenvolvimento de software

Os processos de *software* são dirigidos a planos (modelos tradicionais) ou processos ágeis.

Na metodologia clássica, tem-se o modelo cascata ou sequencial, que aborda a forma linear de desenvolvimento; já no modelo incremental, tem-se a combinação da forma linear e paralela de desenvolvimento. Na metodologia ágil, tem-se o XP (Extreme Programming, “programação extrema”), em que se aplica um método de desenvolvimento de *software* com poucos artefatos, reduzindo, consideravelmente, a fase de projeto, com entregas rápidas das versões implementadas e aprovadas pelo cliente. Outra metodologia de grande importância para o estudo do processo ágil de desenvolvimento é o Scrum, utilizada para o gerenciamento ágil de projetos, sendo possível sua combinação com outras metodologias, como, por exemplo, a XP.

Metodologias clássicas

O estudo dos métodos clássicos (metodologias clássicas ou comuns) é resultado da necessidade de um processo de *software* com qualidade e da necessidade de resolução de problemas e automatizações de processos, dirigidos a planos (modelos tradicionais) que devem ser alinhados aos requisitos dos clientes.

Se quisermos que um computador resolva um problema ou automatize um processo, é necessário programar o computador para que ele entenda o que fazer. Mas antes de escrever um programa, nós precisamos organizar passos para resolver este problema de maneira que o computador possa executá-lo. Para auxiliar nesse processo, desenvolvemos algoritmos [...] (OKUYAMA; MILETTO; NICOLAO, 2014, p. 42).

Nesse sentido, do mesmo modo que os algoritmos ou pseudocódigos auxiliam no passo a passo da codificação do problema, as linguagens de modelagem auxiliam na representação dos requisitos funcionais para a codificação do *software*. Esse conjunto de processos cadenciados, com objetivos específicos, formam um modelo ou método de desenvolvimento de *software*, que pode ser de abordagem clássica ou ágil.

Modelos comuns

Dentre os modelos comuns, isto é, clássicos, o modelo cascata compreende atividades iterativas, passando por várias versões, mas não incrementais, ou seja, constituído parte por parte.

O modelo cascata, também denominado Waterfall Development Model, como ilustra a Figura 2.9, considera as atividades fundamentais do processo de especificação, codificação, validação e evolução, sequencial e sistemático, e representa cada uma das etapas distintas, por exemplo: especificação de requisitos, projeto de *software*, implementação, testes, e assim por diante.

Observando a Figura 2.9, no primeiro nível da cascata, temos a etapa de '**comunicação**', que se refere ao início do projeto e ao levantamento de requisitos. No segundo nível da cascata, temos a etapa de '**planejamento**', que se refere às tarefas de estimativas, cronograma e acompanhamento. No terceiro nível, temos a etapa de '**modelagem**', que se refere às tarefas de análise e projeto. No quarto nível, temos a '**construção**', referente ao desenvolvimento do código e aos testes. No último nível da cascata, '**entrega**', temos as tarefas de entrega, suporte e feedback.

Um ponto crucial relacionado ao modelo cascata é que nenhuma das etapas é finalizada até que toda a documentação tenha sido terminada e os entregáveis aprovados por um grupo que garanta a qualidade do software.

Iterativos

Na abordagem iterativa de desenvolvimento de *software*, as especificações do projeto são elaboradas em cada iteração do incremento (1..n), ou seja, a definição dos requisitos funcionais e não funcionais do sistema (fluxo de levantamento de necessidades da engenharia de requisitos) é feita a cada início de iteração (etapa de comunicação e planejamento) do incremento (1,2..) até o último. O uso desse modelo deve ser evitado para sistemas críticos e que envolvem equipes de trabalhos em locais diferentes, pois é característica do paradigma incremental a iteração entre os membros da equipe de desenvolvimento de *software*.

Iterativos e incrementais

O modelo incremental faz uso de um modelo de vida iterativo e incremental, ou seja, é realizada a combinação dos fluxos de processo linear (modelo cascata) e paralelo dos elementos. A Figura 2.10 mostra que o modelo incremental aplica sequências lineares de forma escalonada, à medida que o tempo vai avançando. Cada sequência linear produz “incrementos” entregáveis de software. O eixo vertical do gráfico apresenta as funcionalidades e os recursos do *software*, e o eixo horizontal apresenta o cronograma do projeto. Cada incremento é composto pelo modelo cascata e seus elementos (comunicação, planejamento, modelagem, construção e disponibilização).

Metodologias ágeis

A engenharia de *software* e as metodologias de desenvolvimento de *software* tiveram inspiração em processos de manufatura para a consolidação de seus métodos de trabalho. Essa percepção era aplicada no desenvolvimento de sistemas corporativos de grande porte, por meio de uma abordagem pesada e de desenvolvimento dirigida a planos, com grandes equipes, dispersas geograficamente, além de longos períodos.

Hoje, as empresas trabalham em ambientes de mudanças rápidas, globais, interativas, inovadoras e de valorização dos funcionários e colaboradores, ou seja, operam por meio de “um pensamento ágil”. A figura ilustra um processo mais ágil através da automação por máquinas e robôs, reduzindo, consideravelmente, os recursos e aumentando a produtividade. Diante disso, as metodologias de desenvolvimento de *software* também evoluíram para um pensamento ágil.

F

A metodologia ágil de *software* é definida como a realização de atividades de levantamento de requisitos, planejamento, projeto, implementação e testes, de fabricação rápida, por meio de uma série de incrementos de funcionalidades, interações de curtos espaços de tempo e de entregas aprovadas pelo cliente. Vejamos, a seguir, o que seria esse pensamento ou conceito ágil dentro de um processo de *software*.

Métodos ágeis

Nos métodos ágeis, o *software* não é desenvolvido como uma única unidade, mas como uma série de incrementos, e cada incremento inclui uma nova funcionalidade do sistema. Os métodos ágeis de desenvolvimento de *software* utilizam uma sistematização mais rápida e objetiva para produzir um software.

Existem diversas metodologias de desenvolvimento de *software* ágil, como o *Extreme Programming* (XP), de programação extrema. O processo ágil de desenvolvimento de *software* se caracteriza pela velocidade de fabricação de um *software* útil, reduzindo a burocracia, por meio de poucos artefatos (documentos).

Com iterações de execução de atividades em curtos espaços de tempo (no máximo, de um mês), é possível realizar alterações junto ao cliente, reduzindo os riscos do projeto. Outra característica do processo de desenvolvimento ágil é o foco nas prioridades e na utilização de protótipos e modelos ágeis, tornando-se, assim, um projeto “leve”.

O Scrum é uma metodologia de gerenciamento de *software* rápido e pode ser combinada a metodologias de desenvolvimento de *software*. Possui uma abordagem interativa, em que a análise inicia assim que alguns requisitos estiverem disponíveis. Após isso, começam as atividades de *design* e codificação, trabalhando com pequenas partes de cada vez, sendo estas testadas, aprovadas e entregues ao cliente.

Hoje, são muitas as metodologias de desenvolvimento e de gerenciamento de *software*, tanto as tradicionais como as ágeis, que podem ser combinadas para uma melhor eficiência na produção (redução de riscos, maior qualidade e menor custo). A metodologia ágil tem um enfoque mais relacionado às pessoas, e a metodologia tradicional é mais relacionada aos processos. No entanto, existem algumas abordagens tradicionais de gerenciamento de projetos, como o PMI, que dão ênfase às pessoas, chegando, inclusive, a inserir em seus processos de gerenciamento um novo elemento, chamado *stakeholders*.

As metodologias tradicionais têm uma abordagem preditiva em vez de adaptativa, característica dos modelos ágeis. A ideia da preditiva é prever, por meio de planos, possíveis riscos, mudanças de curso no escopo, custos e tempo de projeto. No paradigma ágil, tem-se que as mudanças vão ocorrer de qualquer maneira, e que o desenvolvimento sofrerá as consequências. As metodologias ágeis apostam que é melhor fazer algo simples, sem previsões imediatas, e pagar um pouco mais

para realizar futuras modificações. Na questão de geração de artefatos em projetos que utilizam o paradigma clássico, é possível produzir uma quantidade grande de documentações necessárias à previsão e ao direcionamento do projeto, mas não de forma obrigatória, pois tudo depende da segurança do líder do projeto e do conhecimento que ele tem com relação a projetos similares.

No paradigma ágil, as entregas são incrementais, com pouca ou quase nenhuma documentação. Assim, aposta-se nas entregas versionadas das funcionalidades priorizadas e aprovadas pelo cliente. Outra questão é sobre a impessoalidade em projetos de paradigmas ágeis, muito questionada por diversos autores. Na metodologia ágil, a colaboração do cliente é essencial e requer maturidade, pois a falta de documentos legais pode causar algumas dores de cabeça com relação ao escopo, ao tempo, ao custo e aos entregáveis do projeto.

Design de *software*: UI e UX

Em um ciclo de vida de *software*, seja ele tradicional ou ágil, é necessário realizar a análise e a avaliação da usabilidade e da experiência do usuário, o chamado contexto UX, e verificar se as interfaces entre os componentes estão adequadas em termos de interação entre o usuário e o sistema. Essas verificações devem ser realizadas de acordo com as solicitações acordadas no início do projeto.

É por meio do projeto de interface que são definidas as interfaces entre os componentes do sistema. Dessa forma, o projeto de interface apresenta uma comunicação com sistemas que operam de forma conjunta e com os usuários que o utilizam. Uma interface define um fluxo de dados e/ou de controle, por meio de um tipo de comportamento específico.

No projeto de interface, determinados modelos comportamentais e de cenários de uso, como os diagramas de sequência da UML (Linguagem de Modelagem Unificada), também são utilizados. A Figura 2.12 ilustra um ciclo de desenvolvimento de uma aplicação web. Observe que é no momento da implementação que o projeto de interface é aplicado. Nesse momento, os três elementos mais importantes são:

1-A interface do usuário (UI, *user interface*), incorporando elementos estéticos ou ergonômicos, como *layout*, cor, imagem, navegação etc.

2-Interfaces externas para outros sistemas, dispositivos, redes ou produtores ou consumidores de informação; e

3-Interfaces internas entre os vários componentes do projeto.

Deve ser considerado, também, o contexto das experiências do usuário (UX), como usabilidade e acessibilidade aplicada à interface.

A usabilidade é um dos principais conceitos que devem ser utilizados no desenvolvimento de uma interface, tendo em vista as questões de interatividade. A usabilidade é uma propriedade de aceitabilidade das interfaces, trazendo conforto com relação às atividades propostas pelo sistema. Da mesma forma, está relacionada aos critérios de facilidade de aprendizado (uso) e de lembrança das funcionalidades do sistema. Isso significa que, para a experiência do usuário, ele deve aprender sem dificuldades, memorizando facilmente determinadas questões.

Nas etapas de levantamento e de análise de requisitos e de pós-implementação do *software*, as tarefas de análise e de avaliação das experiências do usuário podem ser aplicadas por meio de métodos e técnicas de mensuração, visando medir a satisfação do usuário de duas formas:

° Como expectativa das aplicações

° Resultado da aplicação do software

Contudo existem formas eficientes para detectar os requisitos do cliente, ou seja, o que o cliente realmente deseja. É por meio de métodos e ferramentas de análise e de avaliação UI e UX que é possível conhecer as expectativas da aplicação e os resultados da implementação do *software*, como infográficos, protótipos, *feedbacks*, iconografia e sinalização.

Design patterns

Para desenvolver *softwares* de forma mais rápida, otimizada e com qualidade, é necessário compreender como funcionam os processos de desenvolvimento e suas metodologias (tradicionais e ágeis), de modo a combiná-las por meio de técnicas de refatoração de código e de

padronização de projetos. Assim, vamos estudar alguns conceitos sobre as técnicas de refatoração e de padronização de projetos.

Essas metodologias cadenciam os processos do ciclo de vida de desenvolvimento, regendo os recursos, as regras e os padrões, direcionando os requisitos necessários à implementação e, assim, descomplicando a tarefa de produzir o *software*. Estamos em uma época na qual desenvolver sistemas não é mais sinônimo de complexidade e de alto custo, já que as metodologias ágeis e sua combinação com as boas práticas de gerenciamento de projetos, se bem utilizadas e compreendidas, promovem, em qualquer tipo de ambiente, a elevação da produtividade no desenvolvimento. Cada vez mais, novas metodologias e suas combinações estão sendo estudadas e melhoradas, trazendo reais benefícios e melhorias às equipes de desenvolvimento em relação aos sistemas funcionais e aplicativos, especialmente para clientes e usuários.

Refatoração

A refatoração é uma tarefa (ou conjunto de tarefas) complementar ao processo de desenvolvimento de *software* e é aplicada, especificamente, na etapa de codificação e implementação. A refatoração tem como objetivos a simplificação e a otimização do código, obtendo melhorias significativas no processo de codificação (Figura 2.13) e, consequentemente, no projeto de desenvolvimento do *software* com entregável e no processo de construção. Santos *et al.* (2021, p. 25) comentam que:

O desenvolvimento orientado ao reúso de software é uma estratégia de produção baseada na reutilização do todo ou de partes de um software existente. Durante as últimas décadas, diversas técnicas e metodologias foram aprimoradas com o intuito de obter vantagens dessa proposta, como reduções dos prazos de produção, melhorias na qualidade do produto e simplificações em futuras manutenções. A diversidade de abordagens se dá ao fato de o reúso se aplicar a diversos momentos do ciclo de vida de produção e, para cada uma dessas etapas, as necessidades são diferentes. O reúso vai desde a utilização de componentes individuais até à reutilização de um sistema completo […].

A técnica de refatoração realiza alterações, modificações e simplificações na estrutura interna, mas seu comportamento externo permanece inalterado. É muito utilizada e combinada com as metodologias ágeis de *software*, especificamente a XP, que estimula, consideravelmente, a refatoração. Contudo, a refatoração pode causar determinados efeitos

colaterais, sendo necessário utilizar ferramentas e testes adequados para verificar se o comportamento externo não foi alterado.

A refatoração é uma técnica de reorganização que simplifica o projeto (ou código) de um componente, sem modificar a função ou o comportamento. Assim, é o conjunto de atividades que realizam mudanças em um sistema de *software*, mas sem alterar o comportamento externo do código ou projeto, melhorando sua estrutura interna. Uma forma de refatorar é dividir, sistematicamente, grandes métodos existentes em métodos menores, ou seja, um código mais fácil de entender, reutilizando e evitando duplicar a lógica em outras áreas do sistema.

Ainda, a refatoração é uma transformação que preserva o comportamento, podendo ser considerada uma alteração na estrutura interna ou uma codificação do *software* para uma melhor compreensão. A refatoração é, também, um conjunto de tarefas que envolve a eliminação de duplicação, a simplificação da lógica condicional e o melhor entendimento do código que porventura não esteja claro.

Nesse sentido, é um processo de otimização que deve ser realizado com segurança, a partir de testes manuais e automatizados. A premissa da técnica de refatoração é realizar o processo em pequenas etapas ou passos, o que auxilia na prevenção da inserção de defeitos no código.

Padrões de projetos

A padronização de projetos, dentro de um processo de *software*, envolve a modelagem, a codificação e os testes, com objetivo de obter uma melhor produtividade, eficiência e qualidade do *software* nessas etapas. Os padrões de projetos de software são aqueles que descrevem um problema em nosso ambiente, com foco na solução, em que é possível realizar diversas iterações com a solução.

Esse padrão de projetos é imperativo em objetos e interfaces aderentes aos usuários; nesse caso, os projetos são orientados a objetos. Um padrão de projeto é constituído de quatro elementos: (1) o nome do padrão; (2) o problema; (3) a solução; e (4) as consequências. A nomenclatura do padrão diz respeito a uma referência que podemos utilizar para descrever um problema de projeto. Já o elemento do problema descreve em que situação será aplicado o padrão. A solução descreve os elementos que compõem o padrão de projeto e seus relacionamentos, bem como suas responsabilidades e colaborações. Por

fim, as consequências são os resultados e as análises das vantagens e desvantagens da aplicação do padrão.

Assim, a combinação da refatoração com a padronização de projetos, apoia o especialista na remoção de duplicação e na simplificação do código, fazendo com que o código comunique seu objetivo. Além dos padrões de projeto, é necessário e importante o uso de uma norma de padronização, como, por exemplo, a norma ISO/IEC 9126.

A ISO 9126 é uma norma que estabelece diretrizes para um software. Uma norma de padronização dentro de um projeto de software [...] ela traz uma espécie de padrão, um modelo a ser seguido, que contém seis características para a qualidade do software: funcionalidade, confiabilidade, usabilidade (BARRETO et al., 2018, p. 61).

Os padrões de projetos podem ser classificados conforme a estruturação dos objetos e classes, e conforme sua interação em uma aplicação de *software*. Eles são classificados em:

1

padrões de criação: ajudam na independência do sistema e em como os objetos são criados, compostos e representados;

2

padrões de estruturas: definem um padrão de *design* de objetos e classes (simplificação e relacionamento entre objetos e classes); e

3

padrões de comportamento: responsável pela padronização da interação entre os objetos e suas responsabilidades.

A padronização de projetos é voltada à execução de uma tarefa de reorganização e refatoração, tendendo a uma redução de volume e a uma otimização da modelagem e codificação, já que é possível padronizar os objetos desde sua criação no modelo até sua codificação, com boas práticas de programação. A *Unified Modeling Language* (UML), além de expressar um projeto orientado a objeto, pode se apoiar na modelagem de um padrão de projeto.

Considerações Finais

Ao longo da unidade, vimos que a tendência das fábricas de *software* é a utilização híbrida de metodologias, métodos e técnicas, na busca por qualidade e velocidade para a entrega do *software*. Nesse sentido, vimos que os processos de *software* são dirigidos a planos tradicionais ou processos ágeis, os quais podem ser combinados entre si ou com técnicas específicas de otimização e padronização, como a refatoração de código e a padronização de projetos.

Contudo, é importante estar atento, pois todo esse arsenal de ferramentas deve ser utilizado de acordo com os tipos de projetos (simples ou complexos), o levantamento e análise dos requisitos do *software*, o tamanho da equipe de projeto, a maneira de realizar os entregáveis do projeto, a redução dos riscos envolvidos, a modelagem do sistema, os *feedbacks* do cliente e a maturidade dos envolvidos no projeto, visando extrair o melhor dos métodos e técnicas disponíveis na engenharia de *software*.