A Engenharia de Software é a base para o desenvolvimento de aplicações, tornando seu domínio essencial para qualquer profissional de TI. Compreender como o software é concebido, desenvolvido e gerenciado é fundamental para se destacar no mercado.

Essa disciplina abrange desde fundamentos até gerenciamento de configurações, capacitando o estudante a projetar software com qualidade. Sua aplicação é ampla, indo do desenvolvimento de aplicativos a sistemas críticos, o que aumenta seu valor na formação acadêmica.

No mercado, cada conhecimento adquirido—como modelos de processos de desenvolvimento e gerenciamento de software—é uma peça essencial para a carreira. Empresas buscam profissionais que dominam esses princípios, tornando essa expertise um diferencial competitivo.

O software é essencial na vida moderna, permitindo a usabilidade intuitiva de dispositivos como smartphones. Seu desenvolvimento envolve engenheiros de software, administradores de banco de dados e web designers.

A Engenharia de Software estuda esse produto, que consiste em: (1) instruções executáveis que fornecem funcionalidades, (2) estruturas de dados para manipulação de informações e (3) documentação descritiva, impressa ou digital.

A relação entre software e hardware levou à "Crise do Software", conforme Dijkstra (1972): com o aumento da capacidade dos computadores, programar tornou-se um desafio cada vez maior. A bibliografia de Pressman (2016) é uma referência fundamental na área.

O avanço do hardware permitiu o desenvolvimento de softwares mais complexos, impactando a indústria e possibilitando a adoção da programação orientada a objetos, que melhora a reutilização, manutenção e escalabilidade do software.

O software tem papel central na geração de informação dentro dos Sistemas de Informação, que incluem hardware, banco de dados, redes e serviços. Ele agrega valor aos dados, auxiliando na tomada de decisões em diferentes níveis de gestão, como financeiro e recursos humanos.

A complexidade do software embarcado é exemplificada em aeronaves, onde um defeito pode ter consequências graves. O software atua tanto como produto utilizado pelos usuários quanto como meio para distribuir informações, essenciais para diversos sistemas.

Os desafios dos engenheiros de software envolvem sete categorias principais:

1. **Software de sistema** – Inclui sistemas operacionais e drivers.
2. **Software de aplicação** – Voltado para necessidades específicas, como ERPs.
3. **Software científico/engenharia** – Usado em cálculos estruturais e processamento de imagem.
4. **Software embarcado** – Controla dispositivos, como painéis digitais de veículos.
5. **Software para linha de produtos** – Desenvolvido para múltiplos clientes, como sistemas emissores de nota fiscal.
6. **Aplicações web e móveis** – Projetadas para dispositivos móveis.
7. **Software de inteligência artificial** – Utiliza redes neurais, aprendizado de máquina e sistemas especialistas.

A complexidade do software, como em sistemas embarcados ou controle de tráfego aéreo, exige metodologias que decomponham problemas de forma sistemática, tornando a Engenharia de Software essencial. Quanto maior a complexidade do produto, mais a engenharia se faz necessária.

Assim como a construção de um edifício inteligente demanda profissionais de diversas áreas, o desenvolvimento de software exige uma abordagem sistemática para ser viável. Apesar de intangível e frequentemente associado apenas ao código, o software possui alta volatilidade devido à constante evolução tecnológica e mudanças nos requisitos, aumentando sua complexidade.

O termo "Engenharia de Software" surgiu em 1968 em uma conferência da NATO. O SWEBOK Guide V3.0 define a Engenharia de Software como a aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de software.

Engenharia: aplicação de métodos científicos ou empíricos à utilização dos recursos da natureza em benefício do ser humano.

A Engenharia de Software é estruturada em quatro camadas:

- **Qualidade**: Garante que os requisitos atendam às expectativas do usuário.
- **Processo**: Define as etapas do desenvolvimento.
- **Métodos**: Especifica técnicas, como elicitação de requisitos e modelagem.
- **Ferramentas**: Utiliza tecnologia CASE para criação de artefatos e geração de código.

Assim como na construção de uma casa, um software exige um processo bem definido. A **camada de processo** é a base da Engenharia de Software, guiando o desenvolvimento conforme as melhores práticas.

O **processo de software** é uma sequência de etapas que guiam o desenvolvimento, sendo essencial para lidar com a complexidade e integrar profissionais como analistas, programadores e gerentes. A escolha do processo depende da complexidade do sistema, e **quanto maior a complexidade, mais formal deve ser o processo**.

A **metodologia de processo genérica** inclui cinco atividades principais:

- **Comunicação**: Interação com usuários para entender o problema e definir requisitos.
- **Planejamento**: Elaboração do Plano de Gerenciamento do Projeto, incluindo cronograma e monitoramento.
- **Modelagem**: Criação de modelos gráficos e textuais, como diagramas de casos de uso.
- **Construção**: Implementação do software, codificação e testes conforme os modelos.
- **Entrega**: Disponibilização do produto final conforme o planejamento.

A **qualidade** é a base que sustenta o processo, garantindo a eficácia do desenvolvimento.

As **atividades básicas** do desenvolvimento de software são complementadas por **atividades de apoio**, essenciais para controle, qualidade e eficiência.

Atividades de apoio:

- **Controle e acompanhamento de projeto**: Monitoramento e correção de desvios conforme planejado.
- **Administração de riscos**: Gerenciamento de eventos que possam impactar o projeto.
- **Garantia da qualidade**: Verificação do cumprimento dos requisitos.
- **Revisões técnicas**: Testes em artefatos como processos, modelos e código.
- **Medição**: Definição de métricas para avaliação do desenvolvimento.
- **Gerenciamento da configuração**: Controle de versões e propagação de correções.
- **Gerenciamento da reutilização**: Otimização do reuso de software, aprimorado pelo paradigma orientado a objetos.
- **Preparo e produção de artefatos**: Documentação dos artefatos gerados conforme a complexidade do projeto e os envolvidos.

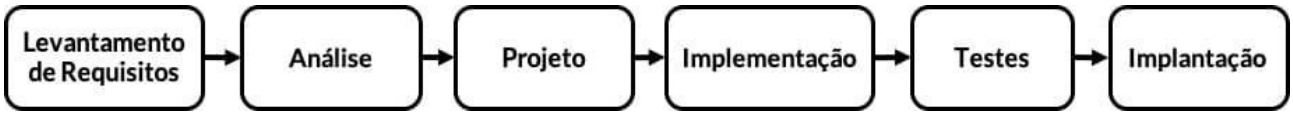
O engenheiro de software deve documentar todas as etapas do processo para garantir a organização e eficiência no desenvolvimento.

O **conceito de abstração** é essencial no desenvolvimento de software, pois o processo começa com especificações abstratas e vai se tornando mais detalhado até chegar ao código.

O engenheiro de software deve definir o **processo de desenvolvimento** a ser utilizado, considerando:

1. **Atividades necessárias** para o projeto.
2. **Sequenciamento** e fluxo dessas atividades.

A figura 4 apresenta as **atividades típicas** de um processo de desenvolvimento, que são fundamentais em qualquer abordagem.



A **Elicitação de Requisitos** é a primeira etapa do desenvolvimento de software, onde o engenheiro deve compreender o problema do cliente. Isso exige comunicação com diferentes usuários e o uso de técnicas como entrevistas, questionários, análise de documentos e etnografia.

Etnografia: mergulhar na vida cotidiana de um grupo social para compreender as suas interações e relações

Tipos de Requisitos:

- **Funcionais**: Serviços oferecidos pelo sistema, como geração de notas fiscais.

- **Não Funcionais**: Restrições operacionais, como banco de dados, linguagem de programação e requisitos de qualidade (usabilidade, confiabilidade, etc.).
- **De Domínio**: Regras de negócio que restringem funcionalidades, como cálculos de médias escolares ou impostos.

A correta identificação dos requisitos é essencial para o sucesso do software.

Sommerville (2007) classifica os requisitos em **requisitos de usuários** (alto nível de abstração) e **requisitos de sistema** (baixo nível de abstração). Os requisitos de sistema são divididos em **funcionais**, **não funcionais** e **de domínio**.

O entendimento inadequado dos requisitos impacta negativamente nas etapas seguintes do desenvolvimento, comprometendo o produto final. A **especificação de requisitos** formaliza um contrato entre o engenheiro e os usuários, definindo o software a ser entregue e permitindo feedbacks sobre falhas na especificação.

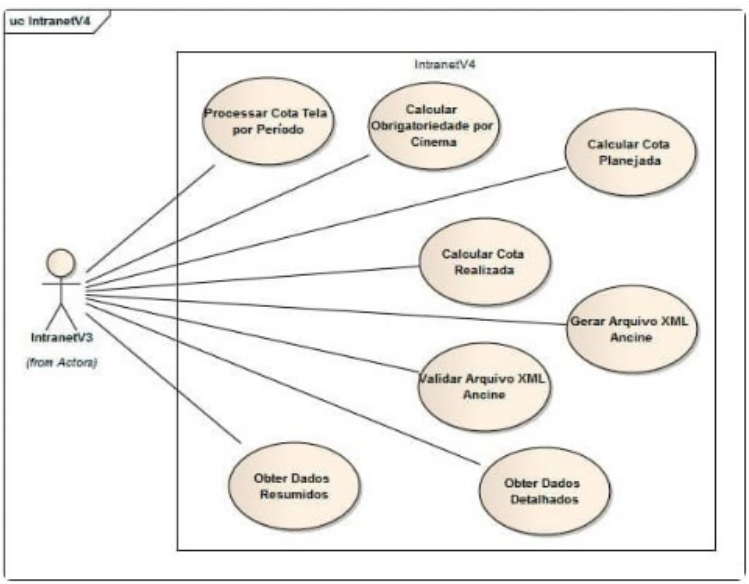
Nesta fase, é gerado o **Documento de Requisitos**, determinando o **escopo do projeto**. A rastreabilidade dos requisitos assegura que as especificações se alinhem com os produtos gerados em etapas subsequentes, como modelos e codificação.

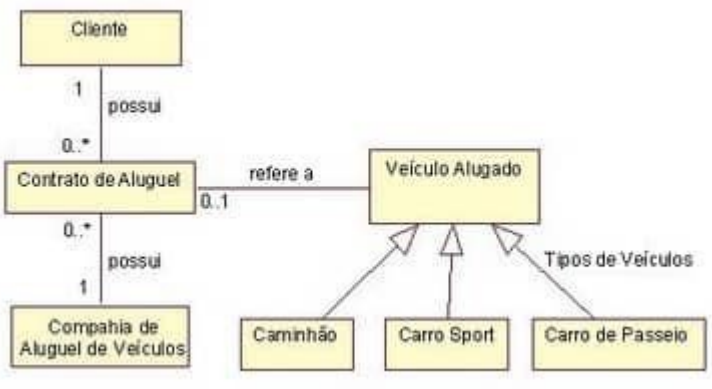
A **etapa de Análise** começa com a conversão das especificações do **Documento de Requisitos** em **modelos de análise**, que incluem artefatos gráficos e textuais. O **Modelo de Casos de Uso**, por exemplo, é composto por diagramas e descrições que representam os requisitos funcionais do sistema. Nessa etapa, o engenheiro de software aplica **alto nível de abstração**, definindo **O QUE** o sistema deverá implementar, sem considerar ainda questões tecnológicas.

Os modelos gerados são usados para comunicar a solução entre os diferentes stakeholders, como analistas, usuários e gerentes. O uso de modelos reduz custos, pois permite detectar problemas mais cedo, evitando que defeitos se propaguem nas fases posteriores, como a codificação. Além disso, esses modelos garantem a **economicidade do projeto**, pois correções em um modelo gráfico (como o modelo de classes) são mais baratas e fáceis de realizar que mudanças no código de produção.

Os **modelos de análise** incluem a representação de requisitos funcionais e a definição de **cenários de utilização** dos casos de uso, detalhados nas descrições. A entrega dessa etapa é o **modelo de análise**, que é **validado pelos usuários** para garantir que o entendimento do problema esteja correto e alinhado às expectativas do cliente.

O engenheiro de software também realiza uma **verificação técnica** dos modelos para garantir sua **correção e balanceamento**. Por exemplo, no **modelo de classes**, ele verifica se os objetos do domínio do problema estão corretamente definidos e se a representação está consistente com o **Modelo de Casos de Uso**. A verificação assegura que os modelos estão completos e funcionais antes de avançar para a próxima etapa.





Etapa de Projeto: Refinamento dos modelos da análise e criação de novos modelos com menor abstração, definindo **COMO** implementar a solução. As principais atividades incluem o refinamento do **modelo de classes**, a construção do **modelo de interação** para definir a comunicação entre objetos, o **mapeamento objeto-relacional** para gerar o modelo lógico do banco de dados, e o **desenho dos componentes e nós computacionais** necessários para a arquitetura do sistema.

Etapa de Implementação: Codificação do software conforme os requisitos do projeto. Devem ser seguidos **padrões de projeto** que representam boas práticas de implementação.

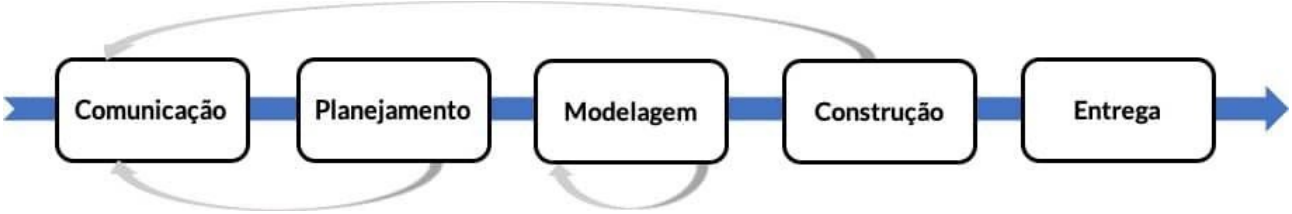
Etapa de Testes: Realização de testes de validação para verificar se o produto está correto. Começa com testes unitários, seguidos de testes de integração e aceitação, até a migração para o ambiente de produção. A etapa deve contar com um **Plano de Teste**, **casos de testes** vinculados aos cenários dos casos de uso, e **automação de testes** para ciclos iterativos.

Etapa de Implantação: Migração do software para o ambiente de produção, com validação da equipe de qualidade. Pode envolver a criação de **manuais**, **migração de dados**, **treinamento de usuários** ou implantação de **call center** em sistemas complexos.

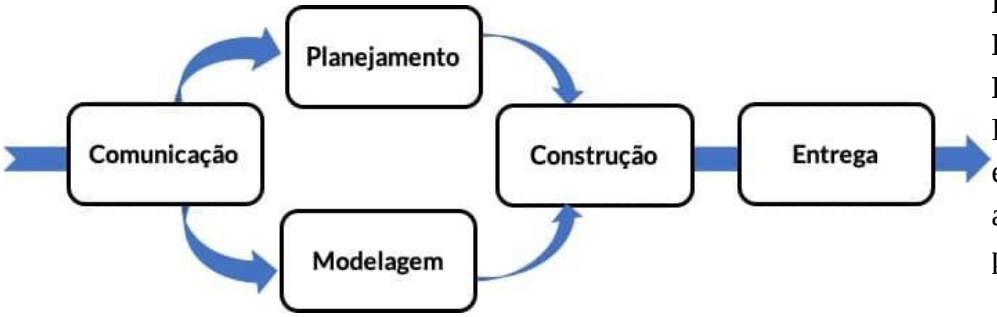
Fluxo de Processo: A especificação do processo de desenvolvimento de software, como requisito não funcional, define as atividades e como elas serão encadeadas (Fluxo de Processo ou Ciclo de Vida).

Fluxo de Processo Linear: No Fluxo de Processo Linear, as atividades são realizadas em sequência, sendo cada uma completada uma única vez.

Fluxo de Processo Iterativo: No Fluxo de Processo Iterativo, uma atividade ou conjunto de atividades pode ser repetido antes de prosseguir para a próxima.

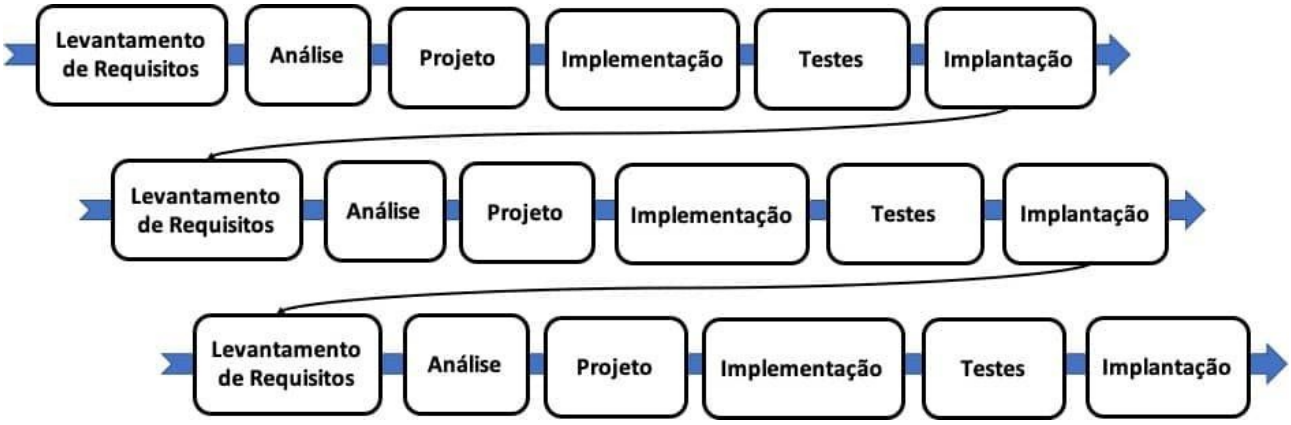


Fluxo de Processo Evolucionário: Cada iteração gera uma nova versão do software, agregando funcionalidades a cada ciclo.



Fluxo de Processo Paralelo: Permite a execução de atividades em paralelo.

Modelo de Ciclo de Vida Iterativo e Incremental: Cada iteração inclui todas as atividades e gera uma nova versão do produto, considerando um subconjunto de requisitos a cada ciclo.



O processo de desenvolvimento de software é a camada principal da Engenharia de Software. Ele inclui atividades como levantamento de requisitos, análise, projeto, implementação, testes e implantação. O engenheiro de software define as atividades e seu sequenciamento, que pode ser linear, iterativo, evolucionário ou paralelo. Sem processo, não há engenharia.

O gerenciamento de projetos aplica metodologias sistemáticas para atender às necessidades dos usuários e gerar o produto desejado. O PMI (Project Management Institute) é uma organização que dissemina as melhores práticas de gerenciamento de projetos, incluindo a certificação PMP® para gerentes, importante para a indústria de software. O Guia PMBOK, publicado pelo PMI, fornece uma base para desenvolver metodologias de projetos.

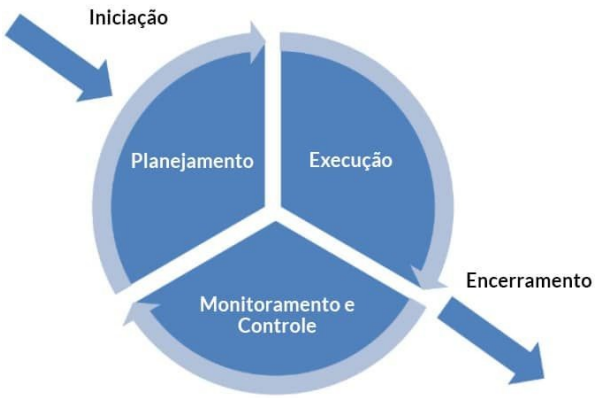
Um projeto é um esforço temporário para criar um produto, serviço ou resultado único. O gerenciamento de projetos envolve aplicar conhecimentos, habilidades, ferramentas e técnicas para cumprir os requisitos do projeto, garantindo eficiência e eficácia nas organizações.

O ciclo de vida do projeto inclui fases genéricas presentes em todos os projetos, relacionadas ao fluxo de processo do software.

Início do Projeto	Organização e Preparação	Execução do trabalho	Término do Projeto

A escolha do fluxo de processo (linear, iterativo, evolucionário, paralelo) define as fases do projeto, desde o início até o fim. Por exemplo, ao selecionar o Fluxo de Processo Evolucionário ou Iterativo e Incremental, o ciclo de vida do projeto é definido, e as entregas ocorrem por iterações.

Esse ciclo é gerenciado por processos de gerenciamento de projetos, que transformam entradas em saídas, utilizando técnicas e ferramentas apropriadas. As saídas podem ser entregas ou resultados.



A figura 14 ilustra as cinco etapas que permitem a "GESTÃO" de um projeto: Iniciação; Planejamento; Execução; Monitoramento e Controle; e Encerramento.

Cada etapa do projeto inclui um conjunto de processos, chamados Grupo de Processos de Gerenciamento de Projetos. O gerente de projetos deve seguir etapas de gestão, como para a fase de Iniciação: selecionar processos, alocar responsáveis, realizar processos conforme o PMBOK, e gerar resultados. Esse ciclo é repetido nas etapas de Planejamento, Execução, Monitoramento e Controle, e Encerramento. Em projetos complexos, as etapas podem ser iterativas.

Os cinco grupos de processos são:

1. **Iniciação**: Define o Termo de Abertura do Projeto, autorizando alocação de recursos.
2. **Planejamento**: Planeja a execução do projeto, com a principal entrega sendo o cronograma.
3. **Execução**: Executa o projeto conforme o planejado, integrando pessoas e conhecimentos.
4. **Monitoramento e Controle**: Acompanha o progresso e desempenho do projeto, com ações corretivas para não conformidades.
5. **Encerramento**: Conclui formalmente o projeto, com entregas, aceitação do produto e avaliação dos resultados.

A complexidade de um projeto de software também é um fator determinante no gerenciamento de projetos, tornando-o multidisciplinar. Uma equipe técnica típica para desenvolvimento inclui engenheiro de software, analistas, programadores, arquitetos de software, entre outros. No entanto, para gerenciar o projeto, são necessários especialistas em áreas como custos, aquisições e recursos humanos, entre outras.

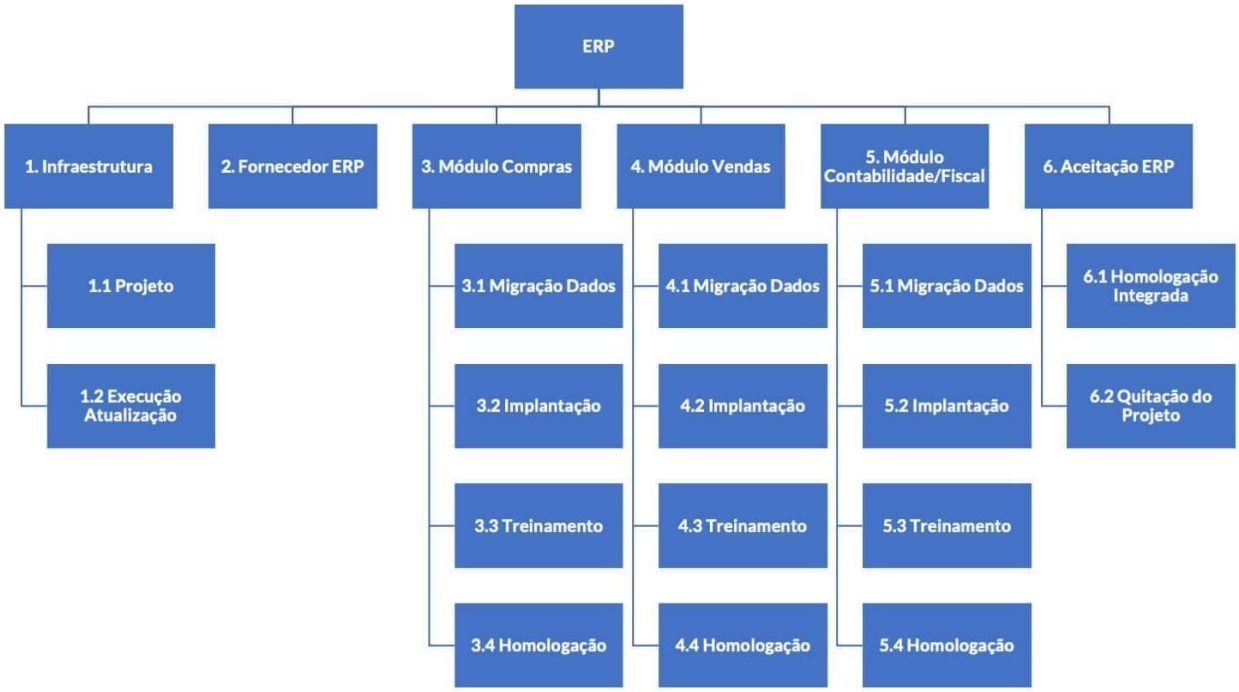
Áreas de Conhecimento	Iniciação	Planejamento	Execução	Monitoramento e controle	Encerramento
Integração	1. Desenvolver o termo de abertura do projeto	2. Desenvolver o plano de gerenciamento do projeto	3. Orientar e gerenciar a execução do projeto	4. Monitorar e controlar o trabalho do projeto 5. Realizar o controle integrado de mudanças	6. Encerrar o projeto ou fase1
Escopo		1. Coletar os requisitos 2. Definir o escopo 3. Criar a EAP		4. Verificar o escopo 5. Controlar o escopo	
Tempo		1. Definir as atividades 2. Sequenciar as atividades 3. Estimar os recursos das atividades 4. Estimar as durações das atividades 5. Desenvolver o cronograma		6. Controlar o cronograma	
Custos		1. Estimar os custos 2. Determinar o orçamento		3. Controlar os custos	
Qualidade		1. Planejar a qualidade	2. Realizar a garantia de qualidade	3. Realizar o controle da qualidade	
Recursos Humanos		1. Desenvolver o plano de recursos humanos	2. Mobilizar a equipe do projeto 3. Desenvolver a equipe de projeto 4. Gerenciar a equipe do projeto		
Comunicação	1. Identificar as partes interessadas	2. Planejar as comunicações	3. Distribuir as informações 4. Gerenciar as expectativas das partes interessadas	5. Reportar o desempenho	
Riscos		1. Planejar o gerenciamento dos riscos 2. Identificar os riscos 3. Realizar a análise qualitativa dos riscos 4. Realizar a análise quantitativa dos riscos 5. Planejar as respostas aos riscos		6. Monitorar e controlar os riscos	
Aquisição		1. Planejar as aquisições	2. Conduzir as aquisições	3. Administrar as aquisições	4. Encerrar as aquisições

O gerenciamento de projetos é multidisciplinar, e o PMBOK divide essa complexidade em 10 áreas de conhecimento, cada uma composta por processos específicos, conforme ilustrado na figura 15.

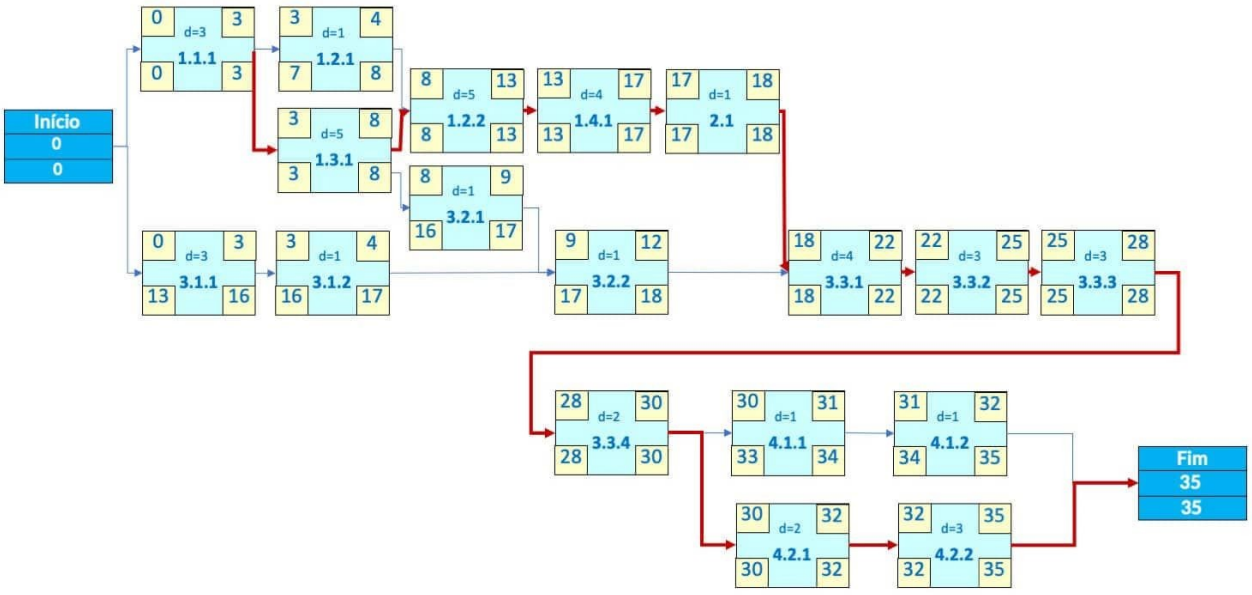
O perfil da equipe de gerenciamento de projeto deve estar alinhado com as áreas de conhecimento descritas a seguir.

Gerenciamento da Integração do Projeto: Envolve processos para integrar as diferentes áreas de conhecimento, sob controle direto do gerente de projetos. Este atua como um maestro, coordenando os resultados das áreas, com visão geral do projeto. A integração está presente em todos os grupos de processos, e o Termo de Abertura do Projeto autoriza a alocação de recursos.

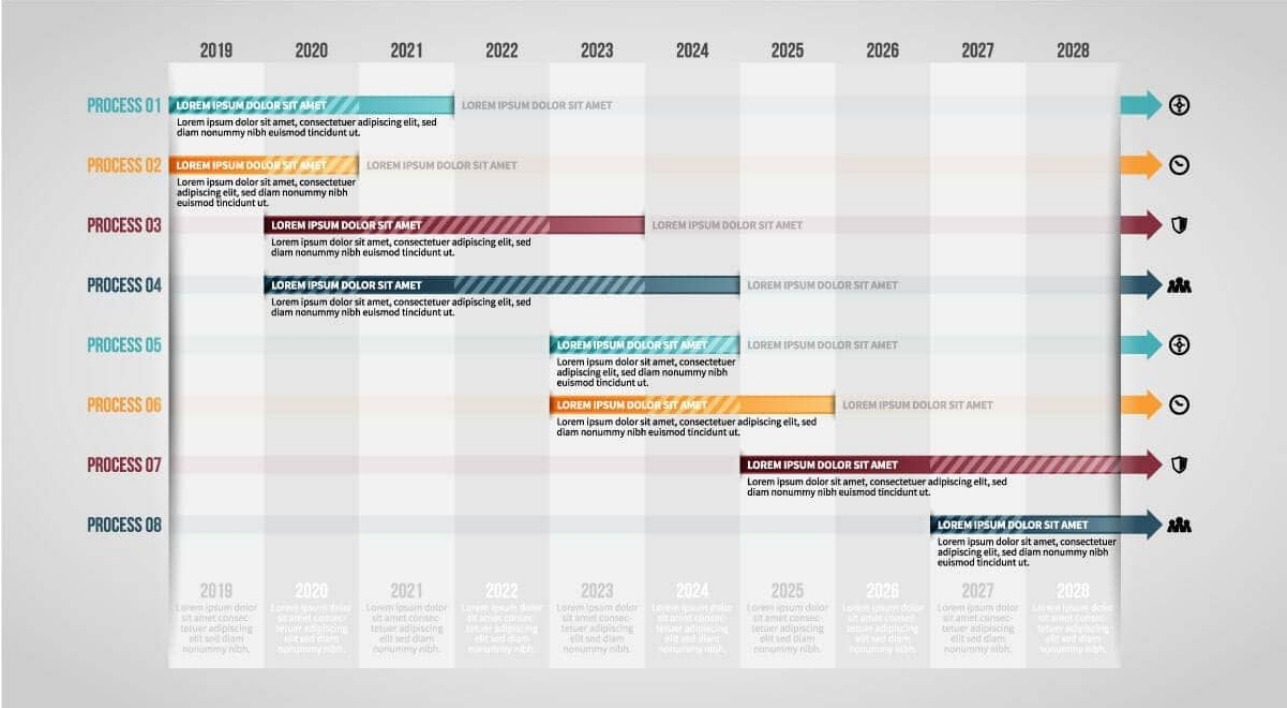
Gerenciamento do Escopo do Projeto: Inclui os processos necessários para garantir que o projeto cubra apenas o trabalho necessário para seu sucesso. A principal técnica é a criação da Estrutura Analítica do Projeto (EAP ou WBS), onde cada elemento é uma entrega, e as entregas não decompostas são chamadas de pacotes de trabalho.



Gerenciamento do Cronograma do Projeto: Inclui os processos para garantir a conclusão pontual do projeto, com o cronograma físico como principal entrega. Durante o planejamento, são identificadas as atividades a partir dos pacotes de trabalho da EAP, especificando a duração e insumos de cada uma. Em seguida, é elaborado o diagrama de rede, determinando as interdependências entre as atividades.



Finalmente, podemos gerar o cronograma do projeto, exemplificado na figura 18.



Gerenciamento dos Custos: Envolve os processos para garantir que o projeto seja concluído dentro do orçamento aprovado.

Gerenciamento da Qualidade: Processos para assegurar que os requisitos do projeto atendam às expectativas dos financiadores.

Gerenciamento dos Recursos: Processos para identificar, adquirir e gerenciar os recursos necessários para o sucesso do projeto.

Gerenciamento das Comunicações: Processos para garantir que as informações do projeto sejam geridas de forma oportuna e apropriada.

Gerenciamento dos Riscos: Processos para gerenciar os riscos do projeto.

Gerenciamento das Aquisições: Processos necessários para adquirir produtos, serviços ou resultados externos ao projeto.

Gerenciamento das Partes Interessadas: Processos para identificar partes interessadas e desenvolver estratégias para seu engajamento no projeto.

A importância do risco no gerenciamento de projetos, com base no PMBOK. O risco é um evento incerto que pode afetar positivamente ou negativamente os objetivos do projeto. A análise de risco envolve identificar possíveis problemas, como falhas em viagens, e avaliar os efeitos negativos ou positivos que podem ocorrer. Por exemplo, uma empresa exportadora pode se beneficiar de uma variação cambial positiva, aumentando seu portfólio de projetos, ou ser prejudicada por uma variação negativa, reduzindo seu capital. No contexto da Engenharia de Software, o foco está nos riscos negativos, que devem ser avaliados e gerenciados adequadamente.

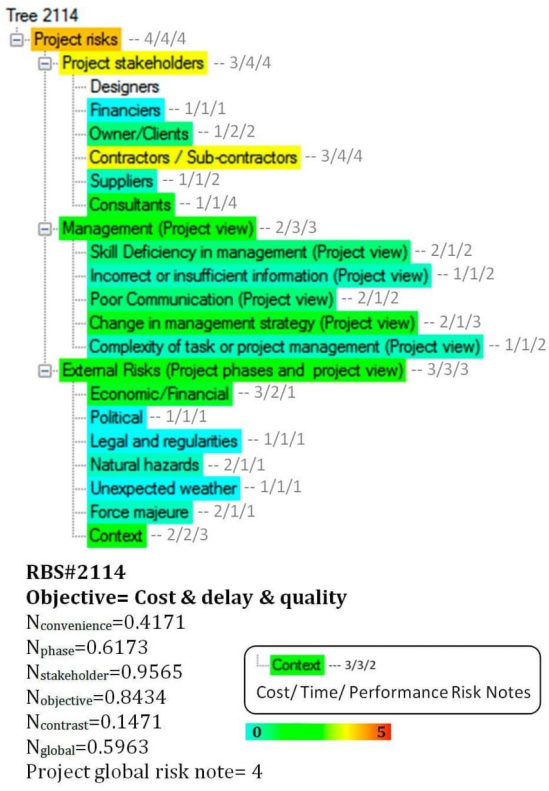
O gerenciamento de risco permite planejar e evitar surpresas nos projetos, lidando com eventos inesperados de forma sistemática. A área de Gerenciamento de Riscos inclui processos para

identificar, analisar e responder aos riscos de um projeto, sendo organizada em um plano estruturado.

Áreas de Conhecimento	Iniciação	Planejamento	Execução	Monitoramento e controle	Encerramento
Integração	1. Desenvolver o termo de abertura do projeto	2. Desenvolver o plano de gerenciamento do projeto	3. Orientar e gerenciar a execução do projeto	4. Monitorar e controlar o trabalho do projeto 5. Realizar o controle integrado de mudanças	6. Encerrar o projeto ou fase1
Escopo		1. Coletar os requisitos 2. Definir o escopo 3. Criar a EAP		4. Verificar o escopo 5. Controlar o escopo	
Tempo		1. Definir as atividades 2. Sequenciar as atividades 3. Estimar os recursos das atividades 4. Estimar as durações das atividades 5. Desenvolver o cronograma		6. Controlar o cronograma	
Custos		1. Estimar os custos 2. Determinar o orçamento		3. Controlar os custos	
Qualidade		1. Planejar a qualidade	2. Realizar a garantia de qualidade	3. Realizar o controle da qualidade	
Recursos Humanos		1. Desenvolver o plano de recursos humanos	2. Mobilizar a equipe do projeto 3. Desenvolver a equipe de projeto 4. Gerenciar a equipe do projeto		
Comunicação	1. Identificar as partes interessadas	2. Planejar as comunicações	3. Distribuir as informações 4. Gerenciar as expectativas das partes interessadas	5. Reportar o desempenho	
Riscos		1. Planejar o gerenciamento dos riscos 2. Identificar os riscos 3. Realizar a análise qualitativa dos riscos 4. Realizar a análise quantitativa dos riscos 5. Planejar as respostas aos riscos		6. Monitorar e controlar os riscos	
Aquisição		1. Planejar as aquisições	2. Conduzir as aquisições	3. Administrar as aquisições	4. Encerrar as aquisições

A identificação de riscos envolve três componentes: evento, probabilidade de ocorrência e impacto. Parte do plano de gerenciamento de riscos, a Estrutura Analítica do Risco (EAR) ajuda a identificar fontes de risco de forma categorizada. As categorias de risco no contexto de software incluem:

- **Tamanho do produto**: riscos relacionados ao tamanho do software.
- **Impacto no negócio**: riscos devido a restrições do cliente ou mercado.
- **Características do envolvido**: riscos na comunicação entre engenheiro de software e cliente.
- **Definição do processo**: riscos de auditoria dos processos pela equipe de qualidade.
- **Ambiente de desenvolvimento**: riscos relacionados às ferramentas usadas no desenvolvimento.
- **Tecnologia a ser criada**: riscos associados a inovações tecnológicas.
- **Quantidade de pessoas e experiência**: riscos relacionados à experiência da equipe.



O processo de identificação de riscos é iterativo, podendo surgir novos riscos durante o projeto. Técnicas comuns incluem:

1. **Brainstorming**: Reunião com especialistas para listar riscos e suas fontes, facilitada por um orientador.
2. **Lista de Verificação**: Riscos baseados em informações de projetos anteriores, ajudando a identificar pontos fracos.
3. **Entrevistas**: Realizadas com gerentes de projeto para levantar questões críticas sobre o compromisso da gerência, requisitos, escopo, equipe e tecnologia.

Se alguma questão for respondida negativamente, o risco é adicionado à lista.

A Análise Qualitativa dos Riscos prioriza os riscos com base na probabilidade de ocorrência e no impacto nos objetivos do projeto. Isso ajuda a determinar quais riscos são mais relevantes, considerando limitações de tempo e recursos. A análise é feita combinando probabilidades e impactos dos riscos para avaliar seu potencial de afetar os resultados do projeto.

Probabilidades:

- Grande chance de ocorrer: 0,95
- Provavelmente ocorrerá: 0,75
- Igual chance de ocorrer ou não: 0,50
- Baixa chance de ocorrer: 0,25
- Pouca chance de ocorrer: 0,10

Graus de impacto:

- Muito grande: 5
- Grande: 4
- Moderado: 3
- Pequeno: 2
- Muito pequeno: 1

Exemplo de aplicação:

1. Problema de saúde: Probabilidade 0,75, Impacto 5, Resultado 3,75
2. Cancelamento da passagem: Probabilidade 0,50, Impacto 3, Resultado 1,5
3. Dólar insuficiente: Probabilidade 0,50, Impacto 4, Resultado 2,0

A análise de prioridades revelou que o risco mais crítico é relacionado à saúde, seguido pela quantidade de dólar e, por último, o cancelamento da passagem.

A Força Aérea Americana define os componentes de risco em projetos de software como desempenho, custo, suporte e cronograma, com categorias de impacto como catastrófico, crítico, marginal e negligenciável.

A probabilidade e o impacto dos riscos são determinados por especialistas, que usam experiências de projetos passados para lidar com incertezas. O gerente de projetos deve integrar conhecimentos de diferentes áreas, embora não precise ser especialista em todas elas.

A Análise Quantitativa dos Riscos avalia numericamente os efeitos dos riscos priorizados, pois podem impactar os resultados do projeto. As técnicas incluem simulações e árvore de decisão. Na Engenharia de Software, a árvore de decisão pode ser usada para avaliar opções como contratar uma fábrica de software ou desenvolver internamente.

No exemplo apresentado, a análise quantitativa revela os custos dos riscos:

1. Problema de saúde: R$ 10.000
2. Dólar insuficiente: R$ 1.000
3. Cancelamento de passagem: R$ 3.000.

O processo de Planejamento de Respostas aos Riscos envolve ações para aumentar oportunidades e reduzir ameaças, gerenciando riscos conforme sua prioridade. As ações incluem tratamentos como mitigação (reduzir impacto) e eliminação (remover impacto).

No exemplo apresentado, as respostas aos riscos são:

1. Problema de saúde: Eliminação - Contratar seguro viagem.
2. Dólar insuficiente: Eliminação - Atualizar cartão de crédito.
3. Cancelamento de passagem: Mitigação - Confirmar reserva em companhia aérea confiável.

O **Gerenciamento de Riscos do Projeto** trata de eventos incertos que podem impactar os objetivos do projeto de forma positiva ou negativa.

A **Estrutura Analítica de Riscos (EAR)** auxilia na identificação e categorização dos riscos. Após identificá-los, é necessário priorizá-los, pois seu tratamento pode afetar o orçamento.

A **análise qualitativa** avalia a **probabilidade de ocorrência** e o **impacto** dos riscos, permitindo sua priorização. Em seguida, a **análise quantitativa** quantifica os impactos financeiros dos riscos priorizados.

O planejamento se encerra com o **Plano de Respostas aos Riscos**, que define ações para eliminar ou mitigar impactos. Esse plano é executado pelo processo **Implementar Respostas aos Riscos** e monitorado pelo processo **Monitorar os Riscos**.

Áreas de Conhecimento	Iniciação	Planejamento	Execução	Monitoramento e controle	Encerramento
Integração	1. Desenvolver o termo de abertura do projeto	2. Desenvolver o plano de gerenciamento do projeto	3. Orientar e gerenciar a execução do projeto	4. Monitorar e controlar o trabalho do projeto 5. Realizar o controle integrado de mudanças	6. Encerrar o projeto ou fase1
Escopo		1. Coletar os requisitos 2. Definir o escopo 3. Criar a EAP		4. Verificar o escopo 5. Controlar o escopo	
Tempo		1. Definir as atividades 2. Sequenciar as atividades 3. Estimar os recursos das atividades 4. Estimar as durações das atividades 5. Desenvolver o cronograma		6. Controlar o cronograma	
Custos		1. Estimar os custos 2. Determinar o orçamento		3. Controlar os custos	
Qualidade		1. Planejar a qualidade	2. Realizar a garantia de qualidade	3. Realizar o controle da qualidade	
Recursos Humanos		1. Desenvolver o plano de recursos humanos	2. Mobilizar a equipe do projeto 3. Desenvolver a equipe de projeto 4. Gerenciar a equipe do projeto		
Comunicação	1. Identificar as partes interessadas	2. Planejar as comunicações	3. Distribuir as informações 4. Gerenciar as expectativas das partes interessadas	5. Reportar o desempenho	
Riscos		1. Planejar o gerenciamento dos riscos 2. Identificar os riscos 3. Realizar a análise qualitativa dos riscos 4. Realizar a análise quantitativa dos riscos 5. Planejar as respostas aos riscos		6. Monitorar e controlar os riscos	
Aquisição		1. Planejar as aquisições	2. Conduzir as aquisições	3. Administrar as aquisições	4. Encerrar as aquisições

O projeto de software possui características que necessitam das melhores práticas de gerenciamento de riscos contidas no PMBOK, com destaque para as atualizações constantes das tecnologias e a alta volatilidade dos requisitos durante o projeto.

Lembre-se, analise os riscos do seu projeto.

A **Engenharia de Software** se dedica ao desenvolvimento sistemático de software, sendo essencial para lidar com sua **complexidade**, como em sistemas embarcados em aeronaves ou no controle de tráfego aéreo.

Diferente de outros produtos de engenharia, o software é altamente **volátil**, devido às constantes mudanças tecnológicas e de requisitos. Para lidar com isso, a Engenharia de Software adota uma abordagem estruturada, organizada em **camadas**:

- **Qualidade**: garante que os requisitos atendam às expectativas dos usuários.
- **Processo**: define as etapas de desenvolvimento do software (base da Engenharia de Software).
- **Métodos**: estabelece técnicas e artefatos de software.
- **Ferramentas**: incentiva o uso de ferramentas CASE.

A **camada de processo** é fundamental, pois estrutura todas as etapas do desenvolvimento de software.

O **desenvolvimento de software** utiliza intensamente a **abstração**, iniciando com especificações de alto nível e reduzindo gradualmente até o código, que representa o nível mais detalhado.

Os **modelos de processos de desenvolvimento** incluem as seguintes **atividades típicas**:

- **Levantamento de requisitos**
- **Análise**
- **Projeto**
- **Implementação**
- **Testes**

Essas etapas estruturam o desenvolvimento de software, garantindo organização e eficiência.

A **Engenharia de Requisitos** faz parte da **Engenharia de Software** e abrange as etapas de **levantamento de requisitos** e **análise**.

Segundo Sommerville (2007), os **requisitos** descrevem os serviços do sistema e suas restrições, refletindo as necessidades dos clientes.

As principais atividades da Engenharia de Requisitos incluem:

1. **Concepção** – Compreensão inicial do problema, identificação das partes interessadas e definição da solução desejada.

2. **Levantamento** – Definição do escopo do projeto, alinhando a visão entre clientes e desenvolvedores. O resultado dessa etapa é a **especificação de requisitos**, que serve como um contrato entre ambas as partes.

O processo completo envolve ainda **elaboração, negociação, especificação, validação e gestão**.

Na etapa de **levantamento de requisitos**, os clientes devem compreender a especificação e fornecer feedback para evitar erros no projeto.

Os requisitos são classificados em três categorias:

- **Funcionais** – Definem os serviços do sistema (ex.: geração de histórico escolar).
- **Não funcionais** – Impõem restrições operacionais (ex.: linguagem de programação, legislação, qualidade).
- **De domínio** – Regras de negócio que restringem os requisitos funcionais (ex.: cálculo da média para aprovação).

Principais técnicas de levantamento de requisitos:

- **Observação (etnografia)** – O engenheiro acompanha a rotina dos usuários.
- **Entrevista** – Questionamentos diretos às partes interessadas.
- **Pesquisa** – Questionário para obtenção de dados quantitativos/qualitativos.
- **JAD (Joint Application Design)** – Reuniões de grupo com usuários e desenvolvedores.
- **Brainstorming** – Sessões para coleta de ideias e necessidades dos envolvidos.

A entrega desta etapa é o **documento de requisitos**, que registra todas as necessidades dos clientes e formaliza o aceite do produto proposto. Esse documento permite a **rastreabilidade dos requisitos**, assegurando a conformidade das especificações até a codificação.

Na **elaboração**, engenheiros de software analisam detalhadamente os requisitos e criam modelos para representar o sistema. A modelagem se baseia em **cenários**, definidos a partir dos requisitos funcionais, descrevendo como os usuários interagem com o sistema.

A **UML (Unified Modeling Language)** é usada para modelagem de **casos de uso**, incluindo **diagramas** (artefatos gráficos) e **descrições** (artefatos textuais) para representar essas interações.

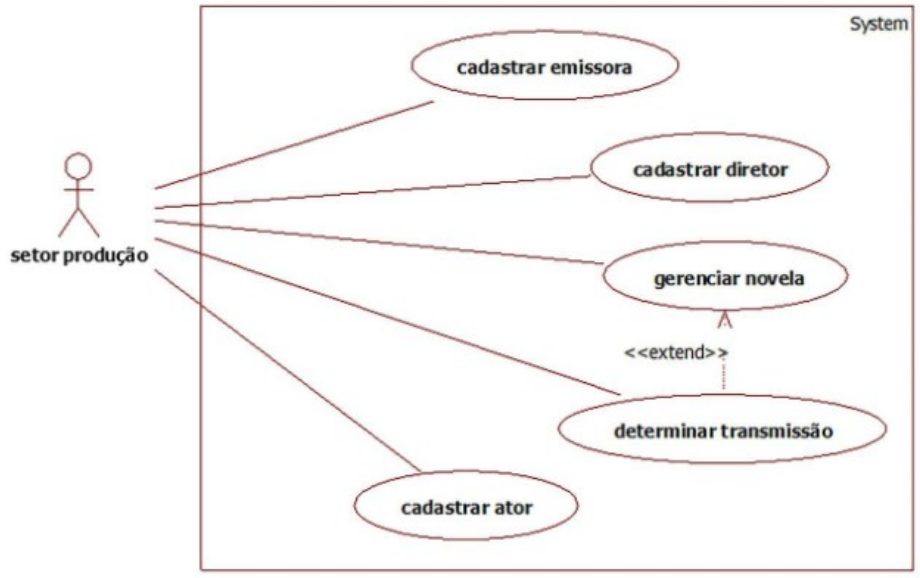
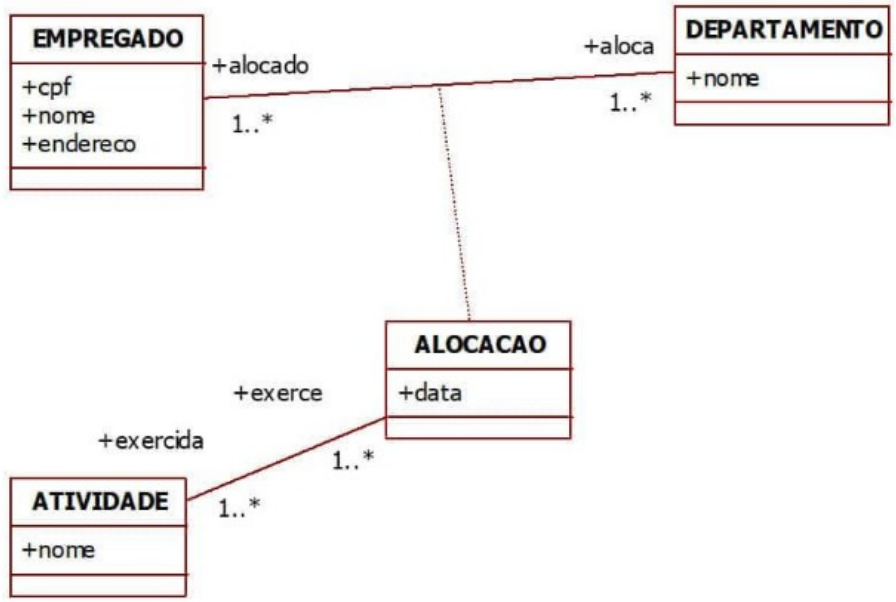


Figura 2 – Exemplo de diagrama de casos de uso.

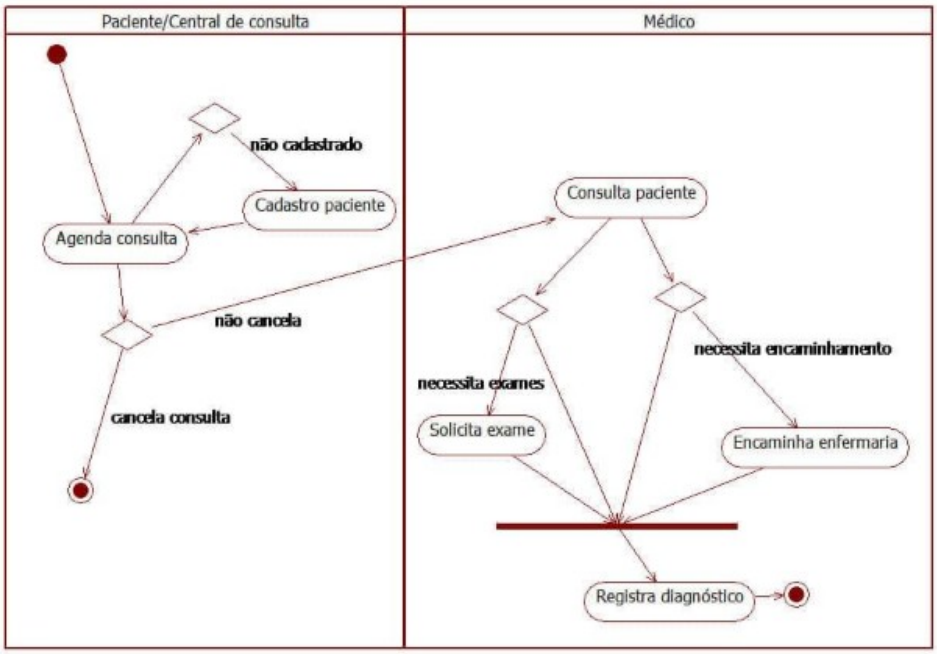
A partir dos **casos de uso**, identificam-se as **classes de análise**, que representam os objetos do negócio ou domínio do problema.

Na **modelagem de análise**, constrói-se:

- **Modelo de atividades**: usado para mapear processos de negócios, descrever graficamente um caso de uso ou definir o algoritmo de um método.
- **Modelo de estados**: representa mudanças significativas de estado e os eventos que as causam.



ura 3 – Exemplo de diagrama de classes.



A **Engenharia de Requisitos** é essencial no desenvolvimento de software, abrangendo **levantamento e análise de requisitos**.

Principais **etapas do processo**:

- **Negociação**: prioriza requisitos, resolve conflitos e avalia custos e riscos.

- **Especificação**: gera um documento detalhado com todos os requisitos e modelos.
- **Validação**: verifica se os modelos refletem as necessidades dos usuários.
- **Gestão**: controla mudanças nos requisitos ao longo do projeto.

Os requisitos são classificados em:

- **Funcionais**: funcionalidades do sistema.
- **Não funcionais**: restrições operacionais e requisitos de qualidade.
- **De domínio**: regras de negócio específicas.

A **matriz de rastreabilidade dos requisitos** permite monitorar a estabilidade dos requisitos ao longo do projeto.

Os principais **modelos da etapa de análise** no desenvolvimento de software são:

- **Modelo de casos de uso**
- **Modelo de classes de análise**
- **Modelo de atividades**
- **Modelo de estados**

A **abstração** é essencial no desenvolvimento, iniciando com especificações de alto nível. Na **fase de projeto**, o nível de abstração dos diagramas de análise é reduzido por refinamentos sucessivos e pela criação de novos modelos.

O termo **"projeto"** pode ser usado tanto no sentido de **design** quanto de **project**, dependendo do contexto.

O **modelo de classes** representa os aspectos **estáticos e estruturais** do sistema. Durante o refinamento, novos elementos são adicionados para reduzir a abstração e permitir a implementação das classes.

Os refinamentos seguem etapas:

1. Definição da classe e nome.
2. Inclusão de atributos.
3. Adição de métodos/funções.
4. Detalhamento de atributos e métodos.

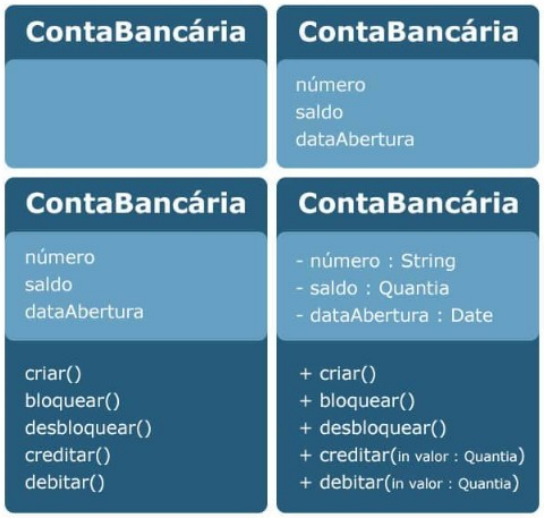
No projeto, os tipos de dados são essenciais para garantir a implementação correta. Além disso, novos elementos podem ser inseridos devido a **associações, heranças e novas classes**.

Também podem ser aplicados **padrões de projeto (design patterns)**, que são templates de boas práticas para solucionar problemas comuns, como o **Factory Method**.

Resumo:

O **modelo de classes** representa os aspectos **estáticos e estruturais** do sistema. Durante o refinamento, novos elementos são adicionados para reduzir a abstração e permitir a implementação das classes.

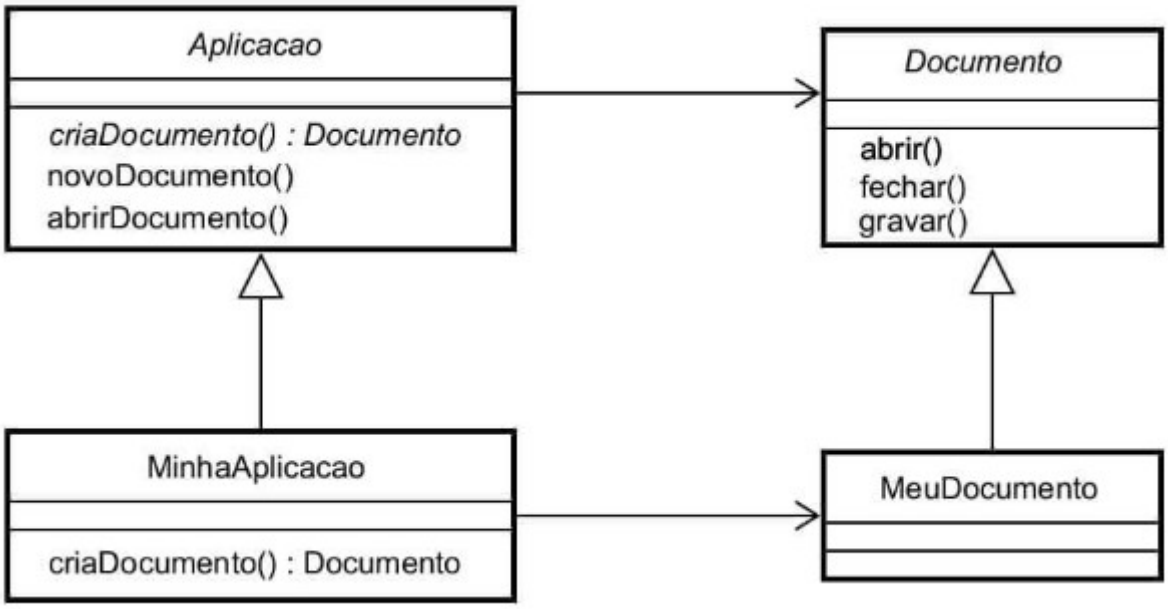
Os refinamentos seguem etapas:



1. Definição da classe e nome.
2. Inclusão de atributos.
3. Adição de métodos/funções.
4. Detalhamento de atributos e métodos.

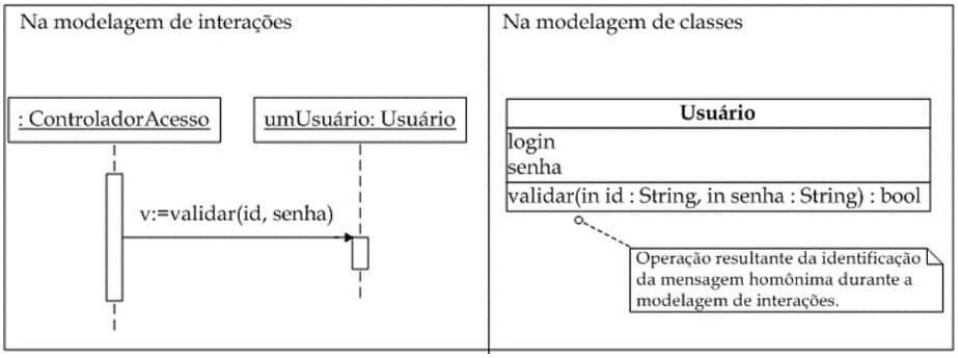
No projeto, os tipos de dados são essenciais para garantir a implementação correta. Além disso, novos elementos podem ser inseridos devido a **associações, heranças e novas classes**.

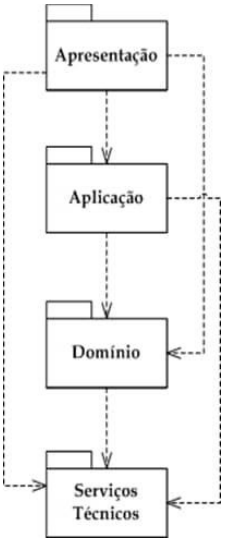
Também podem ser aplicados **padrões de projeto (design patterns)**, que são templates de boas práticas para solucionar problemas comuns, como o **Factory Method**.



Os **aspectos dinâmicos** do sistema referem-se à comunicação entre **objetos**, fundamental na **orientação a objetos**. Para representar essa dinâmica, utilizam-se **modelos de interação**, como o **diagrama de comunicação** e o **diagrama de sequência**, que têm a mesma abstração, mas representações distintas.

O **modelo de interação** é desenvolvido em paralelo ao **modelo de classes**. Os **métodos** nas classes são identificados pelas **mensagens** no modelo de interação. Exemplo: no diagrama de sequência, a mensagem "validar" entre os objetos "controlador acesso" e "usuário" resulta na implementação do método "validar" na classe "usuário".





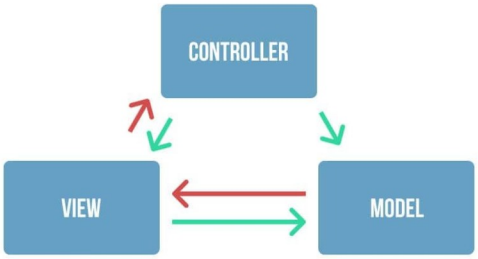
No desenvolvimento de software, a **fatoração** envolve dividir sistemas complexos em subsistemas menores para facilitar o controle da complexidade. O **projeto de arquitetura** define a estrutura do sistema, identificando esses subsistemas e estabelecendo um framework de controle e comunicação entre eles. A arquitetura é descrita em duas abstrações: **lógica** e **física**.

A arquitetura em camadas é uma abordagem comum, que divide o sistema em camadas: **apresentação**, **aplicação**, **domínio** e **serviços técnicos**. A camada superior depende das inferiores, permitindo uma melhor gestão da complexidade, **baixo acoplamento** (minimização de dependências) e **alta coesão** (maior interação interna entre classes).

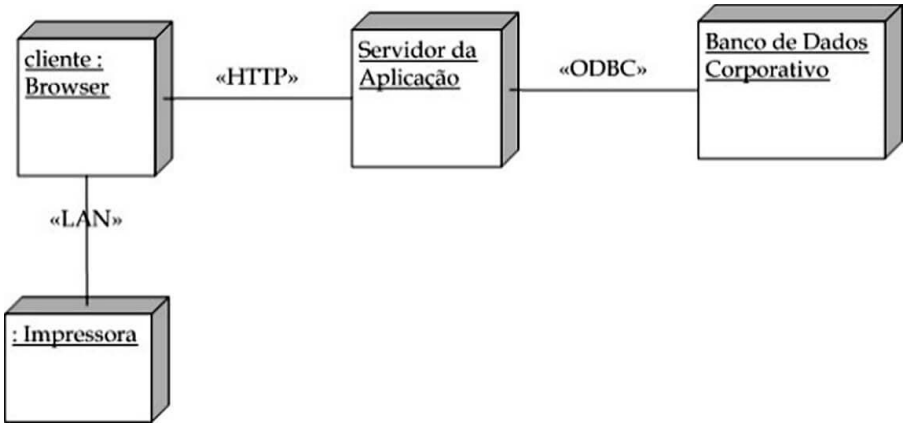
- **Camada de apresentação**: inclui classes de fronteira que interagem com os usuários.
- **Camada de aplicação**: inclui classes de controle que fazem a intermediação entre a apresentação e o domínio.
- **Camada de domínio**: composta por classes que representam o problema e têm métodos para alterar o estado.
- **Camada de serviços técnicos**: inclui serviços genéricos, como persistência de dados.

Além disso, existem **padrões de arquitetura** aplicados à organização do sistema, como o **MVC (Model-View-Controller)**. Este padrão, desenvolvido na década de 1970, foca na separação de responsabilidades:

- **Modelo** (Model): mantém os dados e notifica mudanças.
- **Visão** (View): exibe os dados.
- **Controlador** (Controller): atualiza o estado do modelo e altera a apresentação da visão.

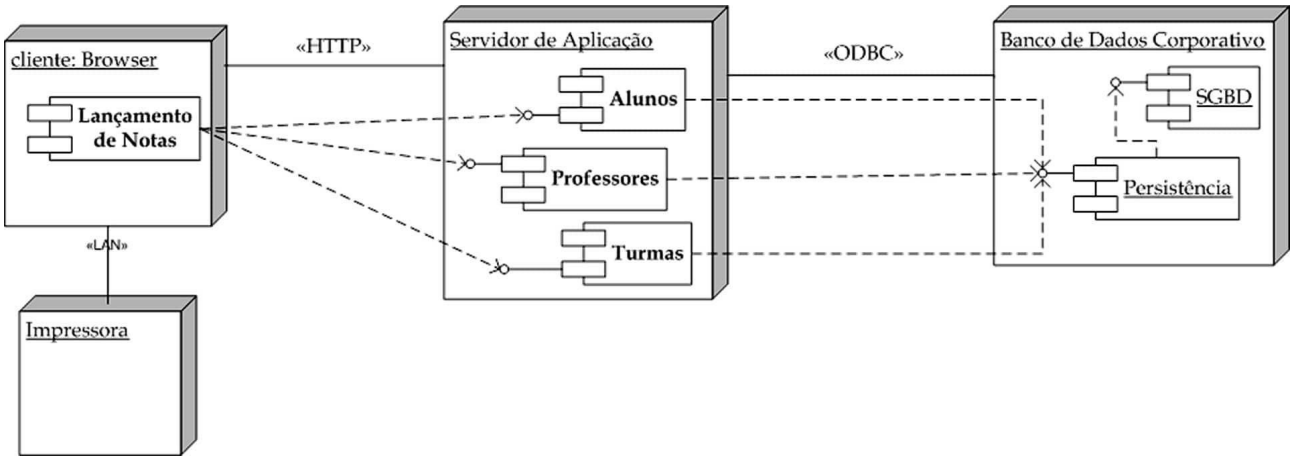


Após definir a arquitetura lógica, o engenheiro de software cria a **arquitetura física** através do **modelo de implantação**, que projeta a infraestrutura de hardware necessária. Embora o engenheiro não especifique o hardware, ele deve definir os nós computacionais para o software, deixando a especificação do hardware para a equipe de infraestrutura. A **Figura 10** ilustra um diagrama de implantação com nós computacionais e conexões.



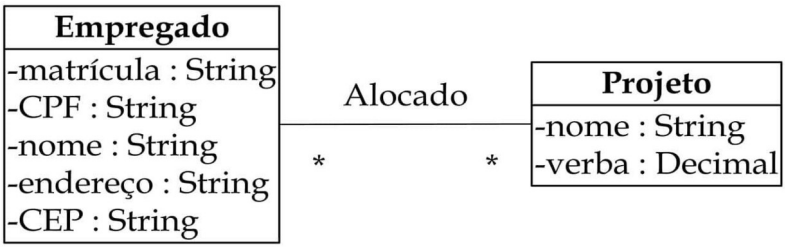
Cada camada é composta por classes, formando componentes independentes. Esses componentes, que podem ser formados por uma ou mais classes, encapsulam funcionalidades e podem compor sistemas complexos. A "componentização" do software é representada no modelo de

implementação. A **Figura 11** mostra o diagrama de componentes integrado ao diagrama de implantação da **Figura 10**, destacando a alocação dos componentes por nó computacional.



O paradigma de software atual é orientado a objetos, que lidam com dados complexos e exigem persistência. Por outro lado, a tecnologia de banco de dados é relacional, com estrutura de armazenamento bidimensional em tabelas.

Para integrar esses padrões, usamos o diagrama de classes como modelo conceitual do banco de dados, gerando o modelo lógico e as tabelas necessárias. Esse processo é chamado mapeamento objeto-relacional. A **Figura 12** ilustra um exemplo de associação muitos-para-muitos e o mapeamento resultante em três tabelas.



Agora, confira o resultado do mapeamento muitos-para muitos:

1 Empregado(id, matrícula, CPF, nome, endereço, CEP, idDepartamento)
2 Alocação(id, idProjeto, idEmpregado)
3 Projeto (id, nome, verba)

Na etapa de projeto da interface gráfica, o **diagrama de casos de uso** define as funcionalidades do sistema, representando interações completas entre o sistema e os atores. Essas interações requerem a definição de uma interface homem-máquina, onde a **usabilidade** é o requisito não funcional fundamental – ou seja, o usuário deve estar "no comando" durante a definição da interface.

Devem ser consideradas questões como:

- **Tempo de resposta**: em sistemas de venda online, um tempo de resposta elevado pode levar à desistência do cliente;
- **Recursos de ajuda**: disponíveis sem que o usuário precise abandonar a interface;
- **Tratamento de erros**: utilizando uma linguagem clara e inteligível;
- **Acessibilidade**: garantindo atendimento a usuários com necessidades especiais.

Este módulo enfatiza a importância da etapa de projeto no desenvolvimento de software, que engloba as seguintes atividades:

1. **Refinamento dos aspectos estáticos e estruturais do sistema**.
2. **Detalhamento dos aspectos dinâmicos do sistema**.
3. **Detalhamento da arquitetura do sistema**.
4. **Mapeamento objeto-relacional**.
5. **Realização do projeto da interface gráfica com o usuário**.

Ao final dessa etapa, as especificações dos modelos de projeto permitem iniciar a implementação do software conforme os requisitos definidos.

A etapa de **implementação** traduz os modelos de projeto (diagramas e especificações textuais) em código executável, assim como uma planta baixa orienta a construção de uma casa. Essa fase utiliza uma ou mais linguagens de programação para desenvolver o software, enfrentando desafios devido à falta de detalhes dos algoritmos e à influência de requisitos não funcionais, como:

- **Linguagens de programação e frameworks** (ex.: Hibernate para mapeamento objeto-relacional);
- **Sistemas de gerenciamento de banco de dados**;
- **Requisitos de qualidade** (confiabilidade, usabilidade) e **reutilização de componentes**.

O aumento da potência dos computadores transformou a programação em um problema cada vez mais complexo, conceito sintetizado na "Crise do Software" de Dijkstra (1972). O avanço tecnológico também impulsionou a evolução do paradigma estruturado para o orientado a objetos, promovendo o reúso e a melhor manutenção das especificações, o que permite o desenvolvimento de softwares mais complexos.

A complexidade em projetos de software, muitas vezes decorrente do tamanho das especificações e das interações entre seus componentes, impacta negativamente o desenvolvimento. A **qualidade** do software é definida como a conformidade com os requisitos, visando satisfazer o cliente por meio da aplicação da Engenharia de Software.

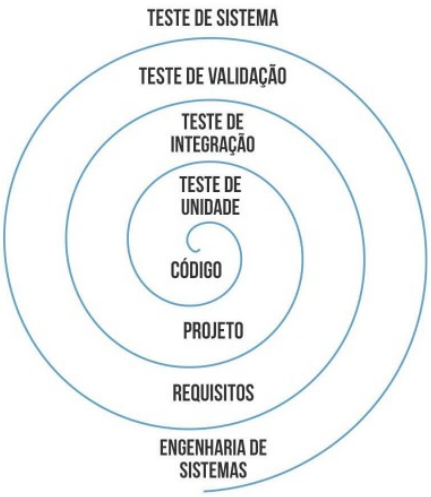
A norma **ISO 9126** identifica seis atributos fundamentais de qualidade:

- **Funcionalidade:** atendimento dos requisitos funcionais.
- **Confiabilidade:** tempo de disponibilidade.
- **Usabilidade:** facilidade de uso.
- **Eficiência:** otimização dos recursos.
- **Facilidade de manutenção:** facilidade na correção.
- **Portabilidade:** adaptação a diferentes ambientes.

A qualidade pode ser avaliada em duas dimensões: **qualidade do processo** e **qualidade do produto**. Na etapa de implementação, o foco está na qualidade do produto, assegurada por meio de testes que garantem a conformidade com os requisitos.

A qualidade do software é assegurada por meio de testes sistemáticos aplicados em diversas fases do desenvolvimento. Esses testes validam a estrutura interna, a conformidade com os requisitos e a integração dos subsistemas (incluindo suas interfaces).

Um **teste de software** é um processo planejado para identificar erros. Em sistemas complexos, os testes podem detectar a presença de erros, mas não garantem sua total ausência. Softwares mal testados podem causar prejuízos significativos, afetando, por exemplo, decisões gerenciais devido a informações incorretas.



Os procedimentos de teste, também conhecidos como testes dinâmicos, geralmente são automatizados, permitindo a simulação de vários cenários. Quanto mais cenários forem simulados, maior a validação e a qualidade do produto.

Durante a codificação, pode ser adotada uma estratégia em espiral (conforme ilustrado na Figura 14), que aumenta o nível de abstração a cada ciclo e detalha as etapas do processo de teste.

Principais tipos de teste de software:

Teste de unidade:

- Valida componentes individuais da aplicação.
- Executa o software para exercitar toda a estrutura interna de um componente, como desvios condicionais e laços de processamento.

Teste de integração:

- Assegura a compatibilidade entre novas unidades (ou modificadas) e os componentes já existentes.
- Detecta erros decorrentes de alterações ou exclusões em rotinas e propriedades públicas, que podem quebrar a integração entre componentes.

Teste de validação:

- Avalia a solução como um todo, considerando que a maioria das falhas já foi identificada nos testes de unidade e integração.
- Realiza testes funcionais, para verificar se o sistema executa casos de uso completos conforme esperado pelo usuário, e testes não funcionais, que avaliam desempenho, segurança, confidencialidade, recuperação de falhas, entre outros.
- Utiliza infraestrutura de hardware mais complexa, simulando o ambiente real de produção.

Os testes de software são planejados a partir dos cenários dos casos de uso identificados na análise, que inicialmente são definidos em alto nível (casos de teste) pelo engenheiro de software. Em seguida, o analista de teste detalha esses cenários, criando casos de teste específicos. Por exemplo, no caso de uso “Realizar saque de conta”, os cenários podem incluir:

- Transação sem problemas, quando o saldo supera o valor do saque;
- Bloqueio por saldo insuficiente;
- Liberação do saque com uso do limite disponível (com juros);
- Liberação com retirada de fundo de aplicação, entre outros.

Nesta etapa, também ocorrem os procedimentos de **aceite**, realizados a cada ciclo de implementação para corrigir pontos não atendidos. O aceite, sendo a última chance de detectar falhas antes da implantação, é dividido em três momentos:

- **Teste Alpha:** Usuários finais operam o software em ambiente controlado, na instalação do desenvolvedor.
- **Teste Beta:** Usuários finais testam o produto em seus próprios ambientes, fora do controle do desenvolvedor.
- **Aceite Formal:** Semelhante ao Teste Beta, mas os próprios clientes definem os testes e validam se os requisitos foram implementados.

A validação é bem-sucedida quando o software atende aos requisitos e funciona conforme o esperado pelo usuário.

Na última etapa da espiral de testes, o **teste de sistema** avalia o software em conjunto com outros elementos do sistema (hardware, base de dados), considerando o contexto amplo da engenharia de sistemas. Nesta etapa, são realizados diversos tipos de teste:

- **Teste de recuperação:**
  Avalia o comportamento do software após erros ou condições anormais, assegurando a restauração do estado inicial da transação.
  Exemplos: saque com defeito em caixa eletrônico ou com queda de energia.
- **Teste de segurança:**
  Detecta falhas que possam comprometer o sigilo, a integridade dos dados ou causar interrupções.
  Exemplos: verificação do uso correto de senhas antes e depois de transações ou solicitação de senha randômica.
- **Teste por esforço:**
  Simula condições atípicas de uso, com volumes de transações que excedem o previsto, gerando picos para avaliar a resposta do software e da infraestrutura.
  Exemplo: simular 10.000 saques simultâneos.
- **Teste de desempenho:**
  Verifica se o software atende aos requisitos de tempo de resposta e performance em situações de pico de acesso e concorrência.
  Exemplo: garantir que a manipulação de dispositivos físicos no saque não ultrapasse 10 segundos.
- **Teste de disponibilização (ou de configuração):**
  Executa o software em diversas configurações de hardware e software, garantindo sua operação em ambientes variados, além de testar procedimentos de instalação e documentação.
  Exemplo: simular saque com impressoras de diferentes fornecedores.

Este módulo destaca a importância das etapas de implementação e testes, que seguem uma estratégia em espiral:

1. **Teste unitário:** valida a estrutura interna de cada componente.

2. **Teste de integração:** confirma a compatibilidade entre novos ou modificados componentes e os previamente validados.
3. **Teste de validação:** realizado em ambientes controlados e depois em produção, para verificar se o sistema atende aos requisitos do usuário, culminando no aceite do sistema.
4. **Teste de sistema:** integra o software com outros elementos do sistema.

O objetivo final dos testes é identificar e corrigir erros antes da implantação, assegurando que o software funcione conforme o esperado e atenda aos requisitos.

A **implantação** é a etapa final do desenvolvimento, onde o software testado é transferido do ambiente de desenvolvimento para o de produção, tornando-se disponível para os usuários finais em suas atividades diárias.

As principais atividades envolvidas na implantação são:

- **Produção de releases externos:** Lançamento de novas versões ou atualizações do software, com decisões sobre a distribuição.
- **Embalagem:** Preparação do software para comercialização, por exemplo, em mídia como DVD.
- **Distribuição:** Entrega do software aos usuários, seja de forma manual ou automatizada.
- **Instalação:** Instalação de todos os arquivos necessários para a execução do software.
- **Prestação de ajuda e assistência:** Suporte aos usuários (como help desk) para resolver dúvidas e problemas técnicos.

Essas ações garantem que o software atenda aos requisitos e funcione conforme o esperado no ambiente real.

No contexto de um fluxo iterativo e incremental de desenvolvimento de software, após os testes, a primeira versão é implantada em produção, permitindo o uso real pelos usuários. Paralelamente, a equipe desenvolve a segunda versão, incorporando novos requisitos. Quando um defeito é identificado na versão em produção, o engenheiro de software enfrenta duas alternativas:

- **Cenário 1:** Corrigir o defeito na versão 2 e implantá-la, fazendo com que os usuários aguardem até a liberação dessa nova versão.
- **Cenário 2:** Realizar manutenção na versão 1 para eliminar o defeito, implantá-la e, em seguida, ajustar a versão 2 em desenvolvimento.

Tecnicamente, o **cenário 2** é preferível, pois minimiza o impacto negativo nos usuários, evitando que eles convivam com o defeito por um período prolongado.

Esse estudo de caso evidencia dois problemas fundamentais: **controle de alterações** e **controle de versões**.

- **Gestão de configuração:**
  Trata-se de um conjunto de tarefas que gerencia as mudanças durante todo o desenvolvimento do software. Uma das tarefas essenciais é o **controle de alterações**, que avalia cada solicitação (ou ECO – Engineering Change Order) quanto ao mérito técnico, aos efeitos colaterais, ao impacto global na configuração, na funcionalidade e ao custo da alteração. Um processo bem definido é vital para corrigir defeitos detectados em produção e evitar o caos em projetos complexos, já que os testes não eliminam todos os erros.

- **Controle de versões (Gestão de releases):**
  Refere-se ao gerenciamento das versões do sistema distribuídas aos clientes. Esse processo determina quando um release será liberado, gerencia sua criação, distribuição e documentação. Um release pode incluir arquivos de configuração, dados, programas de instalação, documentação (eletrônica ou impressa), empacotamento e publicidade associada.

Em resumo, a integração de práticas robustas de controle de alterações e de versões é essencial para uma implantação bem-sucedida, garantindo que o software em produção seja estável e atenda aos requisitos dos usuários.

Após a implantação bem-sucedida do software, inicia-se a **manutenção**, a etapa mais longa do ciclo de vida do software. Ela abrange a correção de defeitos não identificados, aprimoramento de subsistemas e implementação de novas funcionalidades baseadas em novos requisitos.

Com o início da manutenção, surgem relatórios de erros, solicitações de adaptações e melhorias, que devem ser planejadas, programadas e executadas. A **gestão de alterações** continua sendo essencial para garantir a execução dessas modificações.

Um desafio na manutenção é a **mobilidade dos engenheiros**, já que a equipe original pode ser substituída, dificultando a continuidade do trabalho. Para resolver isso, é crucial que todas as etapas do desenvolvimento sejam bem estruturadas, criando modelos e documentação que permitam a compreensão do sistema por engenheiros que não participaram do desenvolvimento.

A **manutenibilidade** é um requisito não funcional importante, pois indica a facilidade de corrigir, adaptar ou melhorar o software. O objetivo da engenharia de software é criar sistemas com alta manutenibilidade.

Após a implantação do software, inicia-se a **manutenção**, que envolve correção de erros e ajustes devido à volatilidade dos requisitos. Com o tempo, a tecnologia pode se tornar obsoleta, tornando a manutenção mais difícil.

Uma solução para isso é a **reengenharia de software**, que visa reconstruir o produto, adicionando mais funcionalidades, melhor desempenho, confiabilidade e manutenibilidade. A reengenharia ocorre em dois níveis:

1. **Reengenharia de processos de negócio**: Foca em melhorar a eficiência e competitividade de uma empresa através de alterações nas regras de negócio.
2. **Reengenharia de software**: Aplica-se quando a modificação de um software existente é suficiente, indo além da manutenção.

Resumindo: A **implantação** envolve a migração do software para o ambiente de produção, enquanto a **manutenção** lida com erros e mudanças nos requisitos. Quando a tecnologia fica obsoleta ou as regras de negócio mudam, a **reengenharia de software** pode ser necessária, ultrapassando a manutenção.

As etapas típicas de desenvolvimento de software são: **levantamento de requisitos**, **análise**, **projeto**, **implementação**, **testes**, **implantação** e **manutenção**.

- **Levantamento de requisitos**: Criação do documento de requisitos (funcionais, não funcionais, de domínio).

- **Análise**: Criação de modelos a partir dos requisitos (ex: casos de uso, classes, atividades).
- **Projeto**: Refinamento dos modelos de análise e criação de novos modelos (ex: interação, implementação, implantação).
- **Implementação**: Codificação da solução com base nos modelos de projeto.
- **Testes**: Validação de unidades de código e do sistema em ambiente de produção.
- **Implantação**: Migração do sistema para o ambiente de produção, onde os usuários finais irão utilizá-lo.
- **Manutenção**: Atendimento às solicitações de alterações devido a defeitos ou mudanças nos requisitos.

A Engenharia de Software é o principal produto dos modelos de Processo de Desenvolvimento de Software. Diferentemente de outras engenharias, o software é altamente volátil devido às constantes evoluções tecnológicas, o que adiciona complexidade. Para tratar essa complexidade, emprega-se uma metodologia que decompõe o problema em partes menores de forma sistemática.

A Engenharia de Software é estruturada em camadas:

- **Camada de qualidade:** Garante que os requisitos atendam às expectativas dos usuários.
- **Camada de processo:** Define as etapas de desenvolvimento do software e é a base da Engenharia de Software.
- **Camada de métodos:** Estabelece técnicas de levantamento de requisitos e gera artefatos de modelagem, como modelos de casos de uso e de classes.
- **Camada de ferramentas:** Promove o uso de ferramentas CASE para o desenho dos artefatos e geração automática de código.

O processo de desenvolvimento de software inicia com especificações e modelos de alto nível, que vão se detalhando até a codificação, representando o menor nível de abstração.

Segundo Pressman (2016), as atividades típicas são:

- **Comunicação:** Interação intensiva com os usuários para entender o problema, definir objetivos e identificar requisitos.
- **Planejamento:** Elaboração sistemática de um Plano de Gerenciamento do Projeto, com destaque para o cronograma das atividades.
- **Modelagem:** Criação de modelos gráficos (diagramas, como o de casos de uso) e descrições textuais, semelhante a uma planta baixa na construção.
- **Construção:** Implementação do software com base nos modelos, envolvendo codificação e testes conforme o planejado.
- **Entrega:** Migração do produto para o ambiente de produção, concluindo o ciclo com a disponibilização do software conforme o planejado.

Os modelos de processos prescritivos na Engenharia de Software sistematizam o desenvolvimento definindo atividades, fluxos e artefatos que proporcionam estabilidade, controle e organização. Embora todas as atividades típicas sejam empregadas, os modelos se diferenciam na ênfase de cada ação, sendo o encadeamento dessas atividades denominado fluxo de processo.

Um exemplo clássico é o **modelo em cascata**, que adota uma abordagem sequencial (conforme ilustrado na Figura 1). Este é o modelo de processo mais antigo, amplamente utilizado na era do desenvolvimento estruturado. Entretanto, ele apresenta limitações:

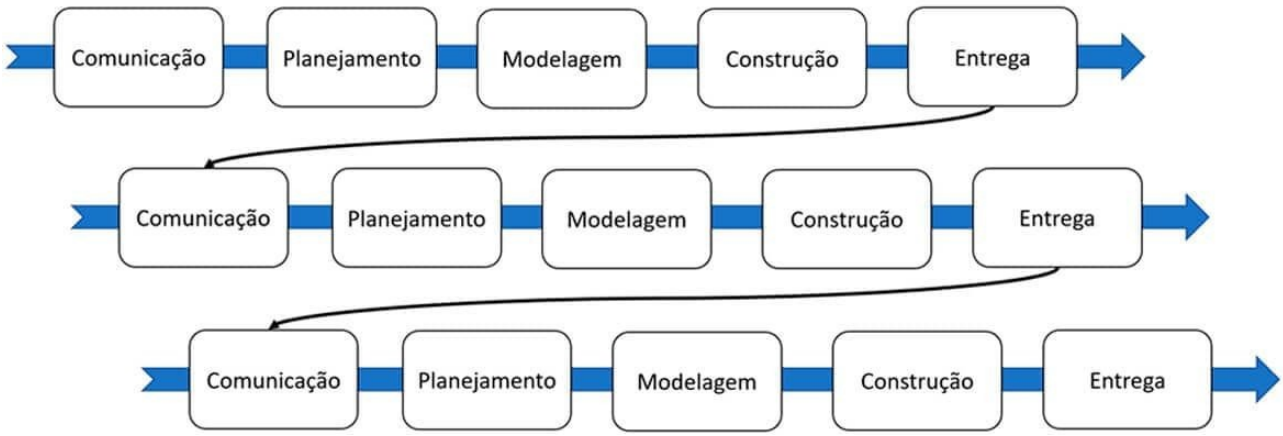
- **Fluxo sequencial inviável:** Projetos reais raramente seguem uma sequência rígida devido à volatilidade dos requisitos.
- **Requisitos detalhados:** Supõe que o cliente consiga definir todas as necessidades antecipadamente, aumentando a possibilidade de propagação de erros.
- **Inflexibilidade:** Permite a liberação de uma versão do software somente ao final do projeto.

Dica: O modelo em cascata é adequado para projetos com requisitos fixos e quando o fluxo de trabalho pode ser realizado de forma sequencial até o encerramento.

Os modelos incremental e evolucionário dividem o desenvolvimento de software em ciclos iterativos, onde cada ciclo engloba todas as atividades típicas e entrega um conjunto de funcionalidades ao usuário final. Cada iteração abrange um subconjunto de requisitos, permitindo que, mesmo com restrições de prazo e alta volatilidade dos requisitos, seja gerada uma versão que agrega valor imediato.

Diferença principal:

- **Modelo incremental:** Entrega inicialmente um produto essencial com iterações bem definidas que adicionam funcionalidades de forma progressiva.
- **Modelo evolucionário:** Baseia-se em um entendimento parcial dos requisitos, permitindo que cada nova iteração aperfeiçoe a compreensão do problema e gere versões atualizadas conforme os requisitos evoluem.



Resumo:

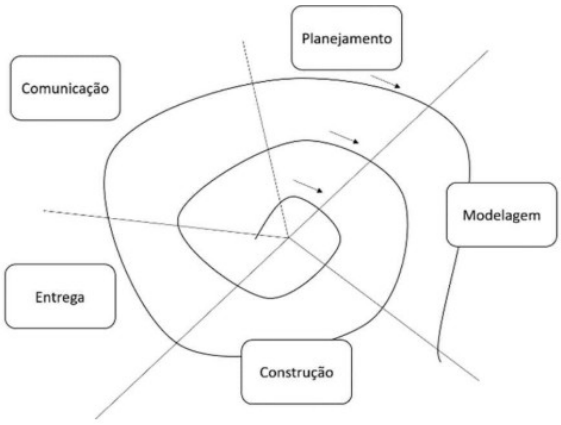
A prototipação em software utiliza o mesmo princípio de construir um protótipo em escala reduzida — como uma maquete de uma barragem — para avaliar esforços e testar hipóteses. No desenvolvimento de software, essa técnica permite que o desenvolvedor interaja diretamente com o

usuário, validando requisitos de forma rápida e iterativa. Por exemplo, se um requisito não funcional exige replicação de dados via tecnologia desconhecida, a prototipação possibilita tratar essa questão sem aguardar o desenvolvimento completo. O protótipo gerado pode ser descartado se servir apenas para validação ou refinado e integrado ao sistema, conforme as peculiaridades do software. Essa abordagem é especialmente útil para solucionar problemas de baixa complexidade.

O Modelo Espiral, proposto por Barry Boehm em 1988, estrutura o desenvolvimento de software em fases representadas por loops na espiral, onde cada quadrante corresponde a uma etapa. O loop mais interno trata da viabilidade do sistema, seguido pelo levantamento de requisitos e outras fases.

As etapas são:

- **Analysis (Primeira etapa):** Definição de objetivos, alternativas, riscos e restrições, com o comprometimento dos envolvidos e a formulação de uma estratégia.
- **Evaluation (Segunda etapa):** Avaliação de alternativas, identificação e solução de riscos, utilizando a prototipação para tratar ameaças.
- **Development (Terceira etapa):** Desenvolvimento do produto.
- **Planning (Quarta etapa):** Avaliação do produto e planejamento para iniciar um novo ciclo.



A abstração permite visualizar o Modelo Espiral como um metamodelo, conforme exemplificado por Pressman (2016), onde a espiral é dividida nas atividades genéricas ilustradas na Figura 1.

Neste módulo, destacou-se a importância dos modelos prescritivos na Engenharia de Software, cuja base é a camada de processos. Esses modelos definem as atividades do projeto e o fluxo de trabalho, incluindo metodologias, tarefas, artefatos, garantia de qualidade e mecanismos de controle de mudanças. O modelo em cascata organiza as tarefas de forma sequencial, enquanto os modelos incremental e evolucionário permitem o versionamento iterativo do software. Além disso, a prototipação é usada para validar requisitos, e o modelo espiral promove um desenvolvimento evolucionário do produto.

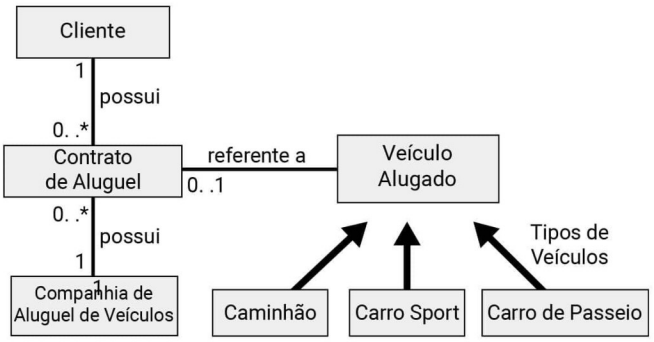
A **Unified Modeling Language (UML)** é uma linguagem padrão para modelar artefatos no desenvolvimento orientado a objetos, essencial para o entendimento do Processo Unificado. Surgiu em 1996 e foi aprovada pelo OMG em 1997, unificando diversas abordagens orientadas a objetos.

Antes da UML, o paradigma estruturado predominava, utilizando funções e diagramas de fluxo de dados. Com o aumento da complexidade do software, a orientação a objetos passou a ser preferida, pois melhora o reuso e a manutenibilidade, tratando o objeto — composto de atributos e métodos encapsulados — como unidade básica.

A UML é uma linguagem visual independente de linguagens de programação e processos de desenvolvimento, permitindo a criação de diagramas que podem ser complementados por descrições textuais.

Uma linguagem permite a comunicação e possui **sintaxe** (estrutura de escrita) e **semântica** (significado). A sintaxe define padrões, como a estrutura de uma frase na linguagem natural (**sujeito + verbo + predicado**) ou a organização de uma classe na UML (**nome da classe, atributos e métodos**).

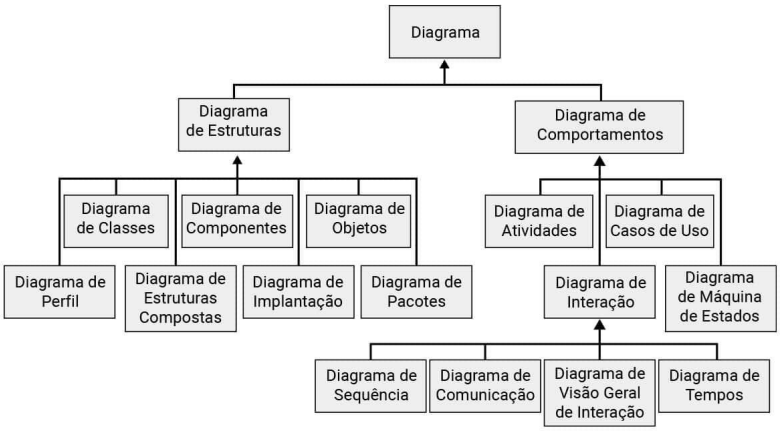
A **semântica** emerge quando um diagrama é construído com múltiplas classes, formando uma **rede semântica**, onde objetos e associações representam uma abstração da estrutura estática de um domínio.



No desenvolvimento de software, um modelo facilita a comunicação entre engenheiros de software, usuários, gerentes e programadores. Um diagrama de classes, por exemplo, pode ser compreendido por qualquer profissional familiarizado com a **orientação a objetos**.

A UML define dois principais tipos de diagramas:

- **Diagramas comportamentais** – Representam a abstração funcional do software.
- **Diagramas estruturais** – Modelam as estruturas de dados e componentes do sistema.

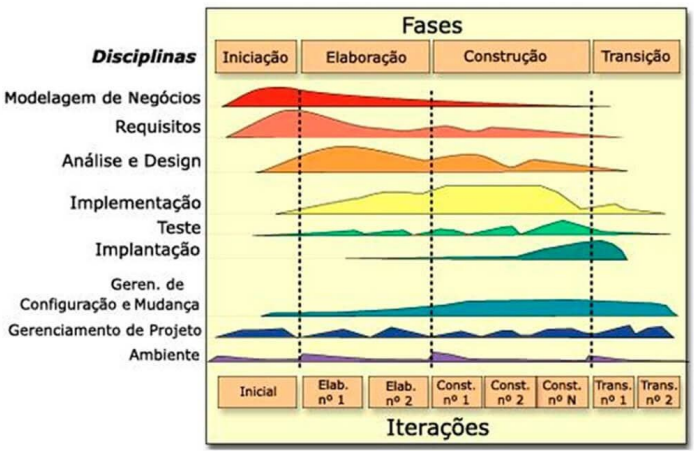


O **Processo Unificado (RUP)**, criado pela Rational Software Corporation (posteriormente adquirida pela IBM), é um processo formal de desenvolvimento de software, aplicável a problemas complexos e adaptável a diferentes escalas de projeto.

O RUP segue três princípios fundamentais:

1. **Guiado por casos de uso** – O desenvolvimento é planejado com base nos serviços esperados pelos usuários, iniciando pela **abstração funcional**.
2. **Centrado em arquitetura** – A arquitetura do software mantém o foco em compreensibilidade, adaptabilidade e reutilização, abrangendo decisões estruturais e comportamentais.
3. **Iterativo e incremental** – O software é desenvolvido em ciclos evolutivos, incorporando novos requisitos e permitindo versionamento contínuo, sem exigir entregas a cada iteração.

O **aspecto bidimensional** do RUP inclui:



- **Eixo horizontal** – Representa as fases do projeto, que estruturam o tempo e podem servir como marcos.
- **Eixo vertical** – Representa as atividades iterativas do processo.

O **Processo Unificado (RUP)** possui quatro fases principais:

Iniciação (Concepção)

Define uma visão geral do sistema e modela os processos de negócio para compreensão do domínio. Identificam-se **requisitos funcionais e não funcionais**, modelam-se **casos de uso críticos e classes iniciais** e delimita-se o **escopo do projeto**. Também são analisados **riscos, viabilidade técnica e financeira**. Caso aprovado, inicia-se o planejamento da próxima fase.

Elaboração

Refina-se a análise dos modelos de **casos de uso e classes**, incluindo a modelagem de interação e um **protótipo da interface**. A arquitetura do software é detalhada em cinco visões:

- **Modelo de casos de uso**
- **Modelo de análise**
- **Modelo de projeto**
- **Modelo de implementação**
- **Modelo de implantação**

Minimizam-se riscos e planeja-se a fase de construção.

Construção

Segue-se o modelo arquitetural para desenvolver **componentes reutilizáveis** e tornar os casos de uso operacionais. Concluem-se os modelos, realiza-se a **codificação e testes (unitários, integração, validação e beta)** para assegurar qualidade.

Transição

Implanta-se o software em produção. Os usuários realizam testes beta em ambiente real, permitindo **correção de defeitos e ajustes**. Também são elaborados **manuais, treinamentos, procedimentos de instalação e suporte**. Ao fim, o software torna-se uma versão de produção utilizável.

O **fluxo de trabalho** no RUP compreende atividades distribuídas por todas as fases do projeto. Cada atividade tem maior intensidade em fases específicas.

- **Modelagem de negócio e requisitos** – Predominam na **concepção**, onde se definem requisitos de alto nível.
- **Análise e design** – Iniciam-se na **concepção**, mas continuam na **construção**, refinando modelos.
- **Implementação na elaboração** – Permite validar requisitos por meio da **prototipação**, incluindo codificação.

O RUP também define três atividades de apoio:

1. **Ambiente** – Configuração de processos e ferramentas de suporte.

2. **Gerenciamento de configuração e mudança** – Controle de versão e rastreabilidade de artefatos.
3. **Gerência de projeto** – Baseada no **PMBOK**, enfatiza boas práticas na gestão de projetos.

Este módulo abordou a importância do **Processo Unificado (RUP)** no desenvolvimento de software. Inicialmente, foi apresentada a **UML**, que padroniza artefatos conforme a orientação a objetos.

O **RUP** é um modelo formal baseado em três princípios:

- **Casos de uso** – Foco nos serviços esperados pelos usuários.
- **Arquitetura** – Organização estrutural do sistema.
- **Iterativo e incremental** – Desenvolvimento evolutivo e adaptável.

O RUP é indicado para problemas complexos, mas pode ser ajustado a diferentes níveis de complexidade.

Até os anos 1990, o desenvolvimento de software seguia processos prescritivos, com planejamento detalhado, garantia de qualidade, modelagem e documentação rigorosa, adequados para sistemas complexos. No entanto, esse modelo nem sempre era viável para projetos menos complexos.

Em 2001, um grupo de especialistas lançou o **Manifesto Ágil**, que prioriza:

- **Indivíduos e interações** sobre processos e ferramentas.
- **Software funcional** sobre documentação extensa.
- **Colaboração com o cliente** sobre negociações contratuais.
- **Adaptação a mudanças** sobre seguir um plano rígido.

Os métodos ágeis enfatizam **entregas incrementais**, comunicação constante com o cliente e rápida validação do software, permitindo ajustes contínuos nos requisitos.

A **Extreme Programming (XP)** é um método ágil de desenvolvimento de software proposto por **Kent Beck (1999)**. Baseia-se nos valores de **comunicação, simplicidade, feedback, coragem e respeito**.

- **Comunicação:** minimiza documentação formal, priorizando interação entre equipe e clientes.
- **Simplicidade:** adota soluções simples para reduzir o custo de mudanças futuras.
- **Feedback:** ocorre por meio de testes e validação dos clientes, permitindo correções.
- **Coragem:** assume erros como naturais, confiando nos mecanismos de prevenção.
- **Respeito:** fundamental para o sucesso do projeto.

A XP também segue cinco princípios: **feedback rápido, simplicidade, mudanças incrementais, adaptação a mudanças e trabalho de qualidade.**

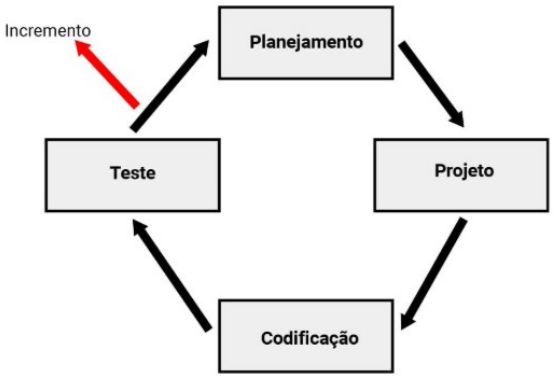
A **Extreme Programming (XP)** adota **12 práticas** alinhadas aos seus valores e princípios:

- **Jogo de Planejamento:** requisitos registrados em cartões ("histórias de usuários"), priorizados pelos clientes, com estimativas da equipe.

- **Pequenas Releases:** entregas frequentes com os requisitos mais importantes, facilitando feedback e correções.
- **Metáfora:** comunicação eficaz com o cliente, traduzindo conceitos para sua realidade.
- **Projeto Simples:** desenvolvimento apenas do necessário, com código bem estruturado e mínimo de classes e métodos.
- **Cliente no Local:** cliente ou representante participando ativamente para esclarecer dúvidas.
- **Semana de 40 Horas:** ritmo sustentável de trabalho, com horas extras limitadas.
- **Programação Pareada:** dois programadores por máquina, um codifica, o outro revisa, reduzindo erros.
- **Propriedade Coletiva:** qualquer desenvolvedor pode alterar o código, garantindo conhecimento compartilhado.
- **Padronização do Código:** regras definidas para uniformidade e compreensão do código.
- **Desenvolvimento Orientado a Testes:** testes unitários antes da implementação ("test first"), testes de integração e validação baseados nas histórias de usuários.
- **Refatoração:** melhoria contínua do código sem alterar funcionalidade.
- **Integração Contínua:** nova funcionalidade integrada ao sistema imediatamente, evitando erros de compatibilidade.

Processo XP

A XP adota o paradigma orientado a objetos, aplicando seus valores, princípios e práticas em um processo com quatro atividades metodológicas: planejamento, projeto, codificação e testes. A Figura 10 ilustra essas atividades metodológicas propostas por Pressman (2016).



Resumo:

O **Extreme Programming (XP)** segue quatro etapas principais:

- **Planejamento:** clientes definem requisitos em "histórias de usuários", priorizando-as conforme o valor ao negócio. A equipe divide essas histórias em tarefas e estima o esforço necessário. Histórias muito grandes são fragmentadas. Após a primeira release, mede-se a velocidade do projeto para futuras estimativas.
- **Projeto:** aplica a máxima "Não complique!". Utiliza **cartões CRC** para modelagem de classes e recomenda prototipação para validar requisitos complexos. A **refatoração** melhora a estrutura do código sem alterar sua funcionalidade.
- **Codificação:** dois programadores trabalham juntos em um computador (**programação pareada**), garantindo qualidade e revisão imediata. O código gerado é integrado continuamente, permitindo rápida identificação de erros.
- **Teste:** antes da codificação, a equipe elabora testes unitários, preferencialmente automatizados, facilitando **testes de regressão**. A **integração contínua** assegura testes frequentes, evitando erros propagados. Clientes elaboram **testes de aceitação** para validar se o sistema atende às necessidades do usuário.

Resumo:

Este módulo abordou as metodologias ágeis, com foco na **Extreme Programming (XP)**. O **Manifesto Ágil** surgiu como resposta ao formalismo excessivo dos processos prescritivos dos anos 1990, estabelecendo quatro valores:

1. **Indivíduos e interações** acima de processos e ferramentas.
2. **Software funcional** acima de documentação extensa.
3. **Colaboração com o cliente** acima de negociações contratuais.
4. **Adaptação a mudanças** acima de seguir um plano.

A maioria dos métodos ágeis prioriza software funcional, processos iterativos e requisitos voláteis.

O **processo XP** segue quatro atividades principais:

- **Planejamento:** baseado em **histórias de usuários**.
- **Projeto:** usa **cartões CRC**, com prototipação e refatoração recomendadas.
- **Codificação:** feita em **pares de programadores**, garantindo revisão contínua.
- **Teste:** os testes são escritos **antes da codificação** e aplicados de forma automatizada, abrangendo **testes unitários, de integração e aceitação**.

O **Scrum** surgiu como um estilo de **gerenciamento de produtos**, inspirado no rugby, e foi adaptado para o desenvolvimento de software por **Jeffrey Sutherland e Ken Schwaber** nos anos 1990. Publicado em **1995**, o Scrum passou a ser amplamente aplicado em projetos de software, seguindo os **valores e princípios do Manifesto Ágil**.

Essa metodologia é ideal para projetos **complexos**, onde os requisitos não podem ser completamente definidos no início. Embora alguns autores considerem o Scrum um **processo ágil de desenvolvimento de software**, ele também é usado em outras áreas de **gerenciamento de projetos**.

O Scrum se baseia em três pilares:

- **Transparência:** A equipe deve compartilhar um **entendimento comum do processo**, utilizando uma linguagem padronizada.
- **Inspeção:** Os artefatos e o progresso devem ser **inspecionados continuamente** para identificar problemas, sem prejudicar a execução das tarefas.
- **Adaptação:** O software e o processo devem ser **ajustados rapidamente** para lidar com mudanças nos requisitos ou falhas no desenvolvimento.

No **Scrum**, a equipe é composta por três papéis principais:

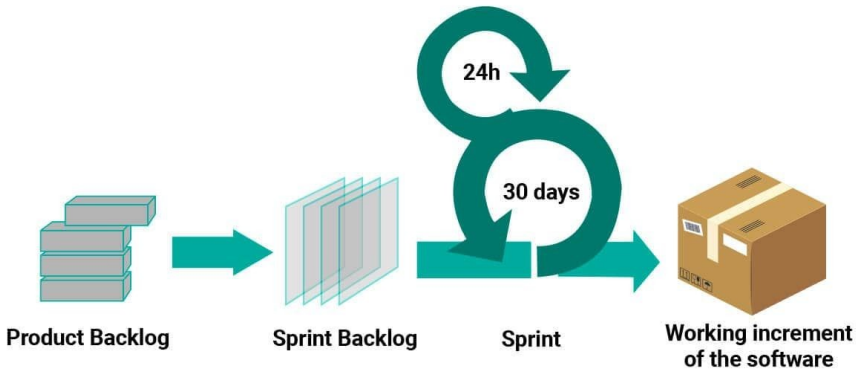
- **Product Owner:** Representa o cliente, define requisitos e funcionalidades, gerencia mudanças e mantém a visão do projeto, esclarecendo dúvidas.
- **Scrum Master:** Garante o cumprimento das regras do Scrum, atua como facilitador e protege a equipe de interferências externas.
- **Scrum Team:** Equipe multidisciplinar com especialistas necessários para o desenvolvimento do produto sem depender de terceiros (ex.: programadores, DBAs, web designers).

Os **artefatos Scrum** garantem o sucesso das equipes na entrega de incrementos. Os principais são:

- **Product Backlog:** Lista dinâmica de requisitos e funcionalidades do produto, gerenciada pelo Product Owner, evoluindo conforme o projeto avança.
- **Sprint Backlog:** Subconjunto do Product Backlog selecionado para uma Sprint, incluindo funcionalidades e plano de entrega. A equipe de desenvolvimento pode ajustá-lo conforme necessário.

Os **eventos Scrum** organizam inspeções e iterações de forma ágil, com duração pré-definida (timebox). Eles minimizam a necessidade de reuniões externas ao Scrum. Os principais eventos são:

- **Sprint:** Iteração de até um mês, resultando em um incremento de produto utilizável.
- **Reunião de Planejamento do Sprint:** Planeja o trabalho do Sprint, com duração máxima de 8h para um Sprint de um mês, dividido em duas partes: o que e como será entregue.
- **Reunião Diária do Sprint (Daily Scrum):** Reunião diária de 15 minutos para ajustar as atividades e planejar as próximas 24 horas.
- **Revisão do Sprint:** Realizada no final do Sprint, inspeciona o incremento e adapta o Product Backlog, com duração máxima de 4h para sprints de um mês.
- **Retrospectiva do Sprint:** Reunião formal de 3h para avaliar o último Sprint e criar um plano de melhorias para o próximo.



O processo Scrum segue o fluxo abaixo:

1. **Reunião de Planejamento do Sprint:** Product Owner, Scrum Master e Scrum Team selecionam e priorizam os requisitos do Product Backlog para o Sprint, decompõem em tarefas e criam o Sprint Backlog.
2. **Desenvolvimento do Sprint:** A Scrum Team começa o desenvolvimento, com reuniões diárias (Daily Scrum) para revisar o progresso e definir prioridades do dia, conduzidas pelo Scrum Master.
3. **Reunião de Revisão do Sprint:** Ao final do Sprint, o Product Backlog é revisado, ajustando requisitos conforme necessário. Participam o Product Owner, Scrum Master e Scrum Team.
4. **Reunião de Retrospectiva do Sprint:** A equipe analisa as lições aprendidas e identifica melhorias para o próximo Sprint.

Resumo:

O Processo Unificado Ágil (AUP) é uma versão simplificada do RUP, com princípios de simplicidade, agilidade, foco em atividades de alto valor, e personalização. Ele adota uma abordagem serial para fases amplas e iterativa para atividades específicas.

Fases do AUP:

- **Concepção:** Identificação do escopo, arquitetura e viabilidade.
- **Elaboração:** Validação da arquitetura do sistema.
- **Construção:** Desenvolvimento iterativo e incremental do software.
- **Transição:** Validação e implantação do sistema.

Atividades Iterativas:

- **Modelagem:** Entendimento do negócio e solução, utilizando UML simples.
- **Implementação:** Transformação dos modelos em código e testes básicos.
- **Testes:** Garantia de qualidade, detecção de defeitos e validação de requisitos.
- **Implantação:** Planejamento e entrega do software com feedback dos usuários.
- **Gerenciamento de Configuração:** Controle de versões e alterações dos artefatos.
- **Gestão de Projetos:** Planejamento e monitoramento das atividades do projeto.
- **Ambiente:** Suporte às atividades com padrões e ferramentas.

Este módulo aborda os processos ágeis Scrum e AUP. O Scrum é uma metodologia amplamente usada em projetos de software, baseada em iterações chamadas de sprints, com entregas de software operacional ao final de cada ciclo. O AUP, derivado do RUP, é uma versão simplificada que segue os princípios ágeis. As fases do projeto no AUP (concepção, elaboração, construção, e transição) são as mesmas do RUP, mas as atividades iterativas incluem modelagem, implementação, testes, implantação, gerenciamento de configurações, gestão de projetos e ambiente.

A Engenharia de Software tem como principal produto o software, cada vez mais presente em nossas vidas. A "Crise do Software", mencionada por Dijkstra (1972), decorre do aumento exponencial da capacidade computacional, tornando a programação um desafio crescente. O avanço do hardware possibilitou softwares mais complexos, impactando a indústria. Um exemplo disso é a transição do paradigma estruturado para o orientado a objetos, que melhora reuso e manutenibilidade. A complexidade, evidente em softwares críticos como os de aviação, exige metodologias para decomposição sistemática de problemas, papel fundamental da engenharia de software.

A qualidade assegura que o software atenda às expectativas do usuário. O processo define as etapas do desenvolvimento, enquanto os métodos determinam os artefatos gerados conforme a modelagem adotada. As ferramentas, como CASE, auxiliam na implementação. A camada fundamental é o processo, pois a qualidade depende dele. Métodos e ferramentas contribuem para garantir a qualidade do software.

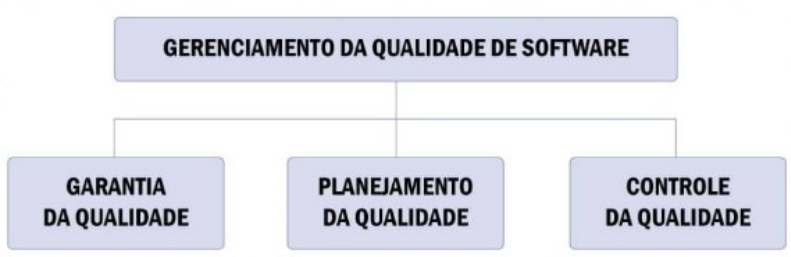
A engenharia de software é estruturada em quatro camadas interdependentes:

- **Qualidade:** Garante que os requisitos atendam às expectativas dos usuários.
- **Processo:** Define as etapas do desenvolvimento do software, sendo a base para a qualidade.

- **Métodos:** Determinam os artefatos gerados conforme a modelagem adotada.
- **Ferramentas:** Incluem o uso de ferramentas CASE para auxiliar no desenvolvimento.

A qualidade depende da existência de um processo bem definido. Métodos e ferramentas auxiliam na obtenção desse objetivo, garantindo maior controle e segurança no desenvolvimento de software.

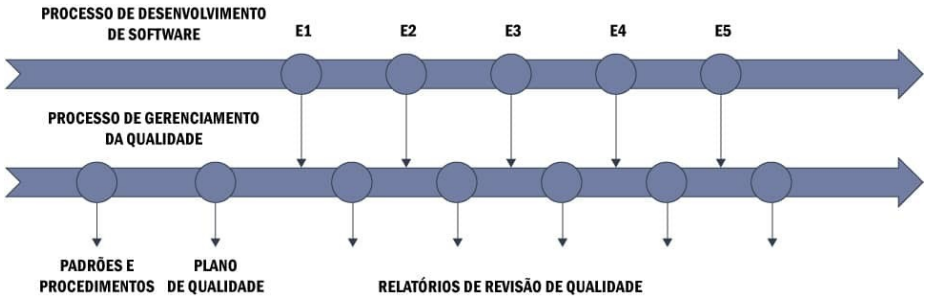
Segundo Pressman (2016), qualidade de software envolve uma gestão eficaz para criar produtos úteis e com valor mensurável para usuários e desenvolvedores. Isso inclui atendimento aos requisitos, ausência de defeitos e benefícios tanto para os usuários quanto para os engenheiros de software, reduzindo manutenção e permitindo foco em novas aplicações.



O gerenciamento da qualidade de software é dividido em três atividades:

- **Garantia de qualidade:** Define procedimentos e padrões organizacionais para garantir qualidade no software.
- **Planejamento de qualidade:** Adapta os padrões e procedimentos a projetos específicos.
- **Controle de qualidade:** Assegura que a equipe siga os padrões estabelecidos.

Sommerville (2019) propõe que a garantia de qualidade seja independente do desenvolvimento para evitar comprometimento da qualidade em situações como atrasos no cronograma.



Processo: O desenvolvimento de software exige decisões em todas as etapas. Garantir a qualidade em cada etapa é essencial para produzir software de qualidade.

Produto: Softwares mal testados podem causar grandes prejuízos, como compras desnecessárias e indicadores falsos, além de comprometer decisões estratégicas de gerentes e executivos. A qualidade das decisões depende diretamente da qualidade das informações fornecidas pelos softwares organizacionais.

O processo é uma sequência de etapas para gerar um produto, no caso, o software. Ele facilita o gerenciamento da complexidade, pois envolve diferentes profissionais, como analistas,

programadores e gerentes. A Engenharia de Software oferece diversos modelos de processos, e a escolha depende da complexidade do sistema—quanto maior, mais formal deve ser o processo.

Segundo Pressman (2016), uma metodologia genérica de desenvolvimento inclui cinco atividades:

- **Comunicação:** entendimento do problema, definição de objetivos e requisitos.
- **Planejamento:** gerenciamento do projeto.
- **Modelagem:** criação de modelos, como casos de uso.
- **Construção:** codificação e testes conforme o planejamento.
- **Entrega:** disponibilização do produto final.

Garantir a qualidade do software exige uma cultura de não tolerância a erros. O processo deve incluir mecanismos para detectar defeitos nos artefatos gerados (documentos de requisitos, modelos e código), evitando sua propagação.

Embora testes sejam geralmente associados ao produto final, eles podem ser aplicados a todas as etapas do desenvolvimento, sendo chamados de testes de verificação ou testes estáticos, avaliando artefatos como diagramas de classes e casos de uso.

Produzir software com qualidade requer um processo bem definido, pois não se pode garantir qualidade sem um desenvolvimento estruturado.

Pressman (2016) descreve os fatores de qualidade do software de McCall, que categorizam aspectos que afetam a qualidade do software nas etapas de revisão, transição e operação. Os fatores incluem:

- **Correção:** satisfação da especificação e necessidades do cliente.
- **Confiabilidade:** capacidade do software de realizar funções com precisão.
- **Eficiência:** uso de recursos e código para desempenhar funções.
- **Integridade:** controle de acesso não autorizado ao software e dados.
- **Usabilidade:** esforço necessário para aprender e operar o software.
- **Facilidade de manutenção:** esforço necessário para corrigir erros.
- **Flexibilidade:** esforço necessário para modificar o software em operação.
- **Testabilidade:** esforço necessário para testar o software.
- **Portabilidade:** esforço necessário para transferir o software para outros ambientes.
- **Reusabilidade:** capacidade do software ou suas partes serem reutilizadas em outras aplicações.
- **Interoperabilidade:** esforço necessário para integrar o sistema com outros.

Esses fatores permitem avaliar de forma sólida a qualidade de um software.

A qualidade do produto é garantida por um plano de testes executado durante a codificação, chamados testes de validação. O processo inclui:

1. Validação da estrutura interna da unidade de software e sua aderência aos requisitos.
2. Validação da integração com unidades anteriores, focando nas interfaces dos componentes.
3. Validação funcional e não funcional do software, com participação dos usuários.

Esses testes, chamados testes dinâmicos, podem ser automatizados, permitindo a criação de ambientes que simulam diversos cenários de uso. Quanto mais cenários simulados, maior o nível de validação e qualidade do software.

O custo da qualidade é dividido em custo da conformidade e custo da não conformidade.

- **Custo da conformidade:** inclui o custo da prevenção e da detecção de defeitos. A prevenção envolve atividades como treinamentos e metodologias para evitar defeitos. A detecção abrange atividades como revisões, testes e auditorias para identificar problemas.
- **Custo da não conformidade:** refere-se aos custos relacionados a erros no processo de desenvolvimento, como retrabalhos, ações corretivas, atrasos e perdas financeiras. Inclui custos de falhas internas (erros antes da entrega) e falhas externas (erros após a entrega ao cliente).

Quanto mais cedo ocorrer a identificação de um defeito, menor o custo.

Este módulo destaca a importância da qualidade na engenharia de software, dividida em duas dimensões: processo e produto. A qualidade do processo exige a definição de um processo de desenvolvimento adequado, com testes de verificação para evitar defeitos nas etapas subsequentes. A qualidade do produto é garantida na codificação por meio de testes de validação, que avaliam desde a estrutura interna até as funcionalidades, com a participação dos usuários.

A garantia de qualidade envolve custos, tanto de conformidade (garantindo qualidade) quanto de não conformidade (falta de qualidade).

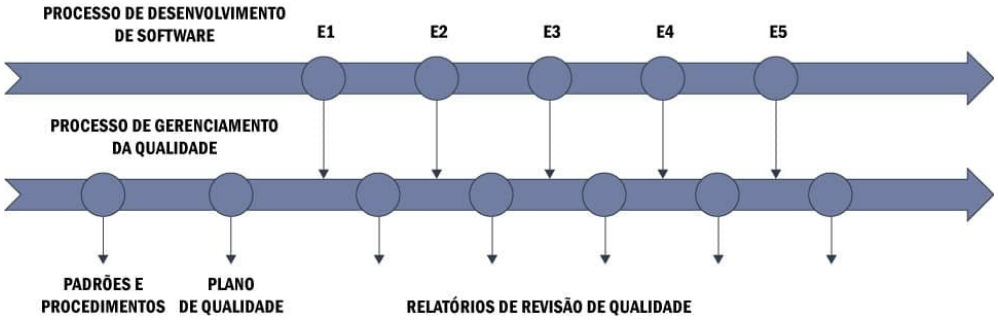
O Sistema de Garantia da Qualidade de Software (SQA) envolve uma estrutura organizacional com responsabilidades, procedimentos, processos e recursos para garantir que os produtos atendam às especificações e expectativas do cliente. Esse processo inclui a definição e seleção de padrões, ferramentas e métodos de apoio.

A equipe de qualidade deve avaliar o software do ponto de vista do cliente, garantindo a conformidade com os requisitos. As principais atividades incluem:

- **Padrões:** Garantir a aplicação de padrões, como os da ISO.
- **Revisões e auditorias:** Identificar erros e verificar a conformidade com os processos.
- **Testes:** Garantir a execução dos testes para detectar erros.
- **Coleta e análise de erros:** Analisar defeitos para melhorar os procedimentos.
- **Gerenciamento de mudanças:** Estabelecer processos para lidar com mudanças.
- **Educação:** Promover programas educacionais para a equipe de desenvolvimento.
- **Gerência dos fornecedores:** Incorporar requisitos de qualidade nos contratos com fornecedores.
- **Administração da segurança:** Proteger dados e garantir a integridade do software.
- **Proteção:** Avaliar o impacto de defeitos ocultos e implementar ações de mitigação.

- **Administração de riscos:** Garantir que a gestão de riscos seja adequada e que planos de contingência existam.

Essas atividades garantem a qualidade e a conformidade do software com as necessidades e expectativas do cliente.



Os padrões de software incluem normas de documentação e codificação, como cabeçalhos de comentários e definições de uso de linguagens de programação. Eles também definem processos, como o de engenharia de requisitos, e os documentos a serem gerados durante o desenvolvimento.

Esses padrões:

- Incluem melhores práticas para evitar erros repetidos.
- Fornecem um framework para garantir que a qualidade seja mantida.
- Garantem a continuidade em caso de mudanças na equipe.

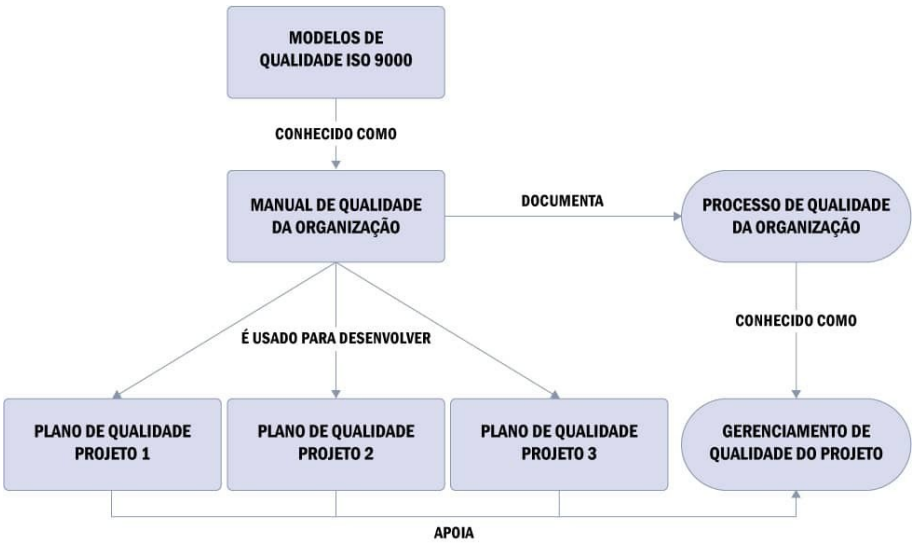
A equipe de qualidade deve criar padrões baseados em modelos nacionais e internacionais, envolvendo a equipe de desenvolvimento, revisando constantemente os padrões e utilizando ferramentas CASE. O processo de desenvolvimento deve ser adaptado à complexidade do problema, e os padrões podem ser modificados ou novos padrões podem ser adicionados conforme necessário, com a aprovação da equipe de qualidade.

A **ISO 9000** é um conjunto internacional de padrões para sistemas de gerenciamento de qualidade aplicáveis a diversos setores, incluindo manufatura e serviços. A ISO 9001 não é específica para o desenvolvimento de software, mas a ISO 9000-3 adapta seus princípios ao contexto de software.

Os requisitos da ISO 9001 incluem:

- Responsabilidade administrativa
- Sistema de qualidade
- Controle de projeto e de processos
- Identificação de produtos e rastreabilidade
- Inspeções, testes e ações corretivas
- Auditorias e treinamentos
- Manutenção e uso de técnicas estatísticas

Uma organização de software certificada pela ISO 9001 deve estabelecer e demonstrar políticas e procedimentos para cumprir esses requisitos. O manual de qualidade (ou Plano SQA) aplica os princípios da ISO 9000 e adapta o processo de qualidade para cada projeto de software.



O padrão ISO 9126 define seis atributos fundamentais de qualidade para software:

1. **Funcionalidade**: Satisfação das necessidades com subatributos como adequabilidade e segurança.
2. **Confiabilidade**: Tempo de disponibilidade, incluindo maturidade e tolerância a falhas.
3. **Usabilidade**: Facilidade de uso, abrangendo compreensão e aprendizado.
4. **Eficiência**: Uso otimizado dos recursos, com foco em tempo e recursos.
5. **Facilidade de manutenção**: Facilidade para corrigir o software, considerando análise e testabilidade.
6. **Portabilidade**: Facilidade de transição entre ambientes, incluindo adaptabilidade e conformidade.

O gerenciamento da qualidade no PMBOK (2017) abrange três processos principais:

1. **Planejamento**: Identificação de requisitos e padrões de qualidade, com a entrega do plano de gerenciamento da qualidade.
2. **Execução**: Implementação das atividades de qualidade conforme o plano de gerenciamento, alinhando as políticas organizacionais.
3. **Monitoramento e Controle**: Avaliação do desempenho das atividades de qualidade para garantir que as entregas atendam às expectativas do cliente.

A abordagem do PMBOK enfatiza a prevenção sobre a inspeção, pois prevenir defeitos é mais barato que corrigir erros identificados.

Existem cinco níveis de gerenciamento da qualidade, que vão desde a detecção e correção de defeitos até a criação de uma cultura organizacional comprometida com a qualidade.

Neste módulo, discutimos a importância da garantia da qualidade no desenvolvimento de software. Um sistema de garantia da qualidade define a estrutura organizacional responsável por garantir que os produtos atendam às expectativas do cliente. O processo de garantia enfatiza a adoção e cumprimento de padrões, como os da ISO, aplicados ao processo e produto de software. A implementação de padrões gera manuais de qualidade, que definem o processo de qualidade da organização, adaptado para cada projeto conforme sua complexidade. O PMBOK apresenta três processos relacionados ao gerenciamento da qualidade de projetos.

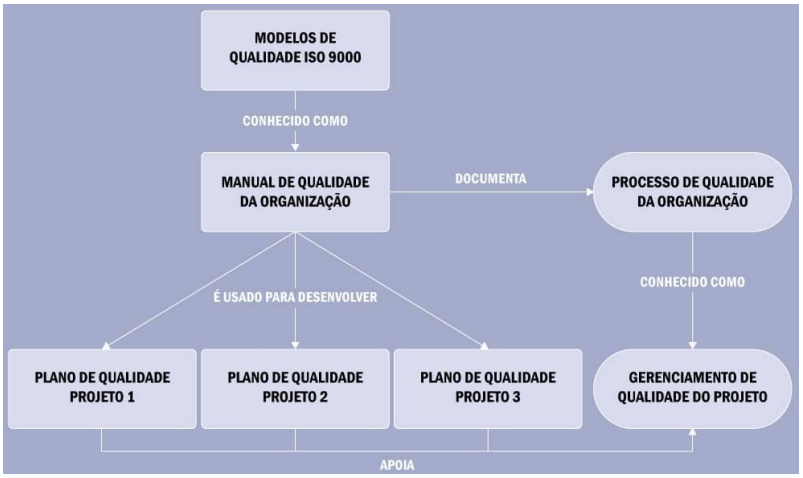
Planejamento de Qualidade

A equipe de SQA (garantia da qualidade) auxilia a equipe de software a entregar um produto de alta qualidade. O SEI recomenda ações de planejamento, supervisão, manutenção de registros, análise e relatórios sobre a garantia da qualidade. Essas ações, orientadas pelo plano de qualidade, incluem avaliações, auditorias, revisões, aplicação de padrões, acompanhamento de erros e feedback à equipe de software.

O plano de qualidade, desenvolvido pela equipe SQA ou de desenvolvimento, serve como um guia para as atividades de garantia da qualidade em cada projeto.

Cada projeto de software tem um plano de qualidade, criado a partir de um manual de qualidade ou política padrão da organização, adaptado conforme a complexidade e o processo de desenvolvimento. O IEEE publicou um padrão que recomenda uma estrutura para o plano, incluindo:

- Propósito e escopo.
- Descrição dos artefatos gerados.
- Padrões e práticas aplicados.
- Ações de garantia da qualidade, como revisões e auditorias.
- Ferramentas e métodos de apoio.
- Procedimentos de administração de configurações.
- Métodos para manutenção de registros de qualidade.
- Papéis e responsabilidades da organização relacionados à qualidade.



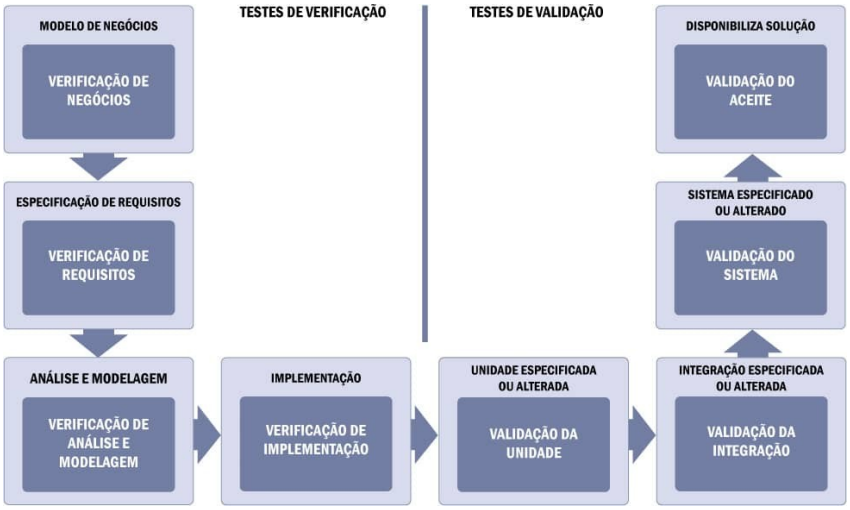
O PMBOK aborda a qualidade por meio da área de conhecimento "Gerenciamento da Qualidade do Projeto", que inclui três processos. O processo "Planejar o Gerenciamento da Qualidade", localizado no grupo de processos de planejamento, visa identificar os requisitos e padrões de qualidade do projeto, documentando como o projeto atenderá a esses requisitos.

O principal benefício é fornecer orientação sobre como a qualidade será gerenciada e verificada ao longo do projeto. As entradas para esse processo incluem o Termo de Abertura do Projeto, o Plano de Gerenciamento do Projeto, documentos do projeto, fatores ambientais da empresa e ativos de

processos organizacionais. As ferramentas e técnicas aplicadas são: opinião especializada, coleta de dados, análise de dados, tomada de decisões e reuniões.

As saídas incluem o Plano de Gerenciamento da Qualidade, métricas de qualidade, atualizações do plano de gerenciamento do projeto e documentos do projeto.

O teste é um processo sistemático planejado para identificar erros. O planejamento de testes e revisões, conforme Bartié (2002), organiza o processo de qualidade em software em duas fases: testes de verificação e testes de validação. A primeira fase, à esquerda do modelo "U", foca na qualidade do processo de desenvolvimento (testes de verificação), enquanto a segunda, à direita, foca na qualidade do produto (testes de validação).



Ambos os tipos de testes visam garantir que o software atenda aos requisitos e produza os resultados esperados, sendo complementares. Testes de verificação asseguram a qualidade do processo e os testes de validação garantem a qualidade do produto final.

Vou enviar o conteúdo de uma página de uma apostila de estudo. Preciso de um resumo claro e direto. Eu não quero que você simplifique as coisas ao ponto de deixar o mínimo de informações possível, não é esse o objetivo aqui, o objetivo é pegar o texto original, e simplificá-lo, tirando quaisquer redundâncias, repetições, ou palavras desnecessárias. Se algo pode ser dito eficientemente com 5 palavras, não há necessidade de usar 10. Texto a ser resumido: