ENGENHARIA DE SOFTWARE

Autor: Leonardo Costa Passos

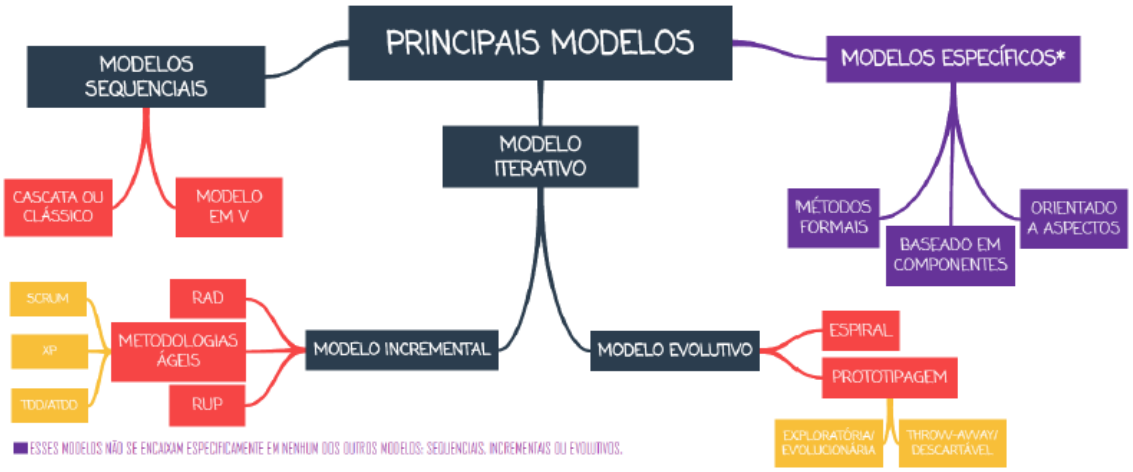
Resumo com o que há de mais importante da disciplina de Engenharia de Software para provas de Concursos Públicos com foco na banca CESGRANRIO.

Se o conteúdo for útil na sua jornada de estudos, você pode me agradecer fazendo um PIX de um valor que considere justo para a seguinte chave:

leonardx@gmail.com

Table of Contents

1. Modelo de Processo de Software

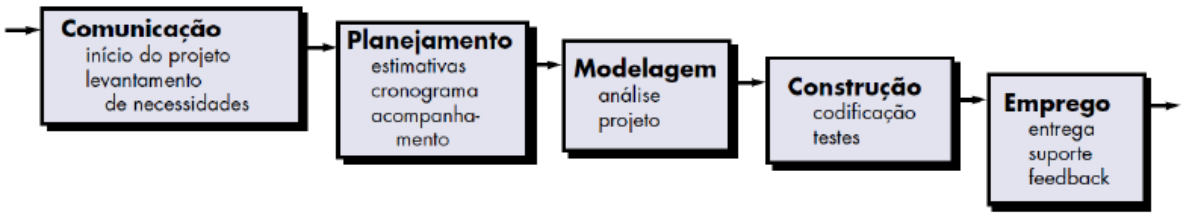


1.1. Modelos Sequenciais

- Desenvolvimento linear e ordenado
- Cada fase é concluída antes de passar para a próxima
- Menos flexíveis às mudanças
- Requisitos geralmente bem definidos no início

CICLOS DE VIDA PREDITIVOS: desde o início do projeto, o escopo, tempo e custo são determinados, e a equipe se esforçará para cumprir os compromissos estabelecidos em cada um destes fatores.

1.1.1. Modelo em Cascata

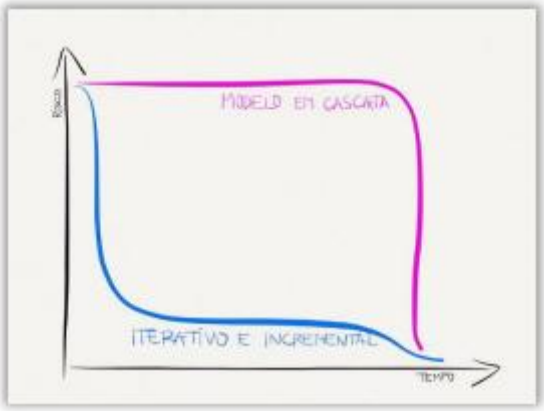


- Também chamado de clássico, ou linear, é o mais tradicional processo de desenvolvimento de software.
- Esse modelo sugere uma abordagem sequencial e sistemática para o desenvolvimento de software, aplicando as atividades de maneira linear. Em cada fase desenvolvem-se artefatos (produtos de software) que servem de base para as fases seguintes.
- Esse nome é devido ao encadeamento simples de uma fase com a outra.
- Na abordagem de execução de projetos em cascata, todas as etapas são seguidas de forma sequencial, dessa forma as fases não se sobrepõem, tampouco se entrelaçam. No modelo em cascata nós só avançamos para a próxima fase quando a anterior é concluída, em outras palavras, o estágio seguinte não deve ser iniciado até que a fase anterior seja concluída. Logo, voltar algumas etapas, dar saltos para frente ou sobrepor atividades não é permitido.

- O modelo em Cascata possui como vantagem principal a simplicidade para a sua aplicação e gerência.
- No Modelo em Cascata, uma fase só se inicia após o término e aprovação da fase anterior, isto é, há uma sequência de desenvolvimento do projeto.
- O modelo em cascata exige que todas as funcionalidades sejam bem definidas antes de iniciar o desenvolvimento.
- Em princípio, o modelo em cascata deve ser usado apenas quando os requisitos são bem compreendidos e pouco provavelmente venham a ser radicalmente alterados durante o desenvolvimento do sistema.
- No entanto, ele reflete o tipo de processo usado em outros projetos de engenharia. Como é mais fácil usar um modelo de gerenciamento comum para todo o projeto, processos de software baseados no modelo em cascata ainda são comumente utilizados.

No entanto, algumas desvantagens podem ser observadas:

- Projetos reais raramente seguem o fluxo sequencial que o modelo propõe. Embora o modelo linear possa conter iterações, ele o faz indiretamente. Como consequência, mudanças podem provocar confusão à medida que a equipe de projeto prossegue.
- Frequentemente, é difícil para o cliente estabelecer explicitamente todas as necessidades. O modelo cascata requer isso e tem dificuldade para adequar a incerteza natural que existe no início de muitos projetos.
- O cliente deve ter paciência. Uma versão operacional do(s) programa(s) não estará disponível antes de estarmos próximos do final do projeto. Um erro grave, se não detectado até o programa operacional ser revisto, pode ser desastroso.
- A natureza linear do ciclo de vida clássico conduz a "estados de bloqueio", nos quais alguns membros da equipe do projeto têm de aguardar outros completarem tarefas dependentes.
- Só é possível verificar se ocorreram erros nas fases finais, que é quando o sistema é efetivamente testado – isso gera grandes riscos! Em outros modelos, os riscos são reduzidos desde as primeiras fases do processo de desenvolvimento.

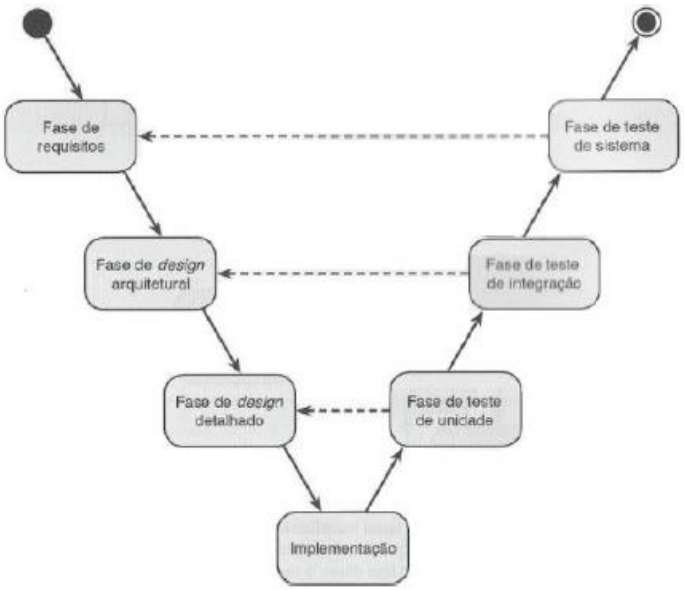


VANTAGENS	DESVANTAGENS
É simples de entender e fácil de aplicar, facilitando o planejamento.	Divisão inflexível do projeto em estágios distintos.
Fixa pontos específicos para a entrega de artefatos.	Dificuldade em incorporar mudanças de requisitos.
Funciona bem para equipes tecnicamente fracas.	Clientes só visualizam resultados próximos ao final do projeto.
É fácil de gerenciar, devido a sua rigidez.	Atrasa a redução de riscos.
Realiza documentação extensa por cada fase ou estágio.	Apenas a fase final produz um artefato de software entregável.
Possibilita boa aderência a outros modelos de processo.	Cliente deve saber todos os requisitos no início do projeto.
Funciona bem com projetos pequenos e com requisitos bem conhecidos.	Modelo inicial (Royce) não permitia feedback entre as fases do projeto.
	Pressupõe que os requisitos ficarão estáveis ao longo do tempo.
	Não funciona bem com projetos complexos e OO, apesar de compatível.

1.1.2. Modelo em V:

Variação na representação do Modelo em Cascata que descreve a relação entre ações de garantia da qualidade e as ações associadas à comunicação, modelagem e atividades de construção iniciais. À medida que a equipe de software desce em direção ao lado esquerdo do V, os requisitos básicos do problema são refinados em representações cada vez mais detalhadas e técnicas do problema e de sua solução.

Uma vez que o código tenha sido gerado, a equipe se desloca para cima, no lado direito do V, realizando basicamente uma série de testes (ações de garantia da qualidade) que validem cada um dos modelos criados à medida que a equipe se desloca para baixo, no lado esquerdo do V. Lembrando que os testes são executados do lado direito do Modelo em V, mas são planejados do lado esquerdo do Modelo em V.



1.2. Modelos Iterativos:

- Desenvolvimento em ciclos iterativos e incrementais
- Permite refinamento e adaptação contínua
- Mais flexíveis às mudanças nos requisitos e na tecnologia
- Requisitos podem ser incertos ou evoluir ao longo do tempo

- No Modelo Iterativo, caso haja cem requisitos, analisam-se, projetam-se, codificam-se, testam-se os cem requisitos, porém os requisitos são entregues incompletos e eu repito esse ciclo de refinamento até chegar ao produto final.

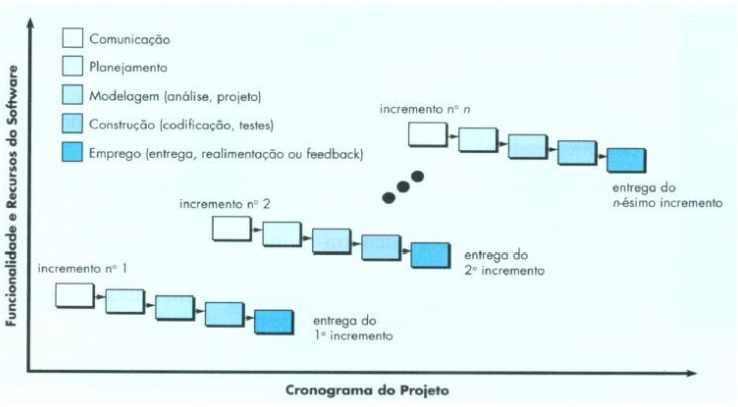
  No modelo iterativo, lança-se a versão 1.0, adicionam-se algumas funcionalidades; lança uma versão 2.0, adicionam-se mais algumas funcionalidades; e assim por diante.

1.2.1. Modelos Incrementais:

- Desenvolvimento em pequenos incrementos ou módulos
- Adição progressiva de funcionalidades ao sistema
- Maior controle e previsibilidade do projeto
- Facilita o gerenciamento de recursos e a entrega contínua
- Exemplos: RAD (Rapid Application Development), RUP (Rational Unified Process), Metodologias Ágeis (Scrum, Kanban, XP, etc.)

- No Modelo em Cascata, caso haja cem requisitos, analisam-se os cem requisitos, projetam-se os cem requisitos, codificam-se os cem requisitos, testam-se os cem requisitos, e assim por diante sequencialmente;
- No Modelo Incremental, caso haja cem requisitos, dividem-se os cem requisitos em vinte miniprojetos de cinco requisitos cada e utiliza-se o modelo em cascata para cada miniprojeto;

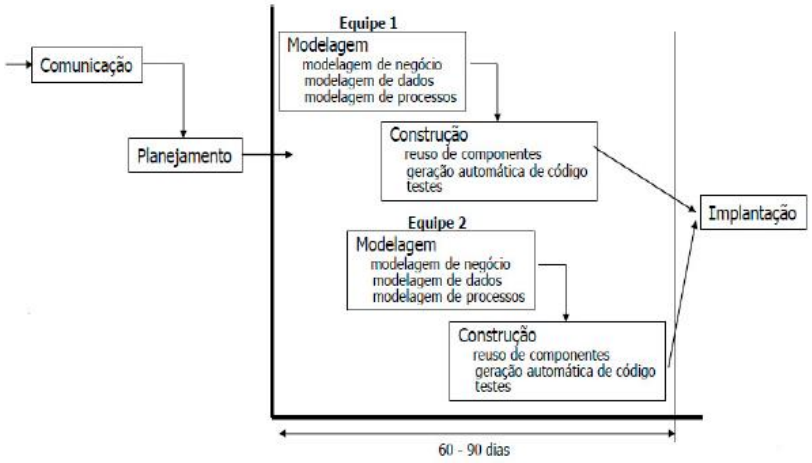
  No modelo incremental, há várias equipes desenvolvendo uma parte do software a serem integradas ao final do desenvolvimento.



- O desenvolvimento incremental é baseado na ideia de desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido. Assim, o modelo incremental é simples: à medida que o projeto vai evoluindo, e as incertezas diminuindo, planeja-se a próxima etapa ou onda, e assim por diante.
- Desenvolvimento incremental é baseado na ideia de desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido. Atividades de especificação, desenvolvimento e validação são intercaladas, e não separadas, com rápido feedback entre todas as atividades.
- Desenvolvimento incremental é uma parte fundamental de abordagens ágeis – é melhor que a abordagem em cascata para a maioria dos sistemas de negócios, e-commerce e sistemas pessoais.
- Cada incremento/versão do sistema incorpora alguma funcionalidade necessária para o cliente.
- Em geral, incrementos iniciais incluem a funcionalidade mais importante ou mais urgente. Isso significa que o cliente pode avaliar o sistema em um estágio relativamente inicial do desenvolvimento para ver se ele oferece o que foi requisitado.

1.2.1.1. MODELO ITERATIVO INCREMENTAL: RAD (Rapid Application Development)

- Trata-se de um modelo de processo de software incremental que enfatiza um ciclo de desenvolvimento curto. É uma adaptação "de alta velocidade" do modelo em cascata, no qual o desenvolvimento rápido é conseguido com o uso de uma abordagem de construção baseada em componentes.
- enfatiza o ciclo de desenvolvimento curto (60 a 90 dias).
- Desvantagens: projetos grandes precisam de muitas equipes; exige comprometimento dos clientes e desenvolvedores; sistemas não modularizados são problemáticos; não é adequado para projetos com grandes riscos técnicos.



VANTAGENS	DESVANTAGENS
Permite o desenvolvimento rápido e/ou a prototipagem de aplicações.	Exige recursos humanos caros e experientes.
Criação e reutilização de componentes.	O envolvimento com o usuário tem que ser ativo.
Desenvolvimento é conduzido em um nível mais alto de abstração.	Comprometimento da equipe do projeto.
Grande redução de codificação manual com *wizards*.	Custo alto do conjunto de ferramentas e hardware para rodar a aplicação;
Cada função pode ser direcionada para a uma equipe separada.	Mais difícil de acompanhar o projeto.
Maior flexibilidade (desenvolvedores podem reprojetar à vontade).	Perda de precisão científica (pela falta de métodos formais).
Provável custo reduzido (tempo é dinheiro e também devido ao reuso).	Pode levar ao retorno das práticas caóticas no desenvolvimento.
Tempo de desenvolvimento curto.	Pode construir funções desnecessárias.
Protótipos permitem uma visualização mais cedo.	Requisitos podem não se encaixar (conflitos entre desenvolvedores e clientes).
Envolvimento maior do usuário.	Padronização (aparência diferente entre os módulos e componentes)

1.2.1.2. MODELO ITERATIVO INCREMENTAL: Rational Unified Process (RUP)

O Rational Unified Process (RUP) é um exemplo de modelo de processo de desenvolvimento baseado no Unified Process (Processo Unificado) desenvolvido pela Rational.

O Rational Unified Process é um processo de desenvolvimento iterativo e incremental, no qual o software não é implementado em um instante no fim do projeto, mas é, ao contrário, desenvolvido e implementado em partes.

O processo unificado consiste em um conjunto de diretrizes que visam a aumentar as chances de um bem sucedido projeto de desenvolvimento de software.

Uma de suas características é gerar como produto de cada etapa de tempo definido e curto um sistema parcial, testável e integrável.

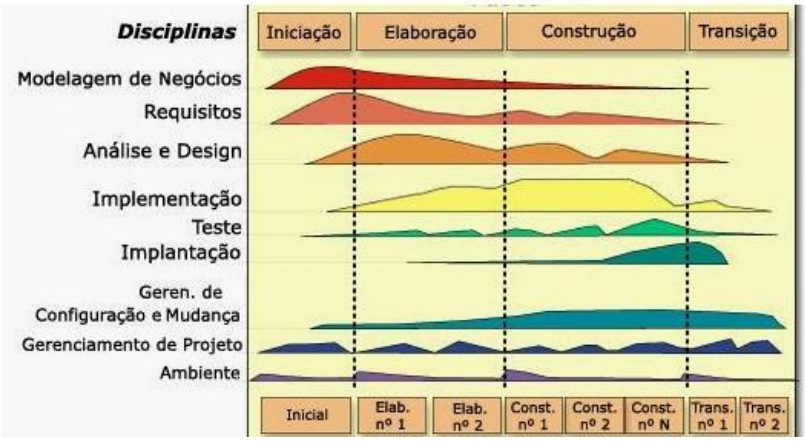
A cada iteração deste processo utiliza-se quatro fases, a saber: Concepção, Elaboração, Construção e Transição.

- Concepção ou Iniciação (Inception):
  - estabelece-se a lógica do domínio da aplicação para o projeto
  - avaliar a contribuição do sistema ao negócio;
  - definir o escopo do projeto.
  - definir aqui o caso de negócio
  - compreender o problema da tecnologia empregada por meio da definição dos use cases mais críticos.
  - obter o comprometimento do patrocinador do projeto para seguir adiante.
  - MARCO: concordância com o escopo e custo.

- Elaboração (Elaboration):
  - coletar requisitos mais detalhados,
  - analisar o domínio do problema,
  - desenvolver o plano do projeto
  - estabelecer a arquitetura,
  - criar um plano para a construção do sistema.
  - faz-se uma análise de risco
  - eliminar elementos de alto risco.
  - nessa fase então que se realiza as principais etapas a respeito de risco.
  - MARCO: arquitetura validada.

- Construção (Construction): consiste de várias iterações, nas quais cada iteração constrói software de qualidade de produção, testado e integrado, que satisfaz um subconjunto de requisitos de projeto. Cada iteração contém todas as fases usuais do ciclo de vida da análise, do projeto, da implementação e do teste. Desenvolve-se o software e prepara-se o mesmo para a transição para os usuários. Além do código, também são produzidos os casos de teste e a documentação.
  - MARCO: produto suficientemente maduro.

- Transição (Transition): nesta fase são realizados os treinamentos dos usuários e a transição do produto para utilização. Este trabalho pode incluir também testes beta e ajuste de performance.
  - MARCO: aceite do cliente ou do final do projeto.



1.2.1.3. MODELO ITERATIVO INCREMENTAL: Metodologias Ágeis

O manifesto Ágil prevê 12 princípios, são eles:

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adéquam a mudanças, para que o cliente possa tirar vantagens competitivas.
3. Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
4. Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
5. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
6. O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
7. Software funcional é a medida primária de progresso.
8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.
9. Contínua atenção à excelência técnica e bom design, aumenta a agilidade.
10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
11. As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.
12. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

TIMEBOX: define o tempo a ser utilizado em um ciclo de desenvolvimento e depois define a funcionalidade que pode ser desenvolvida naquela “caixa” de tempo.

A técnica de timebox envolve a definição de um período de tempo fixo (a "caixa de tempo") e, em seguida, a definição das tarefas ou funcionalidades que podem ser realizadas dentro desse período. Esta técnica é comumente usada em metodologias ágeis, como Scrum e Extreme Programming (XP), onde cada sprint ou iteração é um timebox.

Scrum:

- Framework ágil para gerenciamento de projetos;
- Baseado em sprints (iterações de duração fixa, geralmente 2-4 semanas);
- Papeis bem definidos (Product Owner, Scrum Master e Equipe de Desenvolvimento);
- Três pilares empíricos do Scrum: transparência, inspeção e adaptação;
- Artefatos
    - Product Backlog: lista ordenada de tudo que preciso desenvolver para entregar o meu produto. Ele sempre está em constante mudança e nunca está completo.
    - Sprint Backlog: Subconjunto do backlog do produto (Product Backlog), listando as tarefas técnicas que serão desenvolvidas no decorrer da Sprint.
        - Durante a Sprint apenas o Time de Desenvolvimento pode alterar o Backlog da Sprint.

- Eventos (Reuniões regulares):
    - 1. Planejamento da Sprint: Reunião realizada antes do início de cada sprint, buscando identificar o que será entregue como resultado e como o trabalho será realizado.
    - 2. Reuniões Diárias (Daily Scrum): As reuniões diárias têm por objetivo identificar a remover impedimentos rapidamente. São realizadas, geralmente, em pé e com duração de no máximo 15 minutos.
    - 3. Revisão da Sprint: Avalia o resultado da sprint (avalia o produto)

- 4. Retrospectiva da Sprint: Avalia o processo da sprint e o que melhorar para as próximas (avalia o processo)

Kanban:

- Sistema ágil de gerenciamento visual de fluxo de trabalho;
- Baseado no princípio "pull" (puxar) para controlar a quantidade de trabalho em andamento (WIP – Work in Progress);
- Foco em limitar o WIP e identificar gargalos no processo;
- Quadro Kanban: divisão de tarefas em colunas representando diferentes etapas do processo;
- Flexível e adaptável, pode ser usado em conjunto com outras metodologias ágeis (como Scrum);

XP (Extreme Programming):

- Metodologia ágil focada na qualidade do código e na eficiência do desenvolvimento;
- Práticas-chave: programação em pares, desenvolvimento orientado a testes (TDD), integração contínua, refatoração;
- Valores fundamentais: comunicação, simplicidade, feedback, coragem, respeito;
- Reuniões de planejamento e revisão para manter o alinhamento com os requisitos e melhorar continuamente;
- Adequado para equipes co-localizadas e projetos com requisitos em constante mudança;

1.2.2. Modelos Iterativos Evolutivo:

- Desenvolvimento através de protótipos ou iterações evolutivas
- Refinamento contínuo dos requisitos e soluções
- Adaptação e aprendizado durante todo o projeto
- Gerenciamento efetivo de incertezas e mudanças
- Exemplos: Modelo Espiral, Prototipagem

- intercala as atividades de especificação, desenvolvimento e validação, permitindo que um sistema inicial seja desenvolvido rapidamente, baseado em especificações abstratas.

- No desenvolvimento evolucionário, as atividades de especificação, desenvolvimento e validação são intercaladas.

- Esta abordagem permite o desenvolvimento rápido de um sistema inicial, que pode então ser refinado e expandido com base no feedback e nas mudanças nos requisitos.

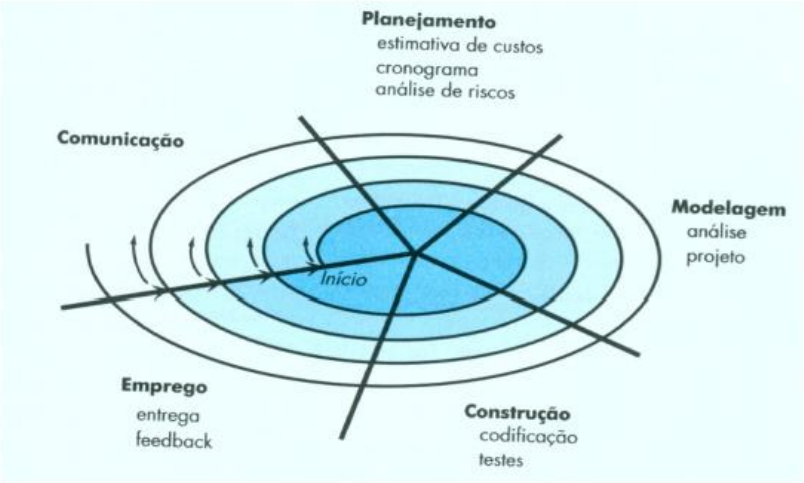
- Os requisitos podem não ser bem definidos no início do projeto e podem mudar ao longo do tempo.

- O modelo evolutivo é adequado para projetos onde os requisitos são incertos ou a tecnologia é nova ou desconhecida.

1.2.2.1. MODELO ITERATIVO EVOLUTIVO: Modelo Espiral

- O processo de desenvolvimento ocorre em ciclos, cada um contendo fases de avaliação e planejamento em que a opção de abordagem para a próxima fase (ou ciclo) é determinada.

- Nesse modelo acrescenta-se a Análise dos Riscos ao ciclo de vida para auxiliar as decisões a respeito da próxima iteração.



- No modelo de processo de desenvolvimento em espiral, cada *loop* na espiral representa uma fase do processo de *software*. Esse modelo exige a consideração direta dos riscos técnicos em todos os estágios do projeto e, se aplicado adequadamente, deve reduzir os riscos antes que eles se tornem problemáticos.

Críticas:

- exige gerentes e técnicos experientes;
- uma vez que o modelo em espiral pode levar ao desenvolvimento em paralelo de múltiplas partes do projeto, as tarefas gerenciais para acompanhamento e controle do projeto são mais complexas;
- é necessário o uso de técnicas específicas para estimar e sincronizar cronogramas, bem como para determinar os indicadores de custo e progresso mais adequados.

1.2.2.2. MODELO ITERATIVO EVOLUTIVO: Prototipação (Prototipagem)

O paradigma de prototipagem começa com a definição dos requisitos. Um projeto rápido é realizado e concentra-se na representação daqueles aspectos que ficarão visíveis pelo cliente. O protótipo é criado e avaliado e é ajustado para satisfazer as necessidades do cliente.

- O uso da prototipação é importante em situações em que os clientes ou usuários têm em mente um conjunto geral de objetivos para um sistema de software, mas não são capazes de identificar claramente as funcionalidades ou informações (requisitos) que o sistema terá de prover ou tratar.

- Um protótipo auxilia tanto o usuário, quanto o técnico a esclarecer e refinar os requisitos que ainda estão obscuros a partir das funcionalidades principais já mapeadas inicialmente.

- Os protótipos podem ser usados nas etapas de elicitação, análise e validação. (Todas menos Especificação).

Nesse contexto:

▪ o cliente não possui uma visão clara de todos os requisitos da aplicação;
▪ o cliente quer avaliar a viabilidade de desenvolvimento da aplicação;
▪ o cliente alocará um usuário-chave no projeto, em tempo integral, a fim de que este possa participar ativamente de todas as fases do projeto;
▪ o cliente gostaria de ter uma versão preliminar do sistema, com base em uma versão inicial dos requisitos, ainda que isto demande um investimento inicial.

▪ Vantagens da prototipagem:
- Maior participação e comprometimento dos clientes e usuários;
- os resultados são apresentados mais rapidamente.

▪ Críticas:
- Forte dependência das linguagens e ambientes utilizados, bem como da experiência da equipe;
- o cliente tende a considerar o protótipo como versão final, podendo comprometer a qualidade do projeto; e
- o desenvolvedor tende a fazer concessões na implementação, a fim de colocar um protótipo em funcionamento rapidamente. Estas concessões podem se tornar parte integrante do sistema.

Prototipação Evolucionária: o protótipo evolui até se transformar no próprio produto entregue ao cliente;

Prototipação Descartável: o protótipo é utilizado para esclarecer requisitos e avaliar riscos, sendo descartado ao final do processo.

1.3. Modelos Específicos:

1.3.1. Desenvolvimento Baseado em Componentes

Desenvolve aplicações a partir de componentes de software pré-empacotados: reuso de componentes de *softwares* já existentes, para conseguir redução no tempo do ciclo de desenvolvimento, bem como redução no custo do projeto;

1.3.2. Test Driven Development (TDD).

Como o próprio nome sugere, essa metodologia é baseada em testes para o desenvolvimento de cada parte do sistema. Ela faz primeiro um código que não passa para depois ir lapidando-o até obter o melhor resultado possível. Veja abaixo como funciona o ciclo:

Ciclo de desenvolvimento
Red,Green, Refactor. Ou seja:

1. Escrevemos um Teste que inicialmente não passa (Red)
2. Adicionamos uma nova funcionalidade do sistema
3. Fazemos o Teste passar (Green)
4. Refatoramos o código da nova funcionalidade (Refactoring)
5. Escrevemos o próximo Teste

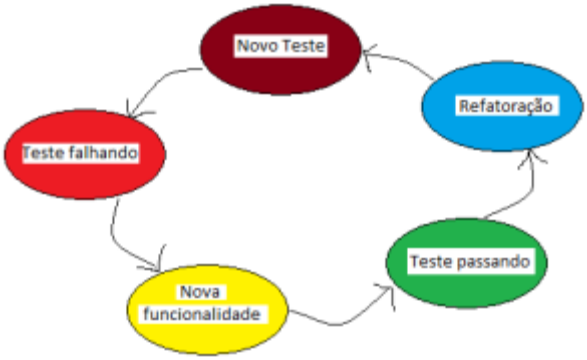


Figura 1. Ciclo de desenvolvimento do TDD

Nós temos, neste tipo de estratégia, um feedback rápido sobre a nova funcionalidade e sobre uma possível quebra de outra funcionalidade do sistema. Assim tempos muito mais segurança para as refatorações e muito mais segurança na adição de novas funcionalidades

- TDD tem que projetar os casos de teste antes de criar o código fonte.

2. Engenharia de Requisitos

FASES	DESCRIÇÃO
CONCEPÇÃO	Após uma necessidade de o negócio ser identificada, busca-se estabelecer um entendimento básico do problema. Trata-se da concepção inicial do software e busca entender o problema, quem são os envolvidos, a natureza da solução e iniciar o processo de comunicação entre clientes e colaboradores.
LEVANTAMENTO	Etapa crítica, utiliza uma abordagem organizada para descobrir o que o cliente deseja em seu sistema. Envolve intensa participação do stakeholders e faz três perquntas: Qual o objetivo do produto? Como o produto se enquadra nas necessidades do negócio? Como o produto será utilizado?
ELABORAÇÃO	Por vezes chamada Análise, informações obtidas do cliente durante a concepção e levantamento são expandidas e refinadas em um modelo, definindo o domínio do problema. Incluem-se modelagens de cenários de interação do usuário com o sistema e modelagens das classes envolvidas.
NEGOCIAÇÃO	Tem por objetivo chegar a um consenso sobre os conflitos entre clientes e usuários, por intermédio de um processo de negociação. Os requisitos são avaliados junto ao cliente e podem se combinar, excluir ou até mesmo inserir novos requisitos.
ESPECIFICAÇÃO	Por vezes chamada Documentação, produto final do engenheiro de requisitos, pode ser um documento escrito, um modelo gráfico, cenários de uso, protótipos, etc. Trata-se da apresentação formal dos dados obtidos até o momento de modo que possa quiar o desenvolvimento futuro do software.
VALIDAÇÃO	Os produtos de trabalho resultantes da engenharia de requisitos são avaliados quanto a sua qualidade por todos os envolvidos (clientes, colaboradores e usuários). Buscam-se erros de interpretação, ambiguidades e omissões.
GESTÃO	conjunto de atividades que auxiliam a equipe de projeto a identificar, controlar e rastrear requisitos e mudanças nos requisitos a qualquer momento . Para projetos de qrande porte, é uma fase essencial na medida em que mudanças em um requisito podem afetar diversos outros requisitos.

2.1. Objetivos da Disciplina de Requisitos

A disciplina de requisitos é uma área crítica na engenharia de software, que se concentra na coleta, análise, especificação e validação dos requisitos de um sistema de software. Ela serve como a ponte entre as necessidades do usuário ou do negócio e a equipe de desenvolvimento de software. Aqui estão os principais objetivos desta disciplina:

1) Entender as necessidades do usuário e do negócio: O primeiro e mais importante objetivo da disciplina de requisitos é compreender completamente o que os usuários e as partes interessadas do negócio desejam do sistema. Isso envolve a coleta de requisitos por meio de várias técnicas, como entrevistas, observações, questionários, workshops, etc.

2) Análise dos requisitos: Uma vez que os requisitos são coletados, eles são analisados para identificar quaisquer inconsistências, conflitos, ambiguidades ou omissões. Além disso, os requisitos são priorizados com base em critérios como valor de negócio, risco, dependências e custo de implementação.

3) Especificação dos requisitos: Este é o processo de documentação detalhada dos requisitos de forma clara, concisa e completa. A especificação de requisitos fornece uma descrição completa do comportamento do sistema e serve como um contrato entre os clientes e a equipe de desenvolvimento.

4) Validação dos requisitos: A validação envolve verificar se os requisitos especificados atendem às necessidades e expectativas dos usuários e das partes interessadas do negócio. Isso é geralmente realizado por meio de revisões de requisitos e prototipagem.

5) Fornece uma base para estimar o custo e o tempo de desenvolvimento do sistema: Os requisitos bem definidos e especificados fornecem informações cruciais necessárias para estimar o tempo e o custo de desenvolvimento do sistema. Eles ajudam a equipe de desenvolvimento a entender a complexidade do sistema, o esforço necessário para implementar os requisitos e as dependências entre os requisitos.

Portanto, a disciplina de requisitos desempenha um papel fundamental no sucesso de qualquer projeto de desenvolvimento de software, pois ajuda a garantir que o sistema desenvolvido atenda às necessidades e expectativas dos usuários e do negócio.

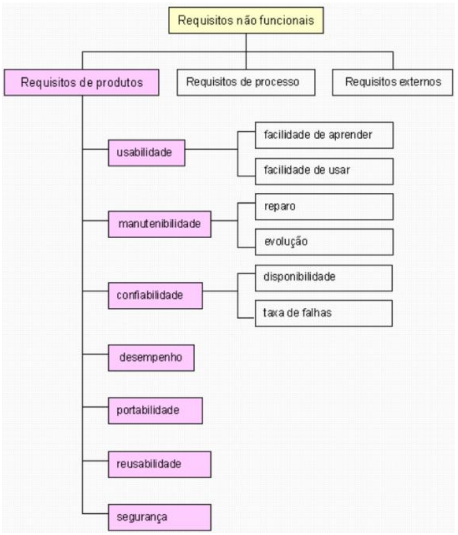
2.2. Tipos de Requisitos

- Requisitos de Negócio: Este é o nível mais alto de abstração. Os requisitos de negócio descrevem as necessidades de alto nível da organização como um todo. Eles estão focados em metas estratégicas e resultados de negócio desejados. Por exemplo, uma empresa pode ter um requisito de negócio para "aumentar a eficiência operacional em 20%". Não há detalhes específicos de implementação aqui; é mais sobre o que a empresa precisa alcançar como um todo.

- Requisitos de Usuário: Este é o próximo nível de abstração. Os requisitos de usuário são baseados nos requisitos de negócio e descrevem o que os usuários finais do sistema precisam que ele faça. Eles são geralmente expressos em linguagem natural e podem incluir diagramas. Por exemplo, baseado no requisito de negócio acima, um requisito de usuário pode ser "Os usuários precisam ser capazes de visualizar relatórios de eficiência em tempo real".

- Requisitos de Sistema: Este é o nível mais baixo de abstração. Os requisitos de sistema são muito mais detalhados e técnicos. Eles descrevem exatamente o que o sistema deve fazer e como deve operar, e são destinados a serem lidos por pessoas com experiência técnica. Por exemplo, baseado no requisito de usuário acima, um requisito de sistema pode ser "O sistema deve consultar a base de dados a cada hora e atualizar o relatório de eficiência em tempo real".

CLASSIFICAÇÃO QUANTO À ORIGEM	
REQUISITOS DE PRODUTO	Especificam o comportamento do produto. Entre os exemplos, estão requisitos de desempenho quanto à rapidez com que o sistema deve operar e quanto de memória ele requer, requisitos de confiabilidade que definem a taxa aceitável de falhas, requisitos de portabilidade e requisitos de usabilidade.
REQUISITOS ORGANIZACIONAIS	São derivados de políticas e procedimentos da organização do cliente e do desenvolvedor. Entre os exemplos, estão padrões de processo que devem ser usados, linguagem de programação ou o método de projeto usado, e requisitos de entrega que especificam quando o produto e a sua documentação devem ser entregues.
REQUISITOS EXTERNOS	Abrange todos os requisitos derivados de fatores externos ao sistema e seu processo de desenvolvimento. Entre os exemplos, estão a interoperabilidade que define como o sistema interage com outros sistemas, requisitos legais que devem ser seguidos, requisitos éticos sistema para assegurar que ele será aceito por todos.



2.3. Requisitos de Produto

Os requisitos de produto se referem a características específicas que o produto de software deve ter. Estes podem incluir requisitos de usabilidade, confiabilidade, desempenho e suporte. Eles são divididos em diferentes categorias, incluindo:

- Usabilidade: os requisitos de usabilidade estão relacionados à facilidade com que os usuários podem usar o produto. Isso pode incluir requisitos para uma interface de usuário intuitiva, suporte adequado para usuários com deficiência, ou facilidade de aprendizado para novos usuários.

- Confiabilidade: os requisitos de confiabilidade estão relacionados à capacidade do produto de funcionar corretamente sob condições normais e anormais. Isso pode incluir requisitos para tolerância a falhas, recuperação de desastres ou disponibilidade do sistema.

- Proteção: os requisitos de proteção estão relacionados à segurança do produto. Isso pode incluir requisitos para controle de acesso, criptografia de dados, ou proteção contra ataques de hackers.

- Eficiência: os requisitos de eficiência estão relacionados à capacidade do produto de fazer o melhor uso possível de recursos como processamento de CPU, memória e espaço em disco. Isso pode incluir requisitos para tempo de resposta rápido, baixo uso de memória, ou capacidade de lidar com grandes volumes de dados.

Estes requisitos são essenciais para garantir que o produto final seja de alta qualidade e atenda às necessidades e expectativas dos usuários. Como tais, eles devem ser claramente definidos e

acordados no início do projeto, e continuamente revisados e atualizados ao longo do ciclo de vida do desenvolvimento do software.

2.4. Fases principais do desenvolvimento de requisitos

- Elucidação (Elicitation):
  - Esta é a fase inicial do processo, onde o objetivo é coletar e entender os requisitos do sistema.
  - Isso pode envolver a realização de entrevistas com os usuários ou partes interessadas, a realização de workshops, a observação do ambiente de trabalho do usuário, a análise de documentos existentes ou a realização de questionários.
  - A principal saída desta fase são os requisitos brutos, que são os requisitos coletados na forma em que foram originalmente fornecidos.
  - A criação de um diagrama de contexto é comumente associada à fase de elucidação (elicitation) de requisitos. Este diagrama ajuda a identificar quais entidades externas (atores) interagem com o sistema e quais são as interfaces de interação.

- Análise:
  - Na fase de análise, os requisitos brutos são estudados e detalhados.
  - Isso envolve a identificação de quaisquer ambiguidades, inconsistências ou omissões nos requisitos.
  - Também envolve a classificação dos requisitos em diferentes categorias (como requisitos funcionais e não funcionais) e a priorização dos requisitos.
  - A principal saída desta fase são os requisitos analisados, que são uma versão mais detalhada e precisa dos requisitos.
  - A priorização dos requisitos é mais comumente associada à fase de análise de requisitos. Durante a análise, os requisitos são estudados, detalhados e priorizados com base em critérios como valor de negócio, risco, dependências e custo de implementação.
  - A identificação de eventos e respostas é geralmente uma parte da fase de análise de requisitos. Durante esta fase, a equipe de desenvolvimento tenta entender como o sistema deve responder a determinados eventos ou condições.

- Especificação:
  - Na fase de especificação, os requisitos analisados são documentados em um formato padrão.
  - Isso fornece um registro permanente dos requisitos e também serve como um contrato entre os desenvolvedores e os usuários ou partes interessadas sobre o que o sistema deve fazer.
  - A principal saída desta fase é o Documento de Especificação de Requisitos.
  - A descrição das regras de negócio é uma prática associada à especificação de requisitos. Durante a fase de especificação, as regras de negócio (ou seja, as políticas, diretrizes e restrições que definem ou restringem algum aspecto do negócio) são formalmente documentadas. Estas regras geralmente são expressas como declarações de fato que definem ou restringem algum aspecto do sistema.

- Validação:
  - Na fase de validação, os requisitos são verificados para garantir que eles são corretos, completos, consistentes e alcançáveis.

- Isso pode envolver a revisão dos requisitos por outras partes interessadas, a realização de protótipos ou a realização de testes de aceitação do usuário.
- A principal saída desta fase é um conjunto de requisitos validados, que foram aprovados por todas as partes interessadas e estão prontos para serem implementados.

Essas fases são iterativas e incrementais, o que significa que elas podem ser repetidas várias vezes ao longo do ciclo de vida do projeto, à medida que novas informações se tornam disponíveis ou as circunstâncias mudam.

2.5. Validação de Requisitos

A validação de requisitos é um processo crucial na engenharia de requisitos, que visa garantir que os requisitos especificados para um sistema atendam às necessidades dos usuários e dos negócios, e que são viáveis tecnicamente. Existem várias técnicas para realizar essa validação, sendo algumas das mais comuns as revisões, inspeções e walk-throughs.

- Revisões de Requisitos:
  - As revisões de requisitos são uma abordagem de verificação informal que envolve a leitura dos requisitos para verificar sua completude, consistência e compreensibilidade.
  - Estas revisões podem ser feitas por membros da equipe do projeto, stakeholders ou especialistas no domínio do sistema.

- Inspeções de Requisitos:
  - As inspeções de requisitos são a abordagem mais formal de validação de requisitos, uma análise estruturada dos requisitos por uma equipe de revisores, com diferentes níveis de experiências, seguindo um processo bem definido.
  - As inspeções geralmente envolvem a preparação para a inspeção, a reunião de inspeção propriamente dita e a correção de defeitos identificados.
  - O objetivo é identificar defeitos nos requisitos, como omissões, ambiguidades ou contradições.

- Walk-throughs de Requisitos:
  - Walk-throughs são uma técnica de validação de requisitos que envolve a apresentação dos requisitos a um grupo de stakeholders, com o objetivo de obter feedback e identificar problemas.
  - O autor dos requisitos geralmente lidera o walk-through, explicando cada requisito e respondendo a quaisquer perguntas que possam surgir.
  - Os walk-throughs são uma boa maneira de envolver os stakeholders no processo de validação de requisitos e de garantir que os requisitos sejam compreendidos por todos.

Cada uma dessas técnicas tem suas próprias forças e fraquezas, e a escolha da técnica apropriada depende de vários fatores, como o tamanho e a complexidade do sistema, o tempo e os recursos disponíveis, e o nível de maturidade do processo de engenharia de requisitos da organização.

- ENTONOGRAFIA: vem da junção de duas outras, do grego ethos (cultura) + graphe (escrita). Ou seja, é o estudo da cultura dos povos, fazendo uma relação para ti lembrar na hora da prova é como se o analista, ou desenvolvedor, imergisse na cultura da empresa para levantar os requisitos que não estão tão claros assim, por exemplo, alguns requisitos derivados da maneira como as pessoas trabalham na empresa, a cooperação entre as pessoas, forma de comunicação, etc...

    - A técnica de etnografia para o levantamento de requisitos implica a imersão do analista no ambiente de trabalho onde o sistema vai ser usado.
- Quality Function Deployment (QFD): que nada mais é do que uma técnica que procura traduzir as necessidades dos clientes, ou seja, elencar os requisitos que possui mais valor

para o cliente. Esta técnica foi adaptada para o desenvolvimento de Software também, cobrindo todas as etapas da Engenharia de software, mas é muito bem aplicada na engenharia de requisitos. Um ponto importante é sua classificação de requisitos em:

- Requisitos formais: refletem objetivos e metas para o produto que garantem a satisfação do cliente.
- Requisitos esperados: requisitos implícitos no produto, que o cliente pode não deixar óbvio, mas sua falta geraria insatisfação.
- Requisitos excitantes: requisitos que vão além das expectativas do cliente.

3. Análise e Projeto

3.1. Análise Estruturada

A análise estruturada é uma atividade de construção de modelos. Utiliza uma notação que é própria ao método de análise estruturada com a finalidade de retratar o fluxo e o conteúdo das informações utilizadas pelo sistema, dividir o sistema em partições ambientais e comportamentais e descrever a essência daquilo que será construído.

A ferramenta central da Análise Estruturada é o Diagrama de Fluxo de Dados (DFD). Trata-se de uma técnica gráfica utilizada na programação estruturada para descrever o fluxo de informação e transformações aplicadas à medida que os dados se movem da entrada para a saída, sem descrever uma representação explícita da lógica procedimental, como loops ou condições.

Para construir um DFD podemos utilizar duas abordagens: top-down e bottom-up.

- TOP-DOWN (do geral para o particular): parte-se do diagrama de contexto (DFD de nível 0 - define apenas os principais processos, os principais depósitos de dados e os fluxos de dados) e da lista de eventos, para decompor sucessivamente os processos, expandindo-os e detalhando-os, de forma a obter processos mais simples, que não podem ser mais decompostos, conhecidos por primitivos funcionais.

- BOTTOM-UP (do particular para o geral): pode-se obter o DFD de nível 0 e o de contexto, por sucessivos agrupamentos de processos que no DFD de nível n tenham ligações entre si através dos mesmos arquivos ou que produzam respostas relacionadas.

Diagramas de Fluxos de Dados (DFD):

- busca identificar os elementos externos que interagem com o sistema;
- busca mostrar o fluxo de informação existente entre o sistema e seu ambiente externo;
- busca estabelecer os limites do sistema;
- busca identificar os eventos que ocorrem no ambiente externo e que provocam uma resposta do sistema.

3.2. Análise Essencial

- A Análise Essencial é um método onde um sistema deve ser modelado através de três dimensões:
  - Dados, que diz respeito aos aspectos estáticos e estruturais do sistema;
  - Controle, que leva em conta aspectos temporais e comportamentais do sistema;
  - Funções, que considera a transformação de valores.

- A Análise Essencial é a técnica que orienta a análise de sistemas para a essência do negócio ao qual se destina independente das soluções de informática que serão utilizadas em sua construção

Os componentes da Análise Essencial são descritos a seguir:

a) Diagrama de Contexto, que tem a finalidade de situar o sistema dentro do negócio da empresa, aonde é demonstrada a finalidade principal do sistema, e as entidades que interagem com o sistema;
b) Lista de Eventos, que é uma lista textual dos estímulos no ambiente externo aos quais o sistema deve responder;
c) Diagrama de Fluxo de Dados (DFD), que apresenta os processos e o fluxo de dados entre eles. Os dados fluem de um nódulo de processamento para outro, onde se modificam.
d) Modelo Entidade-Relacionamento, que fornece uma visão simples e gráfica do sistema para os usuários que não necessitam saber dos detalhes funcionais do sistema;
e) Dicionário de Dados, que é um repositório de informações sobre os componentes dos sistemas. Os dicionários de dados fornecem a informação em forma de texto a fim de auxiliar a informação gráfica mostrada no DFD.

O analista inicia a análise com a criação de um Modelo Essencial, isto é, um modelo com um grau de abstração que não considera restrições tecnológicas. Posteriormente, será inserido o Modelo de Implementação. Este é derivado do Modelo Essencial e contém especificações do sistema considerando restrições tecnológicas.

- Modelo Essencial: apresenta o sistema em um nível de abstração completamente independente de restrições tecnológicas. É formado por:
  - Modelo Ambiental: define a fronteira entre o sistema e o resto do mundo – consiste basicamente de quatro componentes: Declaração de Objetivos; Diagrama de Contexto; Lista de Eventos; e Dicionário de Dados Preliminar (Opcional).
  - Modelo Comportamental: define o comportamento das partes internas do sistema necessário para interagir com o ambiente – consiste de cinco componentes:
    - Diagrama de Fluxo de Dados (DFD) Particionado;
    - Diagrama Entidade-Relacionamento;
    - Diagrama de Transição de Estado;
    - Dicionário de Fluxo de Dados Particionado;
    - Especificações de Processos.

    O diagrama de Fluxo de Dados irá exibir, de forma gráfica as funções, processos (por uma bolha) que serão executados pelo modelo.
    - Para desenvolver o DFD Preliminar é necessário ilustrar um processo para cada evento da lista de eventos. Cada processo deve possuir o mesmo nome da resposta do sistema ao evento e, por fim, ilustrar as entradas e saídas dos processos.

    Já o diagrama de entidade relacionamento é utilizado para definir e ilustrar o relacionamento entre os dados do modelo comportamental.
    - Para o diagrama de entidade relacionamento, também é possível utilizar a lista de eventos para definir as entidades.
  - Métodos Envolvidos: Modelagem de Dados e Modelagem Funcional.

- Modelo de Implementação: tem como objetivo definir a forma de implementação do sistema em um ambiente técnico específico. Apresenta o sistema num nível de abstração completamente dependente de restrições tecnológicas. Possui como artefatos gráficos: Processo, Depósito de Dados, Entidade Externa e Fluxo de Dados.

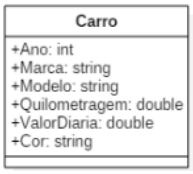
4. UML (Unified Modeling Language)

TIPOS DE DIAGRAMAS	DESCRIÇÃO
DIAGRAMAS ESTRUTURAIS	Representam aspectos estáticos do sistema sob diversas visões diferentes. Em outras palavras, esses diagramas apresentam a estrutura do sistema inalterada há qualquer momento por não levarem em consideração o tempo em sua representação. São eles: Componente, Classes, Implantação, Perfil, Objetos, Estrutura Composta e Pacotes.
DIAGRAMAS COMPORTAMENTAIS	Representam aspectos dinâmicos do sistema como um conjunto de mudanças. Podemos dizer, em outras palavras, que esses diagramas apresentam como os processos e funcionalidades do programa se relacionam. São eles: Máquina de Estados, Casos de Uso, Atividade, Sequência, Comunicação, Interação Geral e Tempo.
DIAGRAMAS DE INTERAÇÃO	São diagramas comportamentais que consideram o relacionamento dinâmico e colaborativo entre os objetos do sistema e suas trocas de informações. Eles enfatizam o controle de fluxo e dados entre as coisas do sistema que estão sendo modeladas (Ex: Objetos). São eles: Sequência, Comunicação, Interação Geral e Tempo.

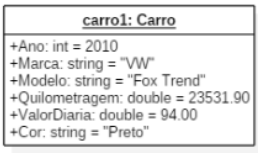
4.1. Diagramas Estruturais

Diagramas Estruturais: Esses diagramas representam a estrutura estática do sistema, mostrando a organização e composição de seus componentes. Incluem:

- Diagrama de Classes: mostra as classes do sistema, seus atributos, métodos e relacionamentos entre elas.



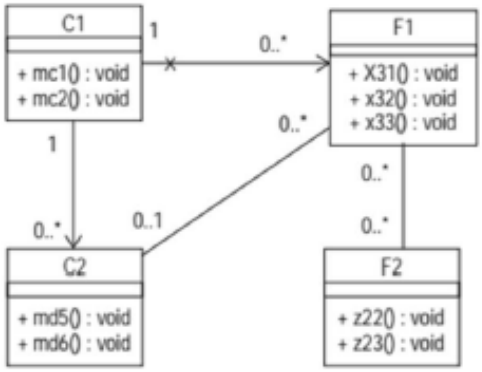
- Diagrama de Objetos: ilustra instâncias de classes, seus atributos e relacionamentos em um momento específico.



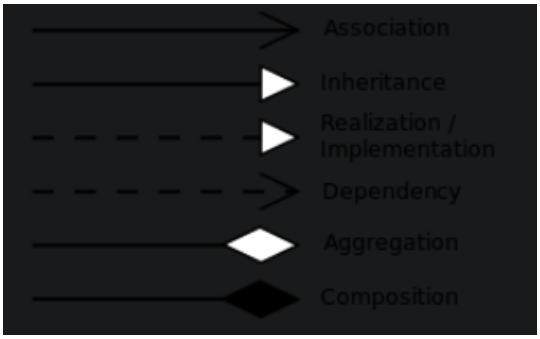
- Diagrama de Componentes: representa os componentes do sistema e suas dependências.
- Diagrama de Implantação: exibe a configuração de hardware e software e como os componentes são distribuídos.
- Diagrama de Pacotes: organiza elementos do modelo em grupos, facilitando o gerenciamento de projetos complexos.
- Diagrama de Estrutura Composta: descreve a organização interna de uma classe, colaboração ou componente.

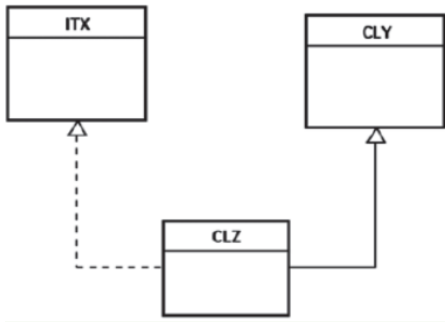
4.1.1. Diagrama de Classe

Diagramas de Classes representam a estrutura estática do sistema, mostrando as classes, seus atributos, métodos e relacionamentos entre elas. Eles são essenciais para identificar os principais componentes do sistema, definindo como os objetos interagem e colaboram. As associações, generalizações e dependências entre classes também são representadas nesses diagramas.



O diagrama de classes da UML possui os seguintes tipos de relacionamentos, que ligam as classes entre si:





- CLZ é uma subclasse de CLY
- CLZ realiza ITX

1) Associação

- Simples: Elementos de uma classe estão ligados a objetos de outra.
- Agregação: A parte existe sem o todo. (ex: Pneu e carro)

- Composição: A parte não existe sem o todo, e este controla o ciclo de vida da parte.

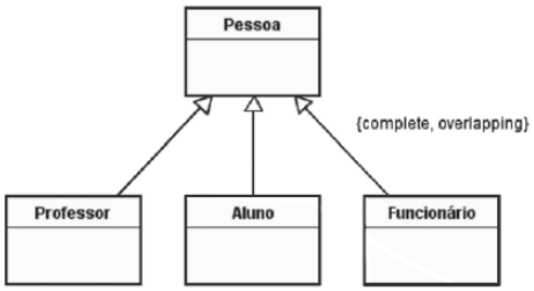
2) Dependência: Mudanças em uma classe pode causar mudanças em outra.

3) Generalização: Quando há uma hierarquia de classes, onde a subclasse herda atributos e comportamentos da classe pai. (Ex: Pessoa e Empregado)

- Uma subclasse é considerada *overlapping*, ou sobreposta, quando uma instância pode ser duas coisas ao mesmo tempo.

  Alguns funcionários podem ser alunos da universidade e Alguns alunos podem ser professores da universidade, portanto, teremos sobreposição.

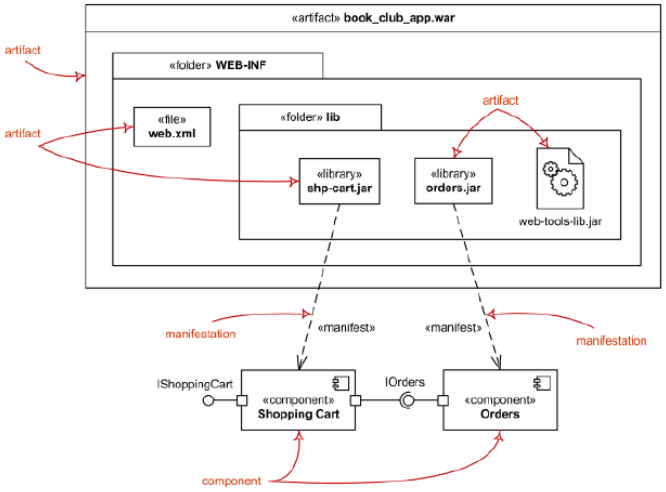
- Uma subclasse é considerada *disjoint* quando só poder ser de um tipo e não de outro.



- Apenas três categorias de pessoas terão acesso às salas: professores, estudantes e funcionários;
- alguns funcionários podem ser alunos da universidade;
- alguns alunos podem ser professores da universidade.

4) Realização: Quando uma classe realiza uma interface.

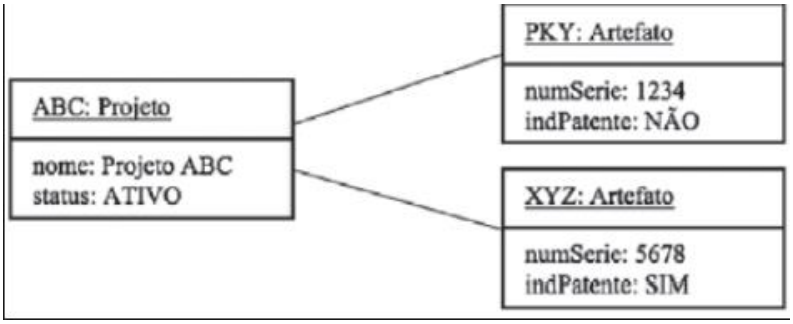
4.1.2. Diagrama de Implantação



- Representa a configuração de hardware e software, além de mostrar como os componentes de software são distribuídos nos nós de hardware.
- Diagramas de Implantação descrevem a distribuição física do sistema, incluindo servidores, dispositivos, redes e outros componentes de hardware.
- Eles também mostram a alocação de componentes de software, como artefatos e executáveis, nos respectivos nós de hardware.
- Diagramas de Implantação são essenciais para compreender a infraestrutura necessária para executar o sistema e auxiliar na identificação de problemas de desempenho e escalabilidade.

4.1.3. Diagrama de Objeto

Diagramas de objetos fornecem uma captura instantânea das instâncias em um sistema e os relacionamentos entre as instâncias.



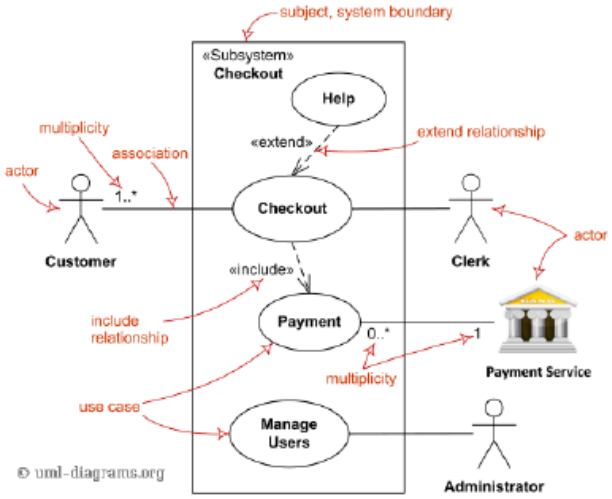
- vemos que no diagrama apresentado contém informações dos atributos das classes. Ou seja, essa classe foi instanciada e virou um OBJETO
- Além disso, o diagrama mostra os relacionamentos entre os OBJETOS

4.2. Diagramas Comportamentais

Diagramas Comportamentais: Esses diagramas demonstram o comportamento dinâmico do sistema, ou seja, como os elementos do sistema interagem e evoluem ao longo do tempo. Incluem:

- Diagrama de Casos de Uso: apresenta as funcionalidades do sistema e as interações entre os atores e o sistema.
- Diagrama de Estados: descreve os estados e transições de um objeto durante seu ciclo de vida.
- Diagrama de Atividades: representa o fluxo de controle e de objetos entre atividades em um sistema, útil para modelar processos de negócio.

4.2.1. Diagrama de Caso de Uso



- muito usado para dar uma "visão geral" (contexto) sobre as funcionalidades existentes em um sistema, através de seus participantes externos e respectivos relacionamentos.
- Responsável por capturar os requisitos funcionais de um sistema, descrevendo seus atores, relacionamentos e casos de uso.
- Diagramas de Caso de Uso demonstram uma interação entre um ator e o sistema
- utilizado para demonstrar quais usuários utilizam determinada funcionalidade do sistema.

Relacionamentos entre casos de uso:

- Inclusão (<<include>>): Indica que um caso de uso inclui outro, ou seja, a execução do primeiro caso de uso implica na execução do caso de uso incluído. A inclusão é usada para modularizar casos de uso comuns, evitando a repetição de comportamentos em diferentes casos de uso.
- Extensão (<<extend>>): Este relacionamento indica que um caso de uso pode ser estendido por outro, acrescentando ou modificando seu comportamento. A extensão é condicional, o que significa que o caso de uso estendido será executado apenas se certas condições forem atendidas.

- *No ato do cadastramento o funcionário que opera o sistema pergunta ao cidadão se ele deseja registrar que ele é doador de órgãos para transplante.*
- *Caso a resposta seja afirmativa, o funcionário seleciona essa opção no formulário de registro, o que fará com que o sistema abra um formulário para que o funcionário registre informações fornecidas pelo cidadão, tais como: tipo sanguíneo, doenças preexistentes, etc.*

- Generalização: Representa uma relação de herança entre casos de uso, onde um caso de uso mais específico herda as características de um caso de uso mais genérico. Esse relacionamento permite a reutilização e a especialização de comportamentos em diferentes casos de uso.

Relacionamentos entre atores:

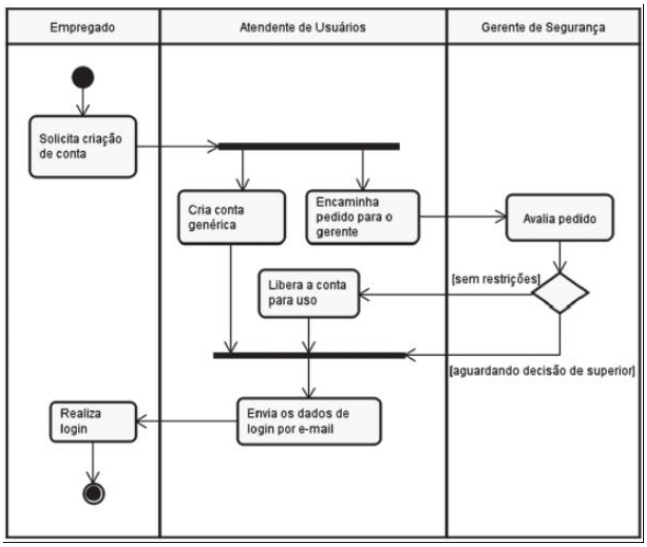
- Generalização: Similar à generalização entre casos de uso, este relacionamento indica que um ator herda as características e associações de outro ator mais genérico. Atores mais específicos podem herdar e estender as funcionalidades de atores mais genéricos, promovendo a reutilização e a modularização.

Relacionamentos entre atores e casos de uso:

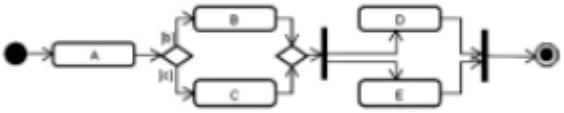
- Associação: Representa a interação entre um ator e um caso de uso. A associação indica que o ator está envolvido na execução do caso de uso, seja como iniciador da ação ou como receptor de informações. As associações podem ser unidirecionais ou bidirecionais, mostrando o fluxo de comunicação entre os elementos envolvidos.

4.2.2. Diagrama de Atividades

O Diagrama de Atividades (FLUXO!) exerce um papel semelhante ao fluxograma, descrevendo a lógica de procedimento e processo de negócio de um dado cenário.



- condições de guarda representam nada menos que as decisões no diagrama de atividades, controlando qual transição ocorre após a conclusão da tarefa.

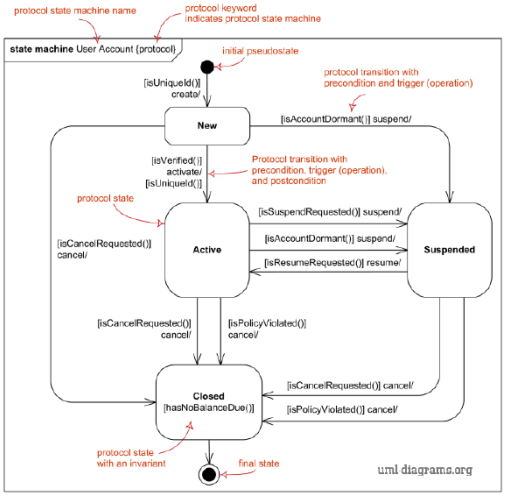


- um pacote, ao chegar a um restaurante, é analisado (atividade A)
- quando se detecta se ele deve ser guardado na geladeira (atividade B) ou no armário (atividade C)
- Após guardado, o pacote deve ser pago (atividade D), e o estoque, atualizado (atividade E), sendo estas duas atividades realizadas em paralelo
- encerrando-se o processo quando essas duas últimas atividades estiverem completas.

4.2.3. Diagrama de (Máquina de) ESTADOS

Diagrama de Transição de Estados, apresenta diversos estados possíveis de um objeto no decorrer da execução de processos de um sistema.

- refere-se a um objeto. É através dele que modelamos as mudanças sofridas em um determinado processo.



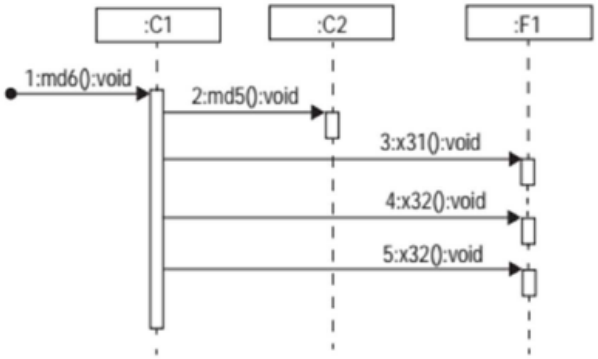
4.3. Diagramas de Interação

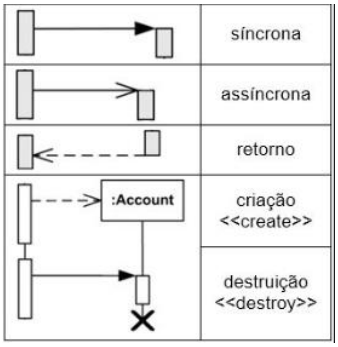
Diagramas de Interação: São um subconjunto dos diagramas comportamentais que focam nas interações entre objetos e componentes. Incluem:

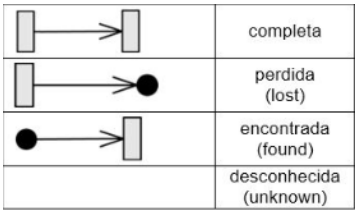
- Diagrama de Sequência: mostra a sequência temporal das interações entre objetos, destacando a ordem das mensagens trocadas.
- Diagrama de Comunicação: foca nos relacionamentos entre objetos e as mensagens trocadas, com menos ênfase na sequência temporal.
- Diagrama de Tempo: exibe o comportamento de um objeto ou sistema em função do tempo, permitindo analisar restrições temporais e eventos.
- Diagrama de Visão Geral de Interação: combina elementos de outros diagramas de interação para fornecer uma visão geral das interações em um sistema complexo.

4.3.1. Diagrama de Sequência

Diagramas de Sequência focam no comportamento dinâmico do sistema, ilustrando a sequência temporal das interações entre objetos. Eles detalham a ordem em que as mensagens são trocadas entre objetos, além de mostrar a criação e destruição de objetos ao longo do tempo. Esses diagramas são úteis para entender o fluxo de eventos e analisar como o sistema responde a diferentes estímulos.







- completa (complete): estão presentes os eventos "remetente" (sendEvent) e "destinatário" (receiveEvent)
- perdida (lost): evento "destinatário" (receiveEvent) é ausente, é interpretada como se o destino da mensagem estivesse fora do escopo da descrição.
- encontrada (found): ausência de evento remetente (sendEvent), é interpretada como se a origem da mensagem estivesse fora do escopo da descrição.
- desconhecida (unknown): nem evento remente nem destinatário estão presentes. Este tipo não possui representação gráfica

Nos diagramas de sequência, fragmentos combinados são agrupamentos lógicos representados por um retângulo, que contém as estruturas condicionais que afetam o fluxo de mensagens. Um fragmento combinado contém operandos de interação e é definido pelo operador de interação. O operador Par define que o fragmento representa uma execução paralela de dois ou mais comportamentos. Vamos relembrar os operadores:

ALT – ALTERNATIVAS	BREAK – QUEBRA	IGNORE – IGNORAR
OPT – OPÇÃO	CRITICAL - REGIÃO CRÍTICA	CONSIDER – CONSIDERAR
PAR – PARALELO	NE - NEGATIVO	SEQ – SEQÜÊNCIA FRACA
LOOP – LAÇO	ASSERTION – AFIRMAÇÃO	STRICT – SEQÜÊNCIA ESTRITA

5. Análise de Ponto de Função

O ponto de função analisa um software de maneira quantitativa e qualitativa baseada na percepção do usuário. Baseado em uma série de fatores, cria pesos para determinadas características do software.

A análise por ponto de função tem como principal objetivo medir a funcionalidade do sistema tendo como base a visão do usuário, de acordo com as seguintes características:

- É independente da tecnologia utilizada;
- Auxilia a produção de resultados consistentes;
- Baseia-se na visão do usuário;
- Tem significado para o usuário final;
- Utiliza-se de estimativas;
- Passível de automação;
- Dificuldade por possuir relativa subjetividade por refletir a visão do usuário.

A primeira entrada de muitos modelos que fazem a estimativa é Pontos de Função é o tamanho, para então derivar o esforço, o custo e o cronograma.

A partir do tamanho sabemos que podemos derivar o esforço necessário, a equipe, o cronograma e o custo. Contudo, essas últimas variáveis irão se relacionar e afetar umas as outras.

6. Verificação, Validação e Teste

PRINCÍPIOS FUNDAMENTAIS	DESCRIÇÃO
TESTES DEMONSTRAM A PRESENÇA DE DEFEITOS...	Um teste pode demonstrar a presença de defeitos, mas não pode provar que eles não existem. Ele reduz a probabilidade de que os defeitos permaneçam, mas mesmo se nenhum defeito for encontrado não quer dizer que ele não os tenha.
TESTES EXAUSTIVOS SÃO IMPOSSÍVEIS...	Testar todas as combinações de entradas e pré-condições é inviável, exceto para casos triviais. Em vez de realizar testes exaustivos, os riscos e prioridades são levados em consideração para dar foco aos esforços de teste.
TESTE O MAIS BREVE POSSÍVEL (ANTECIPADO)...	Os defeitos encontrados nas fases iniciais do processo de desenvolvimento de software são mais baratos de serem corrigidos do que aqueles encontrados já em fase produção. Há, inclusive, técnicas de testes antes mesmo da implementação.
AGRUPEM OS DEFEITOS MAIS SENSÍVEIS...	Seguindo o Princípio de Pareto, 80% dos defeitos são causados por 20% do código. Ao identificar essas áreas sensíveis, os testes podem priorizá-las, de forma a ter alta probabilidade de encontrar defeitos.
PARADOXO DO PESTICIDA...	Caso os mesmos testes sejam aplicados repetidamente, em determinado momento eles deixam de ser úteis, ou seja, não conseguem encontrar nenhum novo defeito. Por isso, os testes precisam ser revisitados com frequência.
TESTES DEPENDEM DO CONTEXTO...	Os testes devem ser elaborados de acordo com o contexto de utilização do software. Ex: um sistema bancário deve ser testado de maneira diferente de uma rede social. Assim como testes de aplicação web têm foco diferente do desktop.
AUSÊNCIA DE DEFEITOS É UMA ILUSÃO	Identificar e corrigir os problemas de um software não garantem que ele está pronto. Os testes foram elaborados para identificar todas as possíveis falhas? O sistema atende às necessidades e expectativas dos usuários? Logo, há outros fatores!

6.1. Tipos de Teste

- Testes aleatórios

O teste aleatório é o chamado teste macaco (monkey). Nele as entradas são aleatórias.

- Testes exploratórios

Um teste onde o analista de teste aprende e testa no mesmo momento, explorando o sistema para aprender durante o teste.

- Testes de mutação

Um tipo de teste onde esses programas levemente modificados servem para identificar testes fracos.

- Testes de perfil operacional

Existe o teste de aceitação operacional que é um teste de qualidade antes da release.

6.1.1. Teste de Unidade

- Também chamado de Teste de Componente/Módulo, focaliza o esforço de verificação na menor unidade de projeto do software: o componente ou módulo.
- Este teste só "se importa" com aquele trecho, ou pedaço de código, por isso que faz o teste de cada um dos módulos de forma isolada.
- Caminhos independentes da estrutura de controle são usados para assegurar que todas as instruções em um módulo tenham sido executadas pelo menos uma vez.
- Finalmente, são testados todos os caminhos de manipulação de erro.
- Geralmente são feitos pelos próprios desenvolvedores de maneira mais informal e, não, por especialistas em testes.

DEFINIÇÕES DE PROVA - TESTES DE UNIDADE
Testes de Unidade são aqueles realizados sobre as menores estruturas de código-fonte, como métodos e classes.
Testes de Unidade consistem em testar individualmente, componentes ou módulos de *software* que, posteriormente devem ser testados de maneira integrada.
Testes de Unidade focalizam cada componente de um software de forma individual, garantindo que o componente funciona adequadamente.
Testes de Unidade focalizam o esforço de verificação na menor unidade de projeto de software, isto é, no componente ou no módulo de software.
Testes de Unidade têm por objetivo explorar a menor unidade do projeto, procurando identificar falhas ocasionadas por defeitos de lógica e de implementação em cada módulo separadamente.
Testes de Unidade enfocam a lógica interna de processamento e as estruturas de dados dentro dos limites de um componente.
Testes de Unidade concentram o esforço de verificação na menor unidade de design de software.
Testes de Unidade concentram-se na lógica de processamento interno e nas estruturas de dados dentro dos limites de um componente.
Testes de Unidade têm por objetivo explorar a menor unidade do projeto, procurando provocar falhas ocasionadas por defeitos de lógica e de implementação em cada módulo, separadamente.
Testes de Unidade têm como foco as menores unidades de um programa, que podem ser funções, procedimentos, métodos ou classes.

6.1.2. Teste de Integração

DEFINIÇÕES DE PROVA - TESTES DE INTEGRAÇÃO
Testes de Integração são caracterizados por testar as interfaces entre os componentes ou interações de diferentes partes de um sistema.
Testes de Integração têm por objetivo verificar se as funcionalidades dos módulos testados atendem aos requisitos.
Testes de Integração visam testar as falhas decorrentes da integração dos módulos do sistema.
Testes de Integração são uma técnica sistemática para construir a arquitetura do software, enquanto, ao mesmo tempo, conduz testes para descobrir erros associados às interfaces.
Testes de Integração têm por objetivo construir uma estrutura de programa determinada pelo projeto a partir de componentes já testados.
Testes de Integração são uma técnica utilizada para descobrir erros associados às interfaces na qual, a partir de componentes testados individualmente, se constrói uma estrutura de programa determinada pelo projeto.
Testes de Integração verificam o funcionamento em conjunto dos componentes do sistema, se são chamados corretamente e se a transferência de dados acontece no tempo correto, por meio de suas interfaces.
Testes de Integração verificam se os componentes do sistema, juntos, trabalham conforme descrito nas especificações do sistema e do projeto do programa.
Testes de Integração são uma técnica sistemática para construir a arquitetura do *software* enquanto conduz testes para descobrir erros associados às interfaces.

- um dos objetivos do teste de integração é detectar problemas junto às interfaces, ou seja, erros que aparecem com a junção das partes do sistema.

- Antes da junção são realizados testes nos pedaços individuais do sistema, os erros encontrados são corrigidos, outros erros aparentemente "menores", nem são notados. Mais tarde, com as partes do sistema integradas, aqueles erros "menores" podem se apresentar não tão pequenos assim, e causar falhas inaceitáveis no sistema".

6.1.3. Teste de Validação (Aceitação)

- Esse teste focaliza simplesmente em ações visíveis ao usuário e também em saídas do sistema reconhecíveis pelo usuário. Se o usuário não vê ou reconhece, não é testado aqui!
- A validação de software é conseguida por meio de uma série de testes que demonstram conformidade com os requisitos.

DEFINIÇÕES DE PROVA - TESTES DE VALIDAÇÃO/ACEITAÇÃO
Testes de Validação focalizam ações e saídas, tais como percebidas pelo usuário final.
Testes de Validação são executados logo após montagem do pacote de software, quando os erros de interface já foram descobertos e corrigidos.
Testes de Validação têm como principal característica verificar o sistema em relação aos seus requisitos originais e às necessidades atuais do usuário.
Testes de Validação avaliam o software com respeito aos seus requisitos e detecta falhas nos requisitos e na interface com o usuário.

6.1.4. Teste de Sistema

- O Teste de Sistema é na realidade uma série de diferentes testes cuja finalidade primária é exercitar totalmente o sistema.
- O teste de integração verifica interfaces entre componentes do mesmo software, já o teste de sistema verifica interfaces entre componentes diferentes de um mesmo sistema – incluindo softwares, hardwares, pessoas e informações, além dos requisitos funcionais e não-funcionais.

DEFINIÇÕES DE PROVA - TESTES DE SISTEMA
Testes de Sistema incluem diversas modalidades de teste, cujo objetivo é testar o sistema computacional como um todo.
Testes de Sistema testam se o sistema cumpre seus requisitos funcionais e não funcionais.
Testes de Sistema avaliam o software com respeito ao seu projeto arquitetural e detecta falhas de especificação, desempenho, robustez e segurança.
Testes de Sistema visam a verificar o sistema, baseado em computador, não se limitando ao software, mas incluindo o processo como um todo, como hardware, pessoal e informação.

6.1.5. Teste de Regressão

- No contexto de uma estratégia de teste de integração, o teste de regressão é a reexecução do mesmo subconjunto de testes que já foram executados para assegurar que as alterações não tenham propagado efeitos colaterais indesejados.
- O teste de regressão ajuda a garantir que as alterações (devido ao teste ou por outras razões) não introduzam comportamento indesejado ou erros adicionais.
- deve ser projetado de forma a incluir somente aqueles testes que tratam de uma ou mais classes de erros em cada uma das funções principais do programa.

6.1.6. Teste de Desempenho

- O teste de desempenho (ou performance) é projetado para testar o desempenho em tempo de execução do software dentro do contexto de um sistema integrado.
- Os testes de desempenho muitas vezes são acoplados ao teste de esforço e usualmente requerem instrumentação de hardware e software.
- Monitorando o sistema com instrumentos, o testador pode descobrir situações que levam à degradação e possível falha do sistema.
- Esse teste pode ser usado também para identificar gargalos, determinar conformidades com os requisitos não-funcionais de desempenho e coletar outras informações, como hardware necessário para a operação da aplicação.
- Em geral, devem ser projetados para assegurar que o sistema pode operar na carga especificada.
- Isso envolve o planejamento de uma série de testes em que a carga é constantemente aumentada até que o desempenho se torne inaceitável.

6.1.7. Teste de Estresse

- o testador que executa teste por esforço pergunta: até onde podemos forçar o sistema até que ele falhe?
- Esse teste usa um sistema de maneira que demande recursos em quantidade, frequência ou volumes anormais. Podem ser realizados testes em condições de alta taxa de entrada de dados, máximo de memória ou processamento, entre outros.
- Em suma, esse tipo de teste é uma forma de teste deliberadamente intenso e completo utilizado para determinar a estabilidade de um determinado sistema ou entidade.
- Envolve o teste para além da capacidade operacional normal, muitas vezes até um ponto de ruptura, a fim de observar os resultados.
- Assim como o teste de desempenho – que está em uma hierarquia superior – o teste de estresse testa basicamente requisitos não-funcionais.

6.1.8. Teste de Carga

- Testes de Carga são a forma mais simples de testes para compreender o comportamento de um sistema sob uma carga específica.
- Capaz de determinar o comportamento de um sistema sob condições especificadas ou acordadas.
- Os testes de carga procuram determinar como o software responderá a várias condições de carga (Ex: número de usuários, número de transações por usuário por unidade de tempo, carga de dados processados por transação) de acordo com os limites de operação do sistema.

6.1.9. Teste de Fumaça

- O teste fumaça deve usar o sistema inteiro de ponta a ponta.
- Ele não precisa ser exaustivo, mas deve ser capaz de expor os principais problemas.
- Simplesmente ver se o software roda e se ele faz o básico, fundamental, o caminho feliz do software.
- Caso ele passe nesse teste, ele estará habilitado a receber testes mais detalhados posteriormente.

6.1.10. Testes Alfa e Beta

- Ambos são conduzidos pelo usuário final e, não, por testadores, programadores ou engenheiros de software;

ALFA: ocorre no ambiente do desenvolvedor e é geralmente realizado por um grupo representativo de clientes e usuários finais e, não, por programadores ou testadores.

BETA: realizado por clientes e usuários finais e ocorrem nas instalações do usuário, isto é, no local real de utilização/trabalho – em um ambiente não controlado.

6.1.11. Teste de Mutação

- gera versões levemente modificadas de um programa sob teste e exercita tanto o programa original quanto os programas modificados, procurando diferenças entre essas formas;

6.2. Técnicas de Testes

6.2.1. Teste Caixa Branca (Estrutural, Procedimental, Orientado à Lógica)

- Analisa caminhos lógicos possíveis de serem executados, portanto é necessário ter conhecimento sobre o funcionamento interno dos componentes. Ela busca garantir que todos os caminhos independentes de um módulo sejam executados pelo menos uma vez;
- Trata de todas as decisões lógicas para valores verdadeiros e falsos, além de executar laços dentro dos valores limites e avaliar as estruturas de dados internas do software;

- Teste de Condição:
    - Teste de condição é um teste caixa-branca que exercita as condições lógicas contidas em um módulo de programa.
    - Sua proposta é avaliar se as variáveis ou expressões lógicas (booleanos – true/false) estão consistentes.

- Teste de Fluxo de Dados:
    - O método de teste de fluxo de dados é um teste caixa-branca que seleciona caminhos de teste de um programa de acordo com as localizações de definições e usos de variáveis no programa.
    - A ideia é identificar e classificar todas as ocorrências de variáveis no programa e então gerar, para cada variável, dados de teste de modo que todas as definições (quando uma variável é definida através de uma leitura ou quando ela aparece do lado esquerdo de um comando de atribuição) e usos (quando a variável é usada na avaliação de uma expressão, ou em um comando de saída, ou afeta o fluxo) sejam exercitadas

- Teste de Caminho Básico

    - O teste de caminho base, uma técnica caixa-branca, usa diagramas de programa (ou matrizes gráficas) para derivar o conjunto de testes linearmente independentes que garantirão abranger todas as instruções do programa.

6.2.2. Teste Caixa Preta (Comportamental, Funcional, Orientada a Dado, Orientado à Entrada/Saída)

- Baseia-se em pré-condições e pós-condições, geralmente sendo utilizada nas etapas posteriores da disciplina de testes.
- Busca funções incorretas ou inexistentes, erros de comportamento ou desempenho, erros de inicialização e interface, entre outros.
- Deriva casos de teste a partir da especificação de requisitos, ignorando detalhes de implementação e se focando nas saídas geradas em resposta a entradas escolhidas e condições especificadas.

Sem saber a lógica, eu sei que:

- Dadas condições específicas, eu devo alcançar resultados específicos;
- Quais dados devem ser gerados como resultado;

Exemplo:

- Se sua internet cair, você pode testar se o cabo de força está conectado na tomada; se o cabo de internet está conectado ao computador, etc.

- Análise de valor-limite

    - Análise do valor limite é uma técnica caixa-preta que leva a uma seleção de casos de teste que utilizam valores limites. Um número maior de erros ocorre nas fronteiras do domínio de entrada e não no "centro"

- Particionamento de Equivalência

    - Particionamento de equivalência é um método de teste caixa-preta que divide o domínio de entrada de um programa em classes de dados a partir das quais podem ser criados casos de teste.

6.2.3. Teste Caixa Cinza

- O Teste Caixa-Cinza é uma versão híbrida entre os Testes Caixa-Branca e os Testes Caixa-Preta;
- Essa técnica analisa a parte lógica mais a funcionalidade do sistema, fazendo uma comparação do que foi especificado com o que está sendo realizado;
- O testador comunica-se com o desenvolvedor para entender melhor o sistema e otimizar os casos de teste que serão realizados;
- Isso envolve ter acesso a estruturas de dados e algoritmos do componente a fim de desenvolver os casos de teste, que são executados como na técnica da caixa-preta.

7. PMBOK6

7.1. DEFINIÇÕES:

PROJETO:

- Um esforço temporário empreendido para criar um produto, serviço ou resultado único.
- Um empreendimento ou evento não repetitivo, caracterizado por uma sequência clara e lógica de eventos, com início, meio e fim, que se destina a atingir um objetivo claro e definido, sendo conduzido por pessoas, dentro de parâmetros pré-definidos de tempo, custo, recursos envolvidos e qualidade.

OPERAÇÃO:

- Atividade repetitiva, permanente e com o final previsível realizadas com o propósito de atender necessidades funcionais.
- Uma função organizacional que realiza a execução CONTÍNUA de atividades que produzem o mesmo produto ou fornecem um serviço repetitivo.
- Exemplos: operações de produção, operações de fabricação e operações de contabilidade.

PROCESSO:

- Quando uma empresa faz sempre as mesmas tarefas (como em uma linha de montagem de automóveis populares) ela está realizando processos.
- Quando uma empresa faz sempre trabalhos diferentes, de acordo com cada pedido (imagine uma empresa que tenha como atividade montar palco de shows) ela trabalha com projetos.

PROGRAMA:

- Um programa é definido como um grupo de projetos, subprogramas e atividades de programa relacionados, GERENCIADOS DE MODO COORDENADO, visando à obtenção de benefícios que não estariam disponíveis se eles fossem gerenciados individualmente.
- Um projeto pode ou não ser parte de um programa, mas um programa SEMPRE terá projetos.
- Os projetos de um PROGRAMA são RELACIONADOS entre si (necessariamente)

PORTFÓLIO:

- Um portfólio refere-se a um conjunto de projetos ou programas e outros trabalhos, AGRUPADOS para facilitar o gerenciamento eficaz desse trabalho, a fim de atingir os objetivos estratégicos de negócios.
- Os projetos ou programas de um PORTFÓLIO PODEM OU NÃO ser RELACIONADOS entre si

7.2. Escritório de Projetos (Project Management Office, PMO ou EGP)

- Escritório de Gerenciamento de Projetos (EGP) é uma estrutura organizacional que padroniza os processos de governança relacionados a projetos, e facilita o compartilhamento de recursos, metodologias, ferramentas e técnicas.

O PMO tem como principal função dar suporte aos gerentes de projetos de diversas maneiras, que incluem, por exemplo:

- gerenciamento de recursos compartilhados entre todos os projetos administrados pelo PMO;
- identificação e desenvolvimento de metodologia, melhores práticas e padrões de gerenciamento de projetos;
- orientação, aconselhamento, treinamento e supervisão;
- monitoramento da conformidade com as políticas, procedimentos e modelos padrões de gerenciamento de projetos por meio de auditorias em projetos;
- desenvolvimento e gerenciamento de políticas, procedimentos, formulários e outras documentações compartilhadas do projeto (ativos de processos organizacionais);
- coordenação das comunicações entre projetos.

O PMBOK (Project Management Body of Knowledge), em sua 6ª edição, descreve três tipos de escritórios de gerenciamento de projetos (PMO - Project Management Office), cada um com um nível de controle e influência distinto:

- PMO Suportivo (Supportive PMO): Este tipo de PMO atua como um repositório de recursos, templates, políticas e procedimentos relacionados à gestão de projetos. Ele fornece suporte de maneira consultiva e tem um nível de controle baixo. É um facilitador e geralmente é escolhido em organizações com uma cultura de gestão de projetos de baixo a moderado.

- PMO Controlador (Controlling PMO): Este tipo de PMO fornece suporte e exige conformidade com certas políticas, metodologias e frameworks. Ele pode fornecer templates e outras ferramentas, mas também exige que as equipes de projeto adiram a uma metodologia de gerenciamento de projetos específica. O nível de controle exercido por um PMO controlador

é moderado. Este tipo de PMO é comum em organizações com um nível moderado de maturidade em gestão de projetos.

- PMO Diretivo (Directive PMO): Este tipo de PMO fornece suporte e toma o controle direto da execução de projetos. Ele fornece recursos e gerentes de projeto para gerenciar projetos e tem um alto nível de controle. O PMO diretivo é comum em organizações com um alto grau de maturidade em gestão de projetos.

7.3. Estruturas Organizacionais

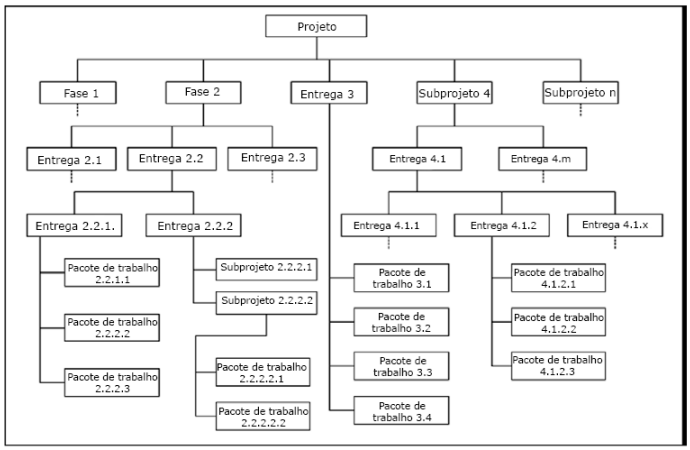
Tipos de estrutura organizacional	Características do projeto					
	Grupos de trabalho organizados por	Autoridade do gerente do projeto	Papel do gerente do projeto	Disponibilidade de recursos	Quem gerencia o orçamento do projeto?	Pessoal administrativo de gerenciamento de projetos
Orgânico ou simples	Flexível: pessoas trabalhando lado a lado	Pouca ou nenhuma	Em tempo parcial; pode ou não ser um papel designado. como coordenador	Pouca ou nenhuma	Proprietário ou operador	Pouco ou nenhum
Funcional (centralizado)	trabalho realizado (ex.: engenharia, fabricação	Pouca ou nenhuma	Em tempo parcial; pode ou não ser um papel designado. como coordenador	Pouca ou nenhuma	Gerente funcional	Em tempo parcial
Multidivisional (pode replicar funções para cada divisão com pouca centralização	Um de: produto; processos de produção; portfólio; programa; região geográfica; tipos de cliente	Pouca ou nenhuma	Em tempo parcial; pode ou não ser um papel designado. como coordenador	Pouca ou nenhuma	Gerente funcional	Em tempo parcial
Matriz – forte	Por função, com gerente do projeto como uma função	Moderada a alta	Função designada em tempo real	Moderada a alta	Gerente do projeto	Full-time
Matriz – fraca	Função	Baixa	Em tempo parcial; pode ou não ser um papel designado. como coordenador	Baixa	Gerente funcional	Em tempo parcial
Matriz – esquilibrada	Função	Baixa a moderada	Em tempo parcial; pode ou não ser um papel designado. como coordenador	Baixa a moderada	Misto	Em tempo parcial

Tipos de estrutura organizacional	Características do projeto					
	Grupos de trabalho organizados por	Autoridade do gerente do projeto	Papel do gerente do projeto	Disponibilidade de recursos	Quem gerencia o orçamento do projeto?	Pessoal administrativo de gerenciamento de projetos
Orientado a projeto (composto, híbrido)	Projeto	Alta a quase total	Função designada em tempo integral	Alta a quase total	Gerente do projeto	Em tempo integral
Virtual	Estrutura de rede com nós nos pontos de contato com outras pessoas	Baixa a moderada	Em tempo integral ou parcial	Baixa a moderada	Misto	Poderia ser em tempo integral ou parcial
Híbrido	Mix de outros tipos	Mista	Misto	Mista	Misto	Misto
EGP*	Mix de outros tipos	Alta a quase total	Função designada em tempo integral	Alta a quase total	Gerente do projeto	Em tempo integral

*EGP refere-se a um porfólio, programa ou escritório/ organização de gerenciamento de projetos

7.4. EAP (Estrutura Analítica do Projeto) ou WBS (Work Breakdown Structure)

- ENTREGAS DO PROJETO
- produtos, serviços ou resultados verificáveis devem ser usados como componentes da EAP.
- organiza e define o escopo total do projeto.
- Cada nível descendente representa uma definição cada vez mais detalhada do trabalho do projeto.
- O projeto é sucessivamente quebrado em níveis de entregas cada vez menores até o menor nível desejado para detalhamento do escopo do projeto.
- Os subprojetos devem estar presente na EAP (representa uma definição cada vez mais detalhada do trabalho do projeto).



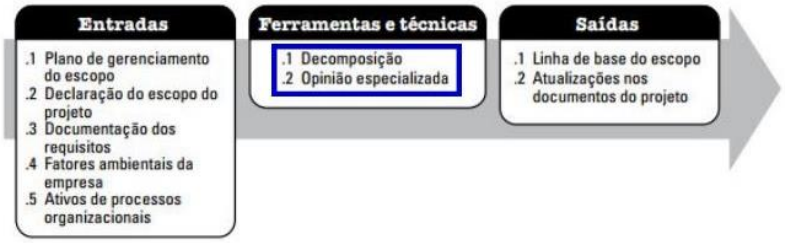


Figura 5-9. Criar a EAP: entradas, ferramentas e técnicas, e saídas

Ferramentas e técnicas da EAP:

- Decomposição:

Decomposição é a técnica usada para dividir e subdividir o escopo do projeto e suas entregas em partes menores e mais facilmente gerenciáveis.

- Opinião Especializada:

A opinião especializada é usada frequentemente para analisar as informações necessárias para decompor as entregas do projeto até as menores partes dos componentes a fim de criar uma EAP eficaz. Tal opinião e conhecimento especializado são aplicados aos detalhes técnicos do escopo do projeto e usados para reconciliar diferenças de opinião sobre a melhor maneira de decompor o escopo geral do projeto.

7.5. Processos de Gerenciamento de Projetos

- Os cinco grupos de processos de gerenciamento de projetos têm as seguintes características e principais ações:

I) INICIAÇÃO: Processos realizados para definir um novo projeto ou uma nova fase de um projeto existente, obtendo autorização para iniciar o projeto ou fase.

II) PLANEJAMENTO: Processos realizados para estabelecer o escopo total do esforço, definir e refinar os objetivos e desenvolver o curso de ação necessário para alcançar esses objetivos.

III) EXECUÇÃO: Processos realizados para completar o trabalho definido no plano de gerenciamento do projeto para atingir os objetivos do projeto.

IV) MONITORAMENTO E CONTROLE: Processos realizados para acompanhar, revisar e regular o progresso e o desempenho do projeto, identificar todas as áreas nas quais mudanças no plano sejam necessárias e iniciar as mudanças correspondentes. A "Validação do Escopo", que é o processo de formalização da aceitação dos entregáveis do projeto, está incluída aqui.

V) ENCERRAMENTO: Processos realizados para finalizar todas as atividades em todos os grupos de processos de gerenciamento de projetos, estabelecer o término formal do projeto ou fase e documentar as lições aprendidas.

- Principais ações dos cinco grupos de processos de gerenciamento de projetos:

I) INICIAÇÃO:
   - determinar os objetivos do projeto;
   - nomear o gerente do projeto;
   - identificar as partes interessadas;
   - documentar e publicar o Termo de Abertura do Projeto.
     - Termo de Abertura do Projeto deve ser elaborado pela entidade patrocinadora. Esse termo de abertura do projeto dá ao gerente do projeto a autoridade para planejar e executar o projeto.
     - fatores ambientais da empresa que podem influenciar o desenvolvimento do termo de abertura do projeto:
       - Fatores Ambientais:
         - Fatores físicos e infraestrutura,
         - Fatores humanos e sociais,
         - Fatores tecnológicos: Os três primeiros estão atrelados à organização (empresa) em si e podem influenciar o desenvolvimento, porém são mais fáceis de controlar.

         - Fatores externos: são os que a organização não tem o controle que são leis, normas, a expectativa das partes interessadas, entre outros.

II) PLANEJAMENTO:
   - Definir equipe
   - documentar e publicar a Declaração de Escopo do Projeto;
   - desenvolvimento da Estrutura Analítica do Projeto (EAP);
   - desenvolvimento do cronograma do projeto;
   - Fazer Diagrama de Rede
   - determinação de necessidades de recursos;
   - definição de compras e aquisições;
   - desenvolvimento do orçamento do projeto;
   - desenvolvimento e publicação do Plano de Gerenciamento do Projeto.
   - Desenvolver plano de gerenciamento de riscos

     Estrutura Analítica do Projeto (EAP):
     - Criar a EAP é o processo de decompor as entregas e o trabalho do projeto em componentes menores e mais facilmente gerenciáveis.
     - visão estruturada do que deve ser entregue.
     - decomposição hierárquica do escopo total do trabalho a ser executado
     - organiza e define o escopo total do projeto e representa o trabalho especificado na atual declaração do escopo do projeto aprovada

III) EXECUÇÃO:
   - alocação e desenvolvimento da equipe de execução do trabalho;
   - alocação dos recursos necessários à execução do trabalho;
   - execução do Plano de Gerenciamento do Projeto;
   - execução do trabalho do projeto.

IV) MONITORAMENTO E CONTROLE

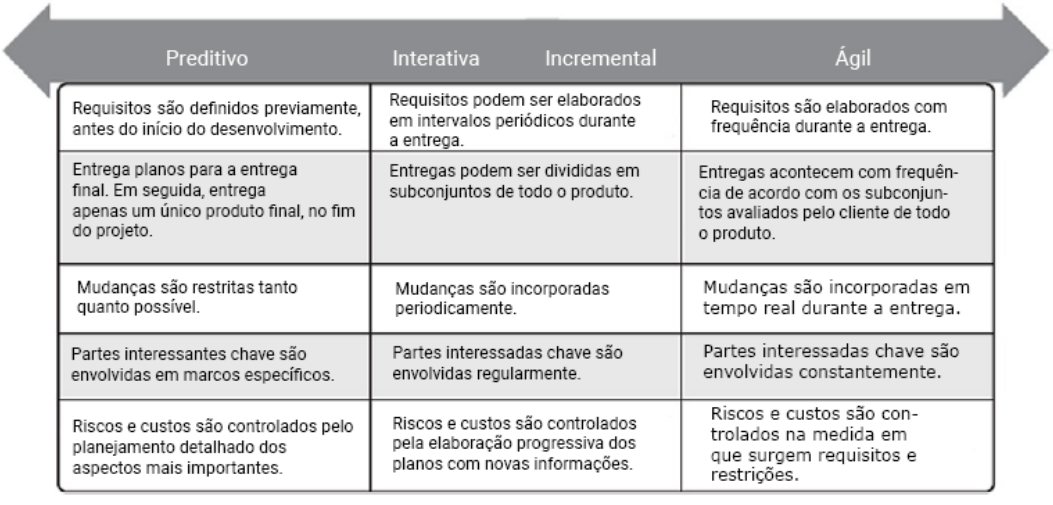
- medição do desempenho do projeto em comparação com as linhas de base;
- determinação de variações e consequentes recomendações de ações corretivas ou preventivas;
- avaliação das ações corretivas adotadas;
- auditoria de riscos;
- execução de relatórios de desempenho;
- administração de contratos.
- A validação do escopo é a principal atividade a ser realizada após a construção e antes de passar para os testes e implantação. O gerente do projeto deve garantir que o produto de software desenvolvido esteja de acordo com os requisitos definidos e obter a aprovação formal do cliente antes de prosseguir.

V) ENCERRAMENTO

- obtenção da aceitação formal das entregas;
- arquivamento dos registros do projeto;
- documentação das lições aprendidas;
- formalização do encerramento do projeto.

7.6. Ciclo de Vida do Projeto

Preditivo (orientados por um plano)	Caracterizam-se pela **ênfase na especificação de requisitos e planejamento detalhado durante as fases iniciais de um projeto.** Marcos para o envolvimento das partes interessadas também são planejados. À medida que a execução do plano detalhado progride, os processos de monitoramento e controle se detém nas mudanças limitantes que podem impactar o escopo, o cronograma ou o orçamento (PMBOK, 2017). **O escopo, prazo e custo do projeto são determinados nas fases iniciais do ciclo de vida.**
Adaptativo ou ágil	Caracterizam-se pela elaboração progressiva dos requisitos com base em planejamento iterativo curto e execução de ciclos. Os riscos e os custos são reduzidos pela elaboração progressiva dos planos iniciais. As partes interessadas chaves estão continuamente envolvidas e fornecem *feedback* frequente, e isso permite reagir mais rapidamente as mudanças e também resulta em melhor qualidade. O escopo detalhado é definido e aprovado antes do início de uma iteração.
Iterativo	O escopo do projeto geralmente é determinado no início do ciclo de vida do projeto, mas as estimativas de prazo e custos são normalmente modificadas à medida que a equipe do projeto compreende melhor o produto. As iterações desenvolvem o produto por meio de uma série de ciclos repetidos, enquanto os incrementos acrescentam sucessivamente à funcionalidade do produto.
Incremental	A entrega é produzida por meio de uma série de iterações que sucessivamente adicionam funcionalidade em um prazo predeterminado. A entrega contém a capacidade necessária e suficiente para ser considerada completa somente após a iteração final.
Híbrido	Combinação de um ciclo de vida adaptativo e um preditivo. Os elementos do projeto que sejam conhecidos ou que tenham requisitos estabelecidos seguem um ciclo de vida de desenvolvimento preditivo, e os elementos que ainda estiverem em evolução seguem um ciclo de vida de desenvolvimento adaptativo.



7.7. Caminho Crítico

- O caminho crítico é a sequência de atividades que devem ser finalizadas nas datas programadas para que o projeto possa ser concluído dentro do prazo final.
- Se o prazo final for excedido, é porque no mínimo uma das atividades do caminho crítico não foi concluída na data programada.
- sequência de atividades do plano, do início ao fim, que nos tomará o MAIOR tempo para sairmos do início e chegarmos ao fim do projeto.
- É necessário prestar muita atenção às tarefas do caminho crítico e aos recursos a elas atribuídos, para garantir que o projeto seja concluído dentro do prazo.

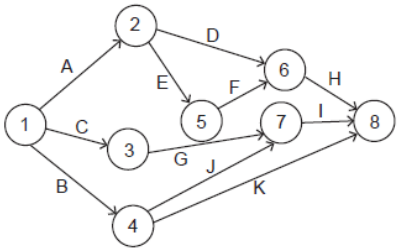
O Diagrama de REDES PERT (Program Evaluation Review Technique) é uma ferramenta de gerenciamento de projetos que fornece uma representação gráfica do cronograma do projeto.

- Princípio básico encontrar a sequência ótima das atividades, com redução de custo e prazo de execução

ETAPAS PERT:

1. Subdividem um projeto em suas tarefas componentes;
2. Classificam as tarefas em uma sequência lógica;
3. Determinam se as tarefas componentes são sequenciais ou simultâneas.
4. Diagramam as tarefas componentes.
5. Determinam o período mais curto em que um projeto pode ser completado (entendido aqui como o caminho crítico).

O CAMINHO CRÍTICO indica as áreas para a aplicação eficaz dos recursos, a fim de completar o projeto mais rápido. Se o administrador pretende concluir o projeto com mais rapidez, precisará encurtar o caminho crítico (identificar gargalos críticos).



- O prazo estimado pelo PERT-CPM é uma média, e a natureza das médias indica que existe uma probabilidade de 50% de o projeto ser concluído antes desse prazo e uma probabilidade de 50% de ser concluído depois. Portanto, pode-se dizer que existe uma probabilidade de 50% de o projeto ser concluído exatamente no prazo estimado.
- O Diagrama de PERT é usado para analisar a sequência de tarefas necessárias para a conclusão de um projeto, para estimar o tempo necessário para a conclusão de cada tarefa e para identificar as tarefas críticas que devem ser concluídas no tempo para evitar atrasos no projeto.
- As tarefas são representadas por nós (ou círculos) e são conectadas por setas que representam as dependências entre as tarefas. A sequência de tarefas que deve ser seguida para completar o projeto é chamada de caminho crítico.

- O Diagrama de PERT leva em consideração a incerteza e a variabilidade nos tempos de conclusão das tarefas, utilizando três estimativas de tempo para cada tarefa: otimista, mais provável e pessimista. Essas três estimativas são usadas para calcular uma média ponderada do tempo de conclusão da tarefa, que é usada para estimar o cronograma do projeto.

- O Diagrama de PERT é tipicamente usado nas fases de planejamento e execução do gerenciamento de projetos. Durante o planejamento, ajuda a definir a sequência de tarefas e a estimar o cronograma do projeto. Durante a execução, ajuda a monitorar o progresso do projeto e a identificar quaisquer desvios do cronograma planejado.

  Fundamentos do PERT-CPM:

  1. Cada atividade é sempre definida por uma única flecha e um par de eventos.

  2. Não pode haver no diagrama duas atividades iniciando e terminando no mesmo par de eventos.

  3. Quando as dependências entre as atividades forem apenas parciais, recorre-se à utilização da atividade fictícia para indicar essas dependências.

  4. Uma rede PERT-CPM não pode apresentar um circuito fechado.

  5. Toda vez que uma determinada atividade admitir deferentes etapas pode-se decompor essa atividade em tantas subatividades quantas forem essas etapas.

  6. Um diagrama PERT-CPM deve se iniciar com um único evento e terminar, também, com um só evento. A rede deve ser fechada.

  7. Tanto quanto possível não se devem cruzar quaisquer atividades.

7.8. Áreas de Conhecimento em Gerenciamento de Projetos

7.8.1. Gerenciamento de Integração do Projeto

- Desenvolver o termo de abertura do projeto;
- Desenvolver o plano de gerenciamento do projeto;
- Orientar e gerenciar a execução do projeto;

- AÇÃO CORRETIVA: orientação documentada para que o trabalho do projeto seja executado de modo que seu desempenho futuro esperado fique de acordo com o plano de gerenciamento.
- AÇÃO PREVENTIVA: uma orientação documentada para a realização de uma atividade que pode reduzir a probabilidade de consequências negativas associadas aos riscos do projeto.
- REPARO DE DEFEITO: a identificação documentada formalmente de um defeito em um componente do projeto com a recomendação para reparar o defeito ou substituir completamente o componente.

- Encerrar o projeto ou a fase.

- Trata da coordenação de todos os processos das 10 áreas de conhecimento e das atividades necessárias para identificar, definir, combinar, unificar e coordenar os vários processos e atividades dentro dos grupos de processos de gerenciamento do projeto.

- Lida com as ações integradoras que são essenciais para o término do projeto, para gerenciar com sucesso as expectativas das partes interessadas e atender aos requisitos.

- O Gerenciamento de INTEGRAÇÃO do Projeto consiste em garantir que TODAS as demais áreas estejam integradas em um TODO único. Seu objetivo é estruturar todo o projeto de modo a garantir que as necessidades dos envolvidos sejam atendidas.

- Para a integração gerencial harmônica do todo, é necessário o comprometimento da organização e o suporte dos altos executivos.

7.8.2. Gerenciamento do Escopo do Projeto

- Coletar os Requisitos" é voltado para determinar, documentar e gerenciar as necessidades e requisitos das partes interessadas a fim de atender aos objetivos do projeto, sendo que o mesmo possui como saídas: "Documentação dos Requisitos" e "Matriz de Rastreabilidade dos Requisitos".

- Sobre a Matriz de Rastreabilidade dos Requisitos, de acordo com o PMBOK (7ª Edição), esta consiste em uma tabela que liga os requisitos dos produtos desde as suas origens até as entregas que os satisfazem.

- O escopo do projeto são as necessidades das partes interessadas que devem ser atendidas ao final do projeto. Em outras palavras, contempla tudo o que deve ser atendido para o sucesso do projeto.

- O gerenciamento do escopo do projeto inclui os processos necessários para assegurar que o projeto inclui todo o trabalho necessário, e apenas o necessário, para terminar o projeto com sucesso.

- Refere-se à definição e ao controle do que está ou não está incluído no projeto.

- Escopo do produto: as características e funções que descrevem um produto, serviço ou resultado;

- Escopo do projeto: o trabalho que precisa ser realizado para entregar um produto, serviço ou resultado com as características e funções especificadas

Técnicas de tomada de decisão em grupo como um processo de avaliação de múltiplas alternativas utilizadas para gerar, classificar e priorizar os requisitos do produto.

Conforme o PMBOK (5º edição), temos:

▪Unanimidade.
Uma decisão alcançada de tal forma que todos concordam com um único curso de ação.

▪Maioria.
Uma decisão alcançada com o apoio de mais de 50% dos membros do grupo. Um grupo com um número ímpar de participantes pode garantir que uma decisão será alcançada, ao invés de resultar em um empate.

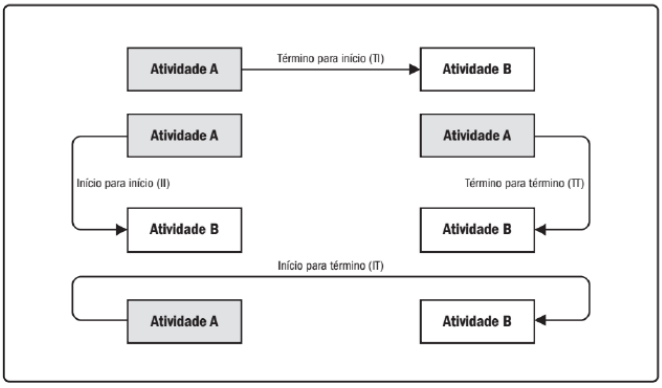
▪Pluralidade.
Uma decisão é tomada pelo maior bloco do grupo, mesmo que a maioria não seja alcançada. Este método é geralmente usado quando o número de opções nomeadas for maior que duas.

▪Ditadura.
Neste método um indivíduo decide pelo grupo.

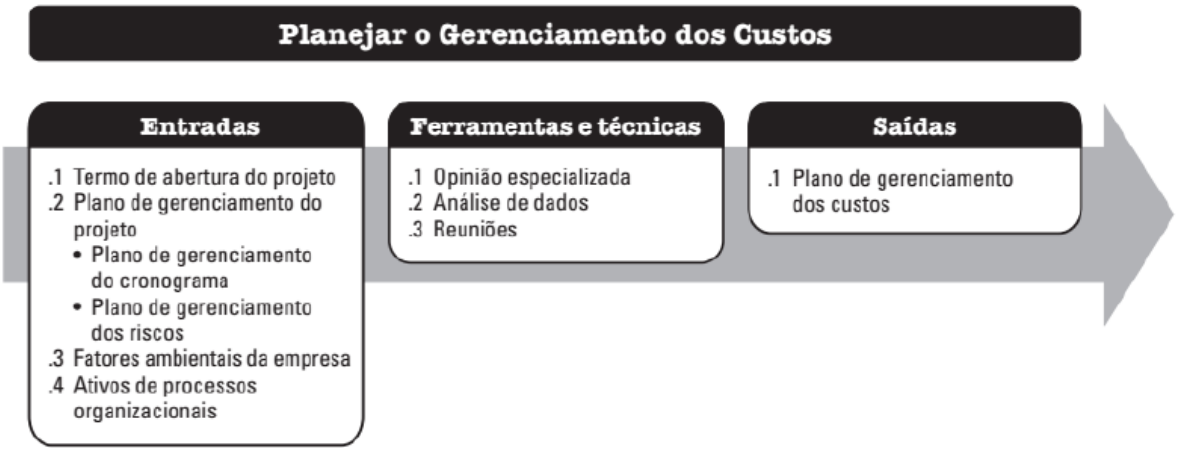
7.8.3. Gerenciamento do Cronograma do Projeto

- Diagrama de rede do cronograma do projeto: Um diagrama de rede do cronograma do projeto é uma representação gráfica das relações lógicas, também chamadas de dependências, entre as atividades do cronograma do projeto.

- Diagrama de Rede do Cronograma do Projeto as Antecipações e Esperas exemplificadas.



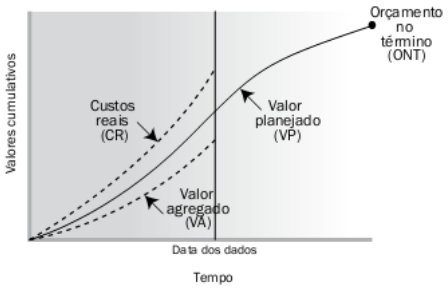
7.8.4. Gerenciamento dos Custos do Projeto



O plano de gerenciamento de custos é um documento que define como os custos serão planejados, estruturados, estimados, orçados e controlados no projeto. Ele faz parte do plano de gerenciamento do projeto e orienta como os custos serão gerenciados e controlados durante todo o ciclo de vida do projeto. O plano de gerenciamento de custos pode incluir detalhes sobre:

- Metodologias para planejar e gerenciar custos.
- Links para políticas de custos organizacionais.
- Descrição dos processos de estimativa, orçamento e controle de custos.
- Diretrizes para medições de desempenho de custos.

Curva S - É uma representação gráfica dos custos cumulativos, horas de mão-de-obra, percentual de trabalho ou outras quantidades, indicando sua evolução no tempo. Usada para representar o valor planejado, o valor agregado, e o custo real de um trabalho de projeto. O nome se origina do formato parecido com um S da curva (mais plana no início e no final, mais inclinada no centro) gerada para representar um projeto que começa lentamente, se agiliza e em seguida diminui o ritmo. Em suma, mostra o que foi planejado e o que foi executado.



As entradas para o processo de planejamento do gerenciamento de custos incluem:

- Plano de gerenciamento do projeto: O plano de gerenciamento do projeto fornece o contexto geral para o plano de gerenciamento de custos e pode conter requisitos ou orientações específicas para o gerenciamento de custos.

- Fatores ambientais da empresa: Estes são as condições que não estão sob o controle da equipe do projeto, mas que influenciam a forma como o projeto é gerenciado. Incluem cultura organizacional, estrutura organizacional, infraestrutura, recursos humanos, mercado, legislação, entre outros.

- Ativos de processos organizacionais: Incluem políticas, procedimentos e planejamentos de custos anteriores da organização que podem ser usados para informar o plano de gerenciamento de custos.

- Termos de Abertura do Projeto (Project Charter): O TAP (Project Charter) é um documento que oficialmente autoriza um projeto ou uma fase do projeto e documenta os requisitos iniciais que satisfazem as necessidades e expectativas das partes interessadas. Ele define a visão geral do projeto, incluindo os objetivos, o propósito, os principais stakeholders, os requisitos de alto nível, os riscos identificados e o patrocinador comprometido em fornecer os recursos necessários. O Project Charter pode influenciar o planejamento do gerenciamento de custos, pois contém informações sobre o escopo, riscos e partes interessadas que podem afetar as decisões de custos.

7.8.5. Gerenciamento da Qualidade do Projeto

- O gerenciamento da qualidade do projeto trabalha para garantir que os requisitos do projeto, incluindo os requisitos do produto, sejam cumpridos e validados. Uma visão geral dos processos de gerenciamento da qualidade do projeto inclui:
    - Planejar o Gerenciamento da Qualidade — O processo de identificação dos requisitos e/ou padrões da qualidade do projeto e suas entregas, além da documentação de como o projeto demonstrará a conformidade com os requisitos de qualidade.
    - Realizar a Garantia da Qualidade — O processo de auditoria dos requisitos de qualidade e dos resultados das medições do controle de qualidade para garantir o uso dos padrões de qualidade e das definições operacionais apropriadas.
    - Realizar o Controle da Qualidade — O processo de monitoramento e registro dos resultados da execução das atividades de qualidade para avaliar o desempenho e recomendar as mudanças necessárias.

O processo de Gerenciar a Qualidade, conforme descrito no PMBOK 6ª edição, utiliza várias técnicas de representação de dados para auxiliar no controle e melhoria da qualidade do projeto. Aqui estão algumas das técnicas-chave:

- Histogramas: Um histograma é uma representação gráfica que organiza um grupo de dados em uma série de intervalos. É útil para visualizar a distribuição de frequências de um conjunto de dados e são comumente usados na gestão da qualidade para identificar padrões que podem não ser evidentes em uma tabela de dados.

- Diagramas de Pareto: Este é um tipo especial de histograma usado para identificar e priorizar áreas problemáticas. Um diagrama de Pareto organiza e exibe informações de tal maneira que é fácil ver quais são os problemas mais significativos. Isso é baseado no princípio de Pareto, que afirma que 80% dos problemas são geralmente causados por 20% das causas.
    - OBJETIVO PARETO: Construção de critérios de prioridade para a correção de defeitos. O diagrama de Pareto é efetivamente usado para estabelecer prioridades. Ele é uma ferramenta gráfica que ajuda a identificar quais problemas devem ser resolvidos primeiro, baseando-se no princípio de Pareto (também conhecido como regra 80/20), que afirma que 80% dos problemas são geralmente causados por 20% das causas.

- Gráficos de controle: Estes são usados para determinar se um processo é estável ou tem desempenho previsível. Eles destacam pontos fora dos limites de controle e identificam a tendência de pontos em relação a esses limites.

- Diagramas de dispersão: Um diagrama de dispersão é usado para mostrar se existe uma relação entre duas variáveis. Ele pode mostrar se uma mudança em uma variável é correlacionada com uma mudança em outra variável.

- Gráficos de caixa (Box Plots): Estes são gráficos que apresentam a distribuição de dados baseada em um resumo de cinco números ("mínimo", primeiro quartil (Q1), mediana, terceiro quartil (Q3) e "máximo").

- Diagramas de afinidade: Estes são usados para organizar grandes quantidades de dados em grupos para revisão e análise. Eles podem ajudar a identificar áreas onde a qualidade pode ser melhorada.

- Fluxogramas: Estes diagramas são usados para analisar como os diferentes elementos de um sistema se relacionam e interagem entre si e determinar relação de causalidade entre etapas que não estão em conformidade.

- Matriz de risco: É uma ferramenta visual que ajuda na classificação e gerenciamento de riscos. Os riscos são classificados com base em sua probabilidade de ocorrência e impacto, ajudando as equipes de projeto a se concentrar nos riscos mais críticos.

Essas técnicas de representação de dados são ferramentas essenciais para a gestão da qualidade, pois permitem que os gerentes de projeto visualizem e compreendam complexidades e tendências nos dados de qualidade do projeto.

7.8.6. Gerenciamento de Recursos do Projeto

7.8.7. Gerenciamento das Comunicações

7.8.8. Gerenciamento de Riscos do Projeto

A análise quantitativa de riscos é uma técnica usada para numerar os riscos de um projeto. Ela oferece uma maneira de avaliar o impacto dos riscos identificados em termos de custo, tempo, escopo e qualidade, e permite que a equipe do projeto determine a probabilidade de alcançar seus objetivos. Algumas das análises quantitativas de riscos mais comumente usadas incluem:

- Análise de Sensibilidade: A análise de sensibilidade ajuda a determinar quais riscos terão o maior impacto no projeto. Isso é geralmente representado como um gráfico tornado ou um diagrama de aranha, e ajuda a entender como a incerteza em diferentes aspectos do projeto pode influenciar seus objetivos.

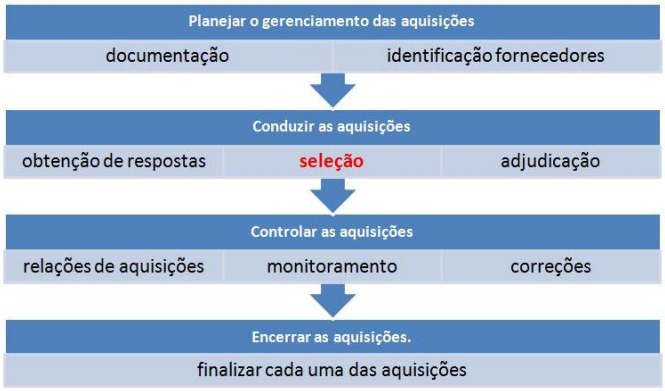
- Modelagem e Simulação: A simulação, geralmente feita através de Monte Carlo, usa uma representação computacional do projeto para simular a combinação de efeitos de diferentes riscos. Através desta técnica, o software calcula e registra o impacto potencial dos riscos, geralmente em termos de custo e programação. Isso permite que os gerentes de projeto avaliem a viabilidade geral do projeto e determinem a probabilidade de concluir o projeto dentro do prazo e do orçamento.

- Análise de Valor Monetário Esperado (EMV): EMV é uma técnica estatística que calcula o resultado médio quando o futuro inclui cenários que podem ou não acontecer (ou seja, cenários de risco). Isso é feito multiplicando o valor de cada resultado possível pelo seu valor de probabilidade e somando esses produtos.

- Análise de Decisão: Esta técnica envolve a avaliação de várias decisões com base em sua qualidade e completude. A análise de decisão pode ser auxiliada por técnicas como a análise de árvore de decisão, que estrutura uma decisão como uma árvore de possíveis opções.

- Análise de Probabilidade e Impacto: Nessa análise, cada risco é avaliado com base em sua probabilidade de ocorrência e seu impacto potencial no projeto se ele ocorrer. Os riscos são então priorizados com base nessa avaliação.

7.8.9. Gerenciamento das Aquisições do Projeto



- Planejar o gerenciamento das aquisições: O processo de documentação das decisões de compras do projeto, especificando a abordagem e identificando fornecedores em potencial.

- Conduzir as aquisições: O processo de obtenção de respostas de fornecedores, seleção de um fornecedor e adjudicação de um contrato.

- Controlar as aquisições: O processo de gerenciamento das relações de aquisições, monitoramento do desempenho do contrato e realizações de mudanças e correções nos contratos, conforme necessário.

- Encerrar as aquisições: O processo de finalizar cada uma das aquisições do projeto.

O processo de Planejar Aquisições, de acordo com o Guia PMBOK 6ª edição, é parte do Gerenciamento de Aquisições do Projeto e envolve a determinação do que será adquirido, quando e como. Isso pode envolver a decisão sobre se deve-se realizar uma atividade internamente ou contratar uma empresa externa para fazê-lo.

Os benefícios do processo de Planejar Aquisições são as definições de:

- o que adquirir;
- como fazer a aquisição;
- a quantidade necessária;
- quando efetuar a aquisição.

7.8.10. Gerenciamento das Partes Interessadas do Projeto

7.8.11. Gestão de Continuidade de Negócio

- visa não permitir a interrupção das atividades do negócio e proteger processos críticos contra os efeitos de falhas ou desastres. Mesmo em situações em que há a interrupção de um serviço ou o comprometimento de seus níveis de qualidade, a gestão da continuidade busca restabelecer as operações normais no tempo mais breve possível.

Planos de continuidade de negócios estão diretamente relacionados com métodos de recuperação de desastres: o primeiro deles corresponde a um plano de manutenção dos processos de negócio em funcionamento, preventivamente, ou seja, sem que um desastre tenha ocorrido. Já o segundo plano aborda técnicas e procedimentos de recuperação dos sistemas para o ponto no qual a empresa se encontrava no momento anterior ao desastre, ou algo próximo deste ponto.

A palavra desastre pode representar ocorrências de fenômenos naturais (enchentes, incêndios e terremotos), mas também é usada para definir desastre computacional, descrito por Hiles como sendo "um evento que causa a perda do serviço computacional, ou uma parte significativa dele, ou de comunicações ou aplicações, por um período de tempo que priva a organização de atingir sua missão ou que coloca em risco os seus negócios".

Infraestrutura computacional pode ser analisada como um conjunto de hardware e software, mas envolve uma estrutura muito maior e mais complexa, como o local físico de hospedagem dos dispositivos físicos, o local de trabalho dos colaboradores da empresa, a rede elétrica e os próprios funcionários de todas as áreas da empresa, inclusive da área de TI.

Assim sendo, desastre pode ser desde a corrupção de uma base de dados até a destruição total da instalação física de uma empresa acometida por um incêndio, o que pode significar até a sua falência, em decorrência do nível de disponibilidade que a empresa possua.