

The background of the slide is a dark, blue-tinted image of an industrial setting. It features several robotic arms and mechanical components, suggesting a manufacturing or automation environment. Overlaid on this background is a dark silhouette of a world map, centered on the Atlantic Ocean. The text is white and positioned in the upper half of the image.

Paradigmas de Linguagens

Aula 01

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO





Não seria melhor usar
este tempo para
estudar uma única
linguagem?



Razões para Estudar Linguagens de Programação

Maior capacidade de desenvolver soluções computacionais para problemas.

- Uma maior **compreensão sobre os conceitos** de uma LP pode **aumentar nossa habilidade** em como **pensar e resolver problemas**.
- Por exemplo, o conhecimento do conceito de tipos abstratos de dados estimulam a utilização desse método de programação mesmo em LPs que não possuem mecanismos específicos para a sua implementação.

Razões para Estudar Linguagens de Programação

Maior habilidade ao usar uma LP

O **maior entendimento** a respeito das **funcionalidades e implementação** de uma LP possibilita ao programador construir **programas melhores e mais eficientes**.

- Por exemplo, conhecendo como as LPs são implementadas pode-se entender porque algoritmos recursivos são menos eficientes que os iterativos correspondentes.

```
5! = 5*4*3*2*1=120
scanf("%d",&numero); // 5
fatorial = 1;
for(i=numero;i>1;i=i-1)
{
    fatorial = fatorial * i;
}
//fim do laço for
printf("fatorial de 5 é: %d",fatorial)
```

```
fatorial = 5//20//60//120//120
l = 5//4//3//2//1//0
```



Razões para Estudar Linguagens de Programação

Maior capacidade para escolher LPs apropriadas

- Conhecer os recursos oferecidos por uma linguagem e saber como esses recursos são implementados pode determinar uma boa escolha da linguagem de programação a ser usada em um projeto.
- Por exemplo, saber que C não realiza checagem dinâmica dos índices de acessos a posições de vetores pode ser decisivo para escolher essa linguagem em aplicações de tempo real que fazem uso frequente de acessos vetoriais.

Razões para Estudar Linguagens de Programação

Maior habilidade para
aprender novas LPs

- Por exemplo, programadores que aprenderam os conceitos de orientação a objetos tem maior facilidade para aprender C++ , C#, e JAVA.

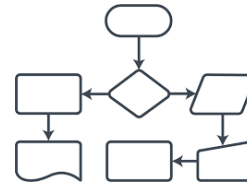


Razões para Estudar Linguagens de Programação

Maior habilidade para projetar novas LPs

- Muito embora poucos profissionais de computação tenham a oportunidade, ao longo de suas carreiras, de participar da criação de uma linguagem de programação de propósito geral, não é raro se depararem com situações nas quais é necessário projetar **linguagens para um propósito específico**.
- Por exemplo, ao construir as interfaces com o usuário de sistemas pode ser necessário projetar e implementar uma linguagem de comandos para comunicação entre o usuário e o sistema.

Algoritmos



Um algoritmo é uma **sequência de passos** que **resolve** algum **problema** ou alcançar algum objetivo, essas **instruções** podem ser **executadas** por um **computador** ou até mesmo por um ser humano.

Uma receita de bolo é um exemplo de algoritmo, onde cada passo da preparação do bolo, corresponde a uma instrução do algoritmo.

É importante salientar que assim como a receita de um bolo, não é um bolo, apenas diz os passos para se construir um bolo, um algoritmo simplesmente **diz o que deve ser feito**, para solucionar algum problema ou alcançar um objetivo.



Programas

Os programas de computadores são **algoritmos** escritos numa **linguagem de computador**.

Existem **diversas linguagens** de programação, tais como Java, C, C++, Pascal.

Essas linguagens são interpretadas e executadas por uma máquina, no caso um computador.

Programas

É importante ressaltar que o computador executa exatamente os comandos escritos nas linguagens de programação. Assim um comando colocado em lugar errado levará ao erro na execução do programa.

Outro fator de erros é a desobediência às regras de escrita, impedindo a execução do programa.



Representação da lógica de programação

Para desenvolver um programa é preciso ter um bom raciocínio lógico, para representarmos esta lógica utilizamos as simbologias dos algoritmos.

Na linguagem computacional temos três tipos de algoritmos mais utilizados, são eles: **descrição narrativa, fluxograma e pseudocódigo ou português.**

Descrição narrativa

A descrição narrativa consiste em entender o problema proposto e **escrever sua solução** através da **linguagem natural**, ou seja, a língua portuguesa.

➤ Vantagens

- Não é necessário aprender **nenhum conceito novo**. Basta escrever da maneira como se fala.

➤ Desvantagens

- Esta linguagem dá margem para **vários tipos de interpretação**. Portanto, se você não for claro o suficiente isto poderá trazer dificuldade a transição desse algoritmo para a programação

Fluxograma

Através de um **conjunto de símbolos** você define os **passos** para a solução do problema apresentado.




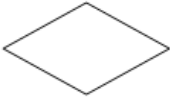


➤ **Vantagens**

- O **entendimento** é bem mais **simples**

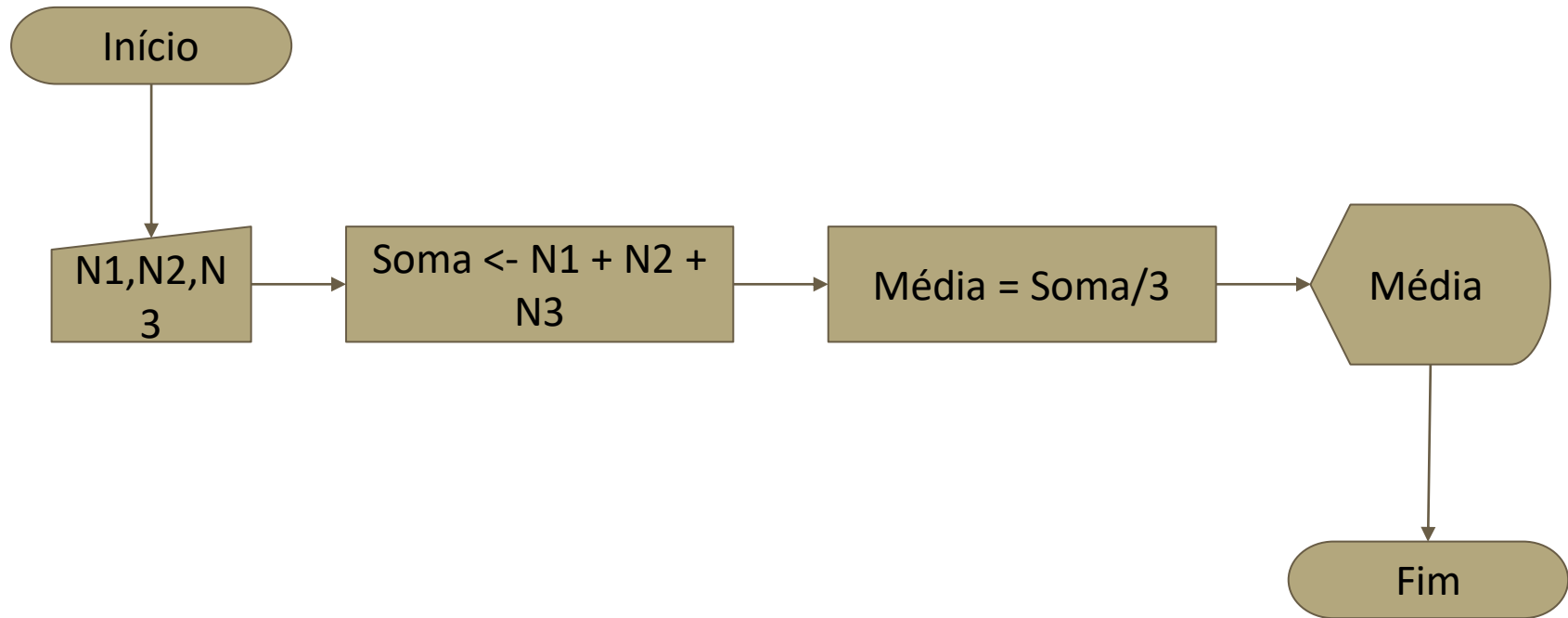
➤ **Desvantagens**

- É necessário que se tenha **conhecimento dos símbolos.**
- **Modificações** e edições **difíceis**

Fluxograma

	Indica Início e Fim do algoritmo
	Indica cálculo e atribuição de valores
	Indica a entrada de dados
	Indica uma decisão com possibilidades de desvios.
	Indica saída de dados
	Indica o fluxo de dados. Serve também para conectar os blocos ou símbolos existentes.

Fluxograma - Calcular a média entre 3 números



Pseudocódigo / Portugol

Esta estrutura é escrita em **português**, muito **próxima à linguagem natural**, utilizando **regras predefinidas** para escrevê-la. Utilizando o enunciado acima, somar dois números.

- **Vantagens**

- O **entendimento** é bem mais **simples**

- **Desvantagens**

- É necessário que se tenha **conhecimento** de **pseudocódigo**, que veremos mais adiante

Pseudocódigo / Portugol

Algoritmo soma_números;

Variáveis

num1,num2,soma: inteiro;

Início

Escreva ("Digite dois números")

Leia num1, num2;

Soma<-num1+num2;

Escreva ("Soma", Soma)

Fim.

Linguagens de Programação

Uma Linguagem de Programação (LP) é um instrumento utilizado pelo profissional de computação para **escrever programas**, isto é, **conjuntos de instruções** a serem seguidas pelo computador para realizar um determinado processo.



Exemplos de Linguagens de Programação

Não confunda LP com IDE

IDE, do inglês *Integrated Development Environment* ou **Ambiente de Desenvolvimento Integrado**, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de **agilizar** este processo.

Exemplos:

Netbeans, Visual Studio, Code::Blocks, Eclipse, Dev-C++

Porque existem tantas linguagens de programação?

Para corrigir
as falhas das
anteriores

Resposta
certa ou
errada?

Porque existem tantas linguagens de programação?

A maioria surgiu para **sanar uma necessidade** diferente numa determinada área **numa época específica**.

Exemplo:

Java

Nasceu para substituir C / C++?

- Nasceu pela orientação a objetos?
- NÃO, nasceu para oferecer Portabilidade

Porque existem tantas linguagens de programação?

Algumas outras linguagens foram criadas para competir.

Exemplo:

C#, veio para competir com Java

Qual é a melhor linguagem de programação?



Resposta: A que tem menos falhas.



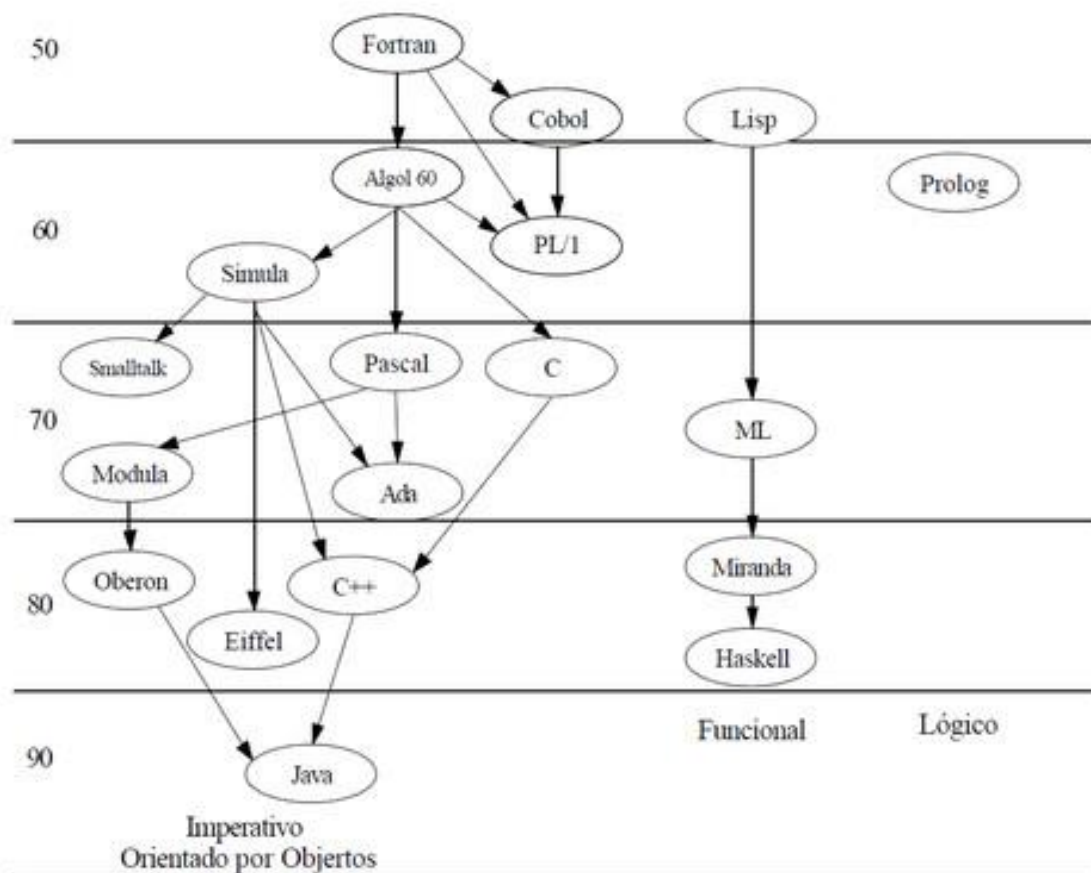
Resposta certa ou errada?

Qual é a melhor linguagem de programação?

Cada linguagem tem pontos fortes e fracos.

Não há a melhor linguagem.

Uma linguagem é a melhor para determinado problema numa determinada situação



Paradigmas

De modo geral, pensamos em um “paradigma” como um **padrão de pensamento que guia um conjunto de atividades relacionadas**.

Dá-se o nome de paradigma a um **conjunto de características que servem para categorizar um grupo de linguagens**. Existem diversas classificações de paradigmas de LPs.

Paradigmas

- Programação imperativa
- Programação orientada a objeto
- Programação funcional
- Programação lógica

Algumas linguagens de programação são intencionalmente projetadas para suportar mais de um paradigma. Por exemplo, C++ é uma linguagem imperativa e orientada a objetos

Programação Imperativa

É o paradigma **mais antigo**, já que está fundamentado no modelo computacional clássico de “von Neumann-Eckert” .

Nesse modelo, tanto o programa quanto as suas variáveis são armazenados juntos, e o programa contém uma série de comandos para executar cálculos, atribuir valores a variáveis, obter entradas, produzir saídas ou redirecionar o controle para outro ponto nessa série de comandos.

Programação Imperativa

A abstração procedural é um componente para a programação imperativa assim como as atribuições, os laços, as sequências, os comandos condicionais e a manipulação de exceções.

As linguagens de programação imperativa predominantes incluem Cobol, Fortran, C, Ada e Perl.

Programação Orientada a Objeto

- A programação orientada a objeto (POO) fornece um modelo no qual um **programa é uma coleção de objetos** que **interagem entre si, passando mensagens que transformam seu estado**.
- Neste sentido, a passagem de mensagens permite que objetos de dados se tornem ativos em vez de passivos.

Programação Orientada a Objeto

- Essa característica ajuda a distinguir melhor a programação OO da imperativa. A classificação de objetos, herança e a passagem de mensagens são componentes fundamentais da programação OO.

Exemplos:

Linguagens orientadas a objetos importantes são Smalltalk, C++, Java e C#.

Programação Funcional

Modela um problema computacional como uma **coleção de funções matemáticas**, cada uma com um **espaço de entrada** (domínio) e **resultado** (faixa). Isso separa a programação funcional das linguagens que possuem o comando de atribuição.

As funções interagem e combinam entre si usando composição funcional, condições e recursão. Importantes linguagens de programação funcional são LiSP, Scheme, Haskell e ML.

Programação Lógica

Permite a um programa modelar um problema declarando qual resultado o programa deve obter, em vez de como ele deve ser obtido.

Às vezes, essas linguagens são chamadas de baseadas em regras, já que as declarações do programa se parecem mais com um conjunto de regras ou restrições sobre o problema, em vez de uma sequência de comandos a serem executados.

Programação Lógica

Interpretar as declarações de um programa lógico cria um conjunto de todas as soluções possíveis para o problema que ele especifica.

A programação lógica também **fornece um veículo natural para se expressar o não-determinismo**, o que é apropriado para problemas cujas especificações sejam incompletas. A principal linguagem de programação lógica é o Prolog.

Tópicos especiais

A conceptual illustration on a dark teal background. A wide staircase leads from the bottom towards the top. At the very top of the stairs sits a golden trophy. A person in a dark suit is seen from behind, walking up the stairs. To the right of the main staircase, another person in a dark suit is running towards the right side of the frame. The overall theme is one of striving and achievement.

Além desses quatro paradigmas, diversos tópicos essenciais para o projeto de linguagem de programação merecem ser estudados.

Esses tópicos tendem a ser universais, na medida em que aparecem em dois ou mais dos paradigmas anteriores, em vez de em apenas um.

Exemplo:

- Manipulação de eventos
- Concorrência



Manipulação de Eventos

A manipulação de eventos **ocorre com programas que respondem a eventos gerados em uma ordem imprevisível**. De certa forma, um programa orientado a evento é apenas um programa cujo comportamento é inteiramente determinado por questões de manipulação de eventos.

A manipulação de eventos está frequentemente acoplada ao paradigma orientado a objeto (por exemplo, Java applets), embora ela ocorra dentro dos paradigmas imperativos também (por exemplo, Tcl/Tk).



Manipulação de Eventos

Os eventos se originam de ações dos usuários na tela (cliques de mouse ou pressionamentos de teclas, por exemplo) ou então de outras fontes (como leituras de sensores em um robô).

Linguagens importantes que suportam a manipulação de eventos incluem Visual Basic, Java e Tcl/Tk.



Concorrência

- A programação concorrente pode ocorrer dentro do paradigma imperativo, orientado a objeto, funcional ou lógico.
- A concorrência **ocorre quando o programa possui uma coleção de elementos assíncronos que podem compartilhar informações** ou sincronizar-se entre si em intervalos de tempo.
- A concorrência também **ocorre dentro de um processo individual como a execução paralela de diferentes iterações de um laço.**



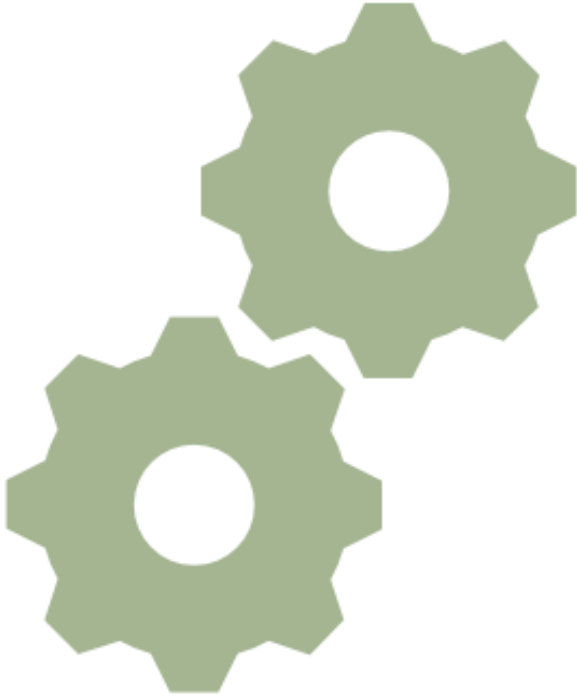
Concorrência

Linguagens de programação concorrente incluem SR (Andrews e Olsson, 1993), Linda (Carriero e Gelenter, 1989) e High Performance Fortran (Adams e outros, 1997).

Todo e qualquer programa escrito em uma LP necessita ser traduzido para a linguagem de máquina para ser executado.

Para fazer isto é necessário aplicar um programa (ou conjunto de programas) que receba como entrada o código fonte do programa a ser traduzido e gere o código traduzido na linguagem de máquina.

Compiladores e Interpretadores



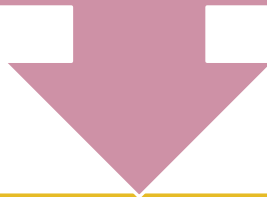
Compilação

O processo de compilação **efetua a tradução integral do programa fonte para o código de máquina**. Uma vez traduzido, o **programa em linguagem de máquina pode ser executado diretamente**.

A grande **vantagem** desse método de tradução é a **otimização da eficiência na execução dos programas**.

Compilação

A execução é rápida porque não se necessita fazer qualquer tradução durante a execução e porque boa parte das verificações de erros já podem ser efetuadas durante a tradução.



Além disso, o próprio tradutor tem mais liberdade para realizar otimizações na geração do código executável, uma vez que pode considerar o código fonte globalmente.

Compilação

A principal **desvantagem** do método de compilação é a **não portabilidade** do código executável para máquinas com arquitetura diferenciada daquela na qual ele foi compilado.

Por exemplo, um programa em C compilado sobre o ambiente Linux não é executado sobre o ambiente Windows e vice-versa.

Interpretação pura

No processo de interpretação pura, um programa **interpretador age como um simulador de um computador virtual** que entende as instruções da LP.

Basicamente, cada **instrução do código fonte é traduzida para a linguagem de máquina** quando essa instrução necessita ser executada. Imediatamente após a tradução, o código gerado é executado.

Interpretação pura

Deste modo, em contraste com a compilação, a tradução e a **execução** de programas interpretados podem ser vistos como um **processo único**, não existindo etapas separadas de tradução e de execução.

Ex: Perl, Python, PHP, JavaScript

Interpretação pura

- **Vantagens:**
 - **facilidade para prototipação**, visto que se pode executar comandos e partes do programa assim que são construídos, verificando se atuam corretamente; a facilidade de depuração, visto que as mensagens de erro podem se referir diretamente ao código fonte;
 - **a facilidade de escrever programas mais flexíveis**, visto que o interpretador da LP está presente durante a execução.

Interpretação pura

- **Desvantagem:**
 - em relação à compilação é a **execução muito mais lenta** do programa. A razão para essa lentidão decorre da necessidade do interpretador decodificar comandos complexos da LP, verificar erros do programa e gerar código em linguagem de máquina durante a própria execução do programa.



Híbrido

Processo que combina **tanto a execução eficiente quanto a portabilidade** de programas através da aplicação combinada dos dois métodos anteriores.

A base para o método híbrido é a **existência de um código intermediário** mais fácil de ser interpretado e, ao mesmo tempo, não específico de uma plataforma computacional



Híbrido

Embora ainda de **execução mais lenta** que o **código compilado**, a **interpretação do código intermediário** é muito **mais rápida** que a **interpretação pura** do código fonte porque **as instruções do código intermediário** são muito **mais simples** que **as do código fonte** e porque a maior parte das verificações de erro é realizada já na etapa de compilação



Híbrido

Por sua vez, como o **código intermediário não é específico para uma plataforma**, os programas já compilados para este código podem ser portados para as mais diferentes plataformas **sem necessidade de adaptação** ou mesmo **recompilação**, bastando que **exista um interpretador do código intermediário** instalado na plataforma onde se deseja executar o programa.

Ex: Java, C#

Perguntas

- 1) Quais são os tipos de paradigmas existentes?
- 2) Qual a diferença entre compilação e interpretação de programas?
- 3) O que é um híbrido? Qual a sua vantagem?
- 4) Por quê existem tantas linguagens?
- 5) Qual é a melhor forma de escolher uma linguagem?

Processo de Desenvolvimento de Programas

O processo de desenvolvimento de software é normalmente compreendido em cinco etapas:

- Especificação de requisitos
- Projeto de software
- Implementação
- Validação
- Manutenção

Especificação de requisitos

Nessa etapa **se identifica o que o software deverá realizar em termos de funcionalidades.**

É necessário especificar o ambiente no qual o software atuará, quais serão as suas atividades, quais os impactos que deverá produzir e de que maneira ele deverá interagir com os usuários.

Especificação de requisitos

Em particular, dever-se-á especificar os **requisitos de desempenho do sistema** em termos de tempo de resposta desejado, espaço de memória requerido e também de interação com outros dispositivos e usuários.

Especificação de requisitos

Outra atividade importante nessa etapa é a realização de um **estudo de viabilidade e custo do software a ser desenvolvido**.

Esse estudo tem por objetivo responder se a confecção e implantação do software é viável técnica, cronológica e socialmente, bem como determinar, através da estimativa do custo de desenvolvimento do software, se a sua construção é viável economicamente.

Tendo por base os documentos de especificação de requisitos, pode-se projetar o sistema de programação.

O projeto envolve a escolha de um **método de projeto e sua aplicação na especificação da arquitetura do software e seus procedimentos**, das suas estruturas de dados e de suas interfaces.

Projeto de software

O **resultado** desta fase é um **documento de especificação do projeto do sistema**, identificando:

- Módulos que compõem a arquitetura do sistema;
- Estruturas de dados utilizadas por cada módulo;
- Interfaces de comunicação entre módulos (interfaces internas), com outros sistemas (interfaces externas) e com as pessoas que o utilizam (interface com o usuário).

Projeto de software

Também faz parte desse documento as descrições procedimentais de cada módulo da arquitetura. O principal papel das LPs nessa etapa é dar suporte ao método de projeto.

Pode-se observar que algumas linguagens de programação são mais adequadas quando se utilizam certos métodos de projeto.

Projeto de software

Por exemplo, enquanto C é mais adequada ao método de projeto hierárquico-funcional, JAVA é mais adequada ao método orientado a objetos.

Existem várias ferramentas CASE que oferecem suporte às atividades dessa etapa. Muitas dessas ferramentas já geram parte da codificação do software em uma LP.

Projeto de software

Implementação

A etapa de implementação é onde ocorre a **programação dos módulos do software**.

Obviamente, LPs são essenciais nessa etapa uma vez que os programas devem ser escritos em uma linguagem.

Implementação

Essa etapa é a mais atendida por ferramentas, tais como editores de texto que destacam os vocábulos da linguagem e identam automaticamente o texto, analisadores léxicos, sintáticos e semânticos de programas e bibliotecas de subprogramas e módulos.

Validação

- O propósito dessa etapa é verificar se o sistema satisfaz as exigências das especificações de requisitos e de projeto.
- Teste do sistema contra as especificações.
- Três tipos de testes:
 - teste de modulo;
 - teste de integração;
 - teste de sistema.

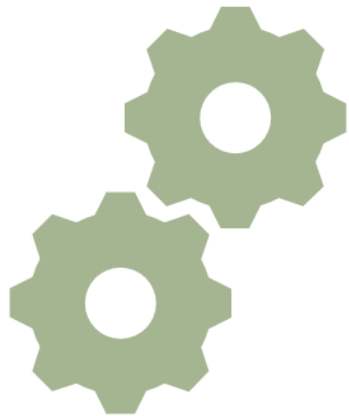


Validação

- **Teste de modulo:** verifica se cumpre o que lhe foi especificado.
- **Teste de integração:** verifica se os módulos se integram apropriadamente, isto é, se eles interagem tal como estabelecido nas especificações de suas interfaces.
- **Teste de sistema:** averigua se o software cumpre as funcionalidades para o qual foi desenvolvido e atende a todos os demais requisitos de usabilidade e eficiência.

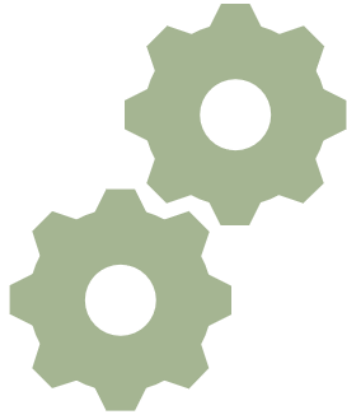


Manutenção



Para que o ciclo de vida de um software possa ser **ampliado é necessário que ele seja capaz de facilitar a correção de erros residuais** (isto é, erros descobertos após a sua liberação para o usuário), **adaptar-se a mudanças no seu contexto de aplicação** (tal como, um novo ambiente computacional) e **atender a demandas por melhoria e inclusão de serviços.**

Manutenção



LPs que oferecem recursos de modularização tendem a gerar programas mais fáceis de serem mantidos uma vez que as alterações em um módulo não interferem nos demais módulos constituintes do software.

Propriedades desejáveis em uma Linguagem de Programação

Algumas propriedades são:

- Legibilidade
- Redigibilidade
- Confiabilidade
- Eficiência
- Facilidade de aprendizado
- Ortogonalidade
- Reusabilidade
- Modificabilidade
- Portabilidade

Legibilidade

Facilidade para se ler e entender um programa

Linguagens que usam Goto normalmente reduzem a legibilidade porque nesse tipo de programação, os programas possuem fluxo de controle e não obedecem a padrões regulares. Tornando difícil acompanhar e entender o que eles fazem.

Uso de mesmo vocábulo da LP para denotar diferentes comportamentos dependendo do contexto é prejudicial à legibilidade e entendimento da LP.

Possibilita ao programador se concentrar nos algoritmos centrais do programa, sem se preocupar com aspectos não relevantes.

Exemplo:

As LPs com tipos de dados limitados requerem o uso de estruturas complexas, o que acaba dificultando a redação de programas. A falta de declaração recursiva e ponteiro em Visual Basic acaba limitando o seu uso para implementar programas com uso de estruturas de árvores, listas e etc.

Redigibilidade

Confiabilidade e

Essa propriedade se relaciona os mecanismos fornecidos pela LP para incentivar a construção de programas confiáveis.

LPs que requerem a declaração de dados permitem verificar automaticamente erros de tipos durante a compilação ou execução. LPs que possuem mecanismos para detectar eventos indesejáveis (Tratamentos de Erros) e especificar respostas adequadas a tais eventos permitem construção de programas mais confiáveis.

Eficiência

Se acordo com as demandas para o tipo de aplicação.
Capacidade da LP de fornecer meios adequados para atingir o objetivo.

Ex: aplicações de automação em tempo real normalmente requerem o uso de Lps que minimizem o tempo de execução e de acesso aos dispositivos periféricos.

Facilidade de aprendizado

O programador deve ser capaz de aprender a linguagem com facilidade.



LPs com muitas características e múltiplas maneiras de realizar a mesma funcionalidade tendem a ser mais difíceis de aprender. Além disso, outro aspecto negativo causado pelo excesso de características é o fato de levar os programadores a conhecerem apenas uma parte da linguagem, o que torna mais difícil a um programador entender o código produzido por outro.

Ortogonalidade

Capacidade de que a LP permita combinar conceitos básicos sem que se produzam efeitos anômalos.

O programador pode prever com segurança o comportamento de uma determinada combinação de conceitos.

Reusabilidade

Possibilidade de reutilizar o mesmo código para diversas aplicações.

Modificabilidade

Possibilita alterar o programa em função de novos requisitos.

Exemplo de mecanismos que proporcionam boa modificabilidade são o uso de constantes simbólicas e a separação entre interface e implementação na construção de subprogramas e tipos de dados abstratos.