CENTRO PAULA SOUZA

GOVERNO DO ESTADO SÃO PAULO

Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia
Análise e Desenvolvimento de Sistemas

ESTUDO DE MODELOS DE LINGUAGEM DE PROGRAMAÇÃO

CAIO PEREIRA

Americana, SP
2013

Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia
Análise e Desenvolvimento de Sistemas

ESTUDO DE MODELOS DE LINGUAGEM DE PROGRAMAÇÃO

CAIO PEREIRA

Caio590@hotmail.com

Trabalho de Graduação desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do Prof. Esp. Antonio Alfredo Lacerda.

Área: Análise e Desenvolvimento de Sistemas

Americana, SP
2013

BANCA EXAMINADORA

Prof. Esp. Antonio Alfredo Lacerda (Orientador)

Prof. Me. Alberto Martins Junior

Prof.ª Esp. Rogerio Nunes de Freitas

AGRADECIMENTOS

Agradeço primeiramente ao meu orientador Prof. Antonio Alfredo Lacerda por me auxiliar durante todo o projeto, pela paciência e pelo aprendizado que obtive com sua orientação. Agradeço também a todos meus colegas de classe, pelo apoio e amizade de todos e principalmente pelo incentivo e companheirismo que sempre tivemos.

DEDICATÓRIA

Dedico este trabalho aos meus pais que sempre me apoiaram, aos meus colegas de sala, a todos os analistas e desenvolvedores de sistemas e para todos que buscam escolher o modelo que se encaixe a suas necessidades.

RESUMO

O A sociedade, atualmente está rodeada por novas tecnologias, que evoluem diariamente junto com as necessidades e consumos exigidos pelas pessoas. Juntamente com as novas tecnologias novos métodos de desenvolvê-las foram criados, com o objetivo de facilitar e acelerar o progresso dos projetos.

Com o avanço da tecnologia e sua utilização o nível de processamento de dados ganhou grandes proporções, gerando a necessidade de manipula-los em alta velocidade, precisão e consistência. A tecnologia da informação tem como objetivo administrar e manter os dados com grande eficiência, com o intuito de gerar as informações esperada pelo usuário.

Essas informações podem ser produzidas através de softwares que utilizam das linguagens de programação e seus modelos para gerenciar os dados. Uma forma padronizada que permite ao desenvolvedor manipular todo conjunto de elementos necessários para processar e elaborar as informações de acordo com as necessidades que consiste seu projeto.

Este trabalho busca comparar os modelos de linguagem de programação, com o objetivo de auxiliar os programadores a escolher o melhor modelo que se encaixe em seu projeto. Os modelos abordados forneceram seus conceitos e aplicações, suas vantagens e desvantagens como uma alternativa na identificação e pesquisa de modelos para o desenvolvimento de sistemas.

Palavras Chave: Linguagem de programação, Software, Modelos de Linguagem de programação, Tecnologia da informação.

ABSTRACT

Now a days society is surrounded by Morden technology, that involves both necessity and required consumptions by everyone, together with new technologies, new methods to develop them were created, with the intension of smoothing and speeding up the process of the projects.

With improved technology and utilization the level of data processing has won great proportions, creating a necessity to manipulate them in great speed, with accuracy and consistency. Information technology has the purpose of administrating and maintaining data with great efficiency, with the intuition of generating information expected by user.

These information can be produced through software that utilize programing languages and templates to manage data. A standardized way to allow the developer to manipulate a set of elements needed to process and elaborate the information according with the necessity that the project consists.

This work aims to compare programing language templates, with the goal to assist programmers a better choice of template that fits within their project. The addressed template will provide concepts and applications, your advantages and disadvantages with an alteration on the identification and a search of templates for the systems development.

Keywords: *programing language, software, Programing language templates, information technology.*

SUMÁRIO

LISTA DE FIGURAS E DE TABELAS

1 INTRODUÇÃO

Os recursos tecnológicos e seus serviços vêm crescendo rapidamente no Brasil e no mundo, sendo utilizados nas mais diversas áreas, tanto para o lazer quanto para o trabalho. Atualmente, a tecnologia se tornou algo essencial no setor empresarial, juntamente com ferramentas e recursos que auxiliam na tomada de decisão, gerenciamento, produção, comunicação entre outros, tornando o mercado cada vez mais competitivo. Além de todos estes fatores, ainda produz entretenimento e conforto para o lazer do homem moderno, com os mais diversos tipos de equipamentos que permitem maior interação e comunicação com seus recursos.

A tecnologia da informação é um dos setores que está ganhando muito espaço nas diversas áreas profissionais. Seu principal foco é administrar e manter as informações, utilizando algum dispositivo ou equipamento que permita o acesso para manipular e armazenar as informações, gerando conhecimento que auxilia para a tomada de decisão.

Essas informações podem ser produzidas através de softwares que são sequências de instruções que serão seguidas e executadas, para manipulação dos dados e informações de maneira que ajudem o usuário a chegar ao resultado esperado. Um programa de computador é basicamente uma sequência de instruções que vão ser interpretadas e executadas pelo processador, portanto podemos considerar os programas de computadores um software.

Podemos encontrar os softwares por toda parte, como em celulares, televisões, vídeo games, computadores, carros, máquinas industriais entre muitos outros. A sua utilização vem facilitando e melhorando a vida do homem dias a dia e cada vez percebe-se sua evolução e eficiência entre os mais diversos setores públicos.

Softwares são desenvolvidos com base nas linguagens de programações e seus modelos. A linguagem de programação é uma forma padronizada de se comunicar através de instruções com o computador, ela é formatada por um

conjunto de regras sintáticas e semânticas que permite o programador definir especificamente os dados que o computador manipulará e as ações que vão ser tomadas. Para que máquina entenda essas instruções, é necessário se comunicar com ela através de códigos binários (código de máquina). A linguagem de programação permite que os programadores expressem com maior facilidade os propósitos que desejam, pois utilizam de expressões e formas linguísticas de fácil compreensão, que posteriormente são convertidos para que o equipamento entenda e processe as instruções.

Por isso, a utilização de modelos de linguagem de programação é essencial para estruturar e melhorar a eficiência de um software. Atualmente, existem vários, como: estruturado (imperativo), orientado a objetos, funcional e lógico. Cada um possui uma determinada forma de abordar os problemas e formular soluções. Os modelos podem ser utilizados em mais de uma linguagem e pode haver dois ou mais modelos sendo utilizados juntos. Cabe ao programador escolher o método mais vantajoso para atender a sua necessidade.

Este trabalho de conclusão de curso tem como objetivo estudar e apresentar as principais características dos modelos de linguagem de programação mais utilizados atualmente, expondo suas vantagens e desvantagens, seus conceitos e aplicações. A descrição desse estudo será realizada com base em estatísticas, trabalhos e artigos acadêmicos. O intuito é identificar os pontos que são mais eficientes em cada modelo e sua principal diferença perante aos outros, contribuindo para formação de conhecimento dos programadores.

O capítulo um apresentará a história da computação, como surgiram as primeiras máquinas e como a tecnologia da computação veio crescendo e de que maneira se deu inicio a era digital em que estamos.

O capítulo dois apresentara o software e como é construído, dando ênfase as linguagens de programação e linguagens algorítmicas e os principais motivos da necessidade delas terem sido criadas e como facilitou e padronizou o desenvolvimento de novos projetos.

No capítulo três, serão abordados os principais modelos utilizados, descrevendo suas principais particularidades, conceitos e a maneira como são utilizados.

O capítulo quatro consiste na comparação dos modelos com base em estatísticas, trabalhos e artigos acadêmicos. Demonstrando suas vantagens e desvantagens, seus conceitos e aplicações.

No capítulo cinco, haverá uma conclusão sobre o estudo, expondo os modelos que mais se destacaram, focando a aceitação dos usuários sobre o modelo, eficiência, flexibilidade, legibilidade, compreensão da estrutura e reutilização de códigos.

1.1 METODOLOGIA

O desenvolvimento deste estudo descritivo e exploratório será realizado com base em trabalhos e artigos acadêmicos. Que terá como principal intuito avaliar os pontos fortes e fracos dos diversos tipos de modelos de linguagem de programação e apresentar as particularidades de cada um e suas características.

1.2 OBJETIVO DA PESQUISA

Atualmente com o surgimento de novas tecnologias e métodos diferentes para o seu desenvolvimento, está cada vez mais difícil escolher a maneira que melhor se adapta ao seu modo de trabalho e qual melhor atende as necessidades do seu projeto.

O estudo realizado nesse trabalho vai abordar as características dos principais modelos de linguagem de programação, apresentando os modelos mais utilizados e suas principais vantagens.

1.3 JUSTIFICATIVA

A justificativa desse estudo será realizada com base em estatísticas, artigos e trabalhos acadêmicos sobre a linguagem de programação e seus modelos. Para

poder contribuir com os programadores e desenvolvedores de sistemas de computação, possibilitando expandir seus conhecimentos sobre os modelos utilizados e as características que cada um apresenta.

2 BREVE HISTÓRIA DA COMPUTAÇÃO

2.1 SURGIMENTO DAS PRIMEIRAS MÁQUINAS

Ao longo da história do ser humano, o homem se deparou com os vários trabalhos e processos cansativos e repetitivos, por isso, sempre buscou novas formas, maneiras ou equipamentos para auxiliar e melhorar seu estilo de vida. Na procura por soluções nasceram diversos dispositivos de cálculos principalmente para a automação dos processos.

O homem sempre tentou utilizar as máquinas ou aparelhos para facilitar seu trabalho e principalmente para aumentar a qualidade, diminuir o esforço e economizar tempo. Assim o nascimento das primeiras máquinas ganhou força não apenas como um tipo de produto do intelecto humano, mas também como parte da intuição humana.

Segundo Magela (2006), Há vários exemplos das primeiras máquinas construídas, como o ábaco (3000 a.C.), um equipamento capaz de resolver problemas matemáticos envolvendo as quatro operações. O quadrante, utilizado para resolver problemas astronômicos, e até um mecanismo misterioso apresentado na figuro 1 chamado Antikythera, uma máquina grega capaz de medir o tempo.

Figura 1 - O Mecanismo de Antikythera
(http://apod.nasa.gov/apod/ap130120.html, 31/05/2013 – 18h 45min)

Porém as principais ideias para o nascimento da primeira máquina mais próxima da atual computação não veio de nenhum matemático, astrônomo ou físico. O francês Joseph-Marie Jacquard era um tecelão que conseguiu produziu os teares de Jacquard, uma máquina capaz de automatizar o processo de tecelagem através de um padrão definido de desenho.

Para o equipamento de Joseph-Marie executar o traçado ela deveria ter um plano de execução (Programa), que informava quando os fios deveriam ir por cima ou por baixo do traçado. Um cartão com buracos (Perfurado) descrevia como era o modelo que devia ser produzido dando ao aparelho o plano de execução para iniciar a produção.

Em uma viagem para a França o matemático e astrônomo Charles Babbage (1792-1871) teve contato com a máquina de teares de Jacquard. Aos 30 anos de idade, Babbage apresentou um projeto ao governo inglês que possibilitaria executar cálculos e que deveria ser construída de madeira e latão. Porém esse projeto acabou não sendo realizado pelo governo, mas, segundos os matemáticos, se esse projeto tivesse sido realizado a computação poderia ter sido antecipada em até um século.

Figura 2 - Máquina diferencial de Babbage
(http://www.biografiasyvidas.com/biografia/b/babbage.htm - 31/05/2013 - 18h 58min)

Babbage idealizou uma máquina com a visão de Jacquard, que permitia fazer uma serie de cálculos. Também notou que, se as informações transmitidas para o equipamento pudessem ser transformadas em números poderia tratar os dados computacionalmente.

De acordo com Magela (2006), a principal funcionalidade da máquina criada por Babbage era calcular tabelas matemáticas. Logo, percebeu a necessidade de visualizar os dados gerados, ou seja, realizar de alguma maneira a impressão dos resultados dando inicio ao desenvolvimento de dispositivos de entrada e saída. Para transferir as informações para o equipamento à entrada de dados era realizada através de três tipos de cartões.

Um cartão numérico, que possuía os números do problema. Um cartão diretivo, com os comandos de movimento da máquina e por fim o cartão de operações, que realizava as operações matemáticas, adição, subtração, multiplicação e divisão.

O equipamento de Babbage foi um referencia inquestionável por dois principais referenciais que existem nos computadores atuais, a “Transferência de controle” e “Modificação dinâmica do programa”.

A transferência de controle é a parte responsável por desviar a maquina para outros partes do programa desenvolvido a partir de comparações matemáticas.

A modificação dinâmica do programa significa a possibilidade que a maquina conseguia modificar as instruções a serem executadas, enquanto ainda estava em execução a partir dos resultados encontrados. A máquina foi denominada *Analitical Engine* pelo próprio Babbage e umas das mais próximas da computação atual, diz Magela (2006).

Em contato com Ada Augusta Byron foi inferido os princípios que iriam dar inicio a programação de computadores, e partir disso considerada a primeira programadora de computadores. Com a ajuda de Ada foi surgindo os conceitos como sub-rotinas, loops e saltos.

Houve também várias outras máquinas e inventores que auxiliaram no crescimento da computação e que influenciaram diversos conceitos e métodos.

Como Herman Hollreth, que teve suas ideias também influenciadas pelo francês Jacuquard, criando um equipamento para processar o censo americano de 1890. Sua invenção foi um sucesso e logo criou a empresa Calculating-Tabulating-Recording (CTR), renomeada para Internacional Business Machine (IBM).

Mas, apenas em 1937 um jovem engenheiro chamado Claude E. Shannon conseguiu estabelecer uma maneira formal entre os circuitos elétricos e a lógica. Desenvolvendo também um mapeamento entre as informações e a matemática que não foi apenas utilizado pela indústria de computadores, mas também pelas indústrias de telecomunicação.

O engenheiro de apenas 22 anos conseguiu perceber uma ligação entre a lógica booleana (0 e 1) com o chaveamento de circuitos elétricos, o que o permitiu ter a visão que os dados como as instruções poderiam ser armazenas eletricamente como 0 e 1, dando inicio a era digital.

2.2 O INÍCIO DA ERA DIGITAL

Com base nas diversas máquinas, conceitos e teorias vários estudioso exploraram formas de reunir os aspectos digitais em um aparelho. Segundo Magela (2006), O Engenheiro Civil Konrad Zuse (1910-1995) foi o primeiro a construir uma máquina de cálculo controlada automaticamente e logo percebeu que um dos grandes problemas era o armazenamento temporário dos cálculos. Criando a necessidade de uma unidade controladora, uma memória e um dispositivo aritmético.

George Stibitz trabalhava nos laboratórios da Bell Telephone, e tinha como principal tarefa melhorar a velocidade dos cálculos presentes nas telecomunicações para o crescimento da empresa. Stibitz e S.B. Willians começaram a utilizar peças eletromecânicas na computação dos cálculos. Seu modelo e o de Zuse foram lançados e ambos utilizavam códigos binários em relês.

Contudo, também houve engenheiros da IBM junto com Howard Aiken, de Havard, que começaram a desenvolver maquinas baseados na memoria de núcleo de ferrite. Essa grande inovação permitiu que fosse escolhido dinamicamente qual

algoritmo iria ser utilizado em determinado calculo, surgindo o conceito de registradores e varias outras funcionalidades, diz Magela (2006).

Essas inovações foram apresentadas no Havard Mark I. Na tentativa de alavancar esses conceitos, e logo surgiram novas tecnologias Frank Hamilton conseguiu aprimorar vários conceitos, e a intuição de programa armazenado foi uma de suas maiores invenções. A construção do computador ENIAC entrou para a historia um dos primeiros computadores totalmente eletrônico, esse computador foi criado na Universidade da Pensilvânia através de uma equipe altamente qualificada de pesquisadores.

No ano de 1945, John Von Neumann ingressou como consultor dessa equipe e logo publicou a “Arquitetura de Von Neumann”, assim estabelecendo os paradigmas no projeto de computadores para as gerações futuras, demonstrando o conceito de programa armazenado e o projeto logico que permitia sua execução. O que deu o nascimento de uma central de coordenação de processamento, conhecida atualmente como CPU.

A figura 3 mostra que com base nesses conceitos foi construído o UNIVAC, o primeiro computador comercialmente disponível que utilizava todos os conceitos antes dele os computadores existentes era denominado computadores de primeira geração.

Figura 3 - Computador UNIVAC em 1951
(http://www.inetdaemon.com/technology/univac-60-years-ago-today, 31/05/2013 - 19h 36min)

Ao passar dos anos foram surgindo novos conceitos e equipamentos para melhorar os computadores. Entre 1956 e 1963 nasceu a segunda geração de computadores, dando inicio a utilização transistor e a novos avanços nos conceitos de computação.

O período de 1964 a 1970 foi marcado pelo surgimento da terceira geração de computadores, com os novos circuitos integrados.

Entre 1970 a 1980 foram instituídos a quarta geração de computadores que tem como marco inicial as tecnologias de LSI, VLSI e ULSL que abrigam milhões de componentes eletrônicos em um pequeno espaço, surgindo assim os chips.

De 1980 até os dias atuais nós encontramos na quinta geração de computadores, é caracterizada pelo grande aumento na capacidade de processamento de dados, armazenamento e taxas de transferência. Aonde a miniaturização vem crescendo e diminuindo o tamanho e buscando aumentar a velocidade de processamento permitindo equipamentos menores e melhores.

3 O SOFTWARE E SUAS LINGUAGENS

3.1 LINGUAGEM ALGORÍTMICA

Havia a proposta de produzir maneiras efetiva e finita que pudesse avaliar a validade das proposições matemáticas. Godel apresentava um teorema provando que não era possível alcançar esse objetivo. Porém, Turing permitiu a criação da abstração denominada Máquina de Turing. Com a máquina de Turing era possível criar um procedimento para resolver um problema computável.

Segundo Magela (2006) Logo, o conceito de procedimento efetivo acabou sendo formalizado e institua a produção de uma sequencia finita de instruções que suportava ser executada por um agente computacional e produzir o resultado desejado. Esse procedimento efetivo acabou sendo denominado algoritmo.

A descrição desses procedimentos deverá ser escrita em uma linguagem algorítmica, para permitir sua execução pelo aparelho computacional. Um dos principais motivos para a formalização era possibilitar a formalidade matemática e o mais importante, eliminar a ambiguidade presente na linguagem natural.

Com o passar do tempo, as instruções e os códigos utilizados para representar os procedimentos em forma algorítmica foram, denominado programação e a linguagem algorítmica, linguagem de programação.

Porém, enquanto a programação não era consiste os procedimentos eram descritos em linguagem de máquina, ou seja, códigos como 0000 0010 0011 1010 (Binários), poderia significar um salto em um programa. Mais tarde, percebeu que a troca de partes da linguagem de maquina era uma forma de facilitar a programação, como por exemplo, trocar 0001 0101 por MOV.

Magela (2006) diz que, essa nova formalização da maneira de programar foi considerada como linguagem de montagem, atualmente muito conhecida como, linguagem Assembly. Mas, embora avanços, que permitia programar com maior facilidade as pessoas estavam utilizando uma linguagem de baixo nível, em outras

palavras, a linguagem estava voltada, mas para o que a maquina interpretava e não para a mente humana, dificultando muito a criação de programas mais complexos.

Preocupados com a interpretação da linguagem lógica, John Von Neumann e Herman H. Goldstine produziram uma metodologia de programação, um conjunto de programas-exemplo, além de consolidar os conceitos de sub-rotinas reutilizáveis e da construção de bibliotecas de sub-rotinas na sua publicação de 1947, *“Planning and Coding Problems of na Eletronic Computing Instrument”.*

Porém um problema continuava, apesar dos avanços para a linguagem de alto nível, o computador suportava apenas um conjunto restrito de instruções. Logo, surgiu a necessidade de traduzir as instruções descritas em alto nível (linguagem próxima à língua natural) para a linguagem de maquina (linguagem binária).

Assim, nasce à ideia de construir um compilador, uma rotina que iria verificar o código descrito em alto nível e transformar em linguagem de máquina. O compilador emularia um computador hipotético possibilitando a execução de instruções de baixo nível através da linguagem de alto nível, as etapas de um processo de compilação são apresentadas através da figura 1.

Figura 4 – Processos de compilação (SEBESTA, 2010, p. 47).

3.2 LINGUAGEM DE PROGRAMAÇÃO

Magela (2006), apresenta que a linguagem de programação deu início no ano de 1954 aonde foi o nascimento da linguagem FORTRAN (IBM *FORmula TRANslation System*), voltada para a resolução de problemas matemáticos. A linguagem trazei o melhor dos dois mundos, rapidez no processo de programação e uma linguagem mais fácil de se utilizar do que a linguagem de máquina.

Essa linguagem é considerada imperativa, uma linguagem que é necessário definir rigorosamente os passos que serão seguidos. Durante a execução das instruções as varias são atualizadas constantemente.

Um ano depois E.K. Bluem desenvolveu a linguagem ADES (*Automatic Digital Encoding System*), com a principal característica de ser uma linguagem declarativa e não imperativa como a FORTRAN.

Logo, foram surgindo diversas ideias e maneiras para se programar, sempre buscando uma maneira mais eficiente, prática e efetiva de se comunicar com a máquina e construir programas mais complexos.

Segundo Magela (2006) no ano de 1960, nasceu a linguagem Algol-60. Essa linguagem buscava representar os algoritmos em forma de programas. Com base nisso surgiu à linguagem Pascal, que possuía uma maneira sistemática e fácil de programar e que poderia ser implementado na maioria dos computadores daquela época.

A linguagem ADA foi desenvolvida depois do Pascal, implementando a ideia de pacotes, mantendo a estrutura modular do Pascal.

Dennis Ritchie criou a linguagem C, que obteve uma grande aceitação principalmente pelo fato de sua portabilidade e que o sistema UNIX utilizou para ser desenvolvido.

Apesar disso, nasceu durante a década de 60 o paradigma da orientação a objetos que tem um conceito diferente da imperativa aonde teria a visão de troca de mensagem entre os objetos e sua consequente mudança de estado. Magela (2006) diz que no inicio esse novo paradigma tinha o objetivo de organizar melhor a programação, aonde se unificava dados e funções em uma única entidade, aplicava o conceito de encapsulamento e protegia os programas de danos indesejáveis.

Com base no conceito de orientação a objetos foi desenvolvida a linguagem SIMULA, criada por Ole-johan e Kristen Nygaard em 1962, essa linguagem possuía uma arquitetura de orientação a objeto, porém a linguagem continuava muito

relacionada ao Algol. Mesmo não causando muita aceitação, Tony Hoare estendeu o projeto iniciado e em 1967, lançou a linguagem orientada a objetos, SIMULA 67.

Entretanto, em 1972, nasceu a linguagem SmallTalk, constituindo o marco da primeira linguagem totalmente orientada a objetos e reconhecida até os dias de hoje. Contudo, ficou claro que para a orientação a objetos se tornar efetivamente parte do mercado, seria necessário preservar os programas feitos até então.

Magela (2006) diz que teoricamente, isso foi possível em 1985 com a criação da linguagem C++. Apesar disso, teve que abandonar alguns conceitos avançados oferecidos pela linguagem Smalltalk para conseguir manter sua compatibilidade com a linguagem C, por exemplo, classes estáticas, necessidade de recompilação e não implementação do conceito de Garbage Colletion (possibilidade da corrupção da memória).

Em seguida, acabou surgindo um monte de linguagens dizendo realmente serem orientadas a objetos, como Object Pascal e a implementação em Delphi.

Porém a mas conhecida e que ganhou seu lugar em destaque foi a linguagem Java, criada por Patrick Naughton e James Gosling, em 1995, pela empresa Sun Microsystems. Seu principal proposito era construir uma linguagem que conseguia ser independente de qualquer eletrônico, ou seja, uma linguagem que não ficasse presa a um tipo de dispositivo.

O Java utilizou de conceitos como, linguagem de Mesa, Modula-3, Objective C, C, C++, entre outras. Com a ideia de possuir uma arquitetura neutra, que seria a produção de um código intermediário determinado, byte code, que seria executado por qualquer computador aonde se criava uma Máquina Virtual Java, como mostra a figura 2.

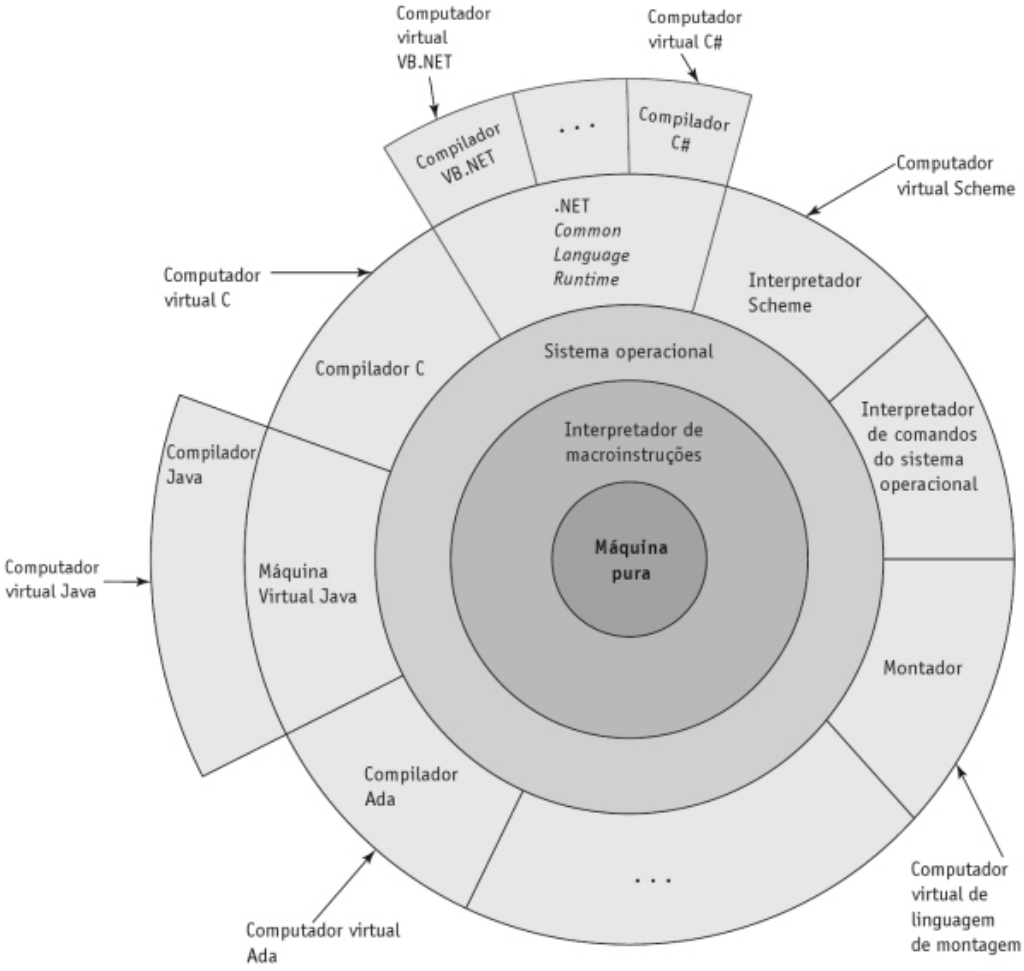

Figura 5 - Interface em camadas de computadores virtuais, fornecidas por um sistema de computação típico (SEBESTA, 2010, p. 45).

Logo, a Microsoft apresentou a linguagem .NET, que demonstra até uma certa independência da linguagem de programação, aonde você consegui um ambiente visual e interativo.

Atualmente, possuímos diversas linguagens de computação, na figura 3 pode-se visualizar uma representação da genealogia das principais linguagens de alto nível. Provavelmente em um futuro próximo, surgirá várias outros tipos de linguagem, com a evolução na programação, acreditamos que ela se torne algo natural e que a escolha da linguagem acabe se tornando uma questão secundária na hora de pensar em um projeto.

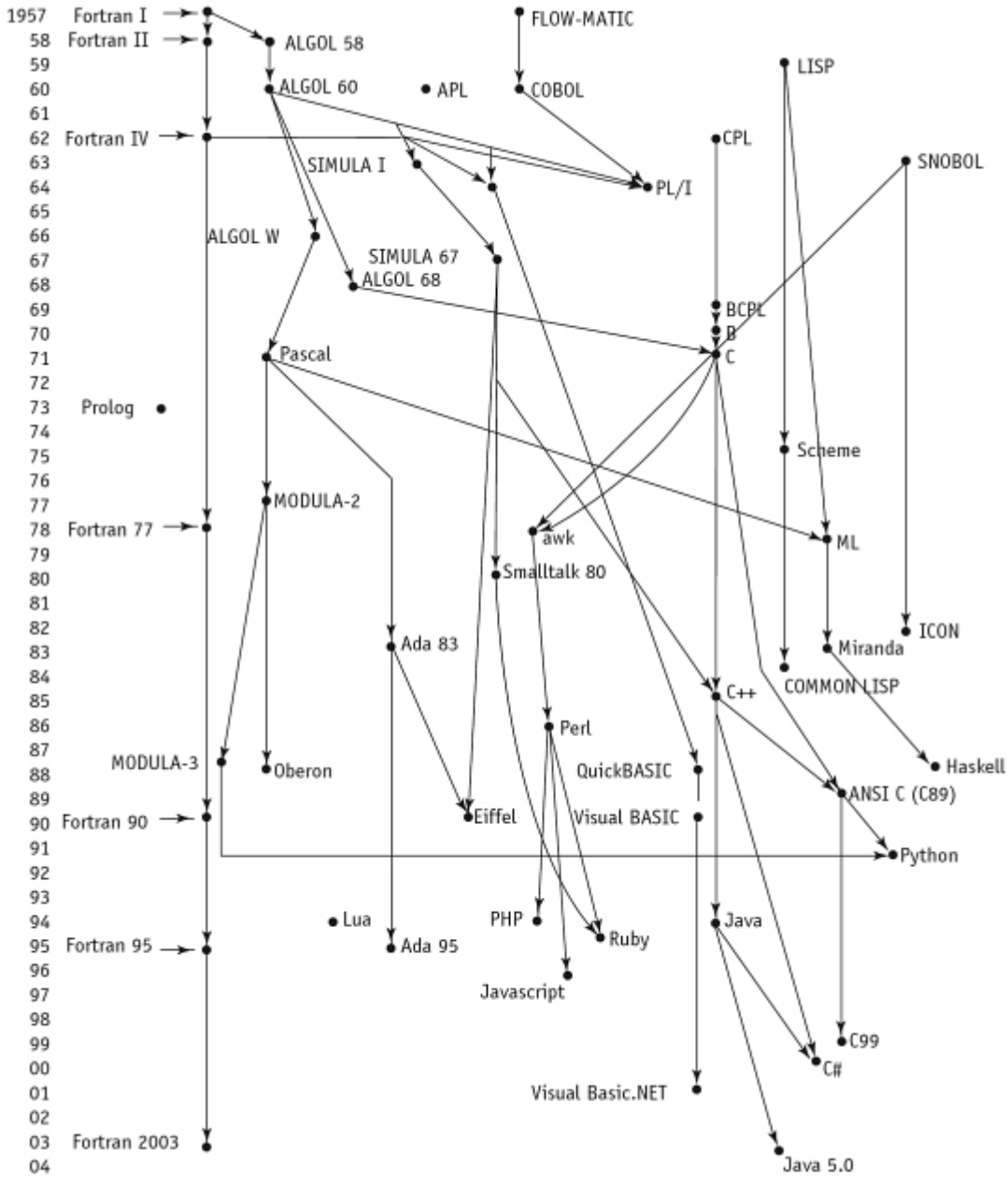

Figura 6 - Genealogia das principais linguagens de alto nível (SEBESTA, 2010, p. 58).

4 MODELOS DE LINGUAGEM DE PROGRAMÇÃO

4.1 PARADIGMAS DE PROGRAMAÇÃO

De acordo com Ascencio e Campos (2007), um paradigma de programação está diretamente relacionado à maneira de como o programador pensa e de que forma ele busca as soluções para os problemas. O paradigma que determina quais técnicas são permitidas durante a programação, possibilitando demonstrar como o programador analisou e abstraiu o problema para ser resolvido e organizado.

Magela (2006) compreende que os problemas são os mesmo, mas depende da maneira em que o observamos ele pode sofre algumas mudanças. De acordo com o paradigma utilizado o problema pode ficar fácil de ser resolvido e outros mais complicados e dependendo até mesmo impossíveis.

Contudo, no início os paradigmas não foram muito bem visto. Mas, com o passar do tempo sua aceitação foi aumentando e conquistaram seu lugar, atualmente existem vários tipos de paradigmas de programação: estruturado, orientado a objetos, lógico, funcional e outros.

4.2 PARADIGMA ESTRUTURADO

Segundo Ascencio e Campos (2007), o paradigma estruturado (também conhecido como imperativo) é constituído de basicamente três tipos de estruturas: sequencial, condicional e iterativa (repetição). Esse paradigma apresenta que qualquer programa pode ser resolvido se baseando nessas estruturas. Também propõe uma forma de quebrar problemas mais complexos em pequenas partes mais simples que, trabalhando conjuntamente, permitam solucionar o problema com maior facilidade.

O Objetivo é utilizar corretamente as estruturas, assim como os recursos de modularização e a parametrização, com o intuito de criar programas com o menor numero de linhas de comando repetidas e facilitando a abstração do problema.

4.2.1 ESTRUTURA SEQUENCIAL

4.2.1.1 ESTRUTURA SEQUENCIAL EM ALGORITMO

Ascencio e Campos (2007) apresentam a estrutura sequencial em algoritmo sendo constituída da declaração de variáveis e o bloco de comandos.

Exemplo:

```
ALGORITMO
        DECLARE
        Bloco de comandos
FIM_ALGORITMO.
```

(ASCENCIO e CAMPOS, 2007, p16).

4.2.1.2 DECLARAÇÃO DE VARIÁVEIS EM ALGORITMO

Essa á parte em que é necessário descrever quais serão as variáveis do seu algoritmo, Ascencio e Campos (2007), diz que as variáveis são declaradas após a palavra DECLARE e os tipos mais utilizados são: NUMÉRICO (utilizado para variáveis que receberão números), LITERAL (utilizado para variáveis que receberão caracteres) e LÓGICO (utilizado para variáveis que receberão apenas dois tipos de valores: Verdadeiro e Falso, também conhecido como, variável Booleana).

Exemplo:

```
DECLARE
 X NUMÉRICO
Y, Z LITERAL
TESTE LÓGICO
```

(ASCENCIO e CAMPOS, 2007, p16).

4.2.1.3 COMANDOS DE ATRIBUIÇÃO EM ALGORITMO

A atribuição é realizada quando se deseja passar algum valor a variável, Ascencio e Campos (2007), apresenta que os comandos de atribuição é utilizado para conceder valores ou operações a variáveis, essa atribuição é representada pelo símbolo ←.

Exemplo:

x ← 4
x ← x + 2
y ← “aula”
teste ← falso

(ASCENCIO e CAMPOS, 2007, p16).

4.2.1.4 COMANDOS DE ENTRADA EM ALGORITMO

Segundo Ascencio e Campos (2007), Esse comando é utilizado para que o usuário passe a digitar os dados necessários para que o programa possa execute sua funcionalidade. Esses dados serão armazenados em variáveis. Esse comando é executado pela palavra LEIA.

Exemplo:

LEIA X
(Um valor digitado pelo usuário será armazenado na variável X)

LEIA Y
(Um valor digitado pelo usuário será armazenado na variável Y)

(ASCENCIO e CAMPOS, 2007, p16).

4.2.1.5 COMANDOS DE SAÍDA EM ALGORITMO

O comando de saída é parecido com o de entrada, mas a diferença é que ao invés de pedir que o usuário digite o valor que será atribuído a variável é o programa

que apresentará o valor da variável com possível resultado ou informação, de acordo com Ascencio e Campos (2007), o comando de saída é utilizado para mostrar dados na tela ou na impressora. Esse comando é representado pela palavra ESCREVA, e os dados podem ser conteúdos de variáveis ou mensagens.

Exemplo:

ESCREVA X
(Mostra o valor que está armazenado na variável X)

ESCREVA “Conteúdo de Y =”, Y
(Mostra a mensagem “Conteúdo de Y =” e em seguida o valor armazenado na variável Y)

(ASCENCIO e CAMPOS, 2007, p17).

4.2.2 ESTRUTURA CONDICIONAL

4.2.2.1 ESTRUTURA CONCIONAL SIMPLES EM ALGORITMO

Essa estrutura é composta por uma condição em que se o dado que foi fornecido satisfazer suas instruções ela irá executar um determinado bloco de comandos.

Exemplo:

SE Condição
ENTÃO Comandos

(ASCENCIO e CAMPOS, 2007, p50).

Ascencio e Campos (2007), diz que o comando só será executado se a condição for verdadeira. Uma condição é uma comparação que possui entre dois valores possíveis: verdadeiro e falso.

Exemplo:

```
SE Condição
ENTÃO INÍCIO
        Comando 1
        Comando 2
        Comando 3
FIM
```

(ASCENCIO e CAMPOS, 2007, p50).

Os comandos 1, 2 e 3 só serão executados se a condição for verdadeira. As palavras INÍCIO e FIM serão necessárias apenas quando dois ou mais comandos forem executados.

4.2.2.2 ESTRUTURA CONCIONAL COMPOSTA EM ALGORITMO

A estrutura composta é utilizada quando se pretende utilizar uma condição em que se ela não for satisfeita executará outro bloco de comando, ou seja, se for verdadeiro irá executar um determinado bloco de comando se não for executará outro.

Exemplo:

```
SE Condição
ENTÃO Comando 1
SENÃO Comando 2
```

(ASCENCIO e CAMPOS, 2007, p50).

De acordo com Ascencio e Campos (2007), se a condição for verdadeira, será executado o comando1 e caso contrário o outro bloco será executado, ou seja, o comando 2.

Exemplo:

```
SE Condição
ENTÃO INÍCIO
        Comando 1
        Comando 2
        FIM
SENÃO INÍCIO
        Comando 3
        Comando 4
        FIM
```

(ASCENCIO e CAMPOS, 2007, p50).

Se a condição for verdadeira, o comando1 e o comando 2 serão executados, caso contrário, o comando3 e o comando4 serão executados.

4.2.3 ESTRUTURA REPETIÇÃO

Uma estrutura de repetição deve ser utilizada quando você deseja que alguma parte do algoritmo ou até mesmo o algoritmo todo se repita variáveis vezes. Ascencio e Campos (2007) apresentam que o número de repetições podem ser fixos ou estar atrelado a uma condição. Assim, existe determinadas estrutura para cada tipo de situação.

4.2.3.1 ESTRUTURA DE REPETIÇÃO PARA NÚMERO DEFINIDO

Normalmente, essa estrutura realiza o bloco de comandos a quantidades de vezes em que foi definido. Segundo Ascencio e Campos (2007), essa estrutura é usada quando se sabe o número de vezes que um trecho de algoritmo deve ser repetido.

Exemplo:

```
PARA I ← Valor inicial ATÉ valor final FAÇA [PASSO n]
INÍCIO
```

Comando 1
Comando 2
. . .
Comandom
FIM

(ASCENCIO e CAMPOS, 2007, p93).

Ascencio e Campos (2007) descreve esse exemplo dizendo que o comando1, comando2 e o comandom serão executados utilizando-se a variável I como controle, e o seu conteúdo vai variar do valor inicial até o valor final. Quando houver apenas um comando não é necessário utilizar os marcadores de INÍCIO e FIM.

A informação do PASSO está entre colchetes porque é opcional. O PASSO indica como será a variação da variável de controle. Se o PASSO for 2 por exemplo, ele aumentara de duas em duas unidades até atingir o valor final, quando esse valor não é fornecido o programa subintende que será aumentado de uma em uma unidade.

Exemplo:

PARA I ← 1 ATÉ 10 FAÇA
ESCREVA I

(ASCENCIO e CAMPOS, 2007, p93).

O comando ESCREVA I será executado dez vezes, ou seja, a variável I irá alterar de 1 a 10. Assim, os valores que serão exibidos por pelo comando serão 1, 2, 3, 4, 5, 6, 7, 8, 9 e 10.

Exemplo:

PARA J ← 1 ATÉ 9 FAÇA PASSO 2
ESCREVA J

(ASCENCIO e CAMPOS, 2007, p93).

O comando ESCREVA J vai ser repetido cinco vezes, ou seja, ele irá alterar de 1 a 2 indo de duas em duas unidades. Os valores exibidos respectivamente serão 1, 3, 5, 7 e 9.

Exemplo:

```
PARA I ← 10 ATÉ 5 FAÇA
ESCREVA I
```

(ASCENCIO e CAMPOS, 2007, p93).

O comando ESCREVE I será repito seis vezes, ele irá alterar de 10 a 5, ou seja, seus respectivos valores serão 10, 9, 8, 7, 6 e 5.

Exemplo:

```
PARA J ← 15 ATÉ 1 FAÇA PASSO -2
ESCREVA J
```

(ASCENCIO e CAMPOS, 2007, p94).

O comando ESCREVA J vai ser repetido oito vezes, variando de 15 a 1 e de duas em duas unidades. A variável J vai apresentar os seguintes valores 15, 13, 11, 9, 7, 5, 3 e 1.

4.2.3.2 ESTRUTURA DE REPETIÇÃO PARA NÚMERO INDEFINIDO DE REPETIÇÕES

A estrutura de repetição para números indefinidos é utilizada, quando não se tem certeza da quantidade de vezes que será necessário repetirmos um determinado algoritmo. Durante a execução será determinado à condição para que a repetição termine ou continue a repetir o bloco de comandos que o compõe.

Ascencio e Campos (2007) diz que, essa estrutura é usada quando não se sabe o número de vezes que um trecho do algoritmo deve ser repetido, embora também possa ser utilizada quando se conhece esse número. Essa estrutura baseia-

se na análise de uma condição. A repetição será feita enquanto a condição mostrar-se verdadeira. Existem situações em que o teste condicional da estrutura de repetição, que fica no início, resulta em um valor falso logo na primeira comparação. Nesses casos, os comandos de dentro da estrutura de repetição não serão executados.

Exemplo:

```
ENQUANTO condição FAÇA
comando1
```

(ASCENCIO e CAMPOS, 2007, p94).

Nesse exemplo o comando1 será repetido enquanto a condição for estiver sendo satisfeita, ou seja, enquanto ela continua verdadeira o bloco de comandos será executado.

Exemplo:

```
ENQUANTO condição FAÇA
INÍCIO
        comando1
        comando2
        comando3
FIM
```

(ASCENCIO e CAMPOS, 2007, p94).

Esse exemplo apresenta quando será necessário o uso dos marcadores de INÍCIO e FIM, ou seja, quando possuir mais de um comando a ser executado.

Exemplo:

```
X ← 1
Y ← 5
ENQUANTO X < Y FAÇA
INÍCIO
```

X ← X + 2

Y ← Y + 1

FIM

(ASCENCIO e CAMPOS, 2007, p94).

Tabela 1 - Resultados da estrutura ENQUANTO. (ASCENCIO e CAMPOS, 2007, p94)

X	Y	
1	5	VALORES INCIAIS
3	6	Valores obtidos dentro da estrutura de repetição
5	7	
7	8	
9	9	

4.2.3.3 ESTRUTURA DE REPETIÇÃO PARA NÚMERO INDEFINIDO DE REPETIÇÕES COM TESTE NO FINAL

Essa estrutura REPITA é usada quando tem a necessidade de verificar a condição no final da execução do bloco de repetição, ou seja, pelo menos uma vez os comandos serão executados sem importar com as condições necessárias.

Ascencio e Campos (2007) apresentam a ideia de que essa estrutura de repetição é utilizada quando não se sabe o numero de vezes que um trecho do algoritmo deve ser repetido, embora também possa ser utilizada quando se conhece esse número.

Essa estrutura baseia-se na análise de uma condição. A repetição sera feita ate a condição tornar-se verdadeira. A única diferença entre o REPITA e o ENQUANTO é que ao menos uma vez o trecho de algoritmo será executado e continuará a repetição se a condição for verdadeira.

Exemplo:

REPITA
comandos
ATÉ condição

(ASCENCIO e CAMPOS, 2007, p95).

O trecho de algoritmo acima mostra que o comandos se repetira ao menos uma vez e continuará sua repetição até a condição se mostra falsa.

Exemplo:

```
X ← 1
Y ← 5
REPITA
X ← X + 2
Y ← Y + 1
ATÉ X >= Y
```

(ASCENCIO e CAMPOS, 2007, p95).

Tabela 2 - Resultados da estrutura REPITA. (ASCENCIO e CAMPOS, 2007, p95)

X	Y	
1	5	VALORES INCIAIS
3	6	Valores obtidos dentro da estrutura de repetição
5	7	
7	8	
9	9	

4.3 PARADIGMA DE ORIENTAÇÃO A OBJETOS

Magela (2006) apresenta que o conceito-chave da linguagem orientada a objetos é denominado polimorfismo. O paradigma da orientação a visa modelar a realidade de acordo com as funções psicológicas do pensamento humano, com o objetivo de separar os objetos de acordo com a sua origem e finalidade.

As linguagens tradicionais que dominaram o ambiente de programação de computadores, se fundamentaram no modelo de John von Neumann, que é divido em cinco partes: memória, central de processamento, unidade aritmética, entrada e saída.

Porém, percebemos que as linguagens de maquina e linguagens tradicionais estão tentando se aproximar ao máximo em direção do pensamento humano. Ou seja, percebeu-se a necessidade de trocar a programação de computadores da linguagem de maquina para uma linguagem mais próxima da natural. E o idioma que predominou foi inglês.

Ao analisar as linguagens tradicionais, percebe-se que ao tentarmos pensar como um ser humano construindo um programa irá ter sérios problemas. De acordo com Magela (2006) o ganho das linguagens tradicionais, no nosso ponto de vista, não foi tão grande quanto parece. Pois, limitamos nosso espaço de representação a apenas linguagem de máquina 0 e 1, independe se nossa linguagem possui a palavra SE (IF), ENTÃO (ELSE) ou qualquer outra expressão.

Acabando drasticamente com o potencial humana por causa dessa abordagem, limitando muitos programadores que se retiram de qualquer ambiente que necessite de algo a mais do que um simples sim e não.

Porém, a orientação a objetos tem o poder de mudar essa visão de produção de softwares, ela possui o potencial de mudar todo esse cenário, mas ainda não o fez.

Magela (2006) diz que, de maneira geral, a orientação a objetos foi reduzida a mais uma estrutura de linguagem. Por exemplo, da mesma forma que a linguagem C temos a palavra reservada STRUCT ou UNION, temos somada a palavra reservada CLASS. Os programadores, as maiores vitimas, usam essa nov apalavra para definição de atributos e operações. Usam também o conceito de herança e polimorfismo continuamente. Ao final, as empresas perguntam-se por que não colheram os benefícios promulgados pela orientação a objetos.

4.4 OBJETOS

Na visão de um programador, um objeto é a transformação da realidade que esse objeto exerce em um ambiente em que nosso problema se encontra. Essa

mudança da realidade pode ocorrer com base da atividade do objeto como suas ações, como passivamente ou responsabilidade do objeto, diz Magela (2006).

A definição de objeto é um elemento diferenciado identificado por suas ações e secundariamente por seus atributos em um universo semântico delimitado.

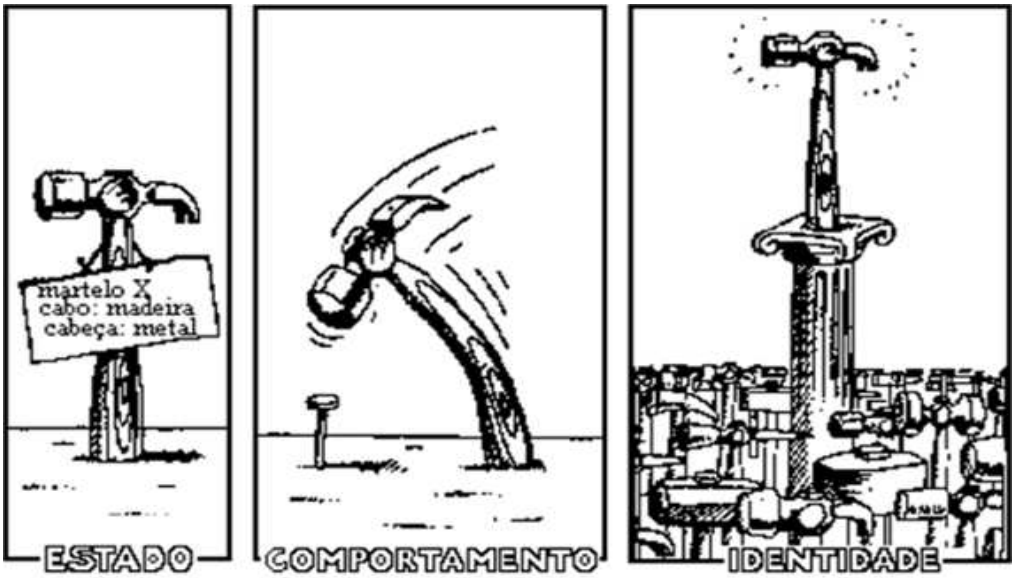

Figura 7 - Representação de um objeto. (Souza, Cleidson)

Dentro do mundo “real” podemos encontrar vários objetos, como um carro, uma moto, uma bicicleta, uma árvore, um lápis entre outros. A nossa visão ultrapassa o conceito de objeto real, tendo em vista que para ser um objeto precisa apenas ter uma responsabilidade.

Magela (2006) apresenta como exemplo, a física que embora o homem nunca tenha visto um átomo, conseguiu produzir a bomba atômica. Porém, referisse que uma entidade abstrata não possui a mesma validade. Pois essas entidades não possuem âncora em nosso pensamento. Algo abstrato pode se tratar de qualquer coisa e se adaptam a qualquer tipo de pensamento. Portanto, descobriremos nossos objetos por suas ações (responsabilidades) em seu próprio ambiente.

Um objeto não é universal, por exemplo, um objetivo chamado bola não dentro de um sistema não é o mesmo em outro o que realmente define um objeto

são principalmente suas ações e secundariamente seus atributos e não simplesmente o nome desse objeto.

4.4.1 CLASSE E INSTÂNCIA

A classe é uma estrutura utilizada para descrever objetos dentro dela, ou seja, é um conjunto de objetos que possuem propriedades parecidas (atributos), ou ações (responsabilidades) similares, a classe atua como um molde (padrão) para a construção dos objetos.

Segundo Magela (2006) o caráter fundamental da computação é sua natureza repetitiva; o caráter fundamental da ciência é a previsão dos resultados. Portanto, a busca por objetos é simplesmente um caminho para chegarmos nas classes que os definem.

Definição: Classe é um conjunto de objetos que respondem às mesmas ações com os mesmos resultados e que possuem os mesmos atributos em um mesmo universo semântico delimitado.

Contudo, quando queremos enfatizar o relacionamento entre um objeto e uma classe, utilizamos a palavra instância.

Definição: Um objeto é um instancia de uma Classe. Ou seja, João é uma instancia da classe Pessoa.

Porém não devemos deixar cair no erro de modelar apenas objetos que são de nossa realidade, ou seja, que tenha uma representação no mundo real. Se percebemos uma mudança de realidade no fluxo da informação, significa que existe uma responsabilidade (ação) e é importante modelá-la.

4.4.2 ENCAPSULAMENTO

É uma maneira de proteger os detalhes da implementação de um objeto, ou seja, é quando um objeto possui uma parte privada como variáveis que só serão

acessadas através dos métodos que sua classe definir, como por exemplo, GET e SET.

Magela (2006) define encapsulamento como uma propriedade de um objeto em esconder seus atributos (estados) de outros objetos, ou propriedades que limita as ações de um objeto a um conjunto específico de atributos, e esses atributos por sua vez só podem ser atualizados por essas mesmas ações.

O método de encapsulamento vem da necessidade de organizar um grande programa de computador, teoricamente o encapsulamento pode ser encontrado em outras abordagens além da orientação a objetos.

A orientação a objetos deu vida a esse tipo de organização de funções e dados dentro de um programa de computador. Essa maneira de organização nasceu quando havia a necessidade de alterar algum ponto de algoritmo desenvolvido e que respectivamente afetava outras partes deixando de funcionar corretamente o programa. Percebeu-se que por causa dessa desorganização em pouco tempo as manutenções superavam o custo do sistema, o que causou grandes perdas financeiras e altos custos das manutenções.

O conceito de objetos consolida em um mesmo elemento suas ações (funções) e seus atributos (dados), o conceito de encapsulamento passou a ser inerente ao conceito de objetos, explica Magela (2006).

4.4.3 HERANÇA

A herança é uma maneira de utilizar uma classe de base para estruturar os comportamentos de outra classe, ou seja, é um mecanismo para reutilizar uma classe com a finalidade de construir outras seguindo os mesmos padrões.

De acordo com Magela (2006), a definição de herança é baseada em um mecanismo pelo qual um objeto especifico incorpora as ações e atributos de objetos mais genéricos.

A utilização de um mecanismo de herança gera um grande aumento na reutilização de códigos e organização de um programa de computador facilitando o processamento e entendimento do algoritmo.

4.4.4 POLIMORFISMO

Magela (2006) define polimorfismo como, sendo um mecanismo pelo qual um estímulo a uma ação pode gerar resultados diferentes em uma mesma hierarquia de herança.

O uso do mecanismo de polimorfismo permite a sobrecarga de métodos, ou seja, a utilização de diferentes parâmetros para um mesmo método possibilitando diferentes ações para uma determinado função e principalmente auxiliando na reutilização de códigos dentro de um programa de computador.

Segundo Davis Jr e Karla AV Borges (1994), o polimorfismo trata-se da possibilidade que um programa possui de utilizar os objetos referentes a diferentes classes de uma forma transparente, interpretando suas características durante o processamento.

4.4.5 RELACIONAMENTO

Um sistema orientado a objetos é construindo com base nos relacionamentos entre os objetos. Segundo Magela (2006), a definição de relacionamento em orientação a objetos é um tipo de mecanismo abstrato que permite algum tipo de comunicação entre os objetos.

Para fins didáticos os relacionamentos serão subdivididos de duas maneiras, existe uma visão estática dos objetos, representada por seus atributos, e uma visão dinâmica dos objetos, representada por suas ações.

Contudo, um terceiro tipo chamado denominado Bijetor, vai ser acrescentado a essa lista. Um relacionamento bijetor é capaz de fazer uma ligação entre os

objetos com base em suas atividades de processo de desenvolvimento, ou seja, uma ligação mais gerencial ou organizacional entre os objetos.

4.4.5.1 RELACIONAMENTO ESTRUTUAL OU ESTÁTICO

Magela (2006) apresenta o relacionamento estrutural ou estático como sendo um mecanismo que está intimamente ligado à própria definição do objeto. Por exemplo, em uma locadora, a definição do objeto filme pode conter o atributo gênero. Esse gênero pode ser visto como um objeto próprio, tendo como representantes alguns gêneros como aventura, suspense, drama e infantil.

Em resumo um relacionamento estrutural é baseado na definição de um atributo de um objeto representando-o por outro objeto, independente de qualquer contexto ou interação em que esse objeto esteja. Outro possível relacionamento estrutural com filme seria o relacionamento com mídia, como DVD, VHS etc.

Um dos principais pontos do relacionamento estrutural é a associação, uma associação representa um relacionamento entre dois objetos. Uma associação é encontrada simultaneamente com o objeto e representa um relacionamento existencial entre eles.

Figura 8 - Associação entre classes (Magela, 2006, p.63).

Outro ponto importante é a especialização ou generalização, Segundo Magela (2006), esse mecanismo representa um relacionamento estrutural entre dois objetos.

Um objeto B especializa em um objeto A quando a definição do objeto A também se aplica ao objeto B, ou seja, um objeto generaliza o outro.

Exemplo:

1. (Animal, Leão)
2. (Pessoa, Funcionário)

(Magela, 2006, p.64).

Magela (2006) diz que, o relacionamento especialista/ generalização na pratica é conhecido relacionamento “é um”, e é na verdade a concretização do conceito de herança.

O exemplo mostra que, o Leão é um Animal e o Funcionário uma Pessoa. Assim, o relacionamento vai em uma única direção ou sentido. Assim, nem todo Animal é um Leão e nem toda Pessoa é um Funcionário.

Em um relacionamento especialista a classe mais especializada provavelmente terá outros atributos e outras operações, estendendo assim a definição da classe menos especializada. Ou seja, Leão possui propriedades especificas de um Leão, contudo obedece aos atributos e operações de um Animal.

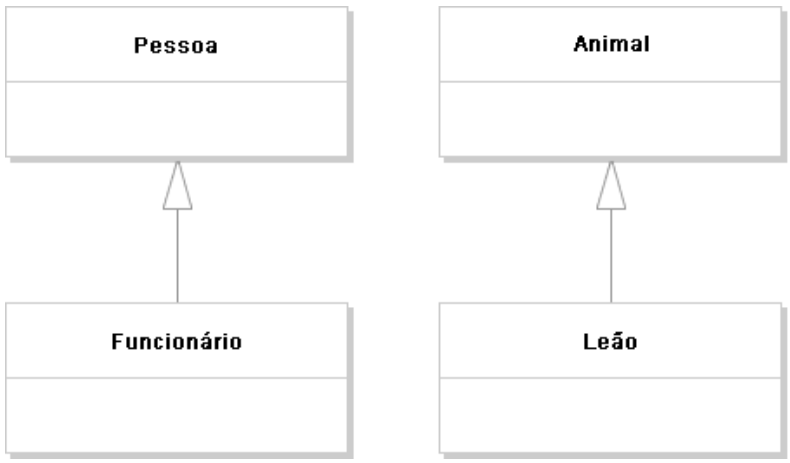

Figura 9 - Exemplo de Especialização/Generalização (Magela, 2006, p.64).

4.4.5.2 RELACIONAMENTO COMPORTAMENTAL OU DINÂMICO

O relacionamento comportamental ou dinâmico está ligado as interações particulares de cada objeto, ou seja, esse relacionamento só existe em um contexto particular, expresso pela interação entre os objetos. Esse relacionamento não faz parte de qualquer definição estrutural do objeto, dessa maneira, se a interação entre eles for retirada, esse relacionamento deixa de existir.

Magela (2006) descreve um exemplo dessa interação, podendo ser representado pelo valor a ser pago por um determinado cliente na locação de um conjunto de filmes, pode ser vista como uma dependência entre esse objeto e os objetos do cliente e filme. Mas, esse relacionamento envolve apenas a interação entre o cliente e o filme, e não faz parte da definição de sua estrutura.

Outro exemplo poderia ser uma classe que calcula a produtividade de um funcionário, essa classe poderá necessitar do auxilio de uma classe que calcule a variância e dispersão matemática dos valores envolvidos.

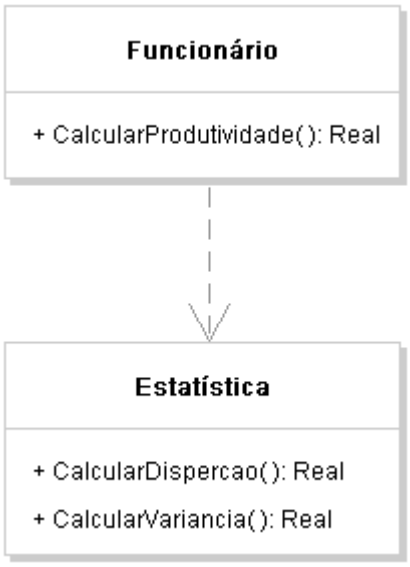

Figura 10 - Exemplo de dependência entre classes (Magela, 2006, p.65).

4.4.5.3 RELACIONAMENTO BIJETOR

Segundo Magela (2006), esse relacionamento faz a ligação entre dois elementos quaisquer de um processo que possuem uma semântica em particular. Ou seja, são relacionamento entre representações diferentes da mesma informação, esses relacionamentos visam manter algum tipo de ligação semântica entre os elementos relacionados.

Existem vários tipos de relacionamento bijetores. Exemplos:

1. Dependência – Um relacionamento geral entre dois elementos de modelagem quaisquer.
2. Realização/Refinamento – Um relacionamento que implica um nível abaixo de abstração.
3. Rastreabilidade – Um relacionamento que significa o antes e o depois da aplicação de um conjunto de técnicas ou um relacionamento entre representações diferentes da mesma função.

(Magela, 2006, p.66).

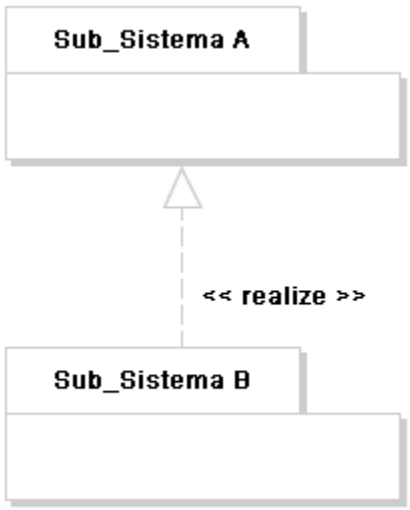

Figura 11 - Exemplo de Realização (Magela, 2006, p.65).

4.5 PARADIGMA FUNCIONAL

Robert W. (2003) determina o paradigma funcional como tendo o principal objetivo de reproduzir as funções matemáticas no maior grau possível, essa abordagem se difere fundamentalmente dos métodos usados com as linguagens

estruturadas. Em uma linguagem estruturada, quando se vai resolver um problema à expressão utilizada é avaliada e o resultado é armazenado em uma parte da memória do computador, representada como uma variável em um programa.

Para executar essa atenção as alocações de células de memória é necessário uma metodologia de programação de baixo nível. Quando se utiliza uma linguagem de montagem para a construção de um programa, muitas vezes, durante sua execução é necessário armazenar os resultados de avaliações parciais de expressões.

Exemplo:

(X + Y) / (A - B)

Durante a execução da expressão acima o valor de (X + Y) é computado primeiro, após sua execução, ele é armazenado enquanto (A - B) é avaliado para posteriormente ser computado. Em linguagens de alto nível, o compilador com o objetivo de ajudar a avaliar o problema, manipula o armazenamento dos resultados intermediários das avaliações de expressões. Porém, o armazenamento intermediário ainda é necessário, mas os detalhes são ocultados do programador.

Contudo, Robert W. (2003) diz que, uma linguagem de programação puramente funcional não usa variáveis ou instruções de atribuição. Isso tira o peso sobre o programador de se preocupar com as alocações de memórias durante a execução de um programa.

Sem a utilização das variáveis, as construções interativas não são possíveis, pelo simples fato e que as instruções e armazenamento de resultados são controlados através de variáveis. A repetição deve ser executada por meio de recursão, ao invés de fazer por meio de lanços de repetição. Programas são definições de funções e de especificação de um programa puramente funcional não tem nenhum estado em termos de semântica operacional. A execução de uma função sempre produz o mesmo resultado quando dados os mesmos parâmetros, isso é denominado transferência referencial.

Uma linguagem de programação funcional apresenta um grupo de funções primitivas, formas funcionais que possibilita a construção de funções mais complexas a partir das funções mais primitivas que são oferecidas, uma operação de aplicação de função e diversas estruturas para representar os dados. Quando se constrói uma algoritmo bem definido com base em linguagem funcional necessita de poucas funções primitivas.

A maioria das linguagens estruturadas (imperativas) oferece um suporte limitado para a programação funcional, inclui algumas tipos de definições de função e facilidades de representação. O principal empecilho para o uso de linguagens imperativas com o intuito de fazer programação funcional é que a função nas linguagens imperativas tem restrições sobre os tipos de valores que podem ser retornados e efeitos colaterais funcionais que podem ocorrer. Linguagens como Pascal e FORTRAN permitem que apenas valores do tipo escalar sejam retornados.

4.6 PARADIGMA LÓGICO

As linguagens de programação lógica são denominadas declarativas, pois os algoritmos desenvolvidos nela são descritos em forma de declarações ao invés de utilizar atribuições e em instruções de fluxo de controle.

Robert W. (2003) afirma que as declarações são, de fato, instruções ou proposições, em logica simbólica. Uma das características essenciais das linguagens de programação logica é sua semântica, chamada de semântica declarativa.

O conceito básico desse modelo é que existe uma maneira simples de determinar o significado de cada instrução, e não depende de que forma é utilizada para resolver um problema. Ou seja, quando se utiliza uma proposição em um programa ela é determinada a partir da própria proposição, diferente das linguagens imperativas que necessitam de informações que não estão contidas ou não estão explicitadas no comando.

Em um ambiente de programação em logica, o programa não apresenta instruções explicitas para a máquina, é estabelecido “fatos” e “regras” sobre a área do problema, utilizando um grupo de sentenças logicas, que são interpretados como programas.

Exemplo:

```
fatorial(0,1).
fatorial(N,Fat):- N>0,
N1 is N-1,
fatorial(N1,Fat1),
Fat is Fat1 * N
```

(Baranauskas, 1993)

O exemplo está ilustrando um programa em Prolog, utilizado para o cálculo do fatorial de um número inteiro. A expressão “fatorial(0,1)” é um fato aonde diz que o fatorial de 0 é 1, As expressões posteriores implica em uma regra aonde o fatorial de N é Fat se o fatorial de N-1 é Fat1 e Fat é Fat1 * N.

Utilizando a programação em lógica, os programas não estabelecem exatamente como um resultado deve ser computado, mas, descrevem os fatos e as regras para que a máquina possa ter um norte de como executar os cálculos. O computador, assume o papel de uma “máquina de inferência”, ou seja, uma que busca uma prova construtiva para uma meta colocada pelo usuário.

5 CARACTERISTICAS DOS MODELOS DE LINGUAGEM DE PROGRAMAÇÃO

5.1 PARADIGMA ESTRUTURADO

5.1.1 CONCEITO

Esse paradigma apresenta que os programas podem ser desenvolvidos com base em três estruturas: sequência, decisão e repetição. O modelo estruturado implica na criação de estruturas simples em seus programas, usado para desenvolver sub-rotinas e as funções. Esse modelo foi o principal paradigma que auxiliou na criação de software entre a programação linear e a programação orientada a objetos.

Contudo, a programação estrutura foi sucedida pela de orientação a objetos. Mas, ainda hoje ela continua sendo utilizada e muito influente, pelo fato de que grande parte das pessoas ainda aprende programação através desse modelo.

5.1.2 LINGUAGENS

As principais linguagens que fazem uso do paradigma estrutura são respectivamente:

- C
- BASIC
- PASCAL
- COBOL

5.1.3 VANTAGENS

A programação estruturada concede grande flexibilidade e a possibilidade de quebrar os problemas em diversos outros subproblemas. Esse modelo permite uma boa legibilidade, é de fácil compreensão, e sua estrutura é bem simples, por esse motivo muitos começam a aprender programação através desse modelo.

5.1.4 DESVANTAGENS

Os dados são separados das funções demonstrando certa lentidão. Diversas operações simples não possíveis, como a implementação da troca de parâmetros, ou seja, mudanças na estrutura dos dados acarretando na alteração de todas as funções que estão relacionadas. Esse modelo gera sistemas difíceis de serem mantidos.

5.2 PARADIGMA ORIENTADO A OBJETOS

5.2.1 CONCEITO

O conceito de programação orientada a objetos é fundamentando na interação e composição de diversas unidades de um software, os denominados objetos. As atividades de sistema com base na orientação a objetos se dão através dos relacionamentos de troca de informações (comunicação) entre os objetos contidos dentro do sistema.

Esses objetos que são denominados classes, o comportamento de cada classe é definida com base em seus métodos e as características ou estados de uma classe são respectivamente seus atributos. A definição do relacionamento entre os objetos são definidas nos métodos e atributos que a classe possui.

5.2.2 LINGUAGENS

As principais linguagens que fazem uso do paradigma estrutura são respectivamente:

- Smalltalk
- Python
- Ruby
- C++
- Object Pascal
- Java

- C#
- Oberon
- Ada
- Eiffel
- Simula
- .NET

5.2.3 VANTAGENS

O paradigma orientado a objetos possui todas as vantagens que o estruturado, flexibilidade, legibilidade, fácil compreensão, quebrar em subproblemas, entre outras. Uma de suas caracterizas é que a alteração de um modulo não gera a necessidade da modificação em outros módulos do sistema, ou seja, quanto mais esse módulo for independe, maior a chance da reutilização do código em outras partes da aplicação.

A sua principal vantagem é a facilidade da reutilização de códigos, possibilitando trabalhar em um nível superior de abstração, e a utilização de um único padrão conceitual durante todo o processo de desenvolvimento do sistema.

5.2.4 DESVANTAGENS

A programação orientada a objetos exige uma forma de pensamento complexa, por isso muitos acabam não compreendendo e ela acaba não sendo usada por grande parte dos programadores. Esse modelo tem um uso elevado da memória da máquina, é mais difícil de modelar e a dependência de funcionalidades já implementadas em outras classes.

5.3 PARADIGMA FUNCIONAL

5.3.1 CONCEITO

Esse paradigma se baseia nas funções matemáticas para o desenvolvimento de algoritmos de programação. Esse modelo evidencia a utilização e funções, que

são tratadas como valores de primeira importância, ou seja, as funções contem parâmetros ou valores de entrada para outras funções e podem ser os valores de retorno ou saída de cada função.

5.3.2 LINGUAGENS

As principais linguagens que fazem uso do paradigma estrutura são respectivamente:

- Lambada
- LISP
- Scheme
- ML
- Miranda
- Haskell

5.3.3 VANTAGENS

A alocação automática das células de memória são uma das suas principais vantagens por causa disso, os efeitos colaterais dos cálculos da função são eliminados. A utilização da recursividade em programação funcional pode assumir varias formas e é considera uma técnica mais poderosa do que os laços de repetição.

Essa linguagem assegura que o resultado será o mesmo para um dado grupo de parâmetros independente da onde, e quando é avaliado em computação independente para execução paralela.

5.3.4 DESVANTAGENS

A programação funcional há grande possibilidade de criar códigos complexos, além de não ser um paradigma muito disseminado. Esse modelo não faz a utilização explicita da alocação de memória e também não há o uso de variáveis, ou seja, falta diversas construções que são essenciais em linguagens estruturadas.

5.4 PARADIGMA LÓGICO

5.4.1 CONCEITO

O conceito de paradigma lógico é baseado nas relações entre entrada e saída de dados. A descrição do seu algoritmo é com base no estilo declarativo, ele utiliza de características de programação estrutura com o intuito de aumentar a eficiência.

Este paradigma não apresenta instruções diretas para a máquina como em outros modelos. O paradigma lógico é utilizado normalmente em aplicações de prototipação, sistemas especialistas, banco de dados, etc.

5.4.2 LINGUAGENS

As principais linguagens que fazem uso do paradigma estrutura são respectivamente:

- Popler
- Conniver
- QLISP
- Planner
- Prolog
- Mercury
- Oz
- Frill

5.4.3 VANTAGENS

Esse modelo tem características parecidas com o paradigma funcional no qual executa a alocação automática das células de memória, mas, possibilita uma abstração de alto nível.

5.4.4 DESVANTAGENS

As variáveis de um programa não possuem tipos, dentro desse paradigma também não é possível uma estruturação modular do conhecimento.

5.5 RAKING DE LINGUAGENS DE PROGRAMAÇÃO

Posição Maio 2013	Posição Maio 2012	Linguagens de programação	Avaliação Maio 2013	Avaliação Maio 2012
1	1	C	18.729%	+1.38%
2	2	Java	16.914%	+0.31%
3	4	Objective-C	10.428%	+2.12%
4	3	C++	9.198%	-0.63%
5	5	C#	6.119%	-0.70%
6	6	PHP	5.784%	+0.07%
7	7	Visual Basic	4.656%	-0.80%
8	8	Python	4.322%	+0.50%
9	9	Perl	2.276%	-0.53%
10	11	Ruby	1.670%	+0.22%
11	10	JavaScript	1.536%	-0.60%
12	12	Visual basic .NET	1.131%	-0.14%
13	15	Lisp	0.894%	-0.05%
14	18	Transact-SQL	0.819%	+0.16%
15	17	Pascal	0.805%	0.00%
16	24	Bash	0.792%	+0.33%
17	14	Delphi/Object pascal	0.731%	-0.27%
18	13	PL/SQL	0.708%	-0.41%
19	22	Assembly	0.638%	+0.12%
20	20	Lua	0.632%	+0.07%

Tabela 3 - Ranking de linguagens de programação Maio 2013 (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, 31/05/2013 – 18h 15min)

6 CONSIDERAÇÕES FINAIS

Com base na analise dos dados apresentados, pode-se perceber que cada modelo possui sua própria particularidade e maneiras para solucionar os problemas. Cada modelo tem suas vantagens e desvantagens e formas diferentes de serem utilizados, cabe ao programador conhecer os modelos e escolher o que melhor se encaixa no seu projeto.

O paradigma estrutural demonstra que um programa pode ser desenvolvido se baseando em três tipos de estrutura: sequência, decisão e repetição. Esse modelo possibilita quebrar o problema em subproblemas uma de suas fortes características é sua legibilidade e fácil compreensão.

A linguagem estruturada é um dos modelos mais utilizados para quem está começando a aprender programação, principalmente nas faculdades e cursos.

A orientação a objetos é caracterizada pelo uso de objetos, classes, métodos e pelo relacionamento que possuem para transferência de informações e dados. Sua principal vantagem é a facilidade da reutilização de códigos em vários ponto do sistema um alto nível de abstração.

A reutilização de código permite que programador economize muito tempo de trabalho e principalmente, evita a alteração de vários pontos do sistema quando se faz necessário à mudança de algum ponto do sistema que é referenciado em outra parte do sistema, possibilitando a alteração uma única vez.

O modelo funcional se baseia totalmente em funções matemáticas, uma linguagem funcional utiliza funções para a passagem de valores de entrada e saída. Além disso as funções se comunicarem entre elas possibilitando a passagem de informações e dados necessários para a execução dos algoritmos do programa e gerar os resultados.

No paradigma lógico é denominado declarativo, pelo fato de seus algoritmos serem descritos em forma de declarações ao invés de utilizar atribuições e instruções de fluxo de controle com a máquina.

Esse conceito consiste em uma maneira simples de determinar o significado de cada instrução, e não depende de que forma é utilizada para resolver o problema. Uma linguagem lógica utiliza de suas próprias proposições para resolver o problema e gerar o resultado, diferente da linguagem estruturada que necessitam de informações que não estão contidas ou não estão explicitas nos comandos.

Contudo, fica na responsabilidade do programador o qual conceito ele irá utilizar e qual linguagem ele vai adotar em seu projeto, porém os conceitos abordados podem ser utilizados em mais de uma linguagem e pode haver dois ou mais modelos sendo usados juntos no sistema.

O programador não tem a necessidade de se limitar apenas a um conceito podemos utiliza-los da forma que nós convém, como por exemplo, usar conceito de orienta a objetos e estruturas em um único projeto.

Esse estudo descreve alguns dos principais modelos de linguagem de programação adotados atualmente e possivelmente em um futuro próximo haverá o surgimento de novos conceitos e modelos. Com base nesse estudo espero agregar conhecimento aos leitores e gerar novas ideias para auxiliar na abstração dos problemas e diminuir a complexidade das linguagens de programação.

.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. Fundamentos da programação de computadores. 2. ed. São Paulo: Pearson Prentice Hall, 2007. 428 p.

MAGELA, Rogério. Engenharia de Software Aplicada: Princípios. 2. ed. Rio de Janeiro: Alta Books, 2006. 335 p.

MENDES, Douglas Rocha. Programação Java com Ênfase em Orientação a Objetos. Novatec, 2008. 456 p.

SEBESTA, Robert W. Conceitos de Linguagens de Programação. 5. ed. São Paulo: Bookman Companhia Ed, 2003. 638 p.

TUCKER, Allen; NOONAN, Robert. Linguagens de Programação: Princípios e Paradigmas. 2. ed. São Paulo: Mcgraw Hill, 2008. 630 p.

Baranauskas, M. C. C. (1993). Procedimento, função, objeto ou lógica? Linguagens de programação vistas pelos seus paradigmas. Computadores e Conhecimento: Repensando a Educação. Campinas, SP, Gráfica Central da Unicamp.

Davis Jr, Clodoveu Augusto, and Karla AV Borges. "GIS Orientado a Objetos na Prática." Anais do GIS Brasil'94 (1994): 18-28.

Dall'Oglio, P. (2007). PHP: programando com orientação a objetos. São Paulo: Novatec.

Jungthon, G., & Goulart, C. M. (2010). Paradigmas de Programação. Faculdade de Informática de Taquara, Taquara.

Souza, Cleidson. "Orientação a Objetos.", Departamento de Informática, Universidade Federal do Pará.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **Citação**: NBR-10520/ago - 2002. Rio de Janeiro: ABNT, 2002.

_____. **Referências**: NBR-6023/ago. 2002. Rio de Janeiro: ABNT, 2002.