Aula 2 - Especificações I

Prof. Dr. Newton Maruyama

Computador, Programas, etc.

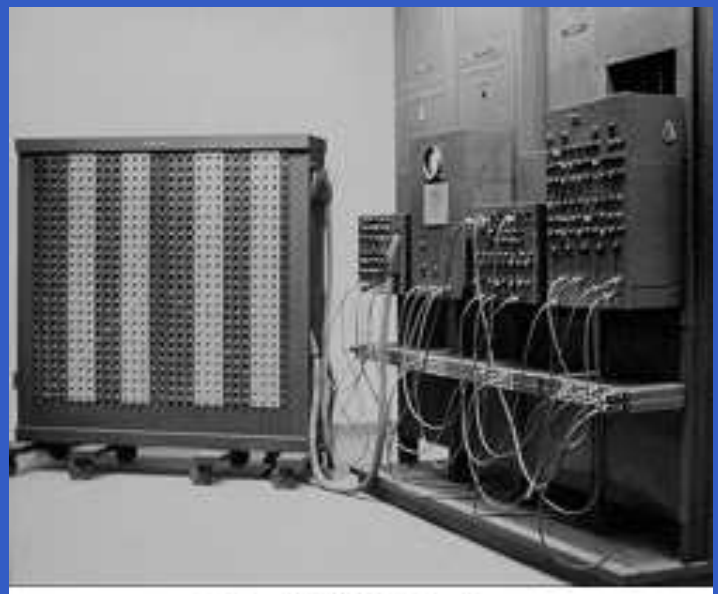
- Máquina de Turing: estrutura matemática para realização de computação, representação de um algoritmo particular. Máquinas de Estado Finitas, por exemplo, são Máquinas de Turing.
- Computadores são Máquinas de Turing Universais, pois podem ser programadas para representar qualquer Máquina de Turing.
- Na verdade, a definição de Máquina de Turing Universal requer a utilização de recursos infinitos.

Microprocessadores

- Microprocessadores são a materialização de Máquinas de Turing Universais através de circuitos eletrônicos (transistores).
- Todo microprocessador possui um conjunto de instruções que permitem programá-lo. As instruções são representadas por códigos numéricos (códigos de máquina).

O início

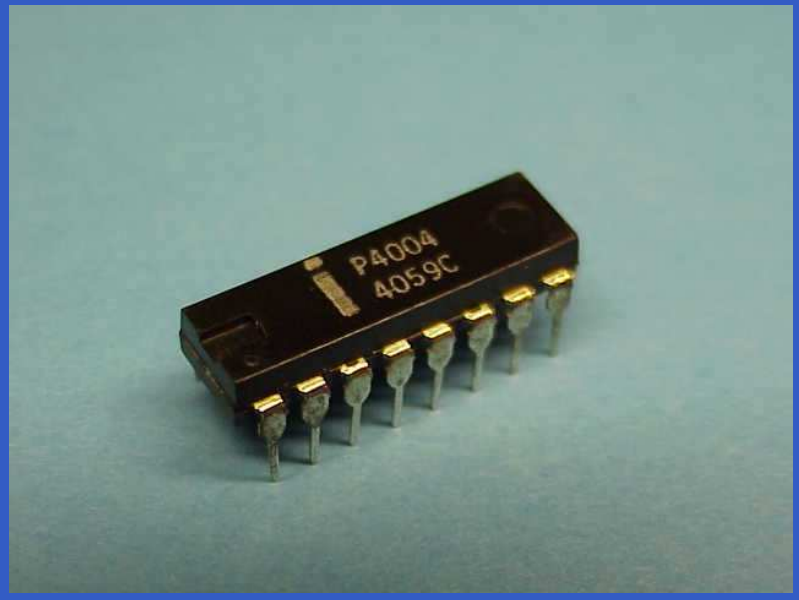
- O ENIAC, construído em 1946, é considerado o primeiro computador.
- 70 mil resistores e 17.468 válvulas a vácuo.
- 30 toneladas.



The ENIAC Today

O primeiro microprocessador

- O Intel 4004 de 4 bits é considerado o primeiro projeto de microprocessador.
- Clock de 108Khz, 2300 transistores, instruções de 8 bits.



O primeiro microcomputador: ALTAIR

- Intel 8080.
- 256 bytes de memória, sem teclado, sem monitor.
- Programação através de chaves.

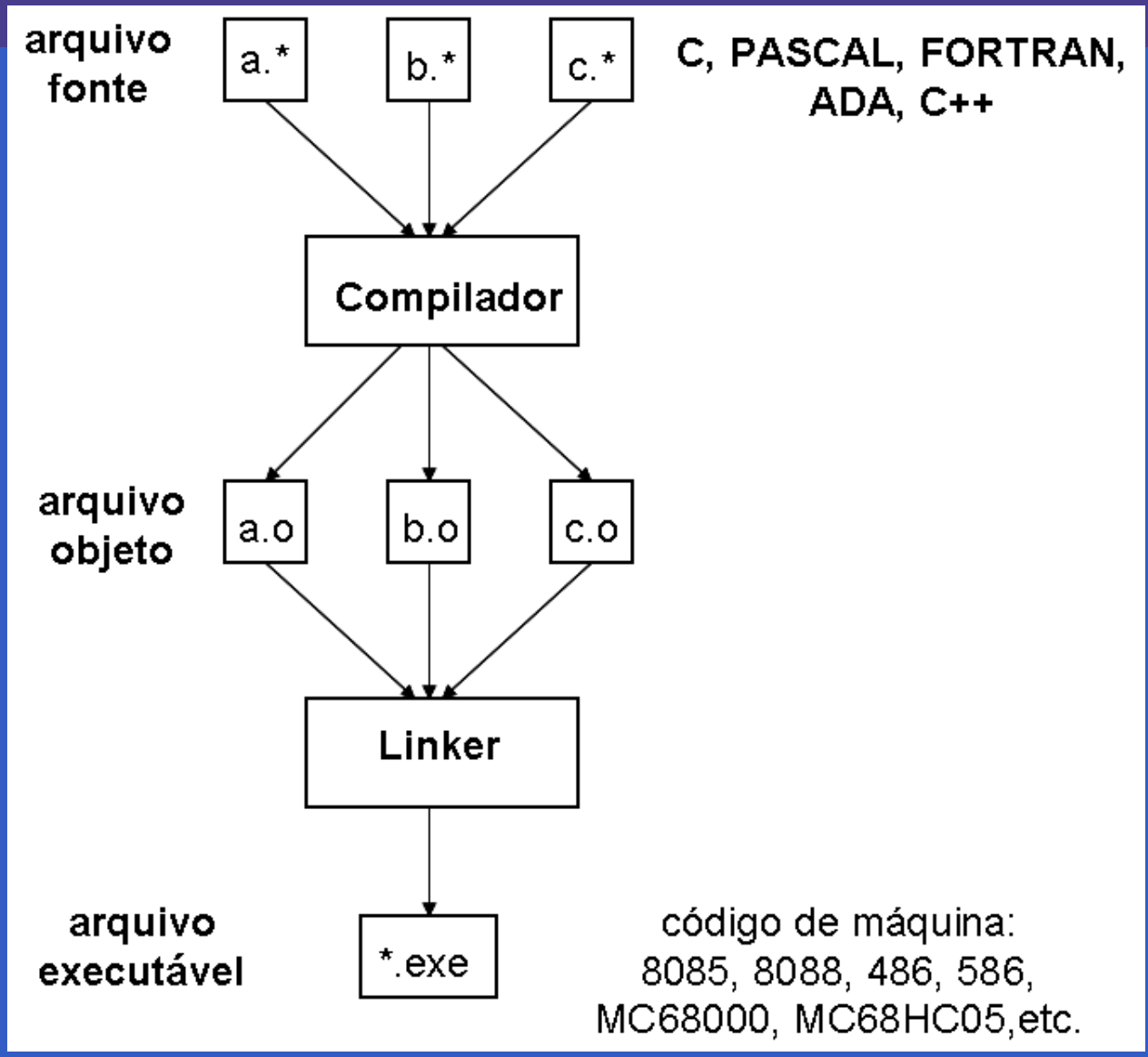
Programas

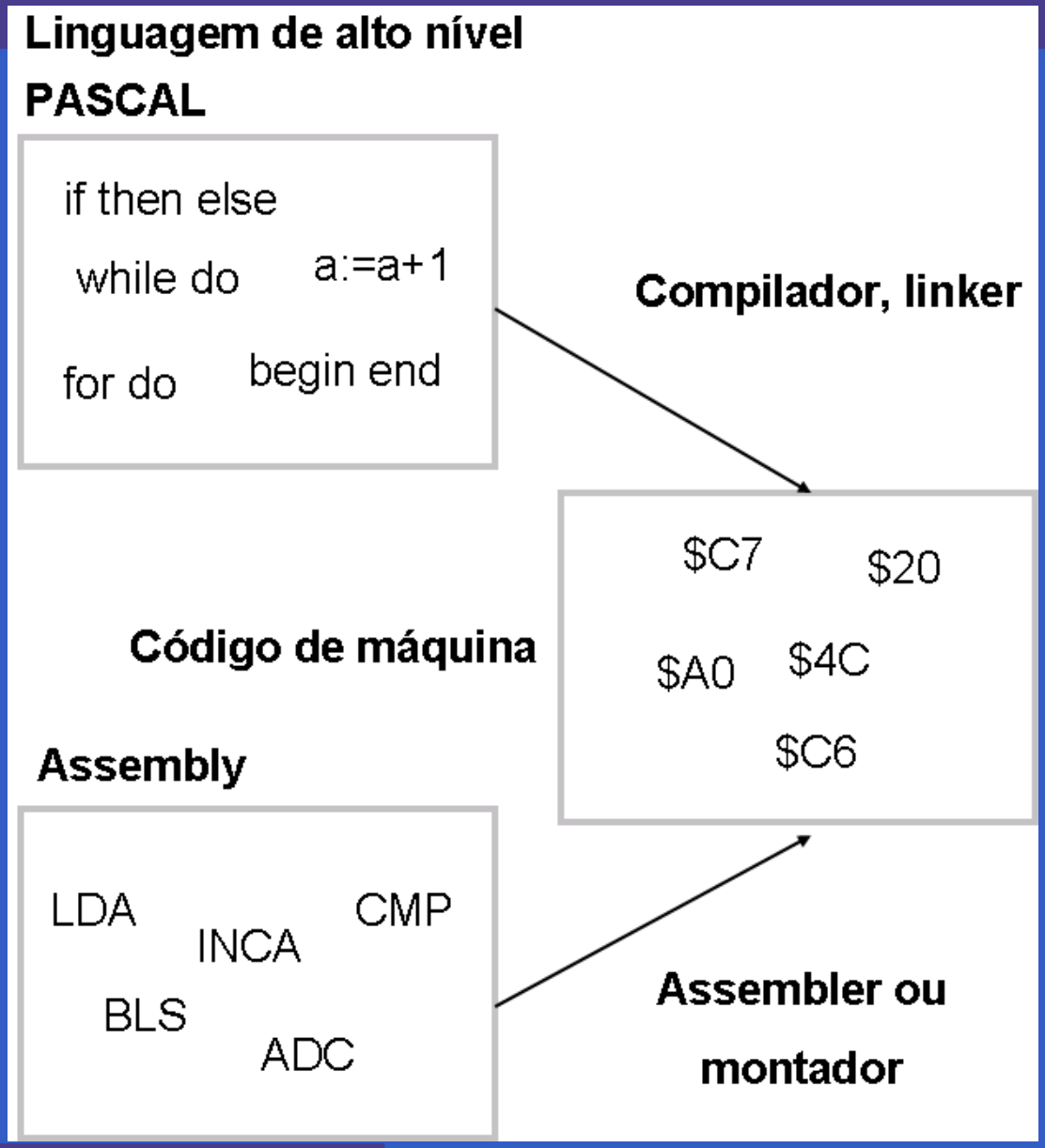
- O ato de programar consiste basicamente em representar uma Máquina de Turing através de uma seqüência de códigos de máquina.
- Em geral, criar programas através de códigos de máquina é muito difícil.

Linguagens de Programação

- Como programar em código de máquina é muito dificil foi necessário criar maneiras mais simples de programar.
- Desta forma, foram criadas as linguagens de programação que permitem abstrações mais poderosas.
- O primeiro passo foi criar as Linguagens Assembly que são equivalentes aos códigos de máquinas.
- Num segundo passo foram criadas as linguagens de alto nível como FORTRAN, PASCAL, ADA, C, C++, Java, etc.

O processo de geração de código




Linguagem de alto nível
PASCAL
if then else
while do
a:=a+1
for do
begin end
Compilador, linker
$C7
$20
$A0
$4C
$C6
Código de máquina
Assembly
LDA
INCA
CMP
BLS
ADC
Assembler ou
montador

```
x := x + 1;
if x > 0 then
    y := y + 1
else
    y := 0;
z := x + y;
```

PASCAL

```
X   EQU $0100
Y   EQU $0101
Z   EQU $0102
    ORG $0200
     :
     :
     :
    LDA   X     C6 01 00
    INCA        4C
    STA   X     C7 01 00
    LDA   #$00  A6 00
    CMP   X     C1 01 00
    BLS   L1    23 06
    LDA   Y     C6 01 01
    INCA        4C
    STA   Y     C7 01 01
    BRA   L2    20 03
L1  LDA   #$00  A6 00
    STA   Y     C7 01 01
L2  LDA   X     C6 01 00
    ADC   Y     C9 01 01
    STA   Z
```

Assembly,
Código de máquina
MC68HC05

Programas de grande porte ?

- A produção de programas de grande porte requer a coordenação de várias atividades envolvendo um grande número de pessoas.
- Os programas são em geral montados como uma coleção de módulos individuais, escritos possivelmente por várias pessoas diferentes e utilizando diferentes linguagens.

Programas de grande porte ? ...

- Por outro lado, programas de grande porte podem envolver até vários milhões de linhas e são escritos por dezenas de programadores num período de muitos anos, além disso envolvem centenas de módulos com interações complexas entre eles.
- Programas com tais características envolvem sistemas de programação de computadores de grande porte, aplicações militares, sistemas de gerenciamento de bancos, etc.

Metodologias

Para o desenvolvimento de programas de porte médio (algumas milhares de linhas) até programas de grande porte se faz necessário a utilização de metodologias de desenvolvimento de software.

Ciclo de Vida

Em geral, reconhece-se que o processo de desenvolvimento de software pode ser dividido em cinco fases:

1. Análise e especificação de requisitos;
2. Projeto de software;
3. Implementação ou codificação;
4. Certificação;
5. Manutenção.

Análise e especificação de requisitos

- o resultado dessa fase é um documento que indica O QUE o sistema deve fazer;

Projeto de software

- iniciando com o documento de requisitos, os programadores devem gerar o documento de especificação de projeto de software identificando todos os módulos e suas interfaces;

Implementação ou codificação

- o código do programa é gerado;

Certificação

- a principal atividade dessa fase é a realização de testes de aceitação para checar se o programa atende as especificações;

Manutenção

- após a entrega do sistema é inevitável a necessidade da realização de manutenção para correções de erros. A importância dessa fase pode ser vista pelo fato de que os custos de manutenção podem atingir custos da mesma ordem de grandeza que as demais fases.

Precisamos de metodologia !

O ponto maior dessa discussão é chamar atenção ao fato que por trás de desenvolvimento de software existem muito mais atividades do que aprender a sintaxe de uma linguagem específica e a habilidade de codificar pequenos trechos de programa.

Custos de manutenção

Em estudo da década de 80, os custos de manutenção foram estimados em 70% do custo total de desenvolvimento.

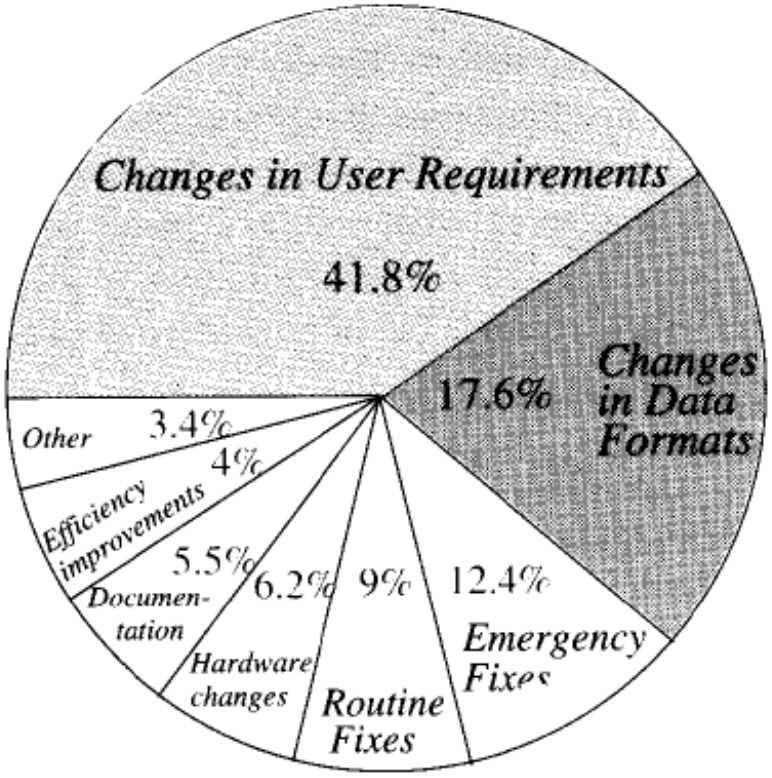


Figura 1: Análise dos custos de manutenção de software [Lientz and Swanson, 1980].

Programação em ponto grande
versus
programação em ponto pequeno

O que é um programa grande ?

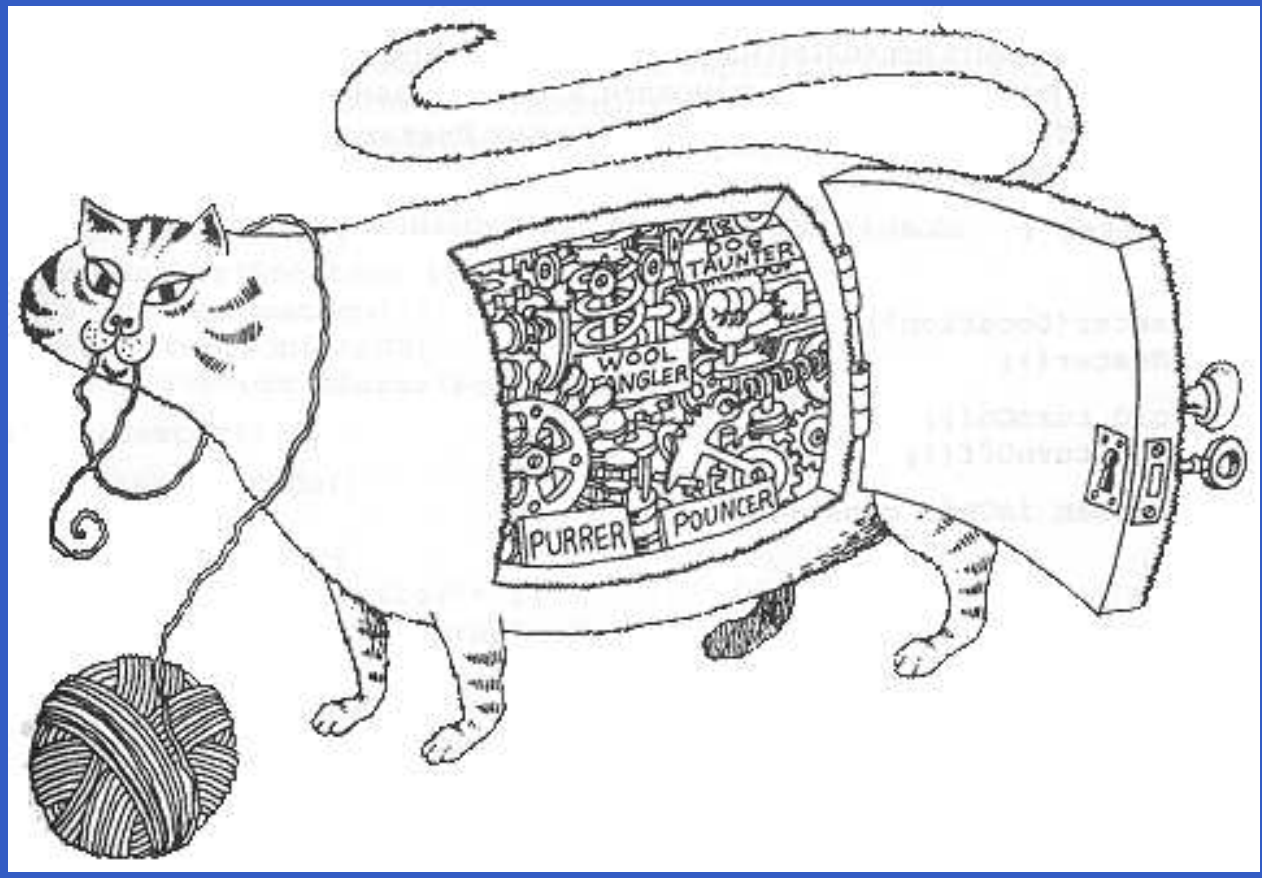
- Um programa grande vai aqui ser definido como um programa contendo no mínimo muitas milhares de linhas com alto grau de complexidade. A idéia de programação em ponto grande está mais associada à complexidade do que ao tamanho do programa, mas usualmente programas grandes são programas complexos.
- A programação em ponto grande requer a utilização de diferentes metodologias para o desenvolvimento de programas quando comparado a programação em ponto pequeno.

É reconhecido que as seguintes metodologias são necessárias para o desenvolvimento de programas grandes:

1. técnicas de gerenciamento para controlar e monitorar o progresso da equipe envolvida;
2. metodologias de projeto para atacar problemas específicos de programação em ponto grande (como modularização e encapsulamento de dados);
3. ferramentas e ambientes para auxiliar na parte criativa e automatizar a parte não criativa do desenvolvimento de software.

Modularização

Encapsulamento de dados



A principal característica de programação em ponto grande é a decomposição do sistema em módulos enquanto que a principal característica de programação em ponto pequeno é a produção de módulos individuais.

Um exemplo

Vamos aqui demonstrar a importância da subdivisão do sistema em módulos através de um exemplo bastante simples. Vamos supor que se deseja desenvolver um programa que calcula a média e desvio padrão das notas de uma classe de alunos.

Num primeiro nível de abstração a seguinte solução poderia ser apresentada:

Program *CalculaEstatisticas*
inicio
Use a Entrada-1 para obter $N, LimInf, LimSup$
Repita 3 vezes
inicio
Use a Entrada-2 para receber N notas e armazene em TabelaNotas
Determine a média utilizando TabelaNotas;
Determine o desvio padrão;
Imprima os resultados;
fim
fim.

Nesse nível de abstração podemos identificar cinco módulos:

1. *Entrada-1*: esse módulo requer ao usuário a entrada dos seguintes valores: *N* (número de notas) , *LimInf* (A menor nota válida) e *LimSup* (A maior nota válida).
2. *Entrada-2*: esse módulo é responsável pela entrada dos valores das notas. É checado se as notas estão dentro dos limites especificados por *LimInf* e *LimSup*.

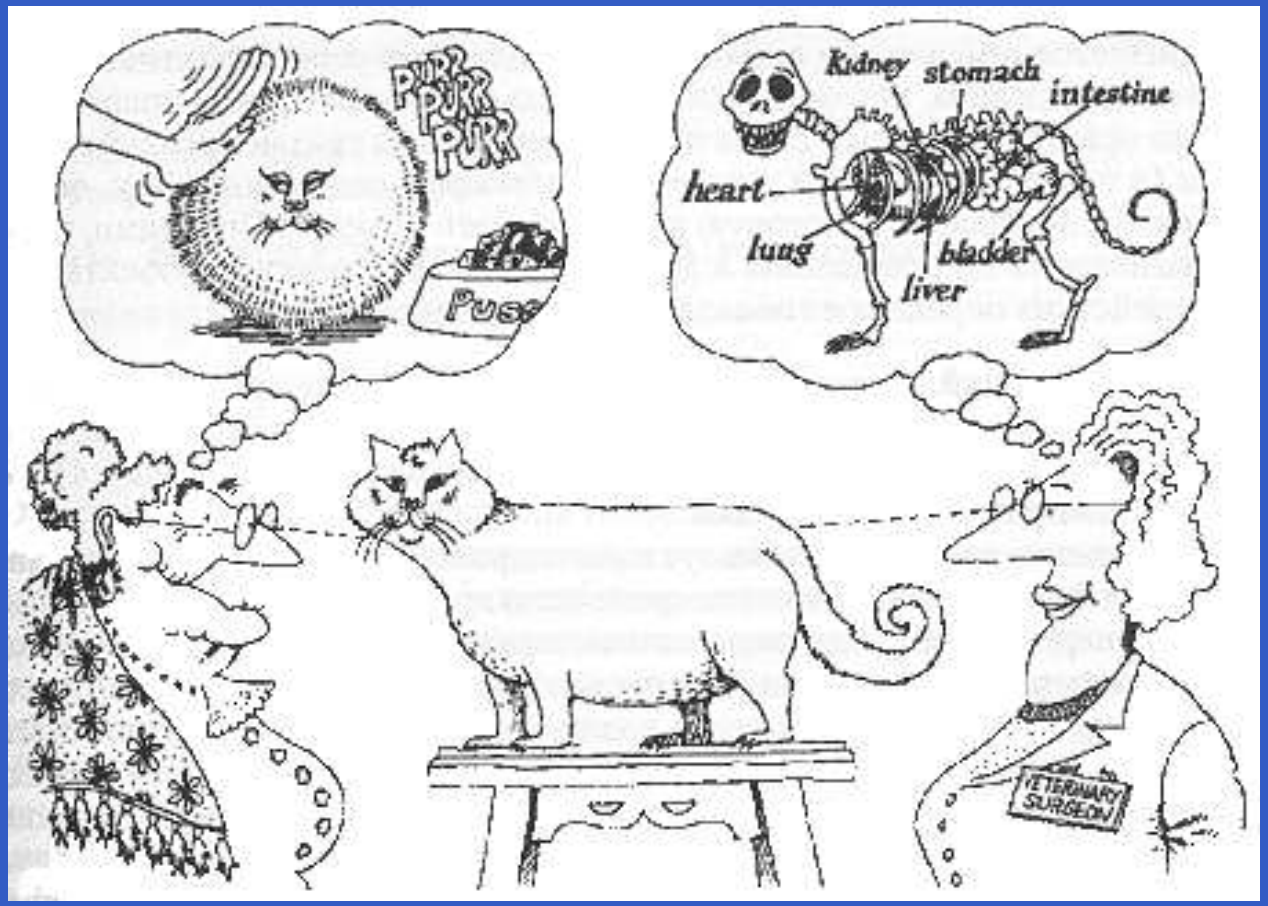
3. *CalculaMedia*: a esse módulo é passado a estrutura de dados *TabelaNotas* com *N* elementos e a média das notas é calculada.
4. *CalculaStd*: esse módulo recebe a estrutura de dados *TabelaNotas* com *N* elementos e a média das notas. Com esses valores o desvio padrão é calculado.
5. *Saída*: imprime os valores *N*, *LimInf*, *Limsup*, *Media* e *Std*

Nesse ponto é possível identificar que a estrutura de dados *TabelaNotas* deve ser capaz de realizar as seguintes operações:

- *Armazena(nota,TabelaNotas,i)*
- *Recupera(nota,TabelaNotas,i)*

Após o projeto do primeiro nível de abstração, deve-se verificar se cada um dos módulos é simples o suficiente ou se são necessários um maior número de refinamentos. Por exemplo, pode-se decidir que o módulo denominado *Entrada-2* necessita de uma posterior decomposição:

Abstração


PURR PURR PURR
Kidney
stomach
intestine
heart
lung
bladder
liver
VETERINARY SURGEON

Detalhamento de Entrada-2

```
proc Entrada-2 inicio
                    i := 0;
                    repita ate para N notas
                    inicio
                       PegaDados(nota, flag);
                       se flag = TRUE então
                                              i := i + 1;
                                              Armazena(nota, TabelaNotas, i);
                                          senão
                                              TratatamentoDeErros;
                    fim;
  fim.
```

Esse refinamento do módulo *Entrada-2* utiliza duas subrotinas:

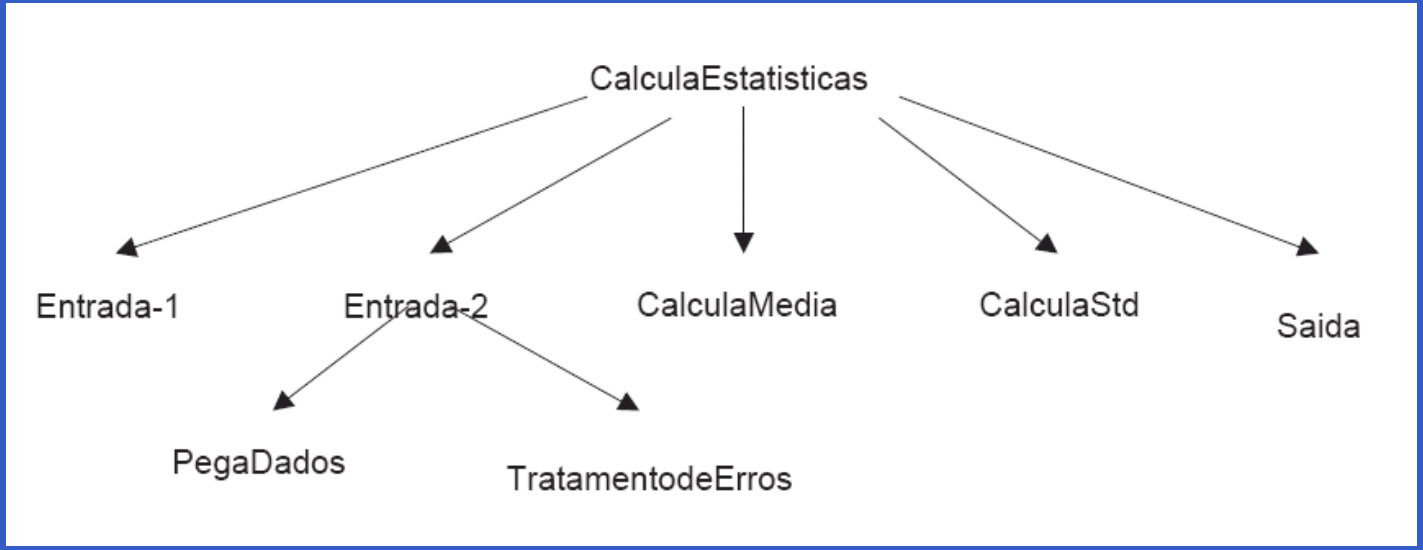
1. *PegaDados:* esse módulo requisita ao usuário a entrada de uma única nota. O valor da nota é checado se pertence ao intervalo dado por *LimInf* e *LimSup*. A subrotina retorna o valor da nota e a variável booleana *flag* que indica se o valor de nota é válido ou não.
2. *TratamentoDeErros:* esse módulo é ativado se um valor de nota não permitido for digitado. Realiza todas as rotinas de tratamento de erro e a impressão das mensagens de erro.

- Além disso deve-se decidir como o tipo abstrato de dados *TabelaNotas* será implementado. Por exemplo pode-se utilizar um vetor ou uma lista ligada.
- Alternativamente pode-se postergar tal escolha e apenas lembrar que o tipo abstrato de dados é definido em termos das duas operações denominadas *Armazena* e *Recupera*.

Um exemplo...

Ao fim desse nível de abstração vamos tentar realizar uma representação hierárquica do programa. A Figura 1 ilustra a representação hierárquica desse nível de abstração através de uma representação em árvore.

Representação hierárquica.



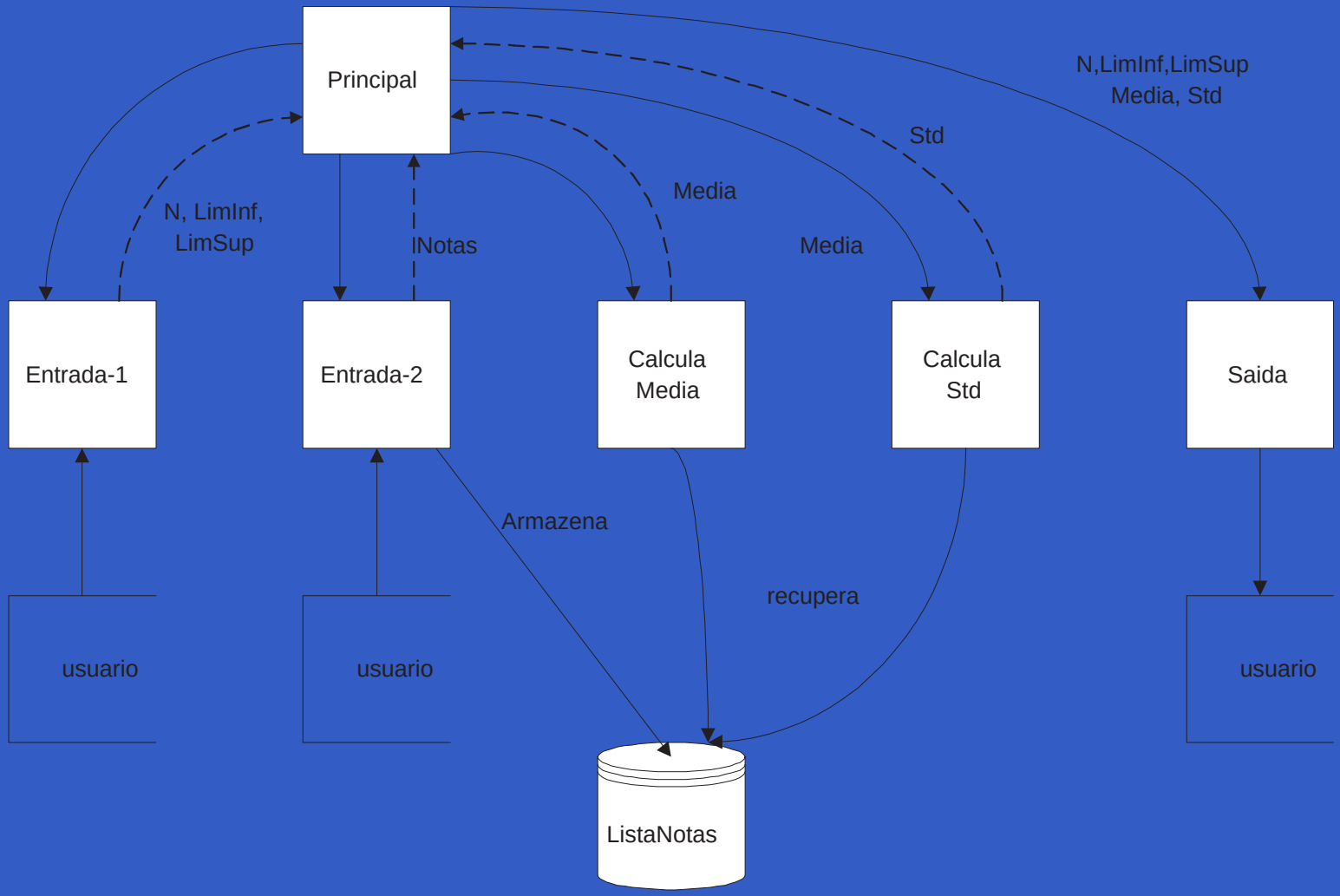
O procedimento de refinamento dos módulos em módulos menores gera um grande número de módulos simples e uma coleção de estrutura de dados.

O documento de projeto do programa - 1

O documento de projeto do programa enumera os componentes individuais contidos no programa e especifica as relações estruturais entre os componentes. Especificamente deve conter três seções principais:

- Diagrama estrutural: é uma representação gráfica das relações estruturais entre os módulos e entre os módulos e a estrutura de dados.
  A Figura abaixo ilustra o diagrama de fluxo de dados correspondente.

Diagrama de Fluxo de Dados



O documento de projeto do programa - 2

- Especificação dos módulos: para cada módulo contido no diagrama estrutural deve existir uma especificação que descreve a interface externa do módulo com o resto do programa.
- Dicionário de dados: para cada estrutura de dados importante deve-se descrever os campos de informação e as operações importantes que podem ser realizadas sobre a estrutura de dados.

Linguagens procedurais x Linguagens declarativas

- Vamos mostrar que existem vários tipos de linguagem de programação e que a escolha da linguagem tem influência decisiva nos métodos de desenvolvimento de software.
- A maioria das linguagens de programação existentes hoje são classificadas na classe das linguagens procedurais ou imperativas.
- As linguagens declarativas são uma outra classe de linguagens que envolve linguagens lógicas e linguagens funcionais.

Linguagens procedurais ou imperativas

- As linguagens imperativas constituem a maior parte das linguagens existentes e envolve linguagens como FORTRAN, Pascal, Algol, Smalltalk, Ada, etc.
- O que caracteriza essas linguagens são as abstrações baseadas na Arquitetura de computadores de *Von Neumann*.

- Essa influência pode ser verificada através das seguintes características:
  1. Variáveis: um dos componentes principais da arquitetura de computadores é a memória onde os dados são armazenados.
  2. Operações de atribuição: Os valores devem ser armazenados, ou seja, atribuídos para uma determinada região da memória.
  3. Repetição: um programa usualmente realiza uma tarefa através da execução de uma seqüência de passos elementares repetidamente.

Linguagens declarativas

- Uma outra classe de linguagens de programação são as *linguagens declarativas* como por exemplo programação lógica (logic programming).
- Usualmente o desenvolvimento de programas se inicia através de uma especificação de requisitos declarativos no qual é enfatizado O QUE o usuário necessita e O QUE o sistema é suposto fazer.

- Progressivamente essas especificações evoluem para uma descrição procedural de COMO o problema é resolvido mecanicamente.
- No presente momento a decomposição do sistema em módulos e a implementação dos módulos internos não pode ser feita automaticamente. Essa atividade requer a criatividade do programador.

- Isso consome muito tempo, é caro, além de ser facilmente sujeito a erros. Tal tarefa é necessária porque computadores convencionais não podem executar diretamente descrições declarativas.
- Portanto, requisitos devem ser traduzidos para uma descrição procedural. A programação em lógica tenta fazer com que as especificações possam ser executadas diretamente sem necessidade de refinamento posterior. O programador declara os *fatos* e as propriedades que descrevem o problema para o qual a solução é procurada.

- Essa informação é então utilizada pelo sistema para inferir a solução sem que seja necessário uma tradução para descrições procedurais.
- A linguagem PROLOG é a linguagem lógica mais popular, entretanto vamos aqui ilustrar um exemplo de especificação declarativa através da linguagem de base de dados relacionais SQL.

- Uma base de dados relacionais pode ser vista como uma tabela de registros denominado *tuples*.
- Em SQL a recuperação de um dado é acompanhado pelo comando SELECT. Por exemplo, se uma relação CLASS contém *tuples* do seguinte tipo:

FirstName	LastName	Sex	WhenBorn	WhereBorn
Jose	Silva	M	1974	SP
Joao	Costa	M	1978	Rio
Maria	Nobre	F	1981	Rio

- Um exemplo da utilização do comando SELECT seria:

```
SELECT *
   FROM CLASS
   WHERE SEX = ''M''
   AND WHENBORN < 1980
```

- É fácil de verificar que a linguagem seleciona a informação armazenada na base de dados através da especificação de relações lógicas que caracterizam a informação desejada.
- Nada é dito a respeito de como acessar a informação na base de dados.

- É interessante observar que as primeiras gerações de linguagens de banco de dados requeriam que o usuário especificasse como acessar o banco de dados através de ponteiros.
- A corrente abordagem declarativa é mais adequada principalmente para usuários que não são programadores.

Linguagens funcionais ou aplicativas

- As *linguagens funcionais* ou *aplicativas* se caracterizam pela utilização de funções e aplicações de funções.
- As linguagens funcionais possuem quatro componentes:

  1. Um conjunto de funções primitivas,

     ```
     ( CAR (QUOTE (A B C))) = A
     ```

     A função QUOTE transforma o seu argumento numa constante enquanto CAR retira o primeiro elemento de uma lista.

2. Um conjunto de formas funcionais: são mecanismos através dos quais funções podem ser combinadas para criar novas funções,

```
(DEFINE ( TOTHEFOURTH ( LAMBDA(X) (SQUARE SQUARE X))))
```

Aqui uma a função TOTHEFOURTH que eleva uma número a sua quarta potência é definida em função da função SQUARE(X).

3. A operação aplicação: é o mecanismo para aplicar a função para os seus argumentos e produzir um valor.

   ```
   ((LAMBDA (X Y) (PLUS X Y)) 2 3)
   ```

   Aqui X=2 e Y=3 resultando em 5 através da execução da função PLUS.

4. Um conjunto de dados.

- Apesar de programação funcional parecer ser de mais alto nível do que programação imperativa a sua execução é bastante ineficiente devido não só ao grande número de chamadas de funções, mas também pelo fato de que muitos objetos são criados e descartados dinamicamente (em tempo de execução).

Programação Estruturada

- A programação estruturada é uma técnica de desenvolvimento de programas baseada em dois elementos básicos:
  1. o refinamento sucessivo,
  2. e o uso racional de GOTO's

Refinamento sucessivo

- Por refinamento sucessivo, entende-se uma técnica para projetar sistemas de software de forma a facilitar o projeto, a implementação, testes e manutenção.
- Dentro dessa técnica, um projeto começa com uma idéia inicial: uma visão de como se gostaria que fosse o sistema já pronto.

- Essa idéia é então refinada sucessivamente em novos subprojetos até que se chegue a um nível de detalhamento de fácil implementação.
- Para isso aplica-se no desenvolvimento do software alguns conceitos largamente utilizados em engenharia:
  1. projeto top-down,
  2. concepção hierárquica
  3. desenvolvimento modular por *caixas pretas*.

Uso racional de GOTOs

- A questão com o GOTO se relaciona com o aumento desnecessário da complexidade dos programas proveniente do seu emprego indiscriminado.
- Em outras palavras, o GOTO possibilita a implementação de infinitas formas de estruturas de controle, entretanto esse aumento de versatilidade provoca uma perda de entendimento do funcionamento do programa.
- A programação estruturada implica na composição de programas a partir de uma base limitada de blocos construtivos simples.

Projeto Top-Down

- Esse é o primeiro conceito que integra a técnica de refinamentos sucessivos. Trata-se de um exercício de abstração, onde em primeiro lugar procura-se extrair do sistema a ser construído as suas características mais importantes.
- A implementação dessas leva a subdividir o sistema inicial em outros menores, os quais, da mesma forma, são abstraídos segundo suas características principais.

- Observe que em cada nível de abstração, tem-se novos sistemas a serem implementados, cujos requisitos são impostos pelo nível superior.
- O detalhe porém é que a medida que desce para os níveis mais baixos, mais simples são os subsistemas, e portanto mais facilmente são implementáveis sem erros.

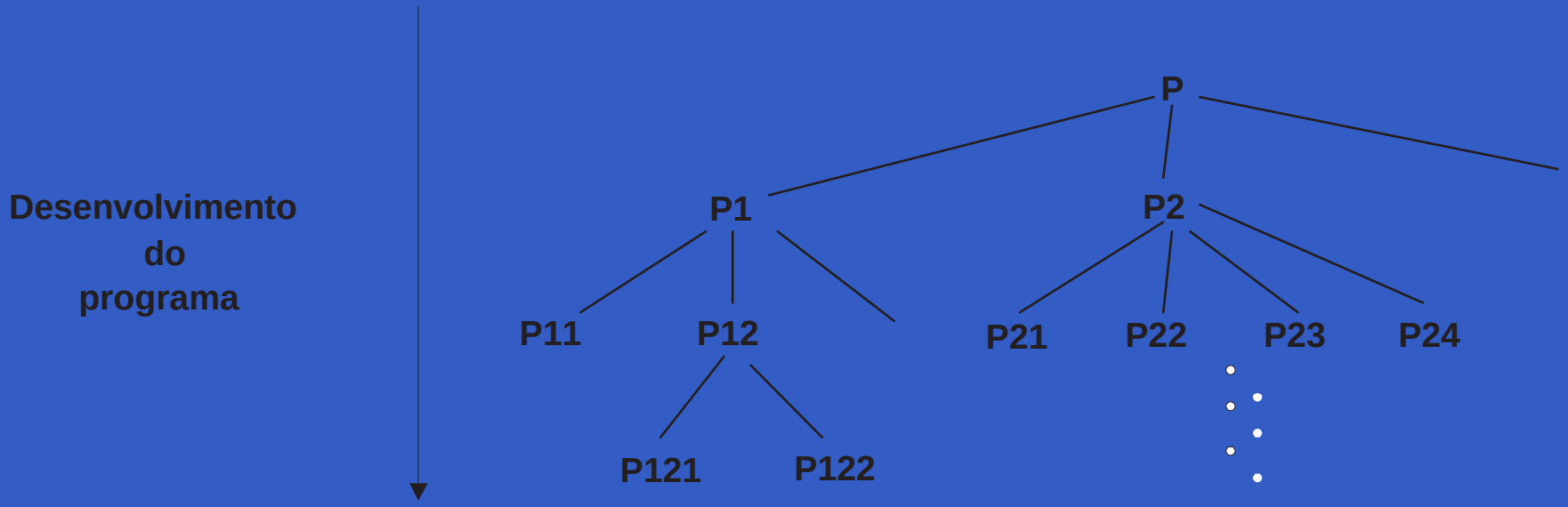


Figura 2: Projeto top-down de um programa.

- Deve-se notar que um projeto puramente TOP-DOWN só é possível quando o projetista tem conhecimento pleno de que cada subprojeto proposto pode ser implementado até o último nível.
- Na prática, porém, conforme uma idéia inicial é refinada, existe a possibilidade do surgimento de problemas técnicos que provocam a necessidade de revisão dos níveis superiores.

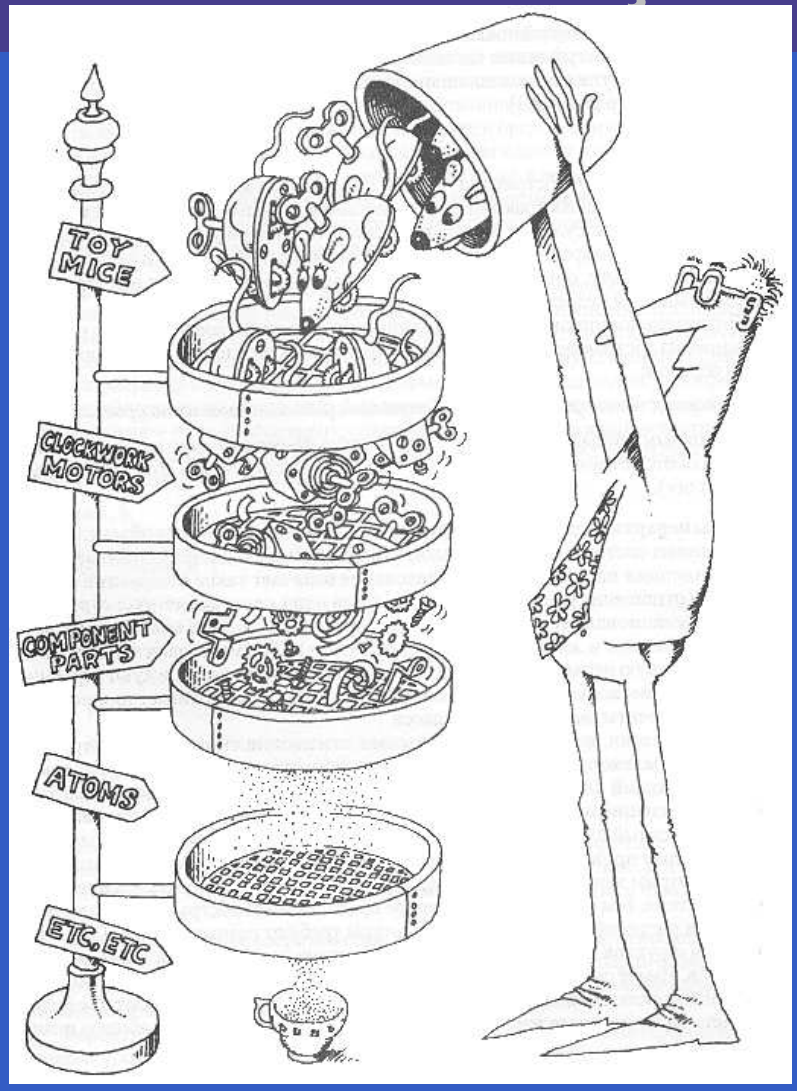
Hierarquia de níveis e projeto modular

- Durante a definição dos níveis, procura-se estruturá-los de uma forma hierárquica. Ou seja, cada nível é subordinado ao seu nível imediatamento superior e controla o funcionamento do imediatamente inferior.
- Pode-se dizer que os subsistemas situados em um nível i oferecem serviços ao nível $i-1$. Para um subsistema qualquer, apenas lhe é relevante o que existe disponível na camada inferior.

- Os serviços das camadas subsequentes não lhe são acessíveis ,ou seja, um subsistema apenas tem o controle direto sobre o nível imediatamente inferior.
- Dessa forma, as alterações que venham a acontecer em uma parte do sistema situadas em um determinado nível só repercutem em camadas inferiores.

- Os subsistemas situados em um mesmo nível não interagem entre si e portanto podem ser desenvolvidos de forma independente.
- Mais ainda, cada subsistema funciona como uma caixa preta: para quem o acessa somente interessa o que fazem e não como fazem. Cada subsistema é suficiente e completamente definido pelo seu comportamento entrada/saída.

Hierarquia de abstrações

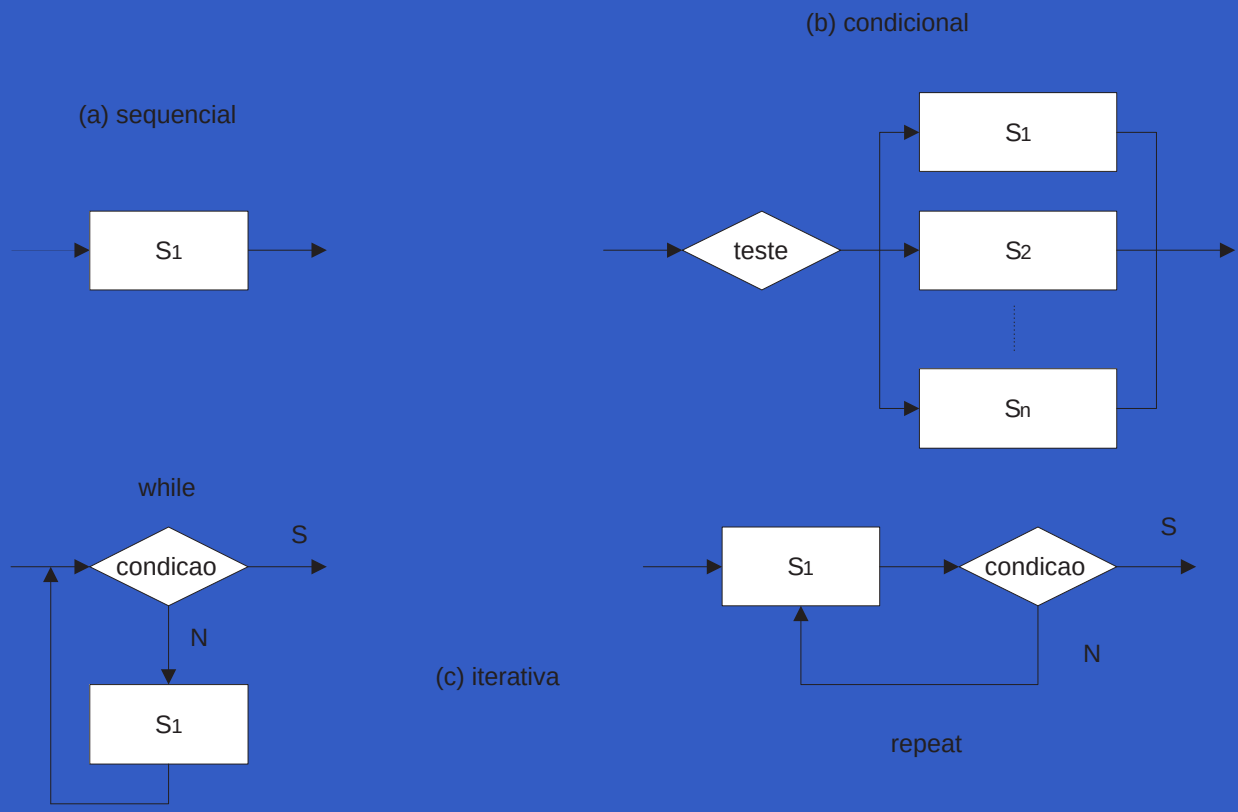


Teorema de estruturação

O desenvolvimento de código estruturado envolve a construção de unidades de programa envolvendo apenas as seguintes declarações:

1. seqüêncial;
2. condicional;
3. iterativa.

Declarações para programação estruturada



Em 1966 C. Boehm e J. Jacobini demonstraram que qualquer programa de computadores pode ser construído utilizando IF/THEN/ELSE, WHILE/DO e operações seqüenciais adequadas (atribuição, entrada e saída, etc.).

O que pode tornar um programa não estruturado ?

Vamos observar com atenção o seguinte trecho de programa em PASCAL onde são utilizados vários comandos GOTO:

```
4: if (a > as) then goto 1;
   if (b > bs) then goto 1;
   if (a1[a] < b1[b] then goto 2;
   goto 3;
2: c1[c] := a1[a];
   a := a + 1;
   goto 9
3: c1[c] := b1[b];
   b := b + 1;
9: c := c + 1;
   goto 4;
```

- Como pode ser observado o uso indiscriminado de comandos GOTO leva a um código de muito difícil compreensão. Uma das grandes vantagens de programação estruturada é a clareza e facilidade de execução de testes.
- De fato a utilização de programação estruturada pode ser traduzida como a utilização de projeto TOP-DOWN e refinamentos sucessivos para a implementação de módulos individuais.

- Trechos de programas estruturados podem em geral ser representados através de um bloco com uma única entrada e uma única saída como indicado na Figura 3(a) o que facilita a localização de erros durante os testes.
- Entretanto, um trecho não estruturado como indicado na Figura 3(b) pode conter várias entradas e várias saídas o que dificulta o isolamento dos erros.

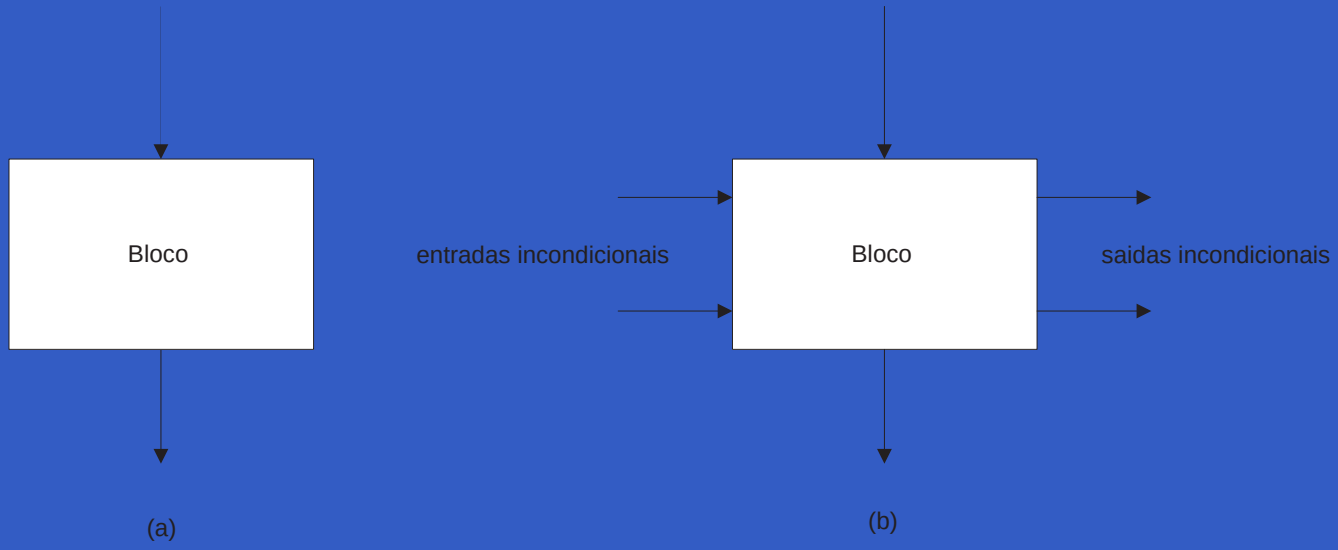
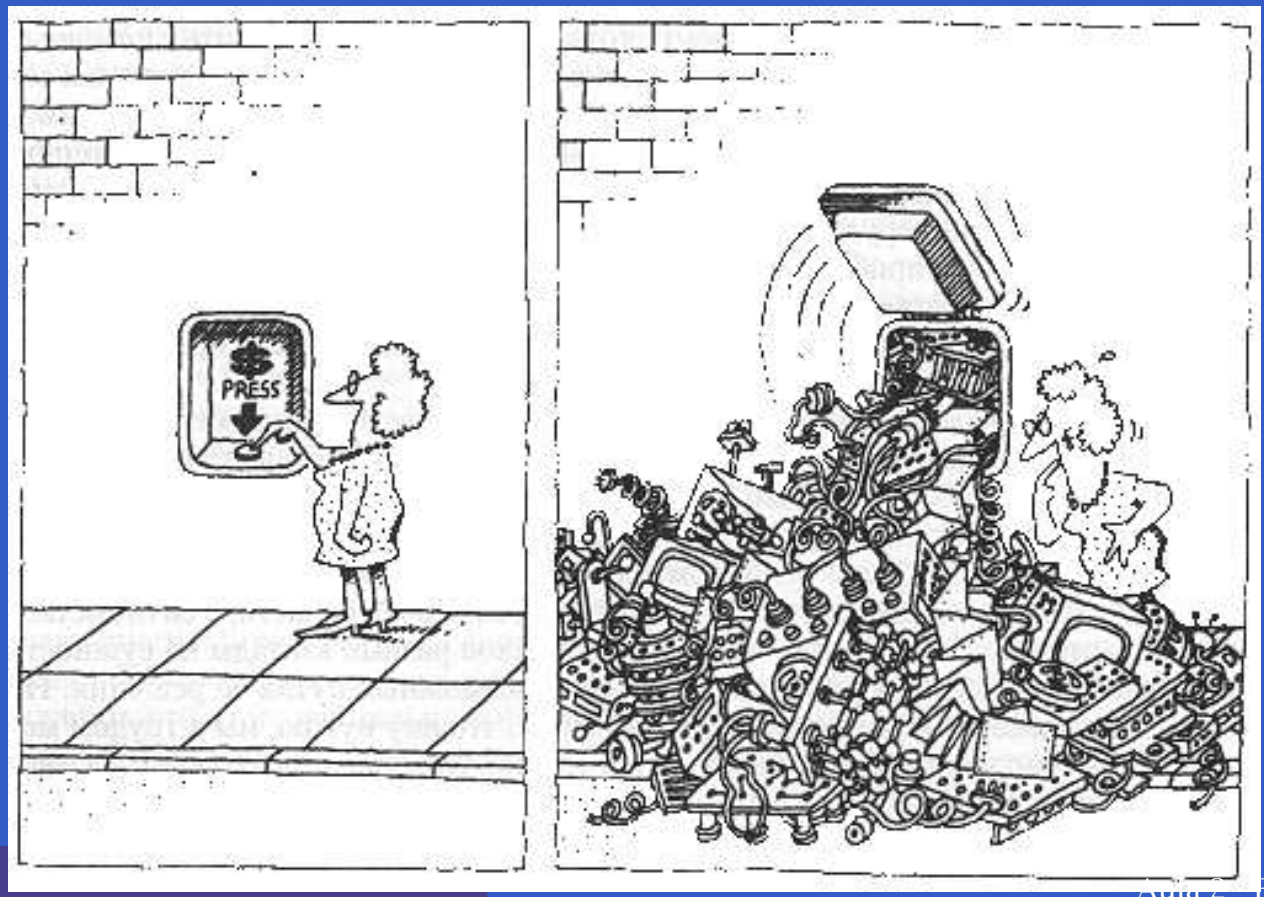


Figura 3: (a) código estruturado. (b) código não estruturado.

Ilusões

- A engenharia de software deve sempre criar a ilusão de simplicidade para os usuários.



Problemas a vista

- O desenvolvimento de software é uma atividade ainda essencialmente artesanal, e por isso facilmente sujeita a erros.
- A fase de manutenção é reconhecidamente a de maior custo de desenvolvimento, sendo a mudança de requisitos do usuário o principal componente.

O calcanhar de Aquiles ?

- A fase inicial do desenvolvimento é a especificação de requisitos.
- Usualmente, essa fase se utiliza de entrevistas com os potenciais usuários, ou seja, muito BLA!BLA!BLA.
- Posteriormente, documentos são gerados, muitas vezes se utilizando de linguagem natural.
- Usualmente, erros importantes relativos a não conformidade do projeto aos reais requisitos são detectados tardiamente, e correcões são realizadas com alto custo.

Solução ?

- É necessário a utilizaçao de técnicas com semântica precisa para descrever O QUE o sistema deve fazer.
- A solução: ESPECIFICAÇÕES FORMAIS, i.e., utilização de lógica matemática.
- Exemplos: Vienna Development Method (VDM), Z, etc.