Concepção de Software

- **Encontrar uma solução que satisfaça os requisitos de software**
- **Objectivos**
  - **Introduzir o processo de concepção de software**
  - **Descrever as diferentes fases no processo de concepção**
  - **Mostrar como as estratégias de concepção orientada para objectos e funcional são complementares**
  - **Discutir alguns atributos de qualidade de desenho**

Instituto Superior Politécnico de VISEU
Escola Superior de Tecnologia
Departamento de Informática

Tópicos Cobertos

- **O processo e métodos de concepção**
- **Estratégias de concepção incluindo concepção orientada para objectos e decomposição funcional**
- **Atributos de qualidade de concepção**

Introdução

- **Definição**: ***"... Processo de aplicação de várias técnicas e princípios com o propósito de definição de um dispositivo, um processo ou um sistema a um nível de detalhe suficiente que permita a sua realização física."***

- **Objectivo**: **Produzir um modelo de implementação do produto a ser construído.**

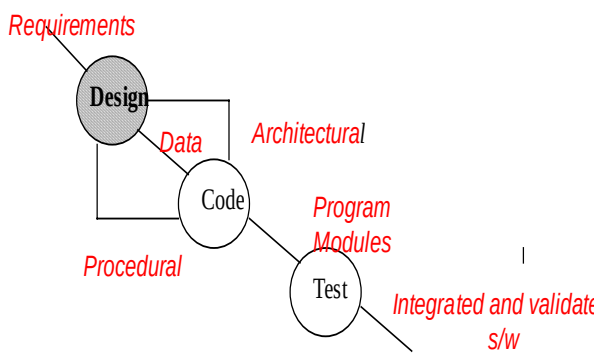
Introdução

- **Facto**: **A concepção de software é uma actividade intelectual e nunca deve ser evitada, dado que "...** ***Encontrar a concepção adequada é muito mais difícil do que implementar a concepção"***
- **Como se realiza a concepção?**
  - O processo de desenvolvimento do modelo combina **intuição** e **avaliação** baseada na **experiência**, num conjunto de **princípios** e **técnicas** e/ou **heurística** para guiar a forma de evolução do modelo, um conjunto de **critérios** para determinar a qualidade do modelo e **procedimentos** interactivos que levem à representação final da concepção.

Introdução

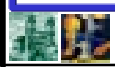
- **Foi dito que "*A mestria na concepção e desenvolvimento de software é semelhante à aprendizagem de como andar de bicicleta. Embora as leis da física descrevam completamente o processo, quase ninguém conseguirá aprender a andar de bicicleta através da mera leitura de um livro. É necessário equilíbrio...*".**
  - Desta forma, temos de aprender através da leitura de livros mas temos igualmente de praticar para ir melhorando. A moral da história (só no caso de ainda não se ter compreendido qual é...) é que a intuição e avaliação só aparece através da experiência... Há que arranjar uma bicicleta!

A Concepção no Processo de Eng. de Software



Quão Importante é a Concepção?

- **Se a qualidade é procurada, então a importância das actividades de concepção nunca serão por demais evidenciadas.**
  - É nesta fase que as decisões tomadas irão afectar o sucesso da implementação de software e a necessidade/facilidade com que o software será mantido.
- **A concepção serve como a base de todas as actividades que a seguem e sem concepção pode correr-se o risco de criar um sistema que não pode suportar nem as mais pequenas alterações os requisitos, pode revelar-se de igual forma de difícil teste e ainda mais difícil manutenção.**



Em que consiste uma boa concepção?

- **Então questão será *Em que consiste um bom software?* A resposta clássica será *"... Um bom programa é aquele que trabalha"* o que está errado.**

- **O engenheiro de software (ou programador de computadores) deve reconhecer a diferença entre um programa que trabalha e um bom programa. O bom software exibe três qualidades que o tornam bom:**
  - trabalha de acordo com os requisitos;
  - é de fácil manutenção;
  - está documentado.

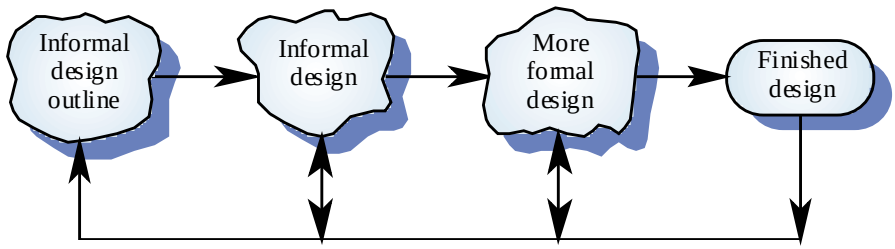
Fases da Concepção

- **Compreensão do problema**
  - **Olhar o problema segundo diferentes ângulos ou perspectivas, já que permitirão encontrar diversas formas para satisfazer os requisitos de concepção**
- **Identificar as características principais de uma ou mais soluções possíveis**
  - **Avaliar as soluções possíveis e escolher a mais apropriada, dependendo da experiência da pessoas que está a fazer a concepção e dos recursos disponíveis. Os factores para identificar soluções pode também ser a disponibilidade de componentes reutilizáveis e simplicidade de soluções. Normalmente são preferidas soluções familiares, ainda que não óptimas, já que se compreendem as vantagens e desvantagens.**
  - **Escolher a solução mais simples se os outros factores forem iguais.**
- **Descrever soluções para abstracções**
  - **Utilizar notações gráficas, formais ou descritivas para descrever os componentes da concepção**
- **Repetir o processo para cada abstracção identificada, até que a concepção seja expressa em termos primitivos. Na prática o processo termina quando a concepção de cada componente puder ser descrito numa única folha de papel.**

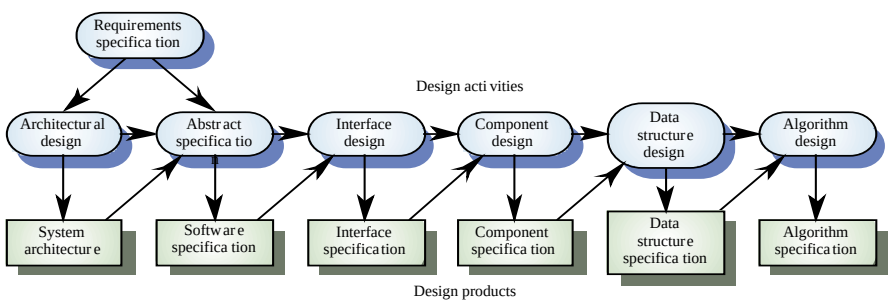
Processo de Concepção

- Cada concepção deve ser modelada como um grafo directo, feito de entidades com atributos que participam em relacionamentos
- O sistema deve ser descrito a vários e diferentes níveis de abstracção, sob a forma de modelos. Ao decompor-se a concepção do sistema, são descobertos erros e omissões havidos em etapas anteriores. Há então um feed-back que melhora a concepção geral.
- A concepção tem lugar em estágios que se sobrepõem. É artificial separá-los em fases distintas, mas alguma separação é normalmente necessária.
- Uma especificação de algum tipo é o output de cada actividade de concepção. Essa especificação pode ser abstracta, formal (produzida para clarificar os requisitos), ou pode ser uma especificação de como partes do sistema devem ser na realidade.
- À medida que o processo continua, são adicionados detalhes à especificação. O resultado final assume a forma de especificações precisas dos algoritmos e estruturas de dados a ser implementados.

Da Concepção Informal à Formal



Fases no Processo de Concepção

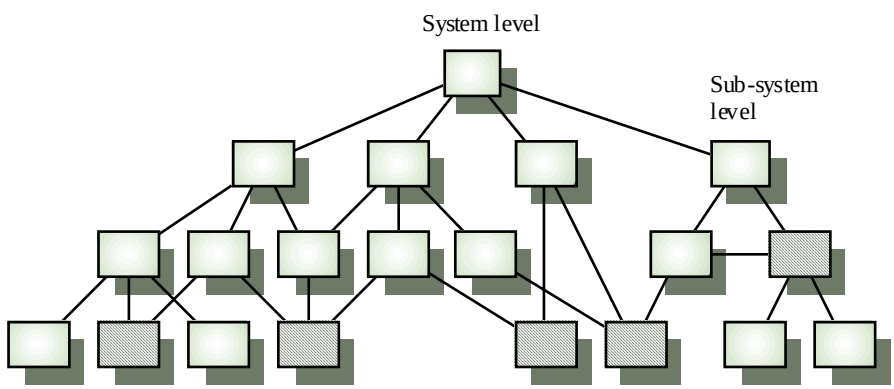


- Obs. na realidade as fases do processo de concepção decorrem em paralelo, não sequencialmente como sugere a figura. A forma sequencial favorece no entanto propósitos de sistematização.

Fases na Concepção

- *Concepção arquitectural* - Identificar e documentar os sub-sistemas e seus relacionamentos.
- *Especificação abstracta* - Especificar os subsistemas e serviços que proporcionam e constrangimentos sob os quais devem operar
- *Concepção da interface* - Descrever as interfaces dos sub-sistemas, de forma a que possam ser utilizado sem conhecer o seu funcionamento
- *Concepção dos componentes* - Decomposição dos sub-sistemas em componentes e alocar os serviços a cada um dos componentes e desenho das interfaces desses componentes
- *Concepção das estruturas de dados* - Conceber a estrutura de dados que suportem os dados do problema
- *Concepção do algoritmo* - Conceber os algoritmos para as funções que possam proporcionar os serviços supostos p/ o componente
- O processo é repetido para cada sub-sistema até que os componentes identificados possam ser mapeados directamente em componentes da linguagem de programação, como packages, procedimentos ou funções.

Estrutura Hierárquica de Concepção



Concepção Arquitectural

- O objectivo primário da concepção arquitectural é desenvolver um programa de estrutura modular e representar os relacionamentos de controlo entre os módulos.
- Exemplo:
  - **Considere-se a construção de uma nova casa. Espera-se que nos mostrem primeiro um esquema da casa, a planta dos andares, e outra informação que proporcionará uma visar arquitectural em vez da própria construção. O mesmo se aplica ao desenvolvimento de software (a construção é substituída pelos detalhes procedimentais e código). Uma boa arquitectura torna a construção fácil. Uma arquitectura deficiente, torna a construção quase impossível. Alterações à arquitectura são caras durante a construção ou mais tarde. O tempo necessário à correcção de um erro na arquitectura de software é, em média, menor do que o necessário à correcção de um erro nos requisitos, mas muito maior do que o necessário para corrigir um erro do código.**

Concepção Arquitectural

- ***Organização do Programa*** Uma panorâmica que descreve o sistema em termos gerais. Deve definir os módulos mais importantes num programa, e o que cada módulo faz deve ser bem definido, a interface de cada módulo deve ser bem definida, deve descrever que outros módulos o módulo pode chamar directamente ou que chama indirectamente e quais não chama. Deve especificar igualmente os dados que o módulo passa e recebe de outros módulos.
- ***Estratégia de alteração*** Alterações irão ocorrer. Desta forma a arquitectura deve ser flexível por forma a acomodar as alterações e deve descrever uma estratégia clara para suportar s alterações.
- ***Criar vs Comprar*** Se a arquitectura não utiliza componentes off-the-shelf, deve explicar bem como os componentes específicos são melhores do que os disponíveis em bibliotecas ou no mercado. Se o software pré-existente for utilizado, a arquitectura deve explicar como é que o software reutilizado poderá ser adaptado.

Concepção Arquitectural

- ***Estruturas principais de dados*** A arquitectura deve descrever os ficheiros principais, tabelas e estruturas de dados a serem utilizados.
- ***Algoritmos chave*** Se a arquitectura depende de algoritmos específicos, deve descrevê-los ou referenciá-los. Po exemplo, se um determinado algoritmo de ordenação for utilizado, deve descrever porque foi este e não outro qualquer.
- ***Desempenho*** arquitectura deve proporcionar estimativas e explicar porque é que o arquitecto acredita que os objectivos (tal como especificados no requisitos) serão atingidos. Caso alguns os objectivos estejam em risco, deverão de igual forma ser mencionados.

Concepção Arquitectural

- ***Funcionalidade genérica. Exemplo:***
- *A interface de utilizador* a arquitectura deve especificar estruturas de comando, formulários de entrada e menus
- Por exemplo, as *Entradas/Saídas* devem descrever o nível a que serão detectados os erros de I/O (campo, registo ou ficheiro)
- *Processamento de Erros* O processamento de erros é correctivo (recuperação de erros, notificação do utilizador) ou meramente de detecção (continuar como se nada se tivesse passado, sair ou notificar o utilizador)? A detecção de erros é activa (teste aos dados introduzidos pelo utilizador) ou passiva (uma combinação de dados introduzidos pelo utilizador produz um overflow)? Como é que o programa propaga os erros (ao detectar um erro, rejeita os dados que o causaram, entra num estado de processamento de erros ou continua e notifica o utilizador no final)? Nível de responsabilidade (é cada módulo responsável pela validação dos próprios dados ou há um grupo de módulos responsáveis pela validação dos dados do sistema)?

Concepção de Dados

- Qualquer método de concepção, como o de fluxo de dados, orientado a objectos, orientado a dados, dão grande ênfase a este assunto. Os conceitos de esconder informação e abstracção de dados são os fundamentos da abordagem à concepção de dados.

- O processo de concepção de dados pode ser resumido desta forma:
  - a actividade que resulta na selecção das representações lógicas de objectos de dados (estruturas de dados) que foram definidas durante as fases de definição e especificação de requisitos.

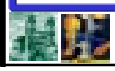
Instituto Superior Politécnico de VISEU
Escola Superior de Tecnologia
Departamento de Informática

Concepção de Dados

- Independentemente das técnicas de concepção a serem utilizadas, foram propostos um conjunto de princípios:
  - Avaliar estruturas de dados alternativas (fluxos de dados, objectos de dados, organização de dados, ...)
  - a concepção de uma estrutura de dados eficiente deve levar em conta todas as operações
  - um dicionário de dados deve ser gerado e utilizado na concepção dos dados e do programa
  - as decisões de concepção dos dados a baixo nível deve ser protelada para o final do processo de concepção
  - a representação de uma estrutura de dados deve ser conhecida só nos módulos que a utilizam directamente (ocultar informação e baixo acoplamento, separação das vistas lógicas e físicas)
  - uma biblioteca de estruturas de dados úteis e das operações que lhe podem ser aplicadas deve ser desenvolvida (torna os dados de tipo abstracto)
  - a concepção de software e linguagem de programação devem suportar a especificação e realização dos dados de tipo abstracto definidos

Concepção Procedimental

- A especificação procedimental define os detalhes algorítmicos. Num mundo ideal uma linguagem natural poderia ser utilizada pois que todas as pessoas ligadas ao desenvolvimento poderiam falar e compreender a linguagem natural mas, infelizmente, iremos cair nos problemas bem conhecidos de ambiguidade, etc. Devido a esses problemas da linguagem natural, deverão ser empregues modos mais restritivos para representação dos detalhes procedimentais.
- *Linguagens de descrição de concepção* As vantagens da descrição utilizando linguagens de alto nível como Ada, Pascal na concepção é exequível.Há, no entanto, uma série de desvantagens. A linguagem escolhida não pode ser estendida para incluir novos conceitos, construções primitivas podem forçar a detalhes desnecessários, além do que o pensamento por trás da concepção fica fortemente influenciado pela linguagem seleccionada, tornando difícil a re-implementação em outra linguagem.

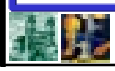


Concepção Procedimental

- ***Programação Estruturada***
  - A programação estruturada é um termo que foi criado no final os anos 60, para significar a programação sem a utilização de instruções goto, programação só com utilização de ciclos Enquanto e instruções Se, como estruturas de controlo e concepção utilizando com abordagem top-down. A adopção da programação estruturada foi importante porque constituiu o primeiro passo no afastar de uma abordagem indisciplinada ao desenvolvimento de software. Com a programação estruturada é menos provável que os programadores façam erros, já que força a um repensar efectivo do programa, já que a sua correcção poderá obrigar a sua completa re-escrita. Também permite que os programas sejam lidos sequencialmente e, desta forma, tornam-se muito mais fáceis de compreender e verificar.

Concepção Procedimental

- ***Um programa estruturado deve então ser escrito, utilizando apenas três componentes:***
  - sequências (normalmente escritas pela ordem em que as instruções serão executadas)
  - selecções (normalmente escritas como se...então...senão)
  - repetições (normalmente escritas como enquanto...fazer)
- Os argumentos contra o goto têm a ver com *clareza e e poder expressivo* pois que os gotos tornam difícil a compreensão da lógica do programa, também a *facilidade de leitura* é afectada pois que costumamos ler da esquerda para a direita e de cima para baixo. Ora andar a ler num cima abaixo, baixo acima sucessivo é contranatura.
- Neste ponto devemos lembrar-nos de que é tão fácil escrever um mau programa mesmo utilizando as três estruturas acima indicadas, como é possível criar um bom programa que empregue instruções goto.



Instituto Superior Politécnico de VISEU
Escola Superior de Tecnologia
Departamento de Informática

Concepção Procedimental

- ***Utilizar:***
  - **Notações gráficas de Concepção como flowcharts.**
  - **Tabelas de decisão**
  - **Linguagens de concepção de programas (PDL) também conhecidas como inglês estruturado ou pseudocódigo) - Linguagem que utiliza o vocabulário de uma linguagem natural e a sintaxe genérica de uma linguagem de programação. Tem uma sintaxe fixa de palavras-chave que proporcionam todas as estruturas construtivas, características de modularidade e de declaração de dados e uma sintaxe livre em linguagem natural para descrever as características de processamento.**

Concepção Procedimental

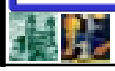
- ***Alguns traços gerais sobre a utilização de PDL de forma efectiva:***
  - **Utilizar instruções em inglês/português que descrevam precisamente operações específicas**
  - **Evitar elementos sintácticos de uma possível linguagem de programação**
  - **Escrever PDL ao nível de intenções. Descrever o significado da abordagem e não como será implementada numa determinada linguagem**
  - **Escrever PDL a um nível suficientemente baixo por forma a que a geração de código a partir dela seja quase automático (refinamento da PDL)**

Concepção Procedimental

- ***Alguns benefícios que podem ser esperados da utilização de PDL:***
  - **As revisões tornam-se mais fáceis dado que não é necessário examinar o código fonte**
  - **Suporta a ideia de refinamento interactivo**
  - **Torna as alterações mais fáceis**
  - **As instruções PDL tornam-se os comentários do código**

Concepção Top-Down

- Em princípio, a concepção top-down traduz-se em começar pelos componentes mais acima na hierarquia e trabalhar na hierarquia, a níveis sucessivamente mais baixos, nível a nível.
- As ligações cruzadas no grafo surgem a níveis mais baixos da árvore de concepção, à medida que são identificadas possibilidades de reutilização de componentes.
- Na prática, a concepção de grandes sistemas nunca é verdadeiramente top-down. Alguns ramos (normalmente aqueles que são mais problemáticos) são desenhados antes de outros. Os melhores conhecidos são decompostos numa fase final. As pessoas encarregadas da concepção reutilizam a experiência (e algumas vezes componentes) durante o processo de concepção.
- A concepção Top-Down baseia-se na decomposição esquemática e é uma abordagem válida quando os componentes estão fortemente acoplados. Contudo, quando uma abordagem orientada a objectos é adoptada, e muitos objectos, já existentes, estão disponíveis para utilização, estes serão utilizados como um referencial de concepção, sendo esta criada a partir deles, não havendo o conceito de um único topo na hierarquia.



Métodos de Concepção

- Os métodos estruturados são conjuntos de notações para expressar a concepção de software e guias para a criação da concepção
- Métodos típicos são por exemplo, a Concepção Estrutural (Yourdon), JSD (método Jackson) e Análise Estruturada de Sistemas (Gane and Sarson)
- Podem ser aplicados com sucesso porque suportam notação standard e asseguram que a concepção segue um formato standard
  - Métodos estruturados podem ser suportados através de ferramentas CASE.

Componentes dos Métodos

- Muitos métodos suportam visões comparáveis de um sistema
- Uma visão de fluxo de dados (diagramas de fluxo de dados) mostram as transformações de dados que têm lugar no processamento
- Uma visão entidade-relacionamento descreve as estruturas lógicas de dados que serão utilizadas
- Uma visão estrutural mostra os componentes do sistema e as suas interacções
- Se o método é orientado a objectos, incluirá um modelo de herança do sistema, um modelo de composição de objectos e o modo como os objectos são utilizados por outros.

Deficiências dos Métodos

- São linhas directoras e não métodos no sentido matemático. Pessoas diferentes a conceber o mesmo sistema, criarão concepções do sistema algo diferentes
- Não ajudam muito na fase criativa inicial da concepção. Em vez disso, ajudam na estruturação e documentação das ideias de concepção.
- Não há um método melhor do que outro. O sucesso de cada um depende da sua vocação a um determinado domínio de aplicação.

Descrição de Concepção

A concepção do software é um modelo de um sistema do mundo real que tem muitas entidades e relacionamentos participantes. Há um conjunto de documentos de concepção que servem para comunicação entre os elementos encarregados da concepção, descrevendo a concepção, proporcionando informação sobre as intenções da concepção a programadores e posterior manutenção.

Há **<u>três tipos</u>** principais de **<u>notações</u>** utilizadas na documentação da concepção:

- *Notações gráficas* - Utilizadas para mostrar as relações entre componentes, relacionando a concepção com o sistema do mundo real que está a ser modelado.
- *Linguagens de descrição de programas (PDL)* - Baseadas em linguagens de programação, mas com mais flexibilidade para representar conceitos abstractos. Estas linguagens permitem texto explicativo que possibilitam expressar a intenção da concepção em vez de detalhes de como a concepção deverá ser implementada.
- *Texto informal* - Descrição em linguagem natural. Informação quanto a razões de concepção ou considerações não funcionais, expressas em linguagem natural (texto).
- Todas estas notações podem ser utilizadas na concepção de grandes sistemas.

Estratégias de concepção

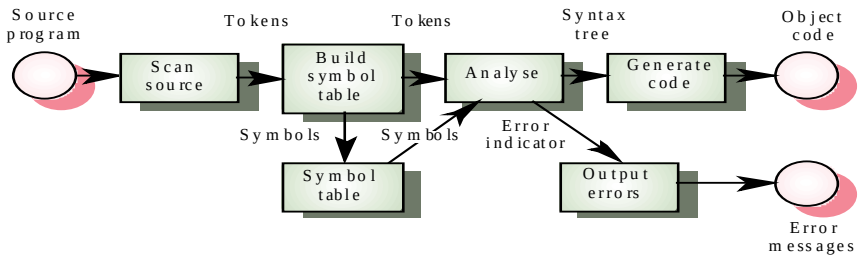
- **Concepção funcional**
  - O sistema é concebido de um ponto de vista funcional. O estado do sistema está centralizado e partilhado entre as funções que operam sobre o estado. A estrutura dos dados é utilizada para determinar a estrutura funcional utilizada para processar esses dados.
- **Concepção orientada a objectos**
  - O sistema é visto como uma colecção de objectos interactuantes em vez de funções. O estado do sistema é descentralizado e cada objecto gere a sua própria informação de estado. Os objectos têm um conjunto de atributos que definem o seu estado e operações que actuam sobre esses atributos. Os objectos podem ser instâncias de uma classe de objectos. Conceptualmente, os objectos comunicam por troca de mensagens; na prática, a maioria da comunicação entre objectos é conseguida através da chamada por um objecto de um procedimento associado por outro objecto.
- **Esta concepção é baseada na ideia de "esconder informação".**

Visão funcional de um compilador

Para ilustrar a diferença entre a aproximação funcional e orientada a objectos, consideremos a estrutura de um compilador.

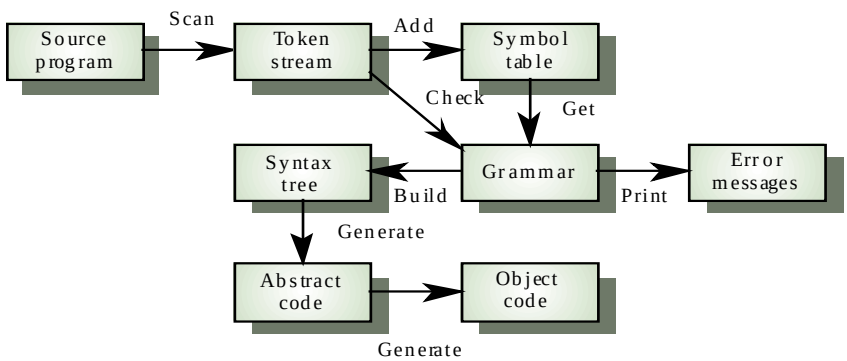
O compilador pode ser visto como um conjunto de transformações funcionais com informação a passar de um processo para outro, como pode ser visto na figura.

Obs. Os componentes principais são identificados como acções (pesquisa, criação, análise, geração).



Visão orientada a objectos de um compilador

Uma visão alternativa, orientada a objectos do mesmo compilador, é centrada nos objectos manipulados pelo compilador, com as operações associadas a cada objecto, mostrada na figura abaixo. Neste caso, os componentes principais são entidades como "token stream", "tabelas de símbolos", "árvore de sintaxe", etc.

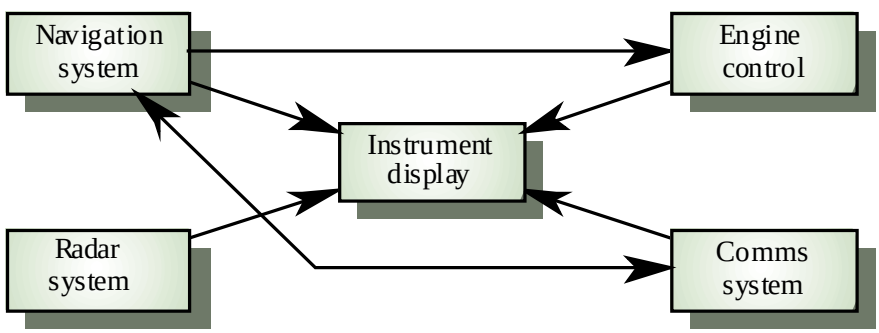


Estratégia de Concepção Mista

- Embora seja por vezes sugerido que uma determinada abordagem à concepção é superior, na prática, as abordagens de concepção orientadas a objectos e orientadas a funções são complementares
- Os bons engenheiros de software devem seleccionar a abordagem mais apropriada para o sistema a ser concebido

Sub-sistemas de um avião

Para ilustrar a estratégia de concepção mista, consideremos os sistemas de software que podem fazer parte de um avião civil moderno. A nossa visão a alto nível considera o sistema como um conjunto de sub-sistemas ou objectos. A este nível de concepção abstracta, uma concepção orientada a objectos é completamente natural (ver figura). Cada objecto é um sub-sistema.



Funções do Sistema (ao nível de sub-sistema)

Quando o sistema é examinado com maior detalhe, a sua descrição natural é um conjunto de funções de interacção, em vez de objectos. Algumas dessas funções podem ser:

- Mostrar rota (sus-sistema de radar)
- Compensar a velocidade do vento (sub-sistema de navegação)
- Reduzir potência (sub-sistema de motores)
- Indicar emergência (sub-sistema de instrumentos)
- Sintonização de frequência (sub-sistema de comunicações)
- ....

Objectos a baixo nível

À medida que a concepção do sistema é sucessivamente decomposta, uma visão orientada a objectos pode tornar-se novamente a forma natural de ver o sistema. Nessa fase, os objectos manipulados podem ser:

- O estado do motor
- A posição do avião
- O altímetro
- O rádio-farol
- ....

Qualidade da Concepção

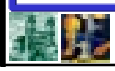
- Não há um acordo geral quanto à noção de uma boa concepção. Além do critério óbvio de que a concepção deve implementar correctamente a especificação.
- A qualidade de concepção é um conceito indefinível, ilusório. A qualidade depende de prioridades organizacionais específicas. Uma "boa" concepção pode ser:
  - **a mais eficiente,**
  - **a mais barata,**
  - **a de mais fácil manutenção,**
  - **a mais fiável, etc..**
- Os atributos aqui discutidos, estão centrados na facilidade de manutenção relativa à concepção.
- As características de qualidade são aplicáveis de forma idêntica a concepções orientadas a funções ou a objectos.

Linhas Gerais Quanto à Qualidade da Concepção

- A concepção deve mostrar uma organização hierárquica que faça um uso inteligente do controlo entre componentes de software
- a concepção deve ser modular, i.e., dividida logicamente em componentes que executam determinadas funções e subfunções
- a concepção deve conter representações distintas e separáveis dos dados e procedimentos
- a concepção deve conduzir a módulos que exibam características funcionais independentes
- a concepção deve conduzir a interfaces que reduzam a complexidade das conexões entre o módulos e com o ambiente externo
- a concepção deve ser conseguida através de uma método conduzido pela informação obtida durante a fase de análise de requisitos de software

Conceitos Relativos à Concepção

- *"O princípio da sabedoria de um engenheiro de software ou de um programador é saber reconhecer a diferença entre o fazer um programa que funcione e um que seja realmente bom"* Michael Jackson.
  - **Os conceitos fundamentais de concepção evoluíram no decorrer das três últimas décadas e embora o grau de interesse de cada conceito tenha conhecido altos e baixos ao longo dos anos, cada um deles proporciona um referencial tendente à criação do 'bom' software. Estes conceitos serão explicados seguidamente.**



Instituto Superior Politécnico de VISEU
Escola Superior de Tecnologia
Departamento de Informática

Conceitos Relativos à Concepção

- ***a) Abstracção:***
  - **a abstracção é uma ferramenta intelectual que permite lidar com conceitos desligando-os de instâncias particulares. O processo de desenvolvimento de software no seu todo, é uma sucessão de níveis de abstracção. Ao mais alto nível de abstracção, uma solução genérica é encontrada , sendo descrita na linguagem própria do ambiente. A um nível mais refinado, o nível da abstracção é reduzido até se atingir o nível mais baixo de abstracção, atingido quando a solução é traduzida de uma forma que o código fonte pode ser gerado. O que temos de recordar é que cada nível de abstracção actua como uma definição para o próximo nível (num prédio de apartamento, o chão de um é o tecto do outro). Há três mecanismos de abstracção principais: *abstracção funcional, de dados e de controlo*.**

Conceitos Relativos à Concepção

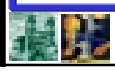
- **Uma <u>abstracção funcional</u> ou (procedimental) é uma sequência nomeada que tem uma função específica e limitada.**
- **Uma <u>abstracção de dados</u> é uma colecção nomeada de dados que descrevem um objecto de dados. Um exemplo de abstracção de dados será um pagamento através de cheque. Este objecto de dados é realmente um conjunto de pedaços de informação (nomes de quem paga e a quem paga, valor total pago, taxas aplicáveis e valores, data de pagamento, data-valor, etc.). Podemos referir-no a todos estes dados, indicando apenas o nome da abstracção de dados. Uma vez definida a abstracção de dados, podemos definir um conjunto de operações (procedimentos) que lhe podem ser aplicados. Também poderemos utilizar a abstracção para descrever outros objectos de dados, sem especificar os seus detalhes.**
- **A <u>abstracção de controlo</u> implica que o mecanismo de controlo do programa, sem especificar detalhes internos. Por exemplo, as instruções *se, enquanto, repetir* são abstracções de implementações em código máquina que envolvem saltos condicionais.**

Conceitos Relativos à Concepção

- ***b) Refinamento:***
  - **O refinamento passo-a-passo é uma estratégia de concepção antiga. A arquitectura do programa é desenvolvida através de refinamento sucessivo de níveis de detalhe procedimental, até atingir instruções de linguagem de programação.**
- ***c) Modularidade:***
  - **O conceito de modularidade em software foi introduzido há quase quatro décadas. O software é dividido em componentes separáveis, endereçáveis e nomeadas chamadas módulos. Este termo é muitas vezes utilizado de forma inconsistente. Uma possível definição será: "*Um conjunto de abstracções em que cada uma trata de um aspecto local do problema a ser solucionado*" Os sistemas modulares consistem em unidades manejáveis bem definidas, com igualmente bem definidas interfaces.**

Conceitos Relativos à Concepção

- **A modularização é um conceito genérico que permite que na concepção:**
  - **o sistema seja decomposto em unidades funcionais,**
  - **impor uma ordenação hierárquica na utilização funcional,**
  - **implementar abstracção de dados e**
  - **desenvolver independentemente sub-sistemas úteis.**
- **As propriedades desejáveis de um sistema modular**
  - **cada abstracção seja um sub-sistema bem definido,**
  - **cada função numa abstracção tenha um propósitos simples e bem-definido,**
  - **cada função não maneje mais do que uma estrutura de dados principal.**
- **Uma característica importante não mostrada é que:**
  - **A Complexidade (problema1+problema2) > Complexidade(problema1) + Complexidade(problema2)**
    - **Argumento do "*dividir e conquistar*". É mais fácil resolver um problema complexo, quando este está dividido em pedaços manejáveis.**

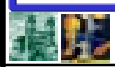


Conceitos Relativos à Concepção

- **É no entanto incorrecto assumir que se um problema for dividido indefinidamente o esforço da sua resolução se tornará sucessivamente menor. À medida que o número de módulos aumenta, crescerá o esforço associado às interfaces. Desta forma há que ser cuidadoso por forma a não haver um sub ou sobre modularização. Qual deve ser então tamanho de um módulo?**
- **Um módulo, quanto mais pequeno é mais fácil de compreender. Mas se os módulos são pequenos, seremos submergidos pelo seu número. Uma sugestão comum é que o seu tamanho não deverá ultrapassar uma página de texto, ou no máximo duas. Esta sugestão tem a ver com o problema psicológico da compreensão do código que passa de uma para outra página.Outra sugestão proposta é 7, baseada na evidência psicológica que o cérebro humano pode armazenar na chamada "memória curta" entre 5 e 9 coisas.**

Conceitos Relativos à Concepção

- **O argumento em detrimento de módulos pequenos tem a ver com o aumento de sobrecarga de comunicação entre componentes.**
  - **De forma clara, a questão é se pretendemos código compreensível ou de mais rápida execução. Uma estratégia sugere que concepção, codificação e teste seja feita por pequenos módulos, sendo depois combinados de forma apropriada.**
  - **Importante é salientar que um sistema deve ser concebido de forma modular, mesmo se a implementação final for monolítica (por exemplo, software em tempo real eu obrigue a mínimo de sobrecarga introduzida pelos sub-programas) . Apesar do que programa não parecer modular, a filosofia dos módulos manteve-se e o programa proporcionará os benefícios de um sistema modular.**

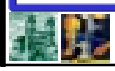


Conceitos Relativos à Concepção

- ***d) Ocultar informação***
  - **O conceito de modularidade conduz ao princípio de ocultar informação. Cada componente (módulo) esconde os detalhes internos das suas actividades de processamento e os componentes comunicam somente através de interfaces bem-definidas. A utilização do ocultar informação como um critério de concepção de sistemas modulares proporciona grandes benefícios quando modificações são necessárias durante a fase de teste e mais tarde, durante a fase de manutenção. Dado que a maioria dos dados e procedimentos estão escondidos de outras partes do software, erros não considerados, eventualmente introduzidos durante as modificações, serão menos susceptíveis de propagação a outras partes do software.**

Conceitos Relativos à Concepção

- Então, o que deve ocultar-se?
  - **Áreas onde poderão ocorrer alterações** - A acomodação de alterações, será um dos aspectos que trará maiores desafios à boa concepção o programa. O objectivo é isolar as áreas mais instáveis, por forma a que os efeitos de uma alteração sejam limitados a um módulo.
  - **Eis algumas áreas onde podem ocorrer as alterações:**
    - dependências do hardware (ecrã, impressoras, discos, facilidades sonoras, etc.)
    - dados complexos
    - lógica complexa



Conceitos Relativos à Concepção

- ***e) estrutura***
  - Permite a decomposição de um grande sistema e unidades mais pequenas e manejáveis, com relacionamentos bem definidos com outras unidades. Poderemos falar em **estrutura do programa** e **estrutura de dados** (definidas em conjunto como a **arquitectura de software**).
  - A *arquitectura de software* é derivada através do processo de divisão que relaciona elementos da solução de software às partes do problema real, definidas implicitamente durante a análise de requisitos. A evolução do programa e estruturas de dados tem início com a definição do problema. A solução ocorre quando cada parte do problema é resolvida por um ou mais elementos de software. Obviamente um problema pode ser satisfeito por muitas estruturas candidatas diferentes.

Coesão

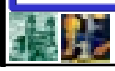
- É uma medida de quão bem o componente "se encaixa"
- O componente deve implementar uma única entidade ou função, devendo todas as partes do componente contribuir para essa implementação
- A coesão é um atributo desejável da concepção do componente, dado que, sempre que uma alteração deva ser efectuada, fica restringida a um único componente coerente
- Vários níveis de coesão foram identificados

Níveis de coesão

- **Coesão coincidente (fraca)**
  - **Partes do componente foram simplesmente empacotados em conjunto, não são efectivamente relacionados** - também chamada "sem coesão" ou coesão caótica"
- **Associação lógica (fraca)**
  - **Componentes que executam funções semelhantes são agrupados num único componente (por exemplo, entradas, tratamento de erros, etc.). Muitas vezes o fluxo de controlo é a única coisa que liga as operações como um conjunto (controlado através de uma flag").**
- **Coesão temporal (fraca)**
  - **Componentes que são activadas ao mesmo tempo são agrupados (por exemplo, start up ou shut down). O componente executa um conjunto de funções relacionadas no tempo.**
- **Coesão Procedimental (fraca)**
  - **Os elementos num componente executam uma única sequência de controlo**

Níveis de coesão

- **Coesão comunicacional (média)**
  - Todos os elementos de um componente operam sobre as mesmas entradas ou produzem as mesmas saídas.
- **Coesão sequencial (média)**
  - O módulo aceita dados de outro módulo, modifica-os e passa-os a outro módulo. A saída de uma parte do componentes é a entrada de outra parte.
- **Coesão funcional (forte)**
  - Todas as operações no módulo contribuem para o desempenho de uma única tarefa bem definida. Cada parte do componente é necessária à execução de uma única função.
- **Coesão de Objecto (forte)**
  - Cada operação proporciona funcionalidade que permite modificar, inspeccionar os atributos do objecto ou utilizada como base para proporcionar um serviço.

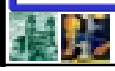


Coesão

- **A coesão é uma característica desejável porque significa que um componente representa uma única parte as solução do problema. A forma sugerida de identificação da coesão é escrever uma declaração em português do que o módulo faz e examinar a declaração. Deve ter um só verbo e um único objecto que é actuado pelo verbo.**
  - Se faz duas coisas, significa que não há coesão funcional, mas há coesão sequencial. Se por outro lado, aparecem vários verbos, isso indica coesão procedimental. Se termos como inicializar são mencionados, podemos estar em presença de coesão temporal. Uma análise deste tipo, não nos mostra como melhorar a estrutura, apenas mostra quão pode poderá ser a nossa estrutura.

A coesão como um atributo de concepção

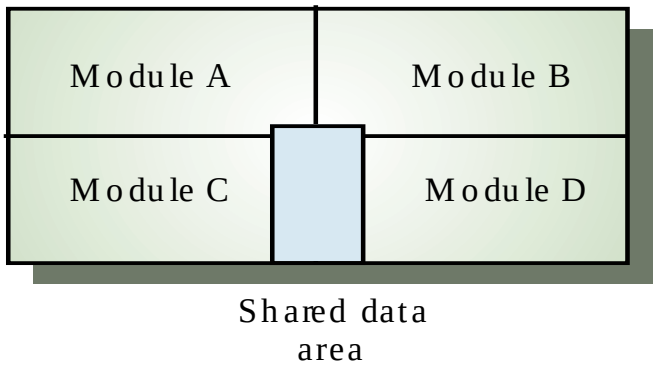
- **Não bem definida. É muitas vezes difícil classificar a coesão.**
- **A herança de atributos de super-classes, enfraquece a coesão, dado que já não poderemos considerar essa classe de objecto como uma unidade auto-contida.**
- **Para compreender um componente, as super-classes além da classe do componente têm de ser examinadas para que a funcionalidade total do objecto seja compreendida**



Acoplamento

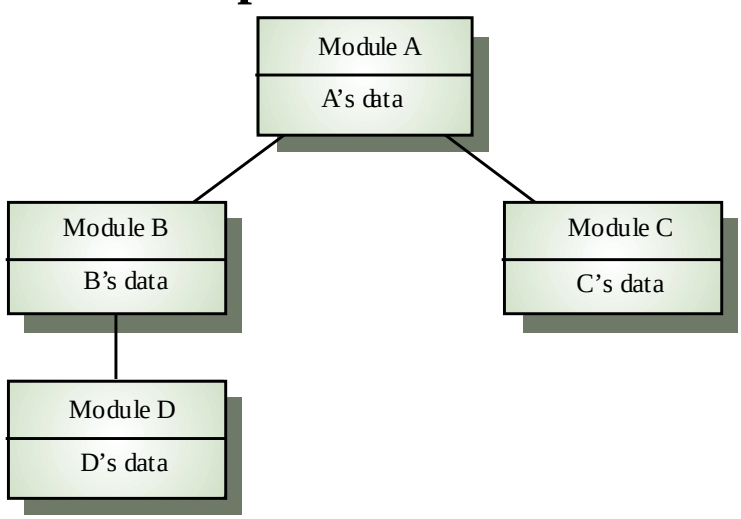
- O acoplamento está relacionado com a coesão.
- É uma indicação da medida da força das interconexões entre os componentes do sistema.
- Sistemas com elevado acoplamento têm interconexões fortes, com unidades de programa dependendo umas das outras.
- Sistemas com fraco acoplamento são feitas de componentes que são independentes ou quase independentes. Isto significa que alterações num componente não deverão afectar outros componentes.
- Variáveis partilhadas ou troca de informação de controlo, conduzem a acoplamento firme
- Acoplamento fraco pode ser conseguido através da descentralização do estado (como nos objectos) e comunicação entre componentes via parâmetros ou passagem de mensagens
- O acoplamento fraco é conseguido assegurando que os detalhes de representação dos dados é deixado dentro do componente. A sua interface com outros componentes é feita através de uma lista de parâmetros. Informação acessível globalmente deve ser evitada sempre que possível.

Acoplamento firme



- Obs. Outra forma de acoplamento firme surge quando valores substituem prematuramente os nomes. Por exemplo, se a taxa do Iva for colocada dentro do programa e não como um parâmetro lido em run-time, alteração da taxa implicará a modificação do programa.

Acoplamento Fraco



Acoplamento e herança

- **Sistemas orientados a objectos são de fraco acoplamento dado que não há nenhum estado partilhado, os objectos comunicam através de passagem de mensagens e qualquer objecto pode ser substituído por outro objecto com a mesma interface.**
- **Contudo, uma classe de objecto está acoplado à sua super-classe. Alterações efectuadas aos atributos ou operações numa super-classe são propagadas a todas as sub-classes. Essas alterações devem ser objecto de um controlo cuidadoso.**

Acoplamento

- **O objectivo é criar componentes com alta integridade interna (alta coesão) e relações pequenas, directas e flexíveis com outros componentes (acoplamento fraco).**
  - Exemplo: o componente *final()* que imprime o relatório final e regista o cálculo final em disco; não é aceitável: o *e*, implica duas funções separadas.
  - O Final() é um nome inadequado. O nome supõe descrever a função, não o quando ou porquê do módulo ser utilizado. Deverão ser criados dois componentes: ImprimeRelatório e GravaCálculos.

Compreensibilidade

A compreensibilidade da concepção é importante, já que, alguém que pretenda alterá-la, deve primeiro compreendê-la

- **Está relacionada com várias características do componente:**
  - *Coesão e acoplamento* - O componente pode ser compreendido em si próprio, sem interferência de outros componentes?
  - *Nomes* - São utilizados nomes significativos? Nomes com significado são aqueles que reflectem os nomes das entidades do mundo real modeladas pelo componente.
  - *Documentação* - A concepção está bem documentada? A documentação relativa ao componente torna claro o mapeamento entre as entidades do mundo real e o componente.
  - *Complexidade* - São utilizados algoritmos complexos? Componentes muito complexos são de difícil compreensão, devendo tentar conceber-se componentes tão simples quanto possível.
- **Informalmente, complexidade alta significa muitos relacionamentos entre diferentes partes da concepção. Dessa forma torna-se de difícil compreensão.**

Adaptabilidade

A adaptabilidade de uma concepção é uma estimativa geral de quão fácil é alterar a concepção.

Uma concepção é adaptável, se:

- Os seus componentes tiverem um acoplamento fraco
- Se estiverem bem documentados e se a documentação estiver actualizada (consistente com a implementação)
- Se houver uma correspondência óbvia entre os vários níveis de concepção (visibilidade da concepção)
- Se cada componente for auto-contido (coesão forte)

- **Para adaptar uma concepção, deve ser possível seguir as ligações entre os diversos componentes da concepção, de forma a que as consequências da alteração possam ser analisadas**
- **Para adaptabilidade óptima, um componente deve ser auto-contido, sem dependências com outros componentes externos. Trata-se contudo de um propósito contrário à possível reutilização.**

Adaptabilidade e herança

- **A herança melhora muito a adaptabilidade. Os componentes podem ser adaptados sem alterações, por derivação de uma sub-classe e modificação dessa sub-classe.**
- **Contudo, à medida que a profundidade da hierarquia da herança aumenta, torna-se sucessivamente mais complexa. Deve ser revista e reestruturada periodicamente.**

Pontos Chave

- **A concepção é um processo criativo**
- **As actividades de concepção incluem o desenho arquitectural, especificação do sistema, concepção dos componentes, concepção da estrutura de dados e concepção de algoritmos**
- **A decomposição funcional considera o sistema como um conjunto de unidades funcionais**
- **A decomposição orientada a objectos considera o sistema como um conjunto de objectos**
- **As decisões quanto ao paralelismo devem ser normalmente decisões de concepção detalhada**

Desenho Arquitectural

- **Estabelecer a estrutura genérica dum sistema de software, identificando os seus sub-sistemas e e conjunto de serviços relacionados e estabelecer o referencial para o controlo dos sub-sistemas e comunicação.**
- **Pode estabelecer-se um paralelo entre o papel desempenhado pelo arquitecto do sistema de software e o arquitecto dum projecto de construção, já que são a interface com o cliente, uma má concepção arquitectural, não pode ser salva por uma boa construção.**

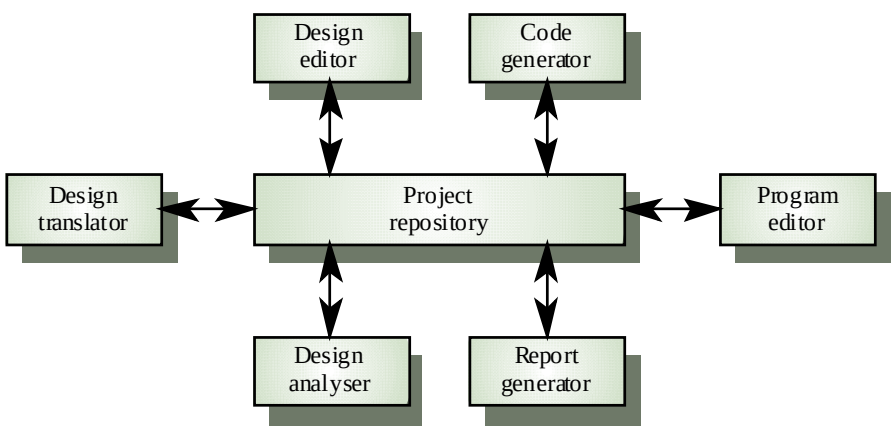
Estruturação do Sistema

- **Lida com a decomposição do sistema em sub-sistemas em interacção**
- **A concepção arquitectural é expressa normalmente como um diagrama de blocos, mostrando uma visão geral da estrutura do sistema**
- **Modelos mais específicos mostrando como os sistemas partilham dados, como estão distribuídos e como é a interface com cada um dos outros podem também ser desenvolvidos.**
- **A estruturação do sistema pode ser efectuada com base em vários modelos: repositório, Cliente-servidor e máquina abstracta.**

Modelo Repositório

- **Os sub-sistemas devem trocar dados. Isto pode ser efectuado de duas formas:**
  - Os dados partilhados são colocados numa base de dados ou repositório central, que pode ser acedida por todos os sub-sistemas
  - Cada sub-sistema mantém a sua própria base de dados e passa dados explicitamente a outro sub-sistema
- **Quando grandes quantidades de dados devem ser partilhados, o modelo repositório é o mais utilizado.**

Arquitectura de ferramenta CASE



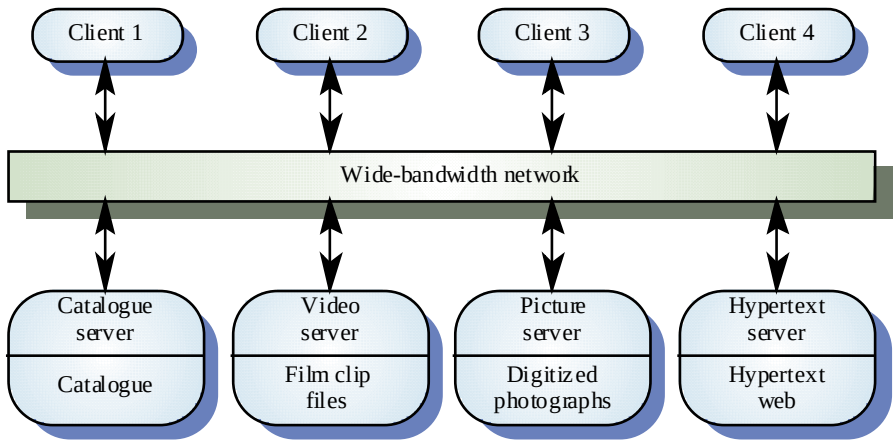
Características do Modelo Repositório

- **Vantagens:**
  - Modo eficiente de partilhar grandes quantidades de dados
  - Os sub-sistemas não precisam de saber como os dados são produzidos
  - Gestão centralizada (backup, recuperação, segurança, etc.)
  - Fácil de integrar outro sub-sistemas, desde que compatíveis com o modelo de dados
- **Desvantagens**
  - Os sub-sistemas devem concordar no modelo de dados para repositório. É inevitavelmente um compromisso que afectará o desempenho.
  - A evolução dos dados é difícil e cara
  - Força à utilização da mesma política de gestão para todos os sub-sistemas
  - Difícil de distribuir o repositório por várias máquinas, podendo levar a inconsistências e redundância.

Modelo Cliente-Servidor

- **Modelo de sistema distribuído que mostra como os dados e o processamento são distribuídos pelos vários componentes**
- **Principais componentes deste modelo:**
  - Um conjunto de servidores stand-alone, que proporcionam serviços específicos como impressão, gestão de dados, etc.
  - Um conjunto de clientes que chamam esses serviços. Normalmente são sub-sistemas, mas podem ser de igual forma várias instâncias de um programa cliente em execução concorrente.
  - Uma rede que permite permite aos clientes aceder a esses serviços.

Biblioteca de Filmes e Quadros



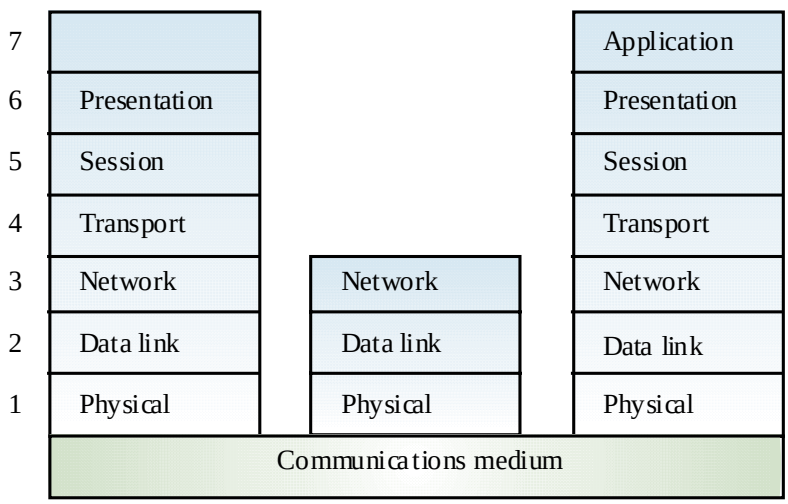
Características do Modelo Cliente-Servidor

- **Vantagens:**
  - Distribuição dos dados simples
  - Faz uso efectivo do sistemas de rede. Pode necessitar de hardware mais barato
  - Fácil de adicionar novos servidores ou actualizar servidores existentes
- **Desvantagens:**
  - Não há um modelo partilhado. Cada sub-sistema pode utilizar diferente organização de dados. O intercâmbio de informação pode revelar-se ineficiente.
  - Gestão redundante em cada servidor
  - Não há um registo central de nomes e serviços - pode tornar-se difícil encontrar que servidores e serviços estão disponíveis.

Modelo Máquina Abstracta

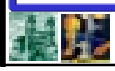
- Utilizada para modelar a interface dos sub-sistemas
- Organiza o sistema como um conjunto de camadas (cada uma implementa uma máquina abstracta) que proporciona um conjunto de serviços
- Suporta desenvolvimento incremental de sub-sistemas nos diferentes níveis. Quando a interface de uma camada se altera, só a camada adjacente é afectada
- Muitas vezes é difícil estruturar os sistemas desta forma

Modelo de Referência OSI



Características do Modelo Máquina Abstracta

- **Vantagens:**
  - Permite o desenvolvimento incremental
  - Fácil de alterar (se a interface for preservada, pode substituir-se uma camada por outra, se a interface se alterar só a camada adjacente é afectada)
  - Portável, já que esta aproximação localiza as dependências da máquina, em camadas baixas, só necessitando de alterar estas.
- **Desvantagens:**
  - Pode ser difícil estruturar sistema desta forma, obrigando a subverter o modelo, se camadas superiores acedem a serviços oferecidos por camadas vários níveis mais abaixo (p.ex. gestão de ficheiros)
  - O desempenho pode ser prejudicado, dada a intervenção de múltiplas camadas, com overhead de gestão de cada camada associado



Modelos de Controlo

- **Lidam com o controlo de fluxo entre os sub-sistemas. Diferente do modelo de decomposição do sistema**
- **Controlo centralizado**
  - Um sistema tem a responsabilidade pelo controlo e início e paragem de outros sub-sistemas
- **Controlo baseado em Eventos**
  - cada sub-sistema responde a eventos gerados externamente por outros sub-sistemas ou pelo ambiente

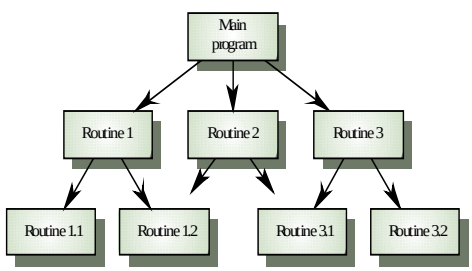
Controlo Centralizado

- **O sub-sistema de controlo tem a responsabilidade de gerir a execução dos outros sub-sistemas. As decisões de controlo são normalmente determinadas por valores d**

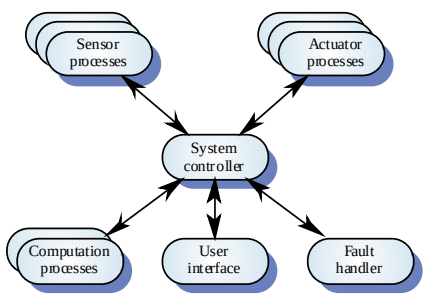
- **Podem ser identificadas duas aproximações:**
  - Modelo chamada-retorno
    - Modelo de sub-rotinas top-down, onde o controlo tem início na sub-rotina no cimo da hierarquia e move-se para baixo. Aplicável a sistemas sequenciais.
  - Modelo gestor
    - Aplicável a sistemas concorrenciais. Um componente do sistema controla o início, paragem e coordenação de outros processos.

Controlo Centralizado

Modelo Chamada-Retorno

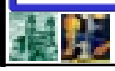


Modelo Gestor



Sistema Conduzido por Eventos

- **Conduzido por eventos gerados externamente, onde a ocorrência do evento está fora do controlo do sub-sistema que processa o evento**
- **Dois modelos principais:**
  - Modelo broadcast. Um evento é enviado a todos os sub-sistemas, ou só aqueles que manifestaram interesse em eventos específicos (broadcast selectivo). Qualquer sub-sistema que possa manejar o evento pode fazê-lo
  - Modelo conduzido por interrupts. Utilizado em sistemas em tempo real, onde os interrupts são detectados por um gestor de interrupts e passados para outros componentes para serem servidos.

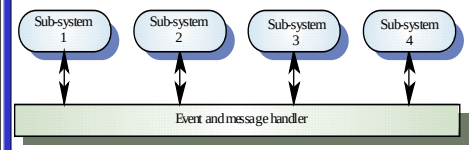


Modelo Conduzido por Interrupts

- **Utilizado em sistemas em tempo real, onde uma resposta rápida a um evento é essencial**
- **Há um conjunto de tipos de interrupts com um “handler” definido para cada tipo**
- **Cada tipo está associado com uma localização de memória um switch por hardware faz a transferência para o “handler”**
- **Permite uma resposta rápida, mas programação complexa e de difícil validação**

Sistemas Conduzidos por Eventos

Modelo Broadcast



Modelo Conduzido por Interrupts