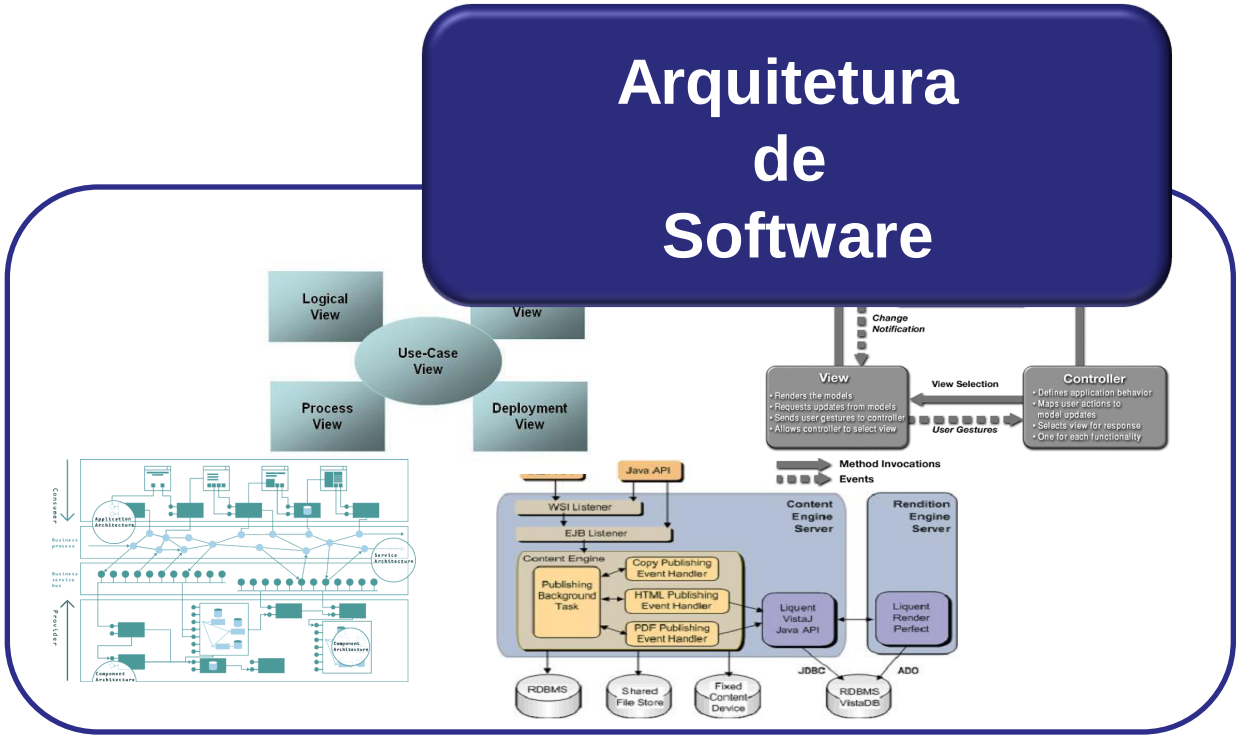
Arquitetura de Software



Princípios de Arquitetura de SW

Curso de Engenharia de SW

Prof: Wilson Wistuba

Introdução

• ***A arquitetura de software proporciona a resolução de requisitos não-funcionais ou ainda capacidades do software/sistema. Nessa aula iremos aprender o significado cada um dos temas; "–idades" em especial, para discussão, compreensão e preparação para os temas vindouros:***

Links da aula de hoje:

- ✓ http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html
- ✓ http://www.ibm.com/developerworks/rational/library/2774.html
- ✓ http://crpit.com/confpapers/CRPITV46Pollard.pdf
- ✓ http://martinfowler.com/articles/designDead.html
- ✓ http://martinfowler.com/eaaCatalog/

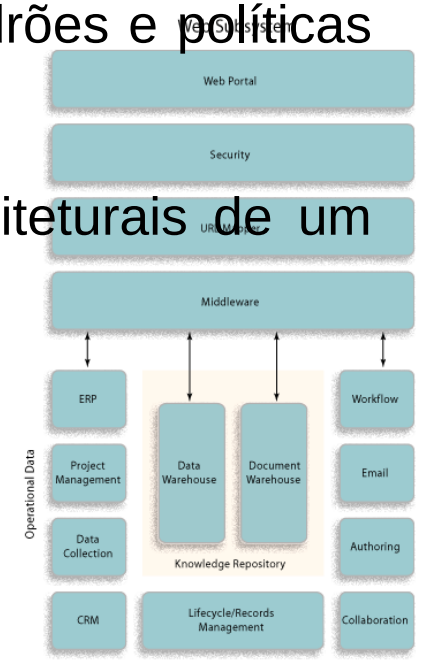
Arquitetura de Sistemas - Resumo

Arquitetura de Sistemas:

a. Define os princípios e diretrizes do(s) sistema(s). Trata uma visão mais abrangente do que do arquitetura de software mas os limites não são tão claros entre as duas. Trata também da estrutura, design das interfaces, layout físico, layout lógico, layout funcional e do processo de suporte a requisitos não funcionais e funcionais.

b. Produz/Propõe a arquitetura de referência, padrões e políticas do sistema.

c. Trata das premissas, restrições e metas arquiteturais de um software



Arquitetura de Sistemas - Resumo

Representação: *Uso formal de linguagens (ADL's) para demonstrar o relacionamento entre diferentes sistemas e os subsistemas além do fluxo da informação na cadeia de negócios.*

Para que serve então?

- *Determinar os subsistemas*
- *Processos de relacionamento entre esses subsistemas*
- *Determinar o modelo de implementação do sistema.*
- *Identificação dos atores (que podem ser outros sistemas) e os casos de uso mais significativos de um software.*
- *Fornecer a visão preliminar do sistema e sua organização*
- *Apoiar as decisões dos "stakeholders" e "sponsors" do projeto*

Existem outros tipos de Arquitetura?

- **Arquitetura de Infra-Estrutura:** Foco no hardware e nas operações de rede

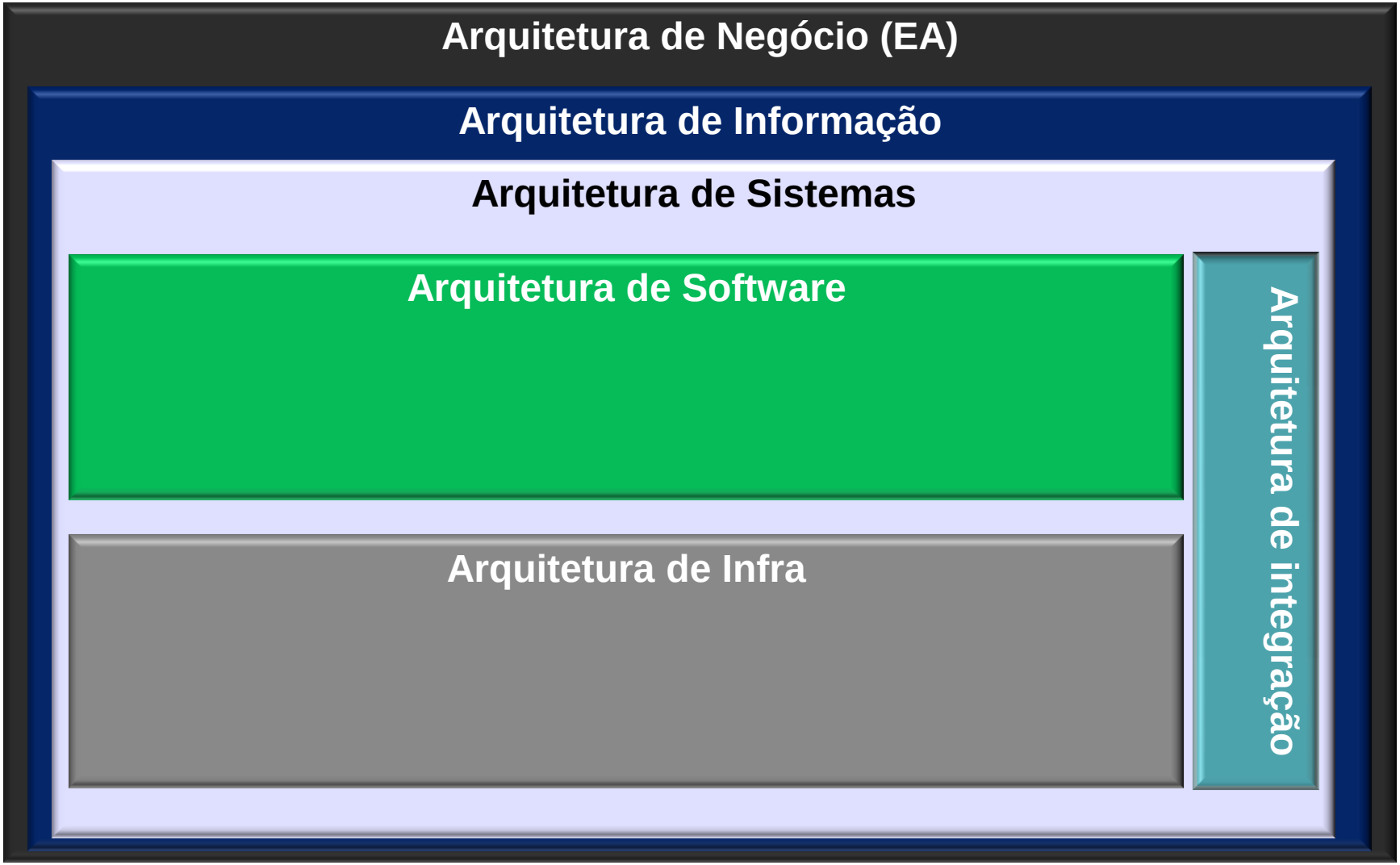
- **Arquitetura de Integração:** Foco no relacionamento entre interfaces de sistemas e no relacionamento entre elas.

- **Arquitetura de Negócio:** Foco nas diretrizes (tecnológicas) de negócio nas áreas e processos de negócio

- **Arquitetura de Informação:** Foco nos dados, nas informações e nos documentos que compõe uma Arquitetura de Negócio

Existem diversas definições de tipos de arquiteturas mas são conceituais e na prática acabam misturando-se dependendo sempre do escopo do problema e da tecnologia

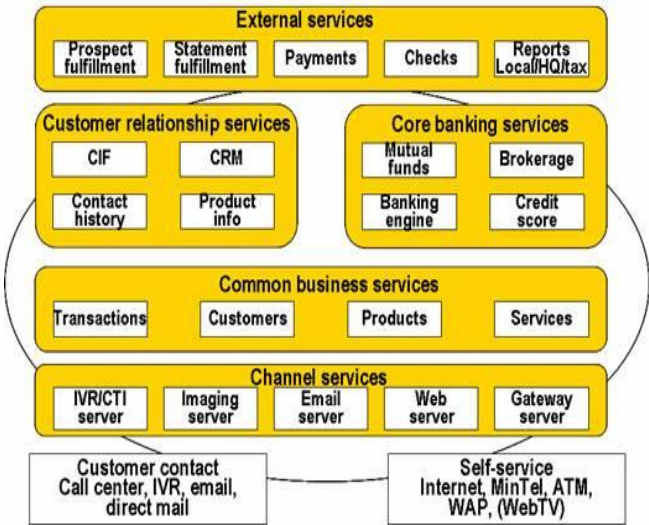
Tipos de arquitetura de software



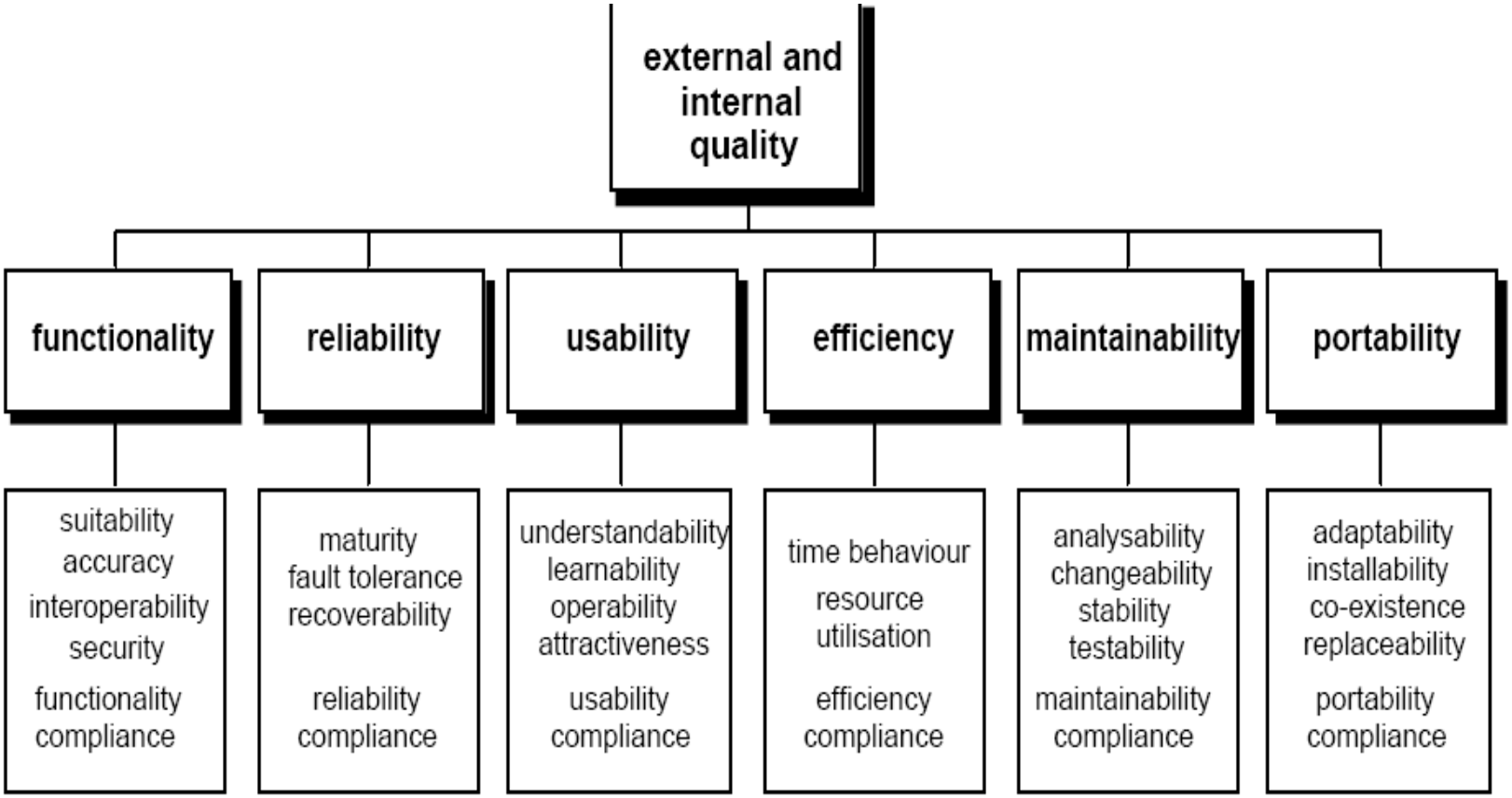
* ***Baseado na visão RM-ODP***

Sistemas Corporativos

- **Sistemas que suportam e implementam processo de negócio**
- **Envolvem dados PERSISTENTES**
- **Tratam dos modelos B2C e B2B (em geral)**
- **Tem idades(tempo) diferentes entre si**
- **Nem sempre são sistemas "grandes"**
- **Trabalham com problemas como:**
    - *i. Multithreading*
    - *ii. Grandes volumes de dados*
    - *iii. Interfaces (ricas) com usuário*
    - *iv. Distribuição do software*
    - *v. Lógica de negócio*
    - *vi. Integração com outros sistemas*
    - *vii. Domínio de dados*
    - *viii. Disponibilidade*
    - *ix. Tolerancia a falhas (robustez)*



Qualidade x Arquitetura de Software



**** ISO 9126 – Categorias de requisitos de qualidade***

Arquitetura de Referência

Definição: É um selecionado das práticas, padrões e normas da organização sobre desenvolvimento de software e arquitetura, que pode estar sendo definido para o projeto ou ter sido definido anteriormente. O uso da arquitetura de referência é uma técnica eficaz de reaproveitar decisões arquiteturais e análises arquiteturais de outros projetos, aumentar a qualidade e resolver requisitos não funcionais. Exemplos dos assuntos encontrados em uma arquitetura de referência:

- ***Divisão de camadas***
- ***Tecnologias utilizadas***
- ***Estrutura de pacotes***
- ***Abordagens de empacotamento***
- ***Governança arquitetural***

Ex: Ao definir que o sistema deve seguir a arquitetura de referência MVC da Sun sobre um servidor de aplicação estamos aproveitando todos os patterns já aplicados (ex: front-controller), estamos automaticamente definindo a decomposição em camadas do sistemas. Mas esta arquitetura é uma referência e pode não resolver todos os requisitos funcionais e não funcionais do seu projeto. É necessário instanciação da arquitetura de referência na arquitetura de sistemas.

"O bom arquiteto não re-inventa a roda ele aproveita os modelos já testados e aprovados para resolver os problemas no seu trabalho!!!"

Exemplo de arquitetura de referência

Visão geral: *O sistema "XPTO" deverá estar organizado em torno de 4 camadas.* ***Interface, Processos, Serviços e Dados***

Interface: Camada que servirá para tratar e renderizar todas as informações vindas das camadas de processos e de serviços. As tecnologias candidatas a essa camada são:

- JEE 1.5
- .Net framework 3.5 ou C#

Processos: Camada que servirá para suporte a uma engine "runtime" de processo de negócio desenhados sobre a linguagem **BPMn** e que deverá acessar somente a camada de serviços.

Serviços: Camada que servirá para suporte aos processo e a interface de negócio. Esta camada deverá ser implementada através de interfaces *WebServices HTTP/SOAP*

Governança de serviços: *Os serviços deverão ser versionados após a entrada em produção respeitando a regra a seguir:*

[Nome_Serviço]_V[nr] Ex: ConsultaXPTO_V1

Qos, SLA e Requisitos não funcionais

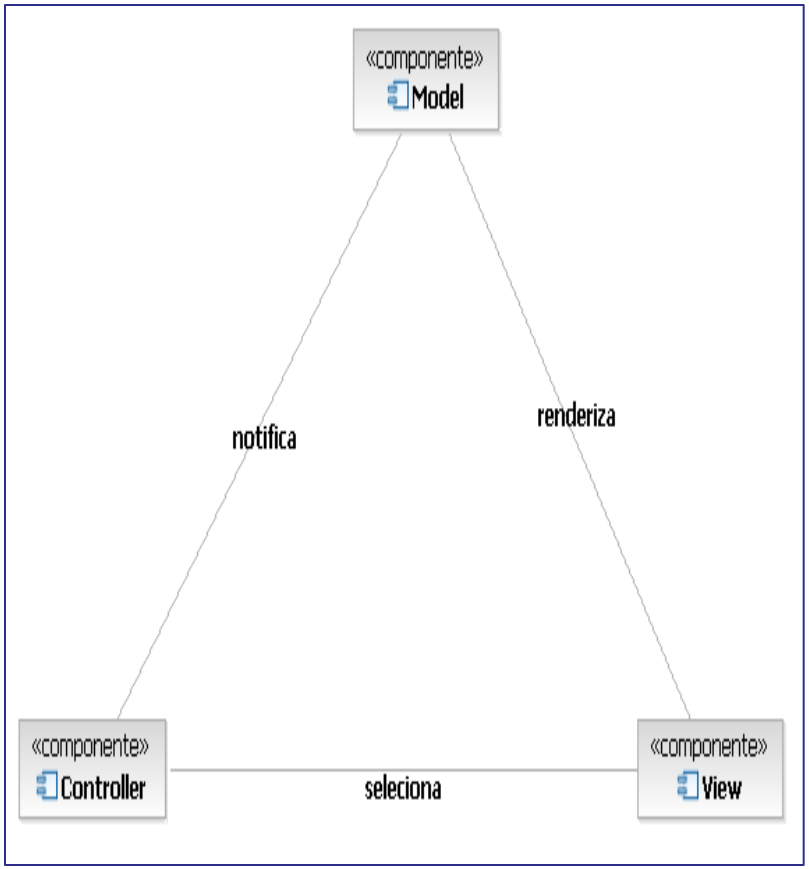
- **QoS:** *Quality of Service*, embora fosse um termo muito aplicado a redes com advento dos WebServices por exemplo começou a fazer sentido para o software e consequentemente para arquitetura. Algumas capacidades do software definem os níveis de QoS como: Performance, Disponibilidade, Segurança e etc

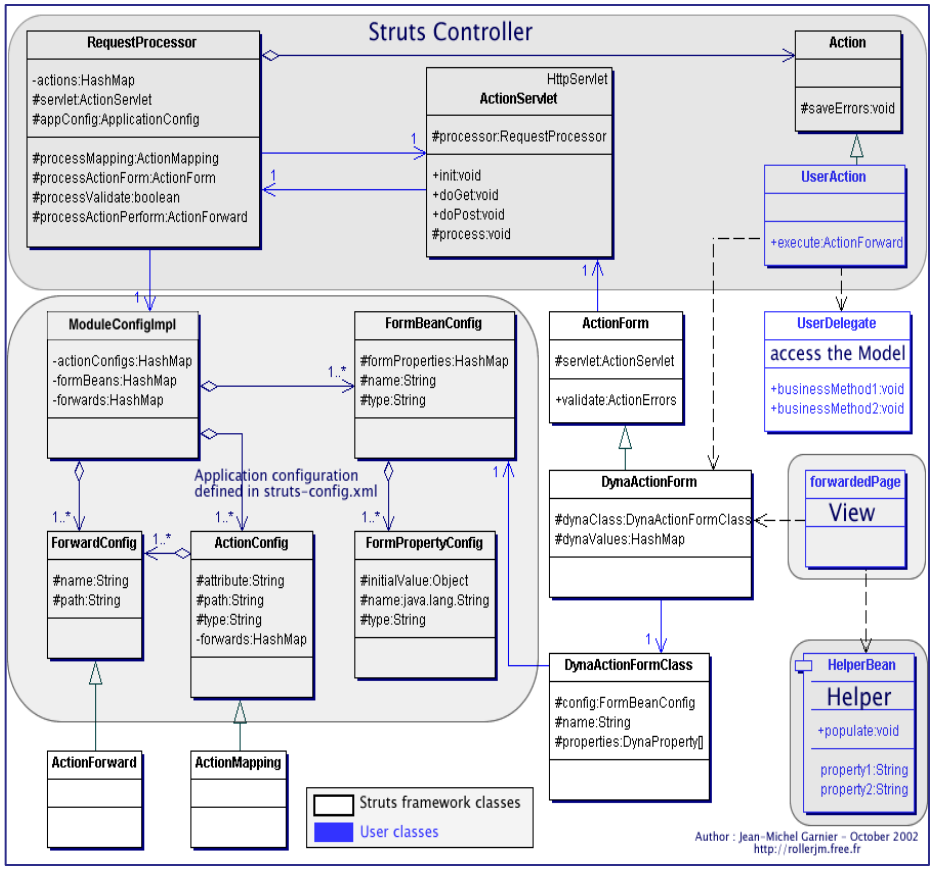
- **SLA:** *Service Level Agreement, é a formalização do nível de de **QoS** de um software, pode estar na forma de um contrato e balizar por exemplo multas comerciais.*

- **Requisitos não funcionais ou suplementares:** *São requisitos não ligados diretamente a um caso de uso mas que são determinantes em todo sistema . Podem ser complexos como disponibilidade e performance ou podem ser mais simples como Internacionalização ou suporte a browsers distintos. Os requistos não funcionais são direcionadores das CAPACIDADES de um software*

Abstração

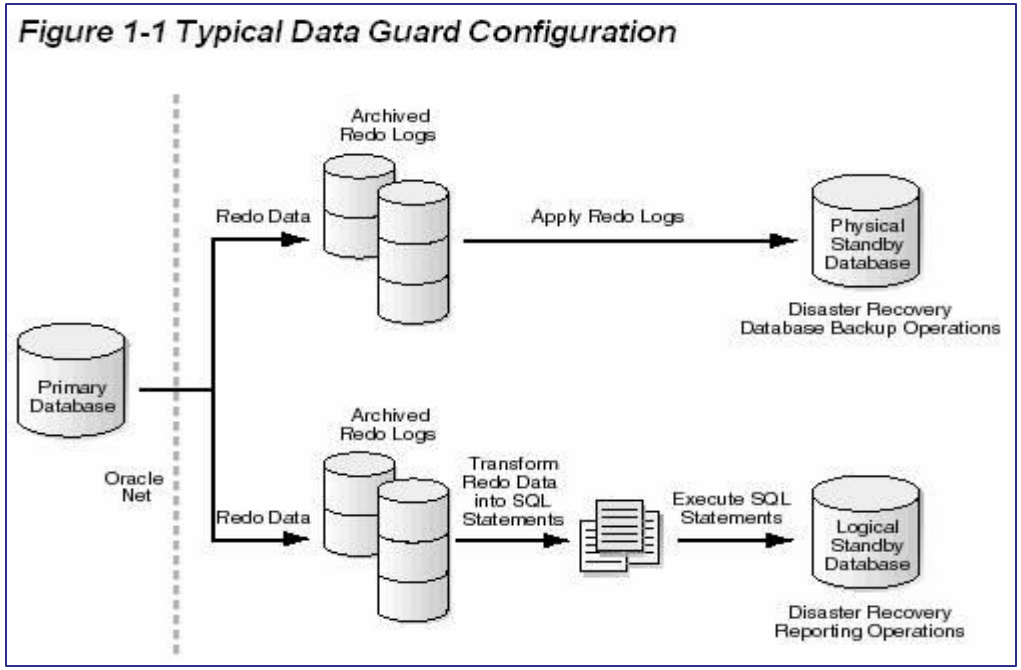
Ao analisar/produzir um modelo de arquitetura é necessário abstrair detalhes de implementação e implantação que embora impactantes são ignorados por algum motivo. É necessário que arquiteto consiga enxergar os componentes de software apenas com uma representação mais simplificada, a isto costuma-se atribuir a capacidade de abstração. Essa é a essência da "***Arquitetura vs Design"!***





Prevenção

A definição da arquitetura de software e de sistemas deve prever mecanismos de recuperação e funcionamento anormal do software. Nem sempre todo os componentes estarão disponíveis e farão seu trabalho da maneira como foram previstos. Uma arquitetura deverá então dispor do uso e manipulação de exceções, mecanismos de dupla verificação, uso de camadas redundantes, timeouts e algoritmos que implementem re-tentivas. A prevênção relaciona-se diretamente as capacidadades de robustez, integridade e confiabilidade do software.



Flexibilidade

A flexibilidade permite que um software seja gerenciável, escalonável e confiável. Basicamente está ligada ao grau de dificuldade de alterar código de acordo com a necessidade do negócio. A independência entre a localização do software e sua execução é um bom parâmetro para flexibilidade. Um exemplo clássico é o uso de HARD CODED's que diminuem a flexibilidade de um componente de SW. Outros exemplos são os usos de VM's que aumentam a flexibilidade de uma plataforma/sistema. A independência de HW vs SW . Ex: SO's e plataformas de HW, tornam as aplicações mais adaptáveis e portáveis. Um uso tradicional são os mecanismos de contectividade a bancos de dados que desacoplam o software e permitem adaptabilidade a vários DATA SOURCES.

Manutenabilidade

Qual o trabalho ou qual a complexidade de corrigir um defeito, um “bug” no software, seja em tempo de desenvolvimento seja pós-produção?

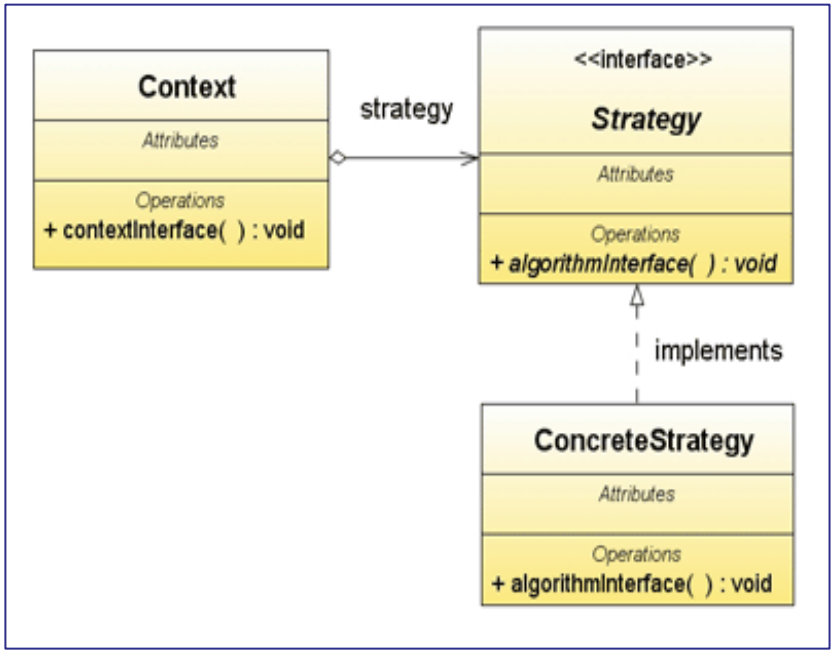
A manutenabilidade permite que o software seja corrigido e evoluído de maneira menos impactante. Algumas das técncnicas sabidas para manter-se a manutenabilidade são modularização ou criação de subsistemas mesmo que na mesma camada de software. Outros pontos que influenciam são a quantidade de documentação e rastreabilidade de requisitos.

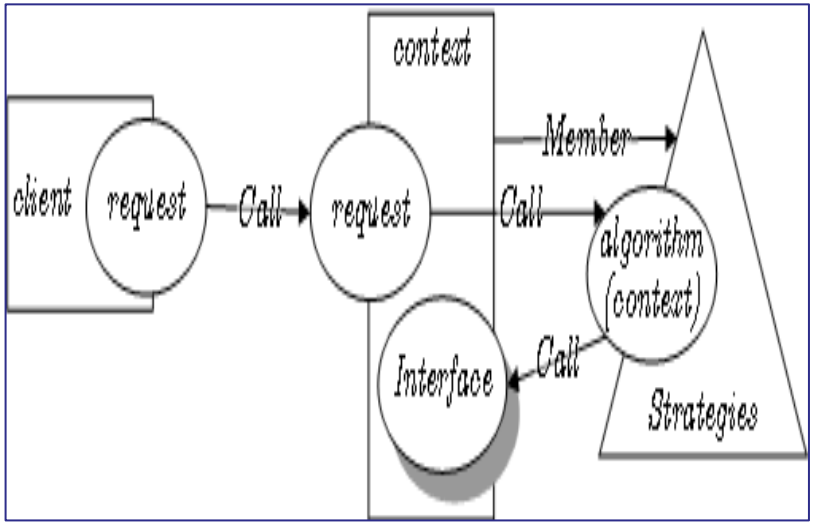
> Ex: *Quando é necessário alterar uma classe em PHP chamada Menu em quantas páginas ela é usada? Quais são suas dependências? Quanto tempo e quanto dinheiro gasto para corrigir o software e colocá-lo em produção passando por ciclo de testes e SQA?*

Ferramentas de desenvolvimento e técnicas de design como delegação e separação de responsabildades ajudam a manter a manutenabilidade

Isolamento

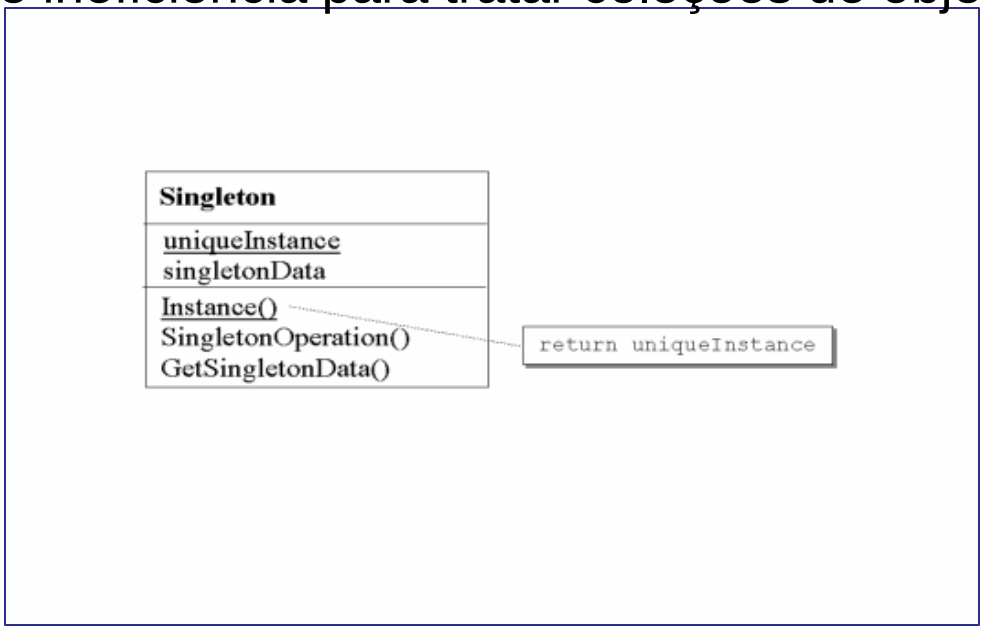
Uma arquitetura de software projeta componentes, camadas e pacotes que “cercam” as características de um software de maneira confiná-lo em um conjunto de classes ou a um conjunto de tipos de objetos ou ainda a uma determinada TIER. Dessa maneira qualquer problema que impacte um sistema pode ficar restrito a um conjunto de artefatos que não prejudique todo o restante ou ainda que propicie o desenvolvimento “em paralelo” de muitas partes desse software. Em muitos aspectos o isolamento é igual ao encapsulamento da OO.





Imutabilidade

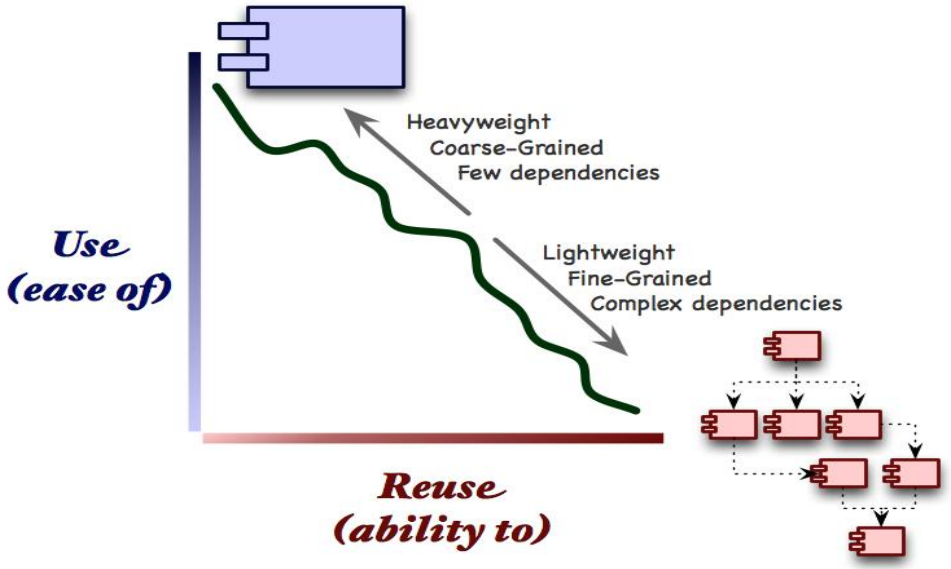
Uma técnica OO útil no design e arquitetura baseia-se na idéia de que o objeto durante todo seu tempo de vida (RUN TIME) só pode ter um valor e portanto não muda. As vantagens dessa técnica facilitam sistemas inteiros. Garante eliminação de problemas de acesso e o não uso do gerenciadores para tratar problemas de concorrência. O design pattern mais aplicado a essa situação é o Singleton (desenho abaixo). Classes e Objetos imutáveis são mais simples para desenvolver e diminuem os erros nos testes. Claro que essa técnica não é aplicável a todos os casos, não existiriam recursos computacionais suficientes para tratar de todo o sistema e o design OO seria esgotado pela multiplicação de classes concretas e ineficiência para tratar coleções de objetos.



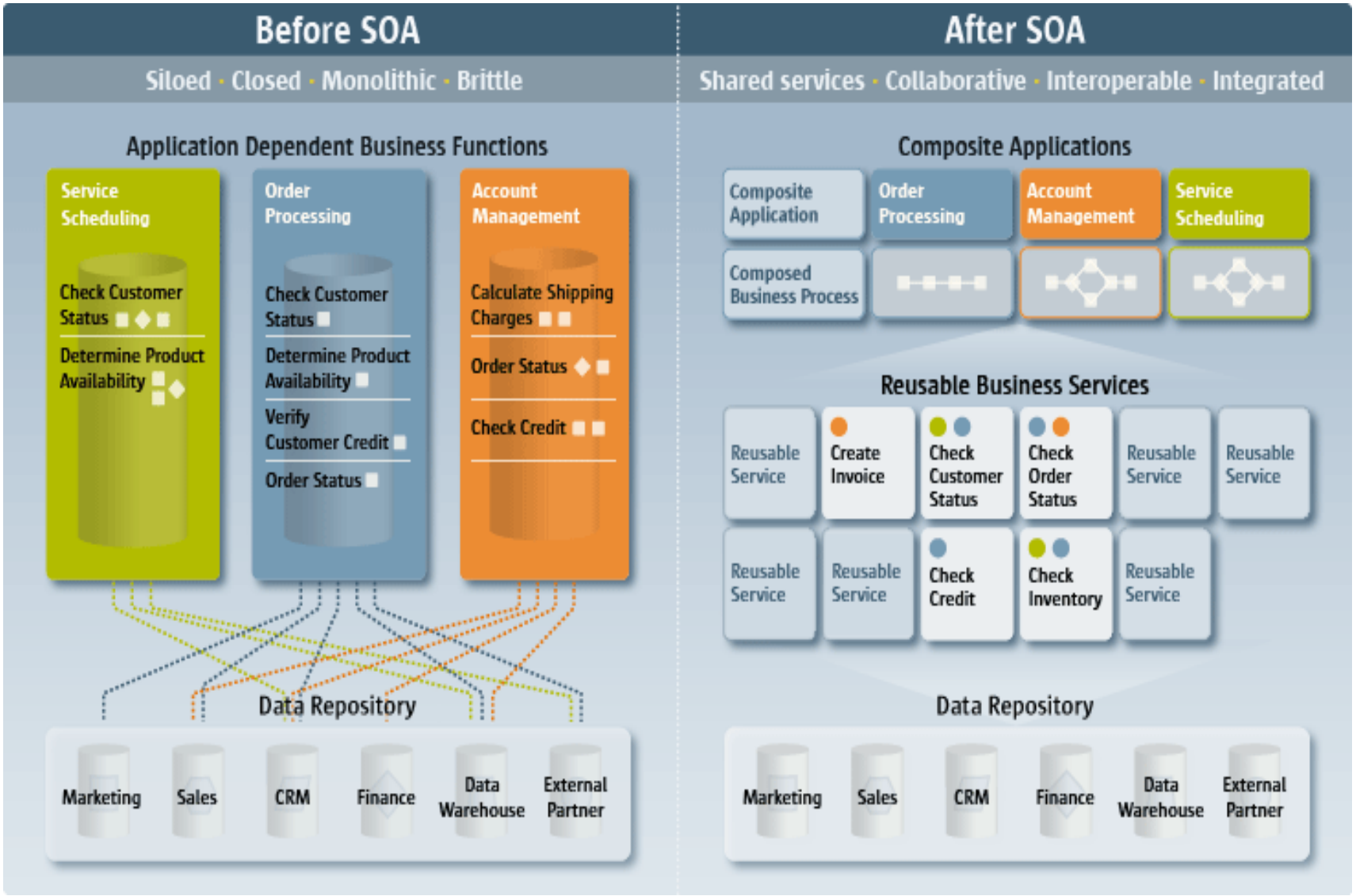
Reusabilidade

O re-uso significa utilizar um design/codificação já desenvolvidos “N” vezes no mesmo domínio e até mesmo em domínios separados. O re-uso pode ser em nível de métodos, classes, objetos em memória, serviços, interfaces, softwares e sistemas. Uma arquitetura flexível permite aumentar o re-uso aumentando assim a produtividade (dimimuição do tempo de programação) e diminuindo o ciclo de testes. Quase todos os frameworks citam o re-uso como principal fator de benefício para seus usos .

Ex: Struts, Log4J, Hibernate e etc. Uma grande parte dos software de “prateleira” ou soluções **BLACK-BOX** usam essa capacidade como motivador principal!!



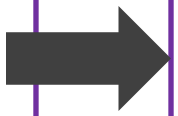
Ex: Reusabilidade



Robustez

Define a capacidade do software tratar condições incomuns e heterôgeneas como por exemplo dados corrompidos, problemas de ambiente. Diz-se que um software é mais robusto quanto mais ele conseguir tratar os erros do ambiente veja o exemplo em Java:

```
public class Aluno {

private String nomeAluno;
private int idAluno;

public Aluno(int idAluno) {
// TODO Auto-generated constructor
stub
}

public String getNomeAluno() {
return nomeAluno;
}

public void setNomeAluno(String
nomeAluno) {
this.nomeAluno = nomeAluno;
}

}
```



```
public class Aluno {

private String nomeAluno;
private int idAluno;

public Aluno(int idAluno) {
// TODO Auto-generated
constructor stub
}

public String getNomeAluno() {
if (this.nomeAluno == null)
return "Aluno SEM NOME";
else
return nomeAluno;
}

public void setNomeAluno(String
nomeAluno) {
this.nomeAluno = nomeAluno;
}

}
```

Performance

A arquitetura impacta diretamente no desempenho de um sistema, em geral problemas de performance são difíceis de resolver depois que o software já está implementado. Ao falarmos de desempenho estamos falando de características como:

a. **Tempo de resposta:** Quanto tempo o software leva para processar algo ou para atender a uma requisição externa

b. **Agilidade de resposta:** Quanto tempo o sistema demora para reconhecer uma requisição e devolver a requisição. Esse é o tempo percebido pelo usuário.

c. **Latência:** É o tempo necessário para se obter qualquer resposta do seu software mesmo que o trabalho seja mínimo ou inexistente. Imagine um sistema que use *WebServices*, toda e qualquer resposta mesmo que seja para retonar o erro envolve sensibilizar um engine *WS* e realizar **marshalling unmarshalling** de dados.

d. **Throughput:** Quantidade de tarefas que um sistema pode executar em um certo tempo. Essa medida pode ser dada em **TPS** ou em **Bytes** por segundo.

Escalabilidade

É a medida com que o acréscimo de recursos de SW ou HW afetam o desempenho do software. Uma arquitetura de sistemas escalável permite a adição de mais hardware melhorando o throughput e os tempos de respostas.

A escalabilidade ganha o nome de **escala vertical** quando são adicnionados mais recursos ao um servidor por exemplo adicionar mais memória.

Quando são adicionados mais servidores chamamos de **escala horizontal**. A escalabilidade no entanto afeta todos os componentes de SW de maneira distinta. Sistema com arquiteturas escaláveis são capazes de priorizar processamento para as requisições menos importantes em momentos de baixa atividade, são capazes de refutar requisições em momentos de stress e até mesmo desativando componentes do SW.

Um sistema com boa escalabildade degrada de forma elegante até 100% de sua capacidade ou seja ocupa todos os recursos de HW e SW mantendo o tempo de resposta. Possuindo facilidade de adicionar mecanismos de I/O até mesmo dobrando sua capacidade conseguindo priorizar o trabalho a ser processado.

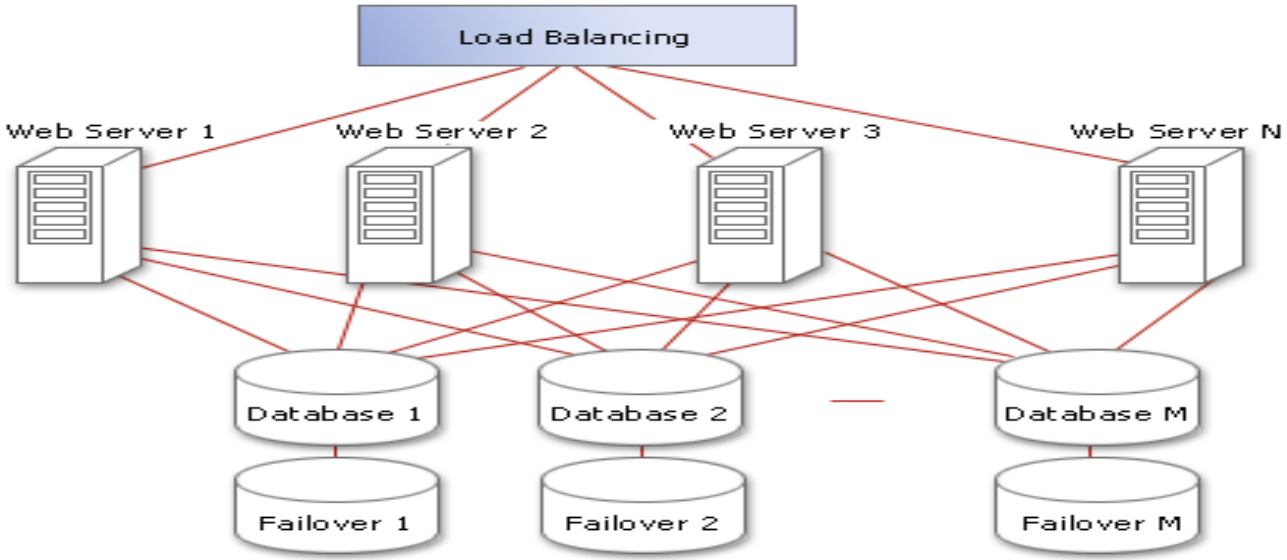
Disponibilidade

É a medida em que o sistema está acessível, quanto tempo o sistema está no “ar” respondendo as requisições do usuário mantendo QoS e SLA’s acordados e mantendo o tempo de resposta. Muitos sistemas costumam ser concebidos com requisito 24x7. No entanto observa-se que esse requisito é **ERRADO**. O requisito **CORRETO** deveria ser 24x7 durante 99% do tempo em um mês, por exemplo, o que significa que a janela de manutenção ou períodos de indisponibildaide são de 7,2horas no mês. O arquiteto deve estar atento pois os sistemas sofrem correções e evoluções que nem sempre podem ser feitas sob operação initerruptas.

Algumas técnicas bem conhecidas de arquitetura de sistemas para aumento de disponilidade sistêmica são o uso de FAIL-OVER, LOAD-BALANCE e CLUSTERING para aumentar a disponibilidade de um sistema. Os impactos nos componentes de SW são grandes pois alguns conceitos de sistemas distribuídos passam a ser incorporados ao escopo do problema.

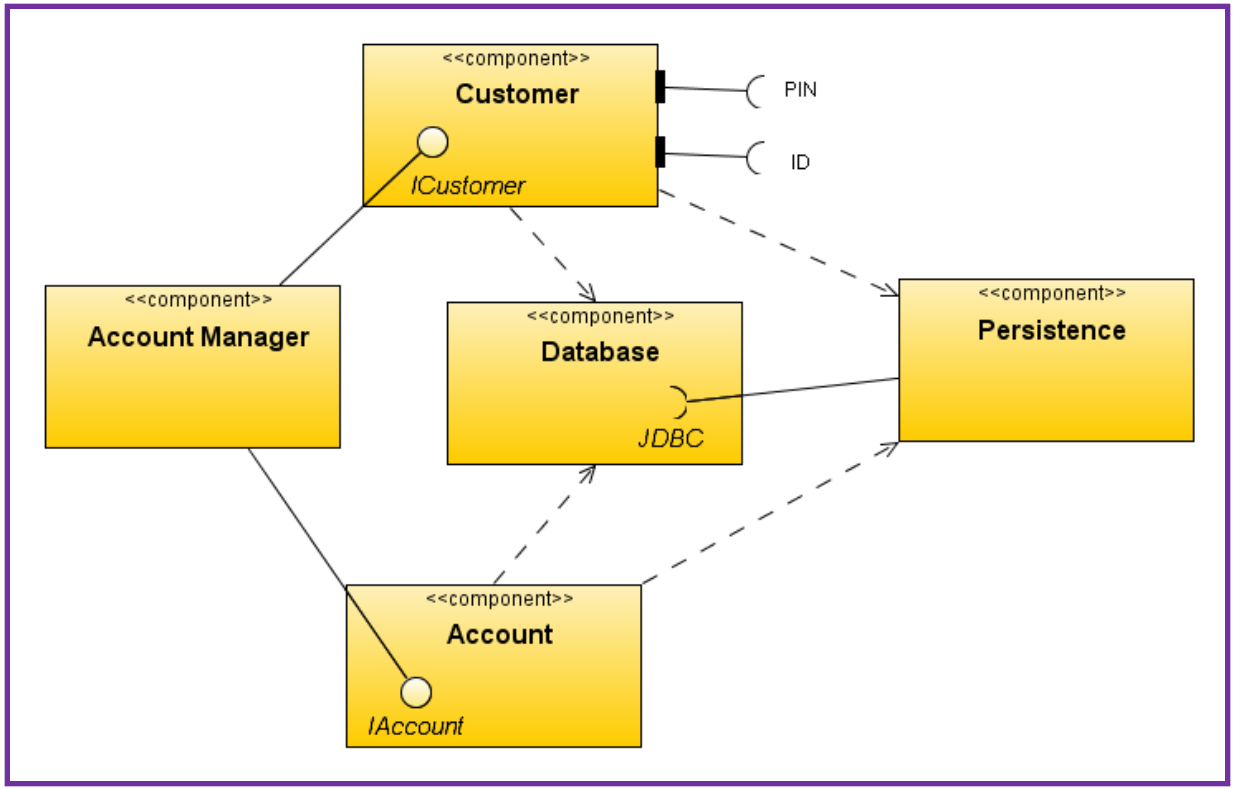
Capacidade

A capacidade está relacionada ao máximo throughput ou máxima carga no sistema em um determinado momento. Ela diz o que um sistema pode executar no máximo de “pressão”. A medida da capacidade e carga pode ser dada em segundos, por exemplo: um sistema com 100 usuários conectados pode ter um tempo de resposta de 0,5 s e com 1000 2s. Logo ele é sensível a carga de 4x degradando seu tempo de resposta. Os cálculos de capacidade de HW e SW envolvem dados como req/s, tp/s análise de memória, processador e etc



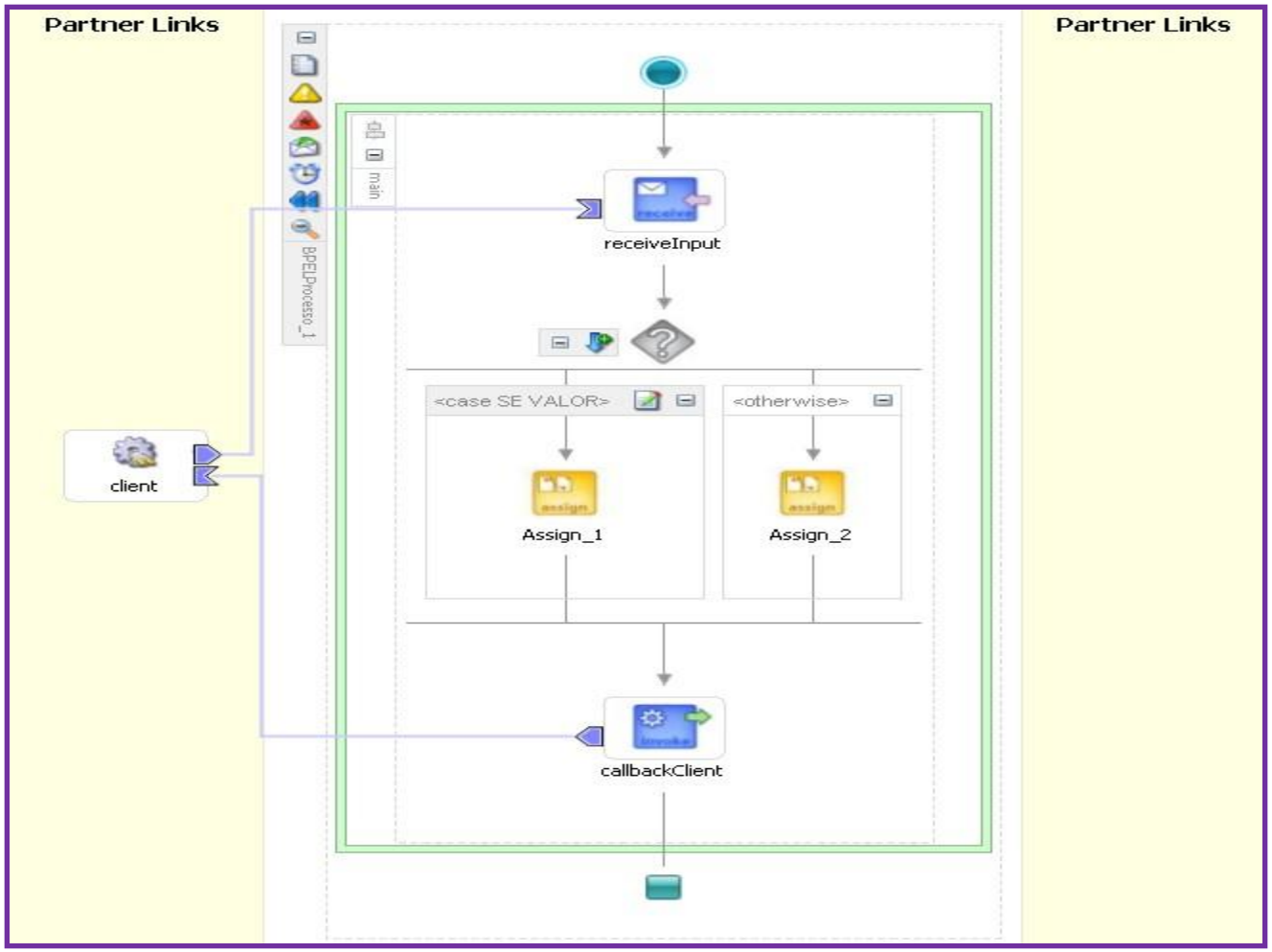
Componentização

É a aplicação prática de abstração mais popular de softwares. Um componete focaliza no aspecto lógico e funcional do software que pode ser composto por vários pacotes, classes, arquivos e etc. É comum a representação de WebServices ou mecanismos de persistência como componentes por exemplo. A diagramação de componentes é essencial para o arquiteto de software!!!



Complexidade

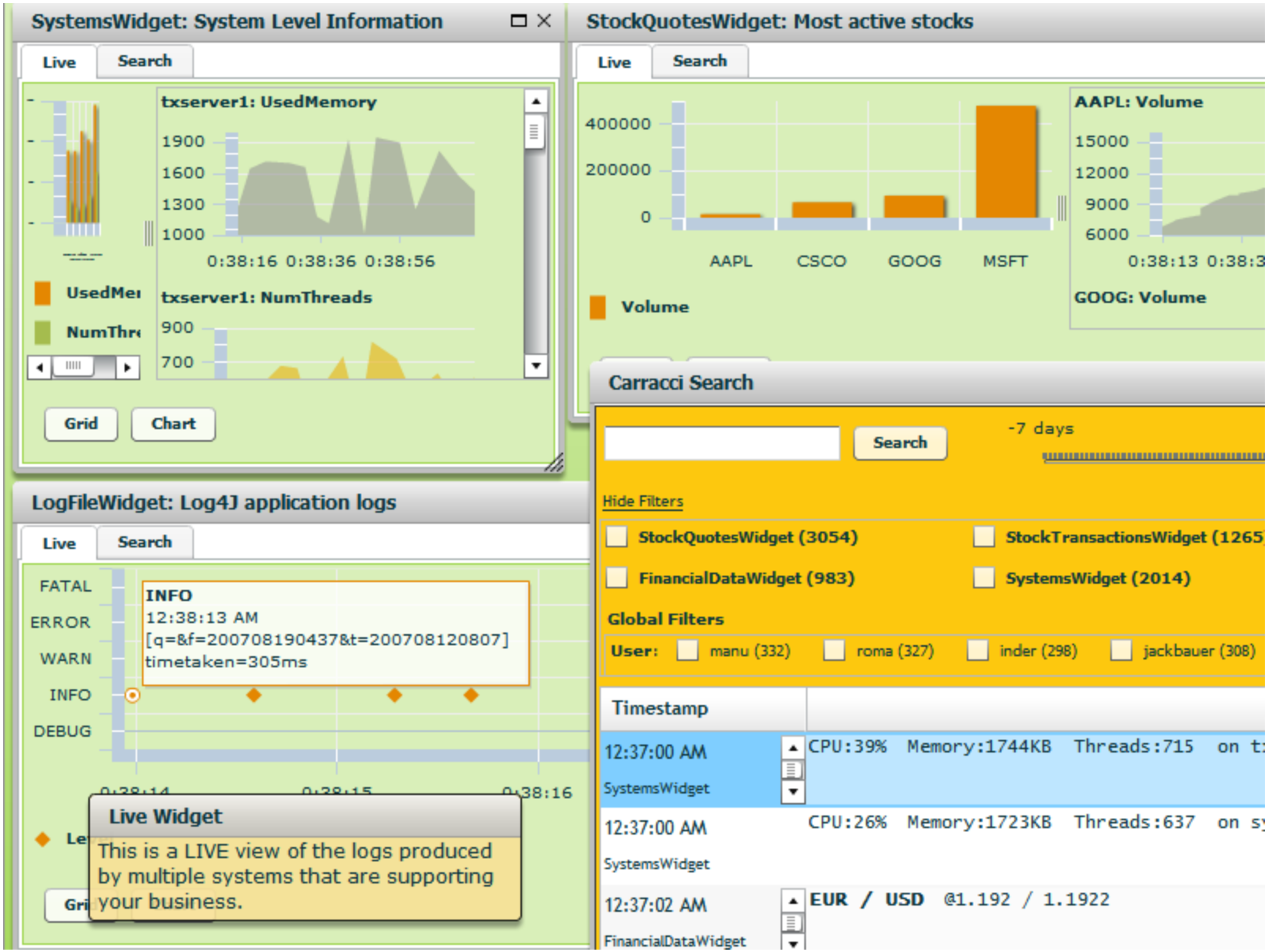
Um software pode ser extremamente simples no entato sua arquitetura pode ser complexa. A complexidade então não pode ser medida somente pelo algoritmo, quantidade de operações, ciclos de processamento. A disposição do software importa sua estrutura também!



Gerenciabilidade

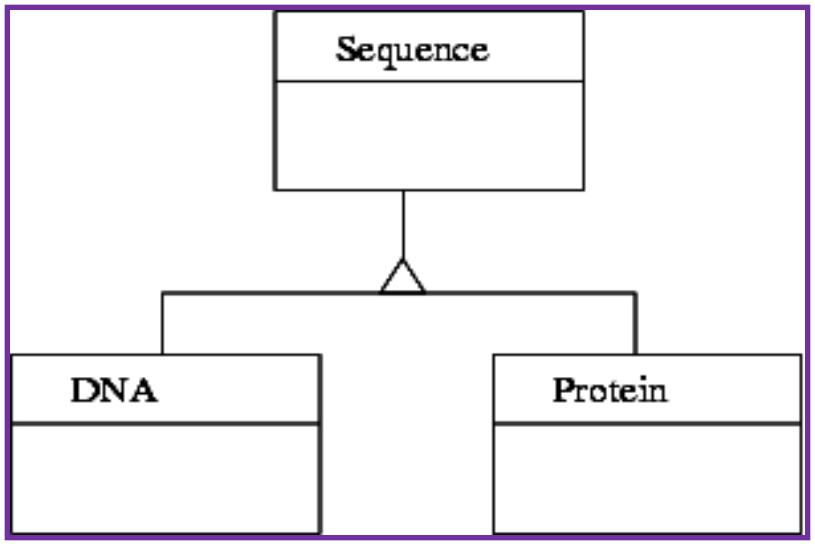
A gerenciabilidade do software envolve a percepção e o processo de trabalho envolvidos em um determinado ambiente. A gerenciabilidade está relacionada a quantidade de mecanismos do software que permitem a ele mudar, ou obter informações em RUN TIME. Para SO's por exemplo em geral a quantidade de telas e recursos para o usuário adicionar recursos de HW e SW e realizar atividades corriqueiras melhoram a gerenciabilidade do SO. Para sistemas de TI em geral a quantidade de "ferramentas" que permitem ao sistema ser monitorado e alterado de acordo com as mudanças de negócio aumenta gerenciabilidade. Um exemplo muito atual de gerenciabilidade do SW são os ***DASH BOARDS (BAM –Business Activity Monitoring)***

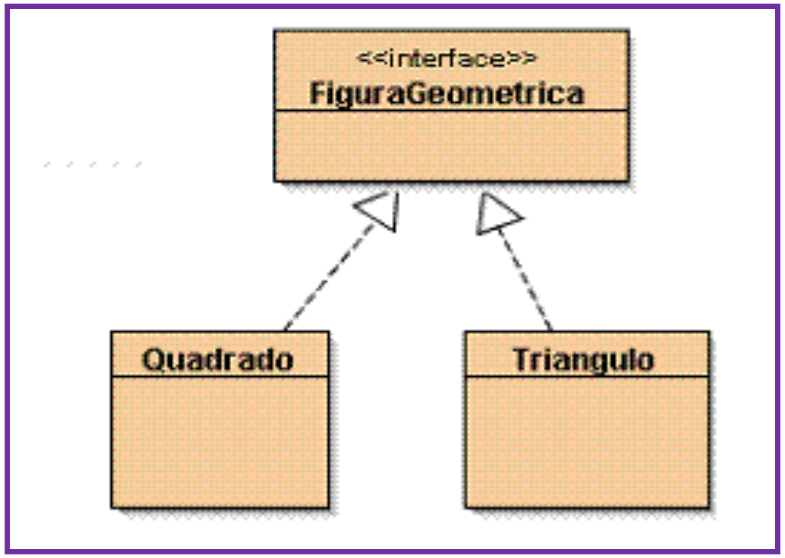
Ex: Gerenciabilidade



Extensibilidade

A extensibilidade do software basicamente trata da capacidade do software ser usado para outros requisitos funcionais que não haviam sido pensados. Ou ainda a propriedade do software “propagar” suas características. A extensibilidade pode dar-se no nível de design como por exemplo com uso de interfaces ou herança em OO. No nível de sistemas o uso de um padrão de comunicação como SOAP que permite que um software receba/envie mensagens de qualquer cliente independente da tecnologia.





Segurança

A segurança é considerada um aspecto não funcional por vários motivos, no entanto o negócio começa cada vez mais a incluir requisitos de segurança como funcionais. Um exemplo disso trata do SINGLE-LOGIN os usuários querem ter uma única senha para autenticar em qualquer sistema das empresas em que trabalham. Ao tratarmos da segurança do sistema falamos dos seguintes aspectos

- **Privacidade:** As informaçõe só são conhecidas pelos interlocutores
- **Integridade:** Os dados não estão corrompidos
- **Autenticidade:** Prova que os participantes forneceram evidencias de que eles são eles mesmos realmente
- **Autorização:** Habilita um usuário a acessar um recurso
- **Auditoria:** Em geral trata dos logs de modificação e acesso
- **Confidencialidade:** Capacidade de impedir que os não autorizados acessem a informação
- **Confiabilidade:** Método para garantir que os sistemas e recursos estejam disponíveis protegidos contra falhas

Exercícios

1. **Descreva 3 capacidades de software relacionando exemplos, técnicas e usos de cada uma.**