



INSTITUTO DE GESTÃO EM  
TECNOLOGIA DA INFORMAÇÃO

---

## **Princípios e Práticas em Arquitetura de Software**

---

Filipe Tório Lopes Ruas Nhimi

2016

## **Princípios e Práticas em Arquitetura de Software**

Filipe Tório Lopes Ruas Nhimi

**©Copyright do Instituto de Gestão em Tecnologia da Informação.  
Todos os direitos reservados.**

---

## Sumário

---

Sumário.....	3
Descrição do Curso.....	4
Capítulo 1 - Introdução à Arquitetura de Software.....	5
Capítulo 2 - Requisitos Arquiteturais .....	13
Capítulo 3 - Riscos e decisões .....	19
Capítulo 4 - Estimativas.....	24
Capítulo 5 - Modelagem e Modelos .....	32
Capítulo 6 - Estilos Arquiteturais .....	46
Capítulo 7 - Padrões de Projeto .....	52
Capítulo 8 - Modelagem Arquitetural.....	57
Referências bibliográficas.....	62

## Descrição do Curso

---

A disciplina de PPAS tem como objetivo introduzir conceitos e práticas em Arquitetura de Software, apresentando o papel do Arquiteto no processo de desenvolvimento de sistemas. Espera-se que o aluno, ao final da disciplina, consiga:

- ✓ Identificar os princípios da Arquitetura de Software e suas práticas
- ✓ Descrever o papel do Arquiteto de Software
- ✓ Entender o artefato documental que descreve a Arquitetura de um Software

## Capítulo 1 - Introdução à Arquitetura de Software

---

### Objetivos do capítulo

- ✓ Definir formalmente Arquitetura de Software
- ✓ Apresentar os objetivos da Arquitetura de Software
- ✓ Discutir o Papel do Arquiteto de Software
- ✓ Apresentar alguns passos da trajetória do Arquiteto de Software

### Conceituando Arquitetura de Software

---

O termo Arquitetura de Software é bastante antigo e foi sugerido pela primeira vez ao final da década de 60 por Edsger Dijkstra, porém, ganhou força e expressão na década de 90. Existem algumas definições de Arquitetura de Software com ideias em comum, mas que de modo geral se completam.



**Figura 1 - Edsger Dijkstra (1930-2002)**

**Fonte:** [http://pt.wikipedia.org/wiki/Edsger\\_Dijkstra](http://pt.wikipedia.org/wiki/Edsger_Dijkstra)

### Definição pela IBM

Uma arquitetura é o conjunto de decisões significativas sobre a organização de um sistema de software, a seleção de elementos estruturais e suas interfaces, juntamente com o comportamento especificado nas colaborações entre estes elementos, a composição destes elementos em subsistemas progressivamente maiores e o estilo arquitetural que guia esta organização. (*The Rational Unified Process: An Introduction*)

### Definição pela Microsoft

Arquitetura de Software é o processo de definição de uma solução estruturada que atende a todos os requisitos técnicos e operacionais e ao mesmo tempo otimiza atributos de qualidade padronizados como desempenho, segurança e gerenciamento. Arquitetura envolve uma série de decisões baseadas em uma vasta gama de fatores e cada uma destas decisões pode provocar um impacto considerável no sucesso ou fracasso da aplicação. (*Microsoft Application Architecture Guide, 2ª edição*)

### Definição pela IEEE

Arquitetura é a organização fundamental de um sistema materializada em seus componentes, na relação entre eles e com o ambiente e nos princípios que

guiam seu projeto e evolução. (ISO/IEC 42010:2007 *Systems and software engineering - Recommended practice for architectural description of software-intensive systems* [IEE00])

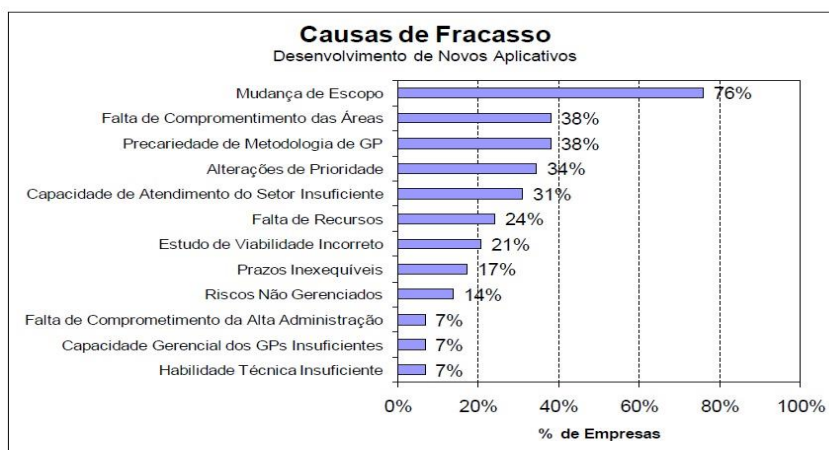
Em resumo, as definições apresentadas concordam que a Arquitetura de Software tem relação com as tomadas de decisões durante o processo de desenvolvimento assim como a estruturação dos elementos que compõe o sistema. Em poucas palavras:

- ✓ Decisões
- ✓ Organização fundamental
- ✓ Elementos de software
- ✓ Relacionamento entre elementos
- ✓ Relacionamento com o ambiente
- ✓ Princípios que guiam o desenho e evolução
- ✓ Pode determinar o sucesso ou fracasso

## Objetivos da Arquitetura de Software

Uma série de causas individualmente ou em conjunto pode fracassar um projeto de desenvolvimento de software. Diversos riscos e restrições associados ao processo aumentam a probabilidade de cancelamento. Falta de controle, escopo volátil, prazos exorbitantes, falta de recursos e entre outros vários motivos.

A figura abaixo mostra uma pesquisa realizada entre setembro e dezembro de 2010 onde participaram 345 organizações brasileiras. O objetivo da pesquisa foi identificar as principais causas de fracasso de projetos de desenvolvimento de novas aplicações.



**Figura 2 - Pesquisa motivos de fracasso no desenvolvimento de novas aplicações**

**Fonte:** <http://analiserequisitos.blogspot.com.br/2011/06/causas-de-fracasso-de-projetos-de.html>

A pesquisa mostra que 76% das organizações pesquisadas tiveram seus projetos fracassos devido à Mudança de Escopo. Já 38% das organizações o problema foi associado à Falta de Comprometimento das Áreas e Precariedade de Metodologia de Gerenciamento de Projetos. Perceba que o somatório dos percentuais é igual a 314% e isso indica que algumas organizações não tiram

seus projetos cancelados por apenas um motivo-problema, e sim, mais de uma causa. Essa pesquisa mostra de modo geral que, a falta de controle nos projetos é forte indício de fracasso.

Os objetivos da Arquitetura de Software são de reduzir os riscos associados ao projeto e das atividades que cercam o Processo de Desenvolvimento de Sistemas. Dentre os diversos benefícios adquiridos pelo processo de Arquitetura, destacam-se:

- ✓ Redução de riscos associados ao negócio
- ✓ Redução de riscos de desenvolvimento e manutenção
- ✓ Alinhamento de expectativas dos stakeholders
- ✓ Construção de aplicações flexíveis e de qualidade
- ✓ Geração de valor aos negócios do cliente
- ✓ Apoio ao Arquiteto na tomada de decisões técnicas

### O Papel do arquiteto de software

O profissional Arquiteto de Software é responsável por uma grande variedade de atividades dentro do processo de desenvolvimento de sistemas. Como é ele quem define a arquitetura do software que será construído, ele tem a missão de garantir que toda a equipe seguirá as diretrizes da arquitetura até o final do projeto. Além disso, faz parte do trabalho de um Arquiteto de Software envolver-se com todo o processo de desenvolvimento, comunicar a arquitetura aos interessados pelo projeto, apoiar desenvolvedores na implementação e por ser um líder técnico, também terá a função de mentor e ajudará na formação de novos arquitetos.

- ✓ Definir, criar e manter a documentação arquitetural para guiar a construção e manutenção do sistema.
- ✓ Definir estratégias, estilos e padrões arquiteturais.
- ✓ Garantir que as diretrizes da arquitetura serão aplicadas até o final do projeto.
- ✓ Envolver-se com todo o processo de desenvolvimento.
- ✓ Comunicar a arquitetura aos stakeholders.
- ✓ Apoiar desenvolvedores na implementação.
- ✓ Atuar como mentor e formar novos arquitetos.

Durante todo o projeto de desenvolvimento de um software, o Arquiteto será cobrado, ou melhor, será exigido de forma a garantir certas necessidades do projeto e atender a determinadas restrições que são impostas do início ao fim. A figura abaixo mostra as possíveis influências sobre a Arquitetura de Software. O Arquiteto será cobrado pelo desenvolvimento ágil, com qualidade e de baixo custo. O atendimento de requisitos funcionais e não-funcionais severos, além de sua própria experiência, do seu time, do contexto organizacional e ambiente técnico que podem influenciar positivamente ou negativamente.



**Figura 3 - Influências sobre a Arquitetura de Software**

Como se não bastasse, o Arquiteto de Software também precisa possuir algumas habilidades interpessoais que são de extrema importância para o seu papel de líder técnico. A figura abaixo mostra as principais habilidades esperadas. Espera-se desse tipo de profissional a capacidade de:

- ✓ Liderança, pois, afinal ele será responsável por um time de desenvolvedores.
- ✓ Habilidade de comunicação para conseguir passar mensagens importantes de forma a sensibilizar os envolvidos a realizarem suas tarefas em conformidades com as exigências da Arquitetura.
- ✓ Negociação para conseguir mudar estratégias, combinar novos prazos e estabelecer consenso entre as partes
- ✓ Proatividade para agir imediatamente quando preciso e antecipar ações antes que os problemas aconteçam.

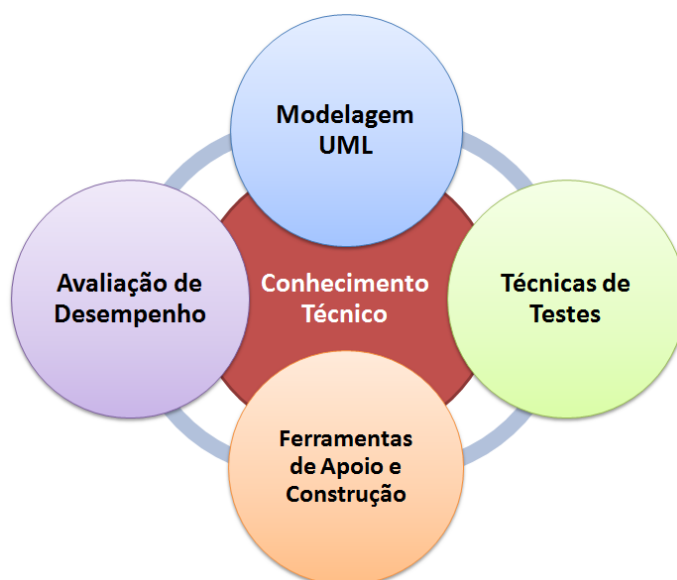


**Figura 4 - Habilidades interpessoais**

Alguns dos conhecimentos técnicos que o Arquiteto de Software precisa ter são mostrados na figura abaixo. Basicamente, ele precisa conhecer de Modelagem UML que será sua principal ferramenta para documentação das

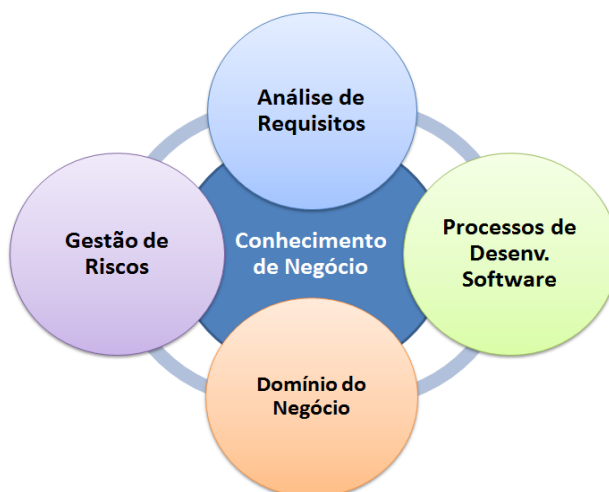


decisões arquiteturais do sistema. Técnicas de Testes para auxiliar na verificação de componentes de software assim como capturar a cobertura de código da aplicação. Essas técnicas de testes também são úteis para Avaliação de Desempenho. O conhecimento de ferramentas apoio e construção são ainda mais importantes, pois, estão diretamente relacionadas com a implementação do software modelado.



**Figura 5 - Conhecimento técnico**

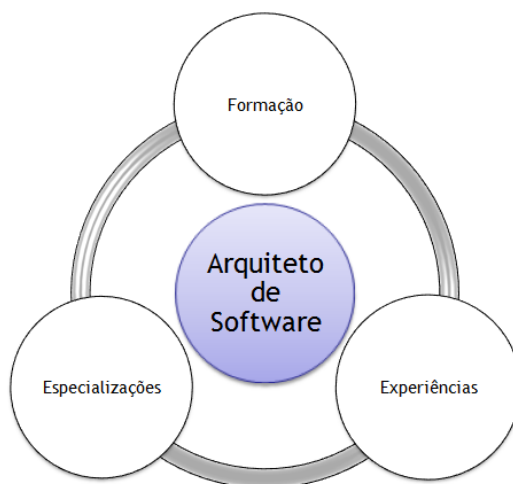
Outros conhecimentos esperados pelo profissional Arquiteto de Software são os conhecimentos relacionados ao Negócio. Análise de Requisitos para que a necessidade do cliente seja traduzida em modelos suficientemente capazes de orientar desenvolvedores a implementarem o sistema. Processos de Desenvolvimento de Software que irá ajudá-lo nos procedimentos de cada fase do projeto. O gerenciamento de riscos, que em sua maioria serão gerenciados pelo Gerente de Projetos, mas o Arquiteto tem a missão de identificar previamente os riscos técnicos associados e criar planos de mitigação. E para que o Arquiteto seja capaz de modelar um sistema e tomar decisões é necessário que ele tenha conhecimentos do negócio que abrange esse sistema.



**Figura 6 - Conhecimento de Negócio**

## Trajetória do Arquiteto de Software

Com base no perfil e papel do Arquiteto de Software, podemos identificar alguns passos que ajudam na trajetória profissional e acadêmica daqueles que buscam construir suas carreiras nesse campo de T.I muito importante e bastante valorizado. A figura abaixo coloca em evidência três características fundamentais no processo de formação de um arquiteto.



**Figura 7 – Trajetória para Formação do Arquiteto de Software**

Uma boa formação acadêmica na área apoia o profissional com conhecimento teórico de fundamentos de computação que são importantes na compreensão de coisas mais avançadas sobre desenvolvimento de software e especificamente arquitetura. A lista abaixo apresenta apenas os cursos mais comuns de graduação e pós-graduação para profissionais de TI, no entanto, existem outras ramificações que são válidas e dependendo do contexto profissional de quem está na trajetória, pode ser até mais interessante.

### **Formação**

#### **Graduação**

- Análise e Desenvolvimento de Sistemas
- Ciência da Computação
- Engenharia da Computação
- Sistemas de Informação

#### **Pós-graduação**

- Arquitetura de Software
- Engenharia de Software

É importante também que o profissional tenha algumas especializações, mas não estamos falando de pós-graduação e sim do conhecimento específico de tecnologias e ferramentas de mercado. Embora o perfil do Arquiteto indique um profissional generalista, sua trajetória exige que em determinados projetos ele seja um especialista, afinal, para projetar boa arquitetura deve-se levar em conta as tecnologias empregadas, fazendo o melhor uso de cada uma delas. Além disso, podemos destacar alguns conhecimentos que devem ser mais profundos para quem busca atuar com Arquitetura de Software. Imagine um

Engenheiro Civil que não conheça bem de padrões e normas de segurança para construção de prédios com mais de dez andares e mesmo assim ele está como responsável pelo projeto. Difícil não é?

Abaixo temos em destaque tipos de tecnologias, ferramentas, conhecimentos específicos e até mesmo algumas certificações para quem deseja uma trajetória em Arquitetura de Software.

## **Especializações**

### **Tecnologias**

- Linguagens
- Banco de dados
- Frameworks Front-end
- Frameworks Back-end

### **Ferramentas**

- IDE's de desenvolvimento
- Gerenciadores de Versão
- Ferramentas de Modelagem
- Ferramentas de Apoio

### **Conhecimentos**

- Arquitetura de Sistemas
- Orientação a Objetos
- Padrões de Projeto
- Integração Contínua
- Análise e otimização
- Gestão de Pessoa

### **Certificações**

- The Open Group IT Architect Certification (ITAC)
- The Open Group TOGAF Certification
- IASA Certified IT Architect (CITA) Program
- The Enterprise Architecture Center of Excellence (EACOE)
- Sun Certified Enterprise Architect (SCEA)

E após uma vasta jornada de investimentos em estudos, ainda é necessário ter experiências que dão a segurança necessária para a tomada de decisões importantes, aplicação do conhecimento adquirido no contexto correto, saber comunicar e fazer com que a equipe trabalhe para manter a arquitetura definida. Claro que, experiência só se adquire com o tempo e muita prática, portanto, na trajetória do arquiteto é importante estar preparado para exercitar isso. Estar preparado nesse caso é ter o conhecimento teórico e procurar oportunidades de colocar a teoria em prova.

## **Experiências**

- Aplicação do conhecimento adquirido
- Vivência em Projetos de Software
- Comunicação e disseminação do conhecimento
- Liderança técnica
- Produção de arquiteturas sólidas

**Resumo**

- ✓ Desenvolvimento de Software é um processo carregado de riscos que podem fracassar um projeto
- ✓ Arquitetura de Software tem o objetivo de virar o jogo aumentando as chances de um projeto ser concluído com sucesso
- ✓ Arquiteto de Software deve ser um profissional versátil e com habilidades interpessoais, conhecimentos técnicos exigidos para o papel, assim como o conhecimento do negócio relacionado ao sistema que será modelado e construído
- ✓ A trajetória para se tornar um Arquiteto de Software indica uma formação básica, especializações em tecnologias e conhecimentos e muita prática em projetos reais

## Capítulo 2 - Requisitos Arquiteturais

---

### Objetivos do curso

- ✓ Definir requisitos funcionais e não-funcionais
- ✓ Mostrar como identificar Requisitos Arquiteturais
- ✓ Apresentar a classificação FURSP+
- ✓ Mostrar alguns exemplos de Requisitos Arquiteturais
- ✓ Mostrar algumas dicas para levantamento de requisitos

### Requisitos funcionais e não-funcionais

---

Antes mesmo de discutirmos o conceito de Requisitos Arquiteturais é importante lembrar o que são Requisitos Funcionais e Não-Funcionais e podemos começar definindo formalmente o que é um Requisito de Software. Segundo o IEEE, Requisitos podem ser considerados como:

- ✓ Uma condição ou uma funcionalidade necessária a um usuário para resolver um problema
- ✓ Uma condição ou funcionalidade que deve ser atingida ou influenciada por um componente de sistema para satisfazer um contrato padrão, especificação, ou outro documento formalmente definido
- ✓ Um requisito de sistema descreve o que é requerido para que o sistema cumpra seu objetivo

**Requisitos funcionais** estão associados às funcionalidades que ditam **o que** o sistema deve fazer. Como exemplos:

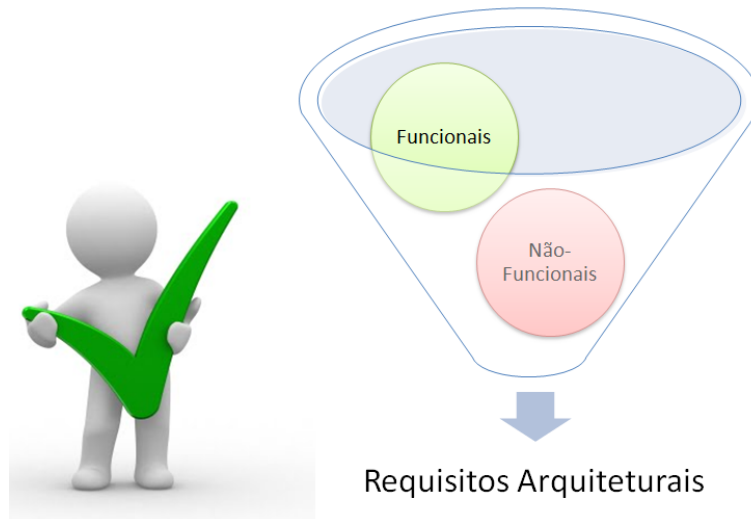
- ✓ O sistema deve possuir um Cadastro de Clientes
- ✓ O sistema deve emitir um Relatório de Contas a Receber
- ✓ O sistema deve permitir o Estorno de Pagamentos

**Requisitos Não-Funcionais** estão associados às restrições de funcionalidades que ditam **como** o sistema deve fazer. Como exemplos:

- ✓ O tempo para cadastrar um cliente não deve exceder 100 milissegundos
- ✓ O sistema deve responder com precisão as coordenadas geográficas das aeronaves
- ✓ O sistema deve possuir uma interface moderna com características de usabilidade, tais como: atalhos para operações frequentes, mensagens indicativas informando ao usuário o tipo de informação esperado para cada campo das telas de cadastro.

Uma importante atividade do Arquiteto de Software e talvez a mais importante, seja o levantamento de requisitos. Nesta fase do projeto, mesmo que não seja possível conhecer todos os requisitos (na maioria dos casos você vai conseguir identificar todos de uma vez e os requisitos têm grandes chances de mudarem ao longo do desenvolvimento), o Arquiteto precisa identificar alguns requisitos para dar início às definições arquiteturais do software a ser modelado e construído.

Além de obter requisitos funcionais e não funcionais, é necessário identificar quais são os Requisitos Arquiteturais do sistema. A figura abaixo mostra uma ideia do trabalho do Arquiteto nessa identificação.



**Figura 8 - Filtro do Arquiteto para Requisitos Arquiteturais**

Os Requisitos Arquiteturais são todos os requisitos, sejam eles Funcionais ou Não-Funcionais que têm impacto direto sobre a Arquitetura do Sistema. Dessa forma, o Arquiteto precisa analisar os requisitos do sistema identificando algumas propriedades e então “filtrando” os Requisitos Arquiteturais. Como auxílio nesse processo de identificação, você pode realizar as seguintes perguntas para os requisitos que já foram levantados:

- ✓ Oferece alto impacto sobre a Arquitetura?
- ✓ Tem escopo abrangente?
- ✓ Oferece alto risco para o negócio?
- ✓ Possui restrições severas?
- ✓ Dita a utilização de alguma tecnologia específica?
- ✓ A implementação do requisito é complexa?
- ✓ Exige a obediência de alguma legislação específica?

Se para alguma dessas perguntas a resposta for “sim”, significa que o requisito tem fortes indícios de ser um Requisito Arquitetural do Sistema. Uma forma de auxiliar o Arquiteto no processo de identificação dos Requisitos Funcionais e Não-Funcionais e extração dos Arquiteturais é a utilização da classificação FURSP+

**FURSP+** é um modelo de classificação de requisitos proposto por *Robert Grady* da *Hewlett Packard (HP)* em seu livro *Practical Software Metrics for Project Management and Process Improvement* (1992). FURSP+ é uma sigla em inglês que faz menção as propriedades e restrições de requisitos de software. Na sequência, são apresentados alguns exemplos de requisitos classificados de acordo com o modelo:

- ✓ **F**unctionality (Funcionalidade)
- ✓ **U**sability (Usabilidade)
- ✓ **R**eliability (Confiabilidade)
- ✓ **P**erformance (Desempenho)
- ✓ **S**upportability (Suportabilidade)
- ✓ **+** => Restrições adicionais importantes

**Functionality** (Funcionalidade):

- ✓ Requisitos mais tradicionais
- ✓ Associados as características principais do sistema

Requisito	Descrição
Vendas	O sistema deve possuir um cadastro de orçamentos com opção para efetivação de venda
Impressão	O sistema deve oferecer opção para impressão de relatório de todas as opções de cadastros disponíveis
Segurança	O sistema deve permitir acesso somente após autenticação de usuários com login e senha
Auditoria	Todas as operações de leitura, inclusão, alteração e exclusão devem ser registradas para fins de auditoria

**Usability** (Usabilidade)

- ✓ Aspectos de Interação entre Usuário e Sistema
- ✓ Cuidado com requisitos subjetivos. Tente obtê-los com descrições objetivas.

Requisito	Descrição
Interface	O sistema deve possuir interface moderna (subjetivo)
Interface	O sistema deve fornecer opções de atalho para proporcionar agilidade e clareza nos trabalhos de rotina do pessoal operacional (objetivo)

### Reliability (Confiabilidade)

- ✓ Disponibilidade do sistema
- ✓ Precisão de resultados

Requisito	Descrição
Disponibilidade	O sistema deverá funcionar em 365x24x7 com no mínimo 99% de disponibilidade
Precisão	O sistema deve oferecer análise de crédito com no mínimo 85% de confiabilidade
Disponibilidade	A operação de simulação de empréstimos deve estar disponível em 99% das tentativas dos usuários

### Performance (Desempenho)

- ✓ Tempo de Resposta
- ✓ Throughput
- ✓ Carga

Requisito	Descrição
Carga e Tempo de Resposta	O sistema deverá suportar no mínimo 20000 usuários conectados simultaneamente no horário de pico e com tempo de resposta da operação de compra em no máximo 500 milissegundos
Throughput	O sistema deve ser capaz de processar no mínimo 2000 transações por segundo



**Supportability** (Suportabilidade)

- ✓ Associados a requisitos não-funcionais:
  - Testabilidade, Adaptabilidade, Configurabilidade, Manutenibilidade

Requisito	Descrição
Testabilidade	As telas do sistema devem suportar automação de testes de interface
Adaptabilidade	O sistema deve ser capaz de adaptar sua interface de acordo com as diversas resoluções de vídeo
Configurabilidade	Todas as taxas de impostos aplicadas sobre transações do sistema devem permitir alteração e aplicação direta sem necessidade de reiniciar o sistema
Manutenibilidade	O processo de restauração de <i>backup</i> deve ser concluído em no máximo 40 minutos

**(+) Restrições adicionais importantes:** Representam algumas restrições sobre o projeto e podem ser classificadas conforme exemplos abaixo:

- ✓ **Restrições ao desenho**
  - "O sistema deve utilizar banco de dados relacional"
- ✓ **Restrições à implementação**
  - "O banco de dados deve ser SQL-SERVER 2008 R2"
- ✓ **Restrições de interface**
  - "Garantir interoperabilidade com outros sistemas por meio de Web Services"
- ✓ **Restrições físicas**
  - "O dispositivo que irá suportar o sistema deve ter tamanho máximo de 100x200 cm e com peso máximo de 600 gramas"

Uma vez que os requisitos foram levantados e devidamente classificados com pelo modelo FURSP+, é possível identificar os Requisitos Arquiteturais verificando suas características conforme as perguntas que foram apresentadas anteriormente. A ideia é identificar requisitos com fortes impactos sobre a arquitetura do software. Abaixo são apresentados alguns exemplos de Requisitos

Funcionais e Não-Funcionais que foram identificados como Requisitos Arquiteturais:

- ✓ O sistema deve oferecer consulta de documentos por linguagem natural **(RF)**
- ✓ A persistência de dados deve ser com banco de dados relacional **(RNF)**
- ✓ O banco de dados deve ser SQL-SERVER 2008 R2 **(RNF)**
- ✓ Disponibilidade 365x24x7 com no mínimo 99% **(RNF)**
- ✓ Todos os relatórios devem ter a opção de exportação para Excel e PDF **(RF)**

O processo de levantamento de Requisitos é de vital importância para o projeto como um todo, e, quando esse trabalho é bem realizado, muitos riscos associados são automaticamente reduzidos. Como sabemos que é praticamente impossível conhecer todos os requisitos do projeto no início, principalmente em projetos gerenciados com metodologias ágeis, é importante ao menos identificar de modo geral quais são os objetivos principais do software e o Arquiteto precisa enxergar amplamente de forma que ele consiga extrair o máximo de informações que serão cruciais para algumas decisões arquiteturais. Na sequência são apresentadas algumas dicas que podem ser úteis no levantamento de requisitos:

- ✓ Sensibilize os envolvidos sobre a importância dessa fase
- ✓ Entreviste os principais interessados no sistema
- ✓ Obtenha descrições objetivas e não subjetivas
- ✓ Crie modelos com o diagrama de Casos de Uso
- ✓ Classifique os Requisitos com FURPS+ e identifique restrições importantes
- ✓ Registre a ordem de prioridade dos Requisitos
- ✓ Analise todos os requisitos observando sua importância, riscos para o negócio, complexidade, baixa estabilidade e extraia os Arquiteturais

## Resumo

- ✓ Requisitos Funcionais dizem o que o sistema deve fazer
- ✓ Requisitos Não-Funcionais dizem como o sistema deve fazer
- ✓ Requisitos Arquiteturais podem ser Funcionais e/ou Não-Funcionais. São aqueles que oferecem alto impacto sobre a estrutura do software
- ✓ A classificação FURSP+ auxilia na classificação, identificação e rastreamento de Requisitos de modo geral e inclusive Arquiteturais

## Capítulo 3 - Riscos e decisões

---

### Objetivos do capítulo

- ✓ Definir formalmente Riscos
- ✓ Apresentar uma Metodologia para Tomada de Decisões
- ✓ Apresentar um Modelo para construção de Planos de Mitigação

### Conceituando riscos

---

Toda e qualquer atividade a ser realizada está sujeita a uma série de riscos em potencial. De modo geral alguns riscos são simples e podem não trazer tantos impactos sobre a atividade, caso eles venham se efetivar. No entanto, algumas vezes existe uma família de riscos sérios e que podem colocar toda a atividade perdida.

No contexto de Desenvolvimento de Software também não é diferente, muito pelo contrário, projetos de sistemas são carregados de uma imensidão de riscos associados e na maioria das vezes esses riscos, quando não conhecidos previamente, pegam toda a equipe de surpresa e causam diversos problemas no projeto e em alguns casos o destino é o fracasso com o consequente cancelamento.



**Figura 9 - Filtro do Arquiteto para Requisitos Arquiteturais**

**Fonte:** [http://www.anunico.com.br/anuncio-de/otros\\_cursos/gestao\\_estrategica\\_do\\_risco-391651.html](http://www.anunico.com.br/anuncio-de/otros_cursos/gestao_estrategica_do_risco-391651.html)

“Sem riscos não há recompensas” (Kontio, 1999). Então o que fazer? Precisamos conviver com os riscos? Como devo agir em determinadas situações? Essas perguntas são um bom começo para que o projeto se organize em conheça os riscos associados. Criar planejamentos para mitigar riscos é uma boa solução para o gerenciamento de riscos. Antes de falarmos de mitigação, vamos definir formalmente o que são riscos.

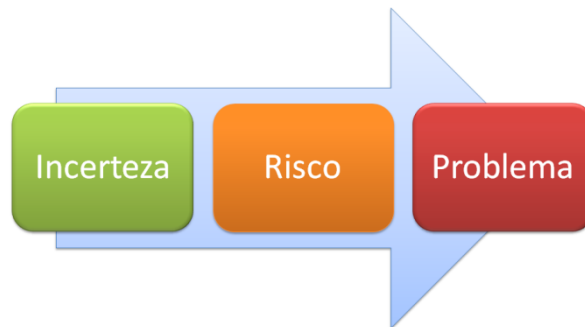
### Definição de Riscos pelo PMBOK, 2004

“Evento ou condição incerta que, se ocorrer, terá um efeito positivo ou negativo sobre pelo menos um objetivo do projeto”

### Definição de Riscos pelo SEI - Software Engineering Institute

“Risco é a possibilidade de sofrer perdas”

Compilando as definições apresentadas sobre riscos, podemos ilustrar através da figura abaixo que o risco nasce quando você tem uma incerteza sobre algo. Por exemplo, se você tem agendada uma viagem a negócios para outro país e você vai de avião, é comum você se preocupar com os atrasos de voos. Se o atraso para a reunião puder lhe ocasionar uma perda ou um efeito negativo sobre seu compromisso, você já tem um risco. A incerteza de que o voo não se atrasará se torna um risco e este risco efetivado gera um ou mais problemas.



**Figura 10 - O surgimento dos Riscos**

Em projetos de desenvolvimento de software, alguns riscos comuns e que são de grande impacto são apresentados conforme exemplos abaixo:

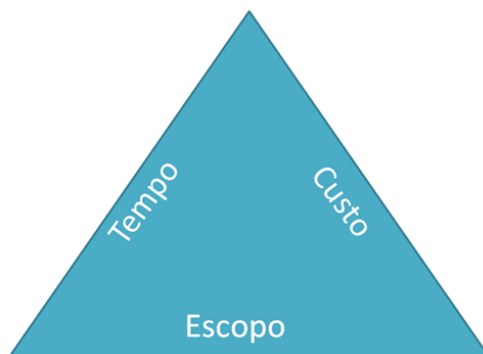
- ✓ Requisitos
  - Falta de definição
  - Requisitos incompletos, mal levantados
- ✓ Tecnologias
  - Defeitos de tecnologias imperfeitas
  - Falta de experiência do time
- ✓ Ideias e conceitos
  - Conhecimento conflitante sobre o negócio
- ✓ Pessoas
  - Perda de pessoal
  - Objetivos diferentes no projeto
- ✓ Prioridades
  - O cliente tende a mudar prioridades atendendo questões estratégicas da organização
  - Isso gera mudança de escopo
- ✓ Planejamentos
  - Cronogramas irreais, otimistas
  - Estimativas falhas
  - Custos descontrolados

## Restrições X Decisões

Todo projeto está sujeito a uma série de riscos e inclusive restrições que limitam recursos para sua execução, restrições legais e etc. Seria muito fácil se não existissem restrições. Se faltassem recursos pessoais era só contratar mais profissionais, se o tempo está acabando é só estender o prazo do projeto e se o dinheiro está ficando curto é só investir mais e tudo fica mais certo. No entanto, não é assim que coisas acontecem. Além disso, as restrições podem ser um

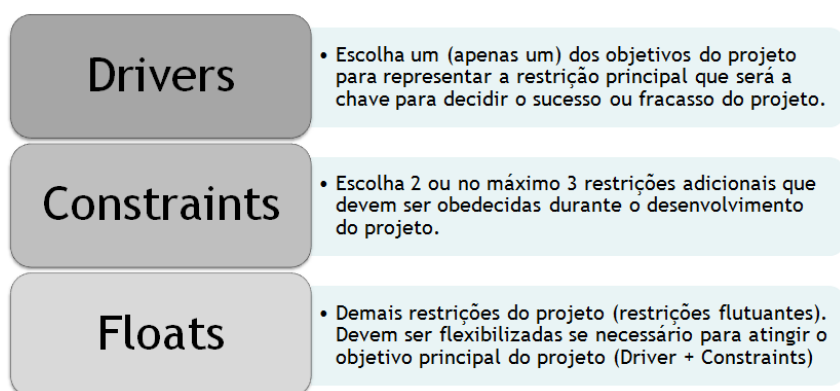
tanto conflitantes entre si. Algumas escolhas devem ser feitas para atender as restrições principais e também para flexibilizar restrições adicionais.

A figura abaixo, conhecida como o Triângulo de Ferro (*The Iron Triangle*) é muito utilizada para descrever restrições comuns em projetos de software, tendo nos seus lados representados pelas restrições de Custo, Tempo e Escopo.



**Figura 11 - Triângulo de Ferro (*The Iron Triangle*)**

O trabalho do Arquiteto de Software é guiar as escolhas de forma que as restrições do projeto sejam compatíveis com o valor de negócio esperado pelo cliente. Uma metodologia sugerida para a tomada de decisões em projetos é apresentada no livro "*Manage It*" escrito por Johanna Rothman, 2007. Esta metodologia é conhecida como "*Drivers, Constraints and Floats*", e é apresentada conforme figura abaixo:



**Figura 12 - *Drivers, Constraints and Floats***

## Mitigação de Riscos

O maior e pior risco é aquele que não é conhecido. Eles pegam os gestores de surpresa exigindo decisões precipitadas e muitas vezes com efeitos negativos sobre o projeto. Riscos conhecidos podem ser mitigados e riscos desconhecidos são dotados de potencial de fracassar e matar um projeto.

Nos últimos anos algumas práticas foram propostas com o objetivo de aumentar a taxa de sucesso de projetos de software. Estas práticas foram desenvolvidas principalmente no contexto de metodologias ágeis. A seguir são listadas as principais ações para mitigação de riscos.


- ✓ Identificar os riscos precocemente
  - Ordene os riscos por impacto e probabilidade

Risco	Impacto	Probabilidade
Atraso na entrega	Alto	Alta
Mudança escopo	Alto	Média
Estimativas erradas	Médio	Alta
Cobertura de código	Alto	Baixa
Perda de pessoal	Médio	Baixa
Mudança de banco de dados	Baixo	Alta
Mudança de prioridade	Baixo	Baixa


- ✓ Divulgar aos principais interessados pelo projeto os dez principais riscos
- ✓ Criar Planos de Mitigação **5W2H**

5W2H	Descrição
<b>What</b>	O que será feito?
<b>Who</b>	Quem vai fazer?
<b>Where</b>	Onde será feito?
<b>Why</b>	Por que será feito?
<b>How</b>	Como será feito?
<b>How much</b>	Quanto vai custar?

Como exemplo da criação de um Plano de Mitigação 5W2H é apresentado abaixo um exemplo onde o cenário propõe que você é o Arquiteto de Software de um projeto que está criando um sistema para auxiliar o atendimento de turistas esperados na próxima copa do mundo que será no Brasil. Você já identificou alguns Riscos associados ao projeto.



O projeto exige implementação com linguagem e tecnologia que o time não tem experiência.



O sistema deve ser entregue para homologação com no mínimo três meses de antecedência da abertura oficial da copa do mundo e só temos 8 meses.

**Figura 13 - Riscos mapeados**

Perceba que o primeiro risco já é claro e provável fonte de problemas, pois o time de desenvolvedores disponível não possui qualquer experiência com a tecnologia exigida pelo projeto. Dessa forma, qual seria o Plano de Mitigação 5W2H para este risco mapeado? A tabela abaixo mostra este plano de ação preenchido.

5W2H	Descrição
What	Vamos contratar um programador com experiência em desenvolvimento de aplicativos para Android
Who	Arquiteto junto ao RH
Where	Recrutamento e Seleção
Why	Para garantir a qualidade e implementação de requisitos complexos
How	Contração imediata
How much	O custo dessa ação é de três semanas para entrevistas e contratação e o tempo dedicado pelo Arquiteto para avaliação técnica.

### Resumo

- ✓ Riscos fazem parte de qualquer projeto, principalmente os complexos
- ✓ Conhecer os riscos antecipadamente pode reduzir os impactos de suas ocorrências
- ✓ Estratégias de mitigação de riscos fazem parte do processo de boa Arquitetura de Software
- ✓ O Plano de Mitigação 5W2H ajuda a criar um planejamento de ações para riscos

## Capítulo 4 - Estimativas

### Objetivos do Capítulo

- ✓ Introdução às estimativas
- ✓ Apresentar técnicas de estimativas

### Introdução as Estimativas

Toda atividade dentro do processo de desenvolvimento de software precisa ser estimada. Através das estimativas é possível construir cronogramas e fornecer uma visão geral a todos os interessados sobre o andamento do projeto. Além disso, a estimativa é uma forma de se calcular o tempo que será gasto e consequentemente o custo. Muitos profissionais serão pagos pela hora de trabalho.

Estimativas são essencialmente previsões de futuro. Espera-se do Arquiteto de Software a capacidade de, junto com o time, dizer com segurança qual será o esforço necessário para a construção do sistema.



**Figura 14 – Estimativas: previsão de futuro**

**Fonte:** <http://orlandobarrozo.blog.br/?p=9364>

Tom DeMarco afirma que “não se consegue controlar aquilo que não se consegue medir”. Além disso, um problema que acontece nas estimativas é que muitas vezes os requisitos não estão amadurecidos o suficiente ou não são conhecidos, cenário típico no início de projetos. Mesmo assim, estimativa precisa acontecer. Existem várias razões para medir e, dentre elas, destacam-se:

- ✓ Melhorar a assertividade das estimativas
- ✓ Construir cronogramas mais precisos
- ✓ Auxiliar gestores na tomada de decisões

Para auxiliar o processo de estimativa, existem algumas técnicas que foram desenvolvidas pela indústria de criação de software ao longo dos anos. Não existe uma técnica mais adequada, o fator que vai determinar a qualidade das estimativas é a experiências dos profissionais, o conhecimento do negócio e principalmente o histórico de atividades similares.



## Estimativa bottom-up

---

Talvez método mais simples e, talvez, mais utilizado de estimativa seja a estimativa Bottom-Up (de baixo para cima). A ideia é dividir o problema em tarefas pequenas, estimar o tempo de cada uma e o somatório é a estimativa final. As vantagens e desvantagens dessas técnicas são apresentadas abaixo.

### **Vantagens**

- ✓ É mais fácil estimar tarefas menores
- ✓ É preciso, pois, utiliza análise detalhada

### **Desvantagens**

- ✓ Em grandes projetos fica complicado gerenciar inúmeras pequenas tarefas
- ✓ Profissionais com pouca experiência podem deixar de fora algumas tarefas
- ✓ Tendência de subestimar a duração de cada tarefa

## Estimativa Delphi

---

Consiste de uma forma mais sofisticada da estimativa Bottom-Up (de baixo para cima). A ideia é dividir o problema em tarefas pequenas, estimar o tempo de cada uma e o somatório é a estimativa final. As vantagens e desvantagens dessas técnicas são apresentadas abaixo.

Nesta estimativa, o Arquiteto ou Gerente de Projetos reúne o time responsável pelo desenvolvimento em uma sala onde é apresentado o escopo do sistema no maior nível de detalhamento possível. Cada membro da equipe escreve sua versão da lista de tarefas Bottom-up. As estimativas são comparadas e o time discute sobre os detalhes percebidos onde cada membro defende seu ponto de vista. Esta discussão normalmente chega a um consenso obtendo-se a lista final de tarefas estimadas. As vantagens e desvantagens dessas técnicas são apresentadas abaixo.

### **Vantagens**

- ✓ Processo pode ocorrer virtualmente
- ✓ Pode focar em detalhes ou no todo do projeto
- ✓ Adequada para obter consenso entre especialistas

### **Desvantagens:**

- ✓ Processo demorado
- ✓ Pode ser necessário estimar antes da existência da equipe
  - Convida-se um grupo de pessoas para ajudar na estimativa
  - Este grupo não vai desenvolver e existe a tendência de subestimar o esforço

- ✓ Nível de experiência heterogêneo, estimativas podem não chegar num consenso
  - Neste caso, o Arquiteto ou Gerente de Projetos deve decidir pela estimativa mais pessimista ou pela estimativa produzida pelos membros mais experientes

## Planning Poker

O Planning Poker é uma ferramenta para o processo de estimativa comumente utilizada em projetos de software com metodologia ágil, tais como *SCRUM* e *XP*. A ideia é apresentar aos participantes da estimativa, normalmente o *Scrum Team* (Desenvolvedores e Arquiteto de Software) as tarefas que precisam ser executadas no *Sprint*.

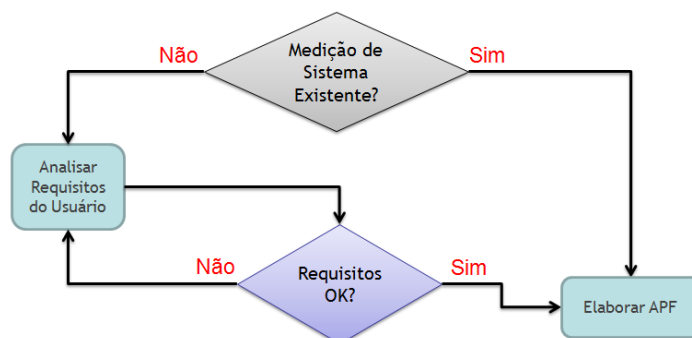
O *Scrum Master* atua como moderador desse rito de estimativa. Ao discutirem as atividades, todos mostram uma carta representando o tamanho da tarefa. Se há discordância entre as estimativas individuais, as partes que estimaram com diferença da maioria devem explicar os motivos da sua escolha.

Por exemplo, imagine uma tarefa "criar um relatório de usuários do sistema". 4 dos 5 membros da equipe estimaram o tamanho da tarefa como sendo "8", no entanto, 1 membro do time estimou "13". Ele deve explicar por que acredita que o tamanho da atividade é "13" e não "8". Se ele conseguir convencer os demais membros que o tamanho ideal é "13" eles chegaram num consenso, caso contrário, o time pode novamente discutir a atividade até chegarem à estimativa final. A figura abaixo mostra um exemplo do baralho Planning Poker com sua sequência de Fibonacci.



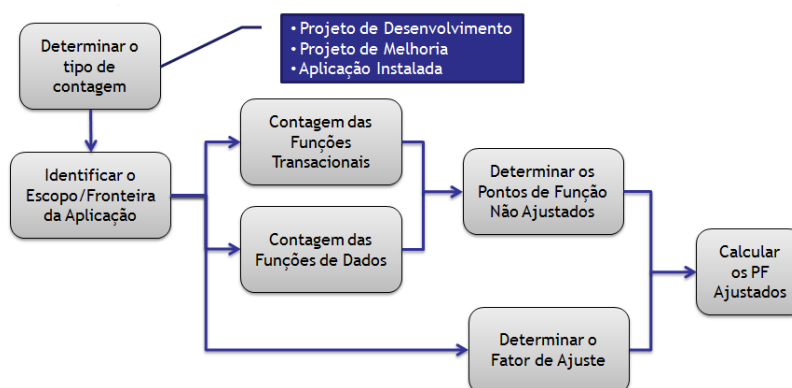
tamanho das funções que serão construídas. É basicamente uma técnica de contagem. Atualmente é regulamentada pelo *IFPUG* (*International Function Points User Group*) e apoiada pelo *BFPUG* (*Brazilian Function Points Group*) na divulgação da técnica.

Uma característica interessante de APF é a possibilidade de efetuar a mediação em qualquer fase do projeto inclusive de sistemas já existentes. A figura abaixo mostra um fluxo de decisão para elaboração da Análise de Pontos de Função.



**Figura 16 - Decisão para elaboração da Análise de Pontos de Função**

O fluxo abaixo representa o processo de elaboração da Análise de Pontos de Função propriamente dita. O primeiro passo é Determinar o tipo de contagem (Projeto de Desenvolvimento, Projeto de Melhoria, Aplicação Instalada). Na sequência, deve ser identificado o Escopo e Fronteira da aplicação para determinar os limites da contagem. Os próximos passos são a Contagem das Funções Transacionais e Funções Dados que serão apresentadas neste capítulo. De posse da contagem têm-se então os Pontos de Função não ajustados. Outro processo é a determinação do fator de ajuste. Este está relacionado diretamente com requisitos não-funcionais do escopo de contagem, no entanto, este processo é pouco utilizado pelo mercado em geral. A conclusão da análise seria o somatório dos Pontos de Função não ajustados com o Fator de Ajuste e o resultado seria os Pontos de Função Ajustados. O forte e o principal objetivo da APF é medir o tamanho funcional do sistema.



**Figura 17 - Fluxo para elaboração da Análise de Pontos de Função**

O processo de Análise de Pontos de Função é bastante complexo e a sua descrição completa pode ser encontrada no site do IFPUG. Daqui a diante vamos apresentar uma visão geral do processo. Abaixo temos os tipos de funções consideradas na contagem da APF.

### ✓ Funções de dados

- Arquivos Lógicos Internos (ALI)
- Arquivos de Interface Externa (AIE)

### ✓ Funções transacionais

- Entradas Externas (EE)
- Saídas Externas (SE)
- Consultas Externas (CE)

Para proceder com a contagem, à metodologia propõe algumas tabelas indicando a complexidade da função e outras tabelas de contribuição com pontos de função para cada função verificada na tabela de complexidade. Como exemplos, abaixo são apresentadas as tabelas de complexidade e contribuição das funções de dados ALI e AIE.

N° de Registros Lógicos	N° de Itens de Dados Referenciados		
	De 1 a 19	De 20 a 50	51 ou mais
Apenas 1	Baixa	Baixa	Média
Entre 2 a 5	Baixa	Média	Alta
6 ou mais	Média	Alta	Alta

**Figura 18 - Tabela de Complexidade - Função de Dados**

A figura abaixo mostra a Tabela de Contribuição de acordo com a complexidade da Função de Dados ALI - Arquivo Lógico Interno. Para cada nível de complexidade da função há um número que representa a quantidade de Pontos de Função correspondente.

Complexidade da Função	Pontos de Função não Ajustados
Baixa	7
Média	10
Alta	15

**Figura 19 - Tabela de Contribuição - ALI (Arquivo Lógico Interno)**

A figura abaixo mostra a Tabela de Contribuição de acordo com a complexidade da Função de Dados AIE - Arquivo de Interface Externa. Para cada nível de complexidade da função há um número que representa a quantidade de Pontos de Função correspondente.

Complexidade da Função	Pontos de Função não Ajustados
Baixa	5
Média	7
Alta	10

**Figura 20 - Tabela de Contribuição - AIE (Arquivo de Interface Externa)**

As figuras abaixo 20 e 21 mostram a Tabela de Complexidade das Funções Transacionais EE - Entradas Externas, SE - Saídas Externas e CE - Consultas Externas. A coluna "Nº de Arquivos Referenciados" indica a quantidade ALI e/ou AIE que são referenciados, e com isso pode ser obtida a complexidade da função verificando também a colunas "Nº de Itens de Dados Referenciados".

Nº de Arquivos Referenciados	Nº de Itens de Dados Referenciados		
	De 1 a 4	De 5 a 15	16 ou mais
0 ou 1	Baixa	Baixa	Média
2	Baixa	Média	Alta
3 ou mais	Média	Alta	Alta

**Figura 21 - Tabela de Complexidade EE (Entradas Externas)**

Nº de Arquivos Referenciados	Nº de Itens de Dados Referenciados		
	De 1 a 5	De 6 a 19	20 ou mais
0 ou 1	Baixa	Baixa	Média
2 ou 3	Baixa	Média	Alta
4 ou mais	Média	Alta	Alta

**Figura 22 - Tabela de Complexidade SE e CE (Saídas Externas e Consultas Externas)**

A figura abaixo mostra a Tabela de Contribuição de acordo com a complexidade da Função de Dados EE - Entradas Externas e CE - Consultas Externas. Para cada nível de complexidade da função há um número que representa a quantidade de Pontos de Função correspondente.

Complexidade da Função	Pontos de Função não Ajustados
Baixa	3
Média	4
Alta	6

**Figura 23 - Tabela de Contribuição EE e CE (Entradas Externas e Consultas Externas)**

A figura abaixo mostra a Tabela de Contribuição de acordo com a complexidade da Função de Dados SE - Saídas Externas. Para cada nível de complexidade da função há um número que representa a quantidade de Pontos de Função correspondente.

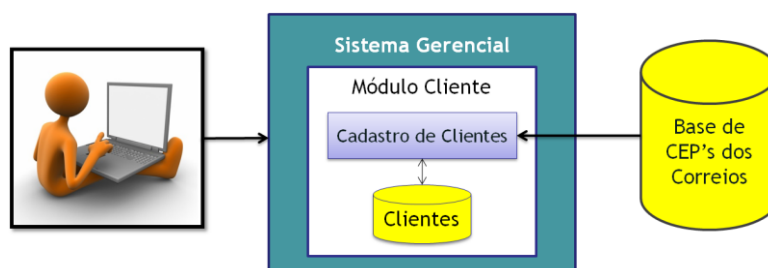
Complexidade da Função	Pontos de Função não Ajustados
Baixa	4
Média	5
Alta	7

**Figura 24 - Tabela de Contribuição SE (Saídas Externas)**

Para exemplificar o uso da contagem e Pontos de Função, vamos para um exemplo prático. Você foi contratado para realizar a Análise de Pontos de Função de um requisito de Cadastro de Clientes com as seguintes funções:

- ✓ Incluir, alterar, excluir e consultar
  - O CEP será consultado no Banco de Dados dos Correios
- ✓ Permitir Impressão dos seguintes Relatórios
  - Relação de todos os clientes
  - Ficha de Cliente constando o total de todas as compras em R\$

Você fez um estudo e identificou o escopo de contagem. Você agora conhece a Fronteira da Aplicação e elaborou o seguinte desenho:



**Figura 25 - Fronteira da Aplicação - Módulo Cliente**

Você determinou o tipo de contagem, a fronteira da aplicação e as funcionalidades que serão contadas. Consultando nas tabelas de complexidade e contribuição para cada função você identificou as seguintes informações:

- ✓ **Tipo de Contagem:** Projeto de Desenvolvimento
- ✓ **Escopo e Fronteira:** Módulo de Clientes
- ✓ **Funcionalidades:**
  - **ALI:** Tabela de Clientes = 7PF
  - **AIE:** Base de CEP's (Correios) = 5PF
  - **EE:** Incluir Cliente = 4PF
  - **EE:** Alterar Cliente = 4PF
  - **EE:** Excluir Cliente = 4PF
  - **CE:** Consultar Cliente = 4PF
  - **CE:** Relação de Clientes = 4PF
  - **SE:** Ficha do Cliente com total de compras = 5PF

Tamanho do Projeto:  
**37PF**

Pronto! Agora você sabe qual é o tamanho do projeto funcional do ponto de vista do usuário (37PF). Dessa forma, se você tiver informações sobre a produtividade do time que irá desenvolver, você saberá quanto tempo será necessário para entregar a funcionalidade para o cliente. Se você sabe quando o cliente cobra por Ponto de Função, você também saberá quanto o projeto vai custar.

✓ **Considere que:**

- Sua equipe é composta por três desenvolvedores experientes
- Eles gastam em média 8 horas para construir 1 PF, logo:
- Seu time constrói três PF em um dia de trabalho
- Sua empresa cobra R\$ 250,00 por Ponto de Função

✓ **Estimativa (Custo e Prazo)**

- Custo do Projeto =  $(37 \text{ PF} * \text{R\$ } 250,00) = \text{R\$ } 9.250,00$
- Prazo de Entrega =  $(37 \text{ PF} / (3 \text{ PF/dia})) = 12,33 \text{ dias}$

Afinal, quais são as vantagens e desvantagens da Análise de Pontos de Função?

✓ **Vantagens**

- Pode ser utilizada em qualquer fase do projeto
- Independente de Tecnologia
- Permite medir unidades de um produto de software
- Calcula custos e recursos necessários para o desenvolvimento/manutenção

✓ **Desvantagens**

- O processo é complexo
- Exige detalhamento dos requisitos em baixo nível (entender a solução)
- A contagem não é facilmente automatizada
- Esforço da medição costuma ser alto
- Pouco detalhista nos Requisitos Não Funcionais

**Resumo**

- ✓ Estimativas são previsões, mas podem ser assertivas se o time tem experiência
- ✓ Não existe uma técnica de estimativa melhor ou pior, é questão de experiência e dados históricos para guiar o processo
- ✓ Análise de Pontos de Função mede o tamanho funcional do software do ponto de vista do usuário

## Capítulo 5 - Modelagem e Modelos

---

### Objetivos do Capítulo

- ✓ Mostrar uma visão geral dos diagramas estruturais e comportamentais da UML
- ✓ Apresentar os conceitos de BPMN e SoaML
- ✓ Mostrar uma visão geral e um exemplo de Modelagem de Requisitos
- ✓ Mostrar uma visão geral da Modelagem de Domínio

### Introdução à Modelagem e Modelos

---

O processo de modelagem consiste do entendimento do negócio e da representação desse conhecimento através de modelos em diagramas e por descrições textuais indicando as decisões arquiteturais do sistema que será construído. Este é o momento onde o Arquiteto de Software possui maior influência no projeto devido à sua experiência e conhecimento técnico.

Nas últimas décadas alguns princípios arquiteturais para Modelagem e Desenho de sistemas surgiram no mercado a partir de experiências em projetos de alta complexidade. Estes princípios devem ser levados em conta nos diversos níveis do projeto, mas possuem especial relevância para a fase de Modelagem.

### Separação de interesses

Um dos principais conceitos da Engenharia de Software moderna e foi introduzido originalmente por Dijkstra em 1974 e refere-se ao fato de que para atacar os diversos aspectos de um sistema complexo é necessário isolar os diversos aspectos (interesses) e estudá-los de forma separada.

O principal objetivo deste princípio é dividir um sistema complexo em unidades que possuam as características de:

- ✓ Alta coesão:
  - Cada unidade/módulo/componente do sistema deve ser responsável unicamente por um conceito chave.
- ✓ Baixo acoplamento:
  - Unidades do sistema devem ser o máximo independente possível. Obviamente, elas precisam trocar mensagem de alguma forma para que o sistema funcione. A ideia não é eliminar todo o acoplamento, mas acoplar somente o necessário para que a unidade consiga dar prosseguimento na sua execução dentro do sistema.

### Modelar apenas o suficiente

Não será necessário criar todos os modelos com todos os níveis de detalhes possíveis. Muitos projetos possuem modelos muito pouco detalhados, sobre os quais não é possível tirar muitas conclusões. Outros são extremamente detalhados e consomem uma enorme quantidade de esforço que não será aproveitado.

A ideia do conceito de Modelar apenas o suficiente é a de que um Modelo possui um valor e também um custo de criação. O valor gerado por um modelo é de acelerar o processo de entendimento do sistema. O custo de criação dos modelos é atribuído ao esforço em homens-hora para sua produção. Quando o custo de criação de um modelo é superior ao valor gerado por ele, significa que



o modelo está exageradamente detalhado. Modelar apenas o suficiente não pode ser confundido com modelar com baixa qualidade. A relação tem que ser melhor custo x benefício.

## UML - Unified Modeling Language

A UML (Unified Modelling Language) é uma linguagem de modelagem que foi lançada pela OMG (Object Management Group) em 1997 e sua versão atual é a 2.2. OMG é uma organização sem fins lucrativos que possui diversos representantes de diversas áreas de indústria de software e que tem como objetivo a criação de padrões de integração de sistemas corporativos. A linguagem UML foi criada com a ideia de quatro objetivos básicos:

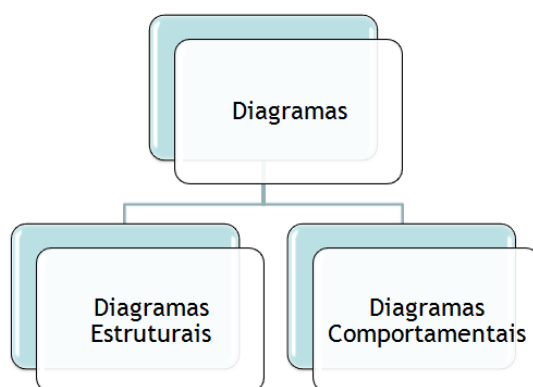
**Visualização:** Permitir a descrição visual de modelos com uma semântica precisa.

**Especificação:** Permitir a criação de modelos precisos, não ambíguos e completos que capturam todas as decisões de Análise, Desenho e Implementação.

**Construção:** Modelos podem ser diretamente mapeados para elementos de linguagens de programação e vice-versa.

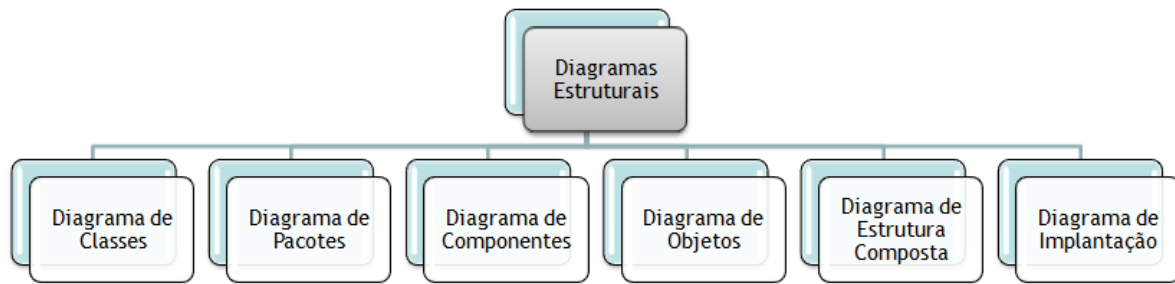
**Documentação:** Diagramas capturam todas as informações coletadas pela equipe, permitindo o compartilhamento do conhecimento.

Os diagramas propostos pela UML são divididos em dois grupos: os diagramas estruturais e os diagramas comportamentais.



**Figura 26 - Tipos de Diagramas da UML**

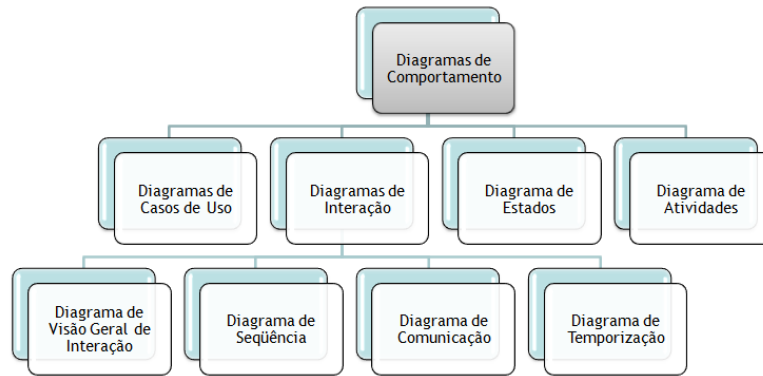
A figura abaixo mostra o grupo de Diagramas Estruturais da UML que são responsáveis pela modelagem do sistema em nível de estrutura e como os elementos que compõe esta estrutura se relacionam.



**Figura 27 - Diagramas Estruturais da UML**

- ✓ **Diagramas Estruturais => Diagrama de Classes**
  - Mostra o conjunto de classes do sistema com seus atributos, métodos e relacionamentos
- ✓ **Diagramas Estruturais => Diagrama de Pacotes**
  - Mostra o conjunto de pacotes do sistema e suas dependências
  - Utilizado para representar conjunto de classes em pacotes
- ✓ **Diagramas Estruturais => Diagrama de Componentes**
  - Permite a modelagem física de um sistema
  - Fornece uma visão dos componentes e seus relacionamentos
- ✓ **Diagramas Estruturais => Diagrama de Estrutura Composta**
  - Mostra a estrutura de um elemento (classes, pacotes, componentes)
  - Útil para modelagem de padrões de projeto (SILVA, 2007)
    - Padrões estruturais de desenho
- ✓ **Diagramas Estruturais => Diagrama de Implantação**
  - Mostra a distribuição de hardware do sistema
    - Identificando servidores como nós do diagrama
    - Os componentes de software são mapeados nesses nós
  - Mostra uma visão da infraestrutura necessária para implantação
    - Servidores
    - Redes
    - Bancos de Dados
    - Componentes

A figura abaixo mostra o grupo de Diagramas Comportamentais da UML que são responsáveis pela modelagem do sistema em nível de comportamento os elementos estruturais do sistema se comportam em execução.



**Figura 28 - Diagramas Comportamentais da UML**

- ✓ **Diagramas Comportamentais => Diagrama de Casos de Uso**
  - Principal Diagrama da UML
  - Modela o comportamento do sistema
    - Descreve o sistema, seu ambiente e a relação entre os dois
    - Utilizados para obter requisitos funcionais a partir da perspectiva do usuário
    - Descreve o que fazer e não como fazer
  - Usuários compreendem o diagrama com facilidade
- ✓ **Diagramas comportamentais => Diagramas de Interação**
  - Mostram como os objetos interagem uns com os outros
    - Modela aspectos dinâmicos do sistema
  - São divididos em quatro tipos de diagramas
    - Diagrama de Sequência
      - Mostra a interação num conjunto de objetos e seus relacionamentos em um único caso de uso
      - Ênfase na ordem temporal das mensagens
    - Diagrama de Comunicação
      - Mostra como os objetos do sistema interagem sobre múltiplos casos de uso
    - Diagrama de Temporização
      - Especializado para sistemas de tempo real
    - Diagrama de Visão Geral de Interação
      - Descreve em alto nível o fluxo principal das interações
- ✓ **Diagramas comportamentais => Diagrama de estados**
  - Especifica aspectos dinâmicos do sistema através de Máquinas de Estados
    - São aplicadas em classes, casos de uso ou até mesmo no sistema todo
- ✓ **Diagramas Comportamentais => Diagrama de atividades**
  - Utilizados para modelar fluxos
    - Processos
    - Processos de Negócio
    - Operações Internas

- Uma atividade representa uma operação de classe que provoca uma mudança no estado do sistema

## BPMN e SoaML

Como vimos na seção anterior, existem diversos diagramas que são úteis no processo de modelagem do projeto de um sistema. Nesta seção vamos apresentar uma visão geral de mais duas ferramentas, uma para modelagem de processos de negócios (BPMN) e outra para modelagem de sistema cuja arquitetura é orientada a serviços (SoaML).

### BPMI – Business Process Management Initiative

Trata-se de uma organização independente que cuida do desenvolvimento de especificações abertas para o gerenciamento de processos empresariais. Em 2005 o Business Process Management Initiative e o Object Management Group anunciaram sua junção. O BPMI desenvolveu três padrões: O BPMN (Business Process Modeling Notation): como um padrão para modelar processos do negócio, BPML (Business Process Modeling Language): como a linguagem padrão de desenvolvimento, BPQL (Business Process Query Language): como uma interface padrão de manutenção para a distribuição e a execução de processos e-Business. Porém, vamos nos concentrar apenas no padrão BPMN que é mais comum e normalmente aparece nos processos de desenvolvimento de software.

### BPMN

Fornece uma notação para expressar os processos de negócio por meio de um diagrama (Business Process Diagram – BPD). Observe a figura abaixo que mostra uma visão do que a UML e BPMN oferecerem para modelagem.

UML	BPMN	
Atividade	Processo	Atividades genéricas, passíveis de decomposição funcional arbitrária na criação do modelo.
	Sub-processo	
Ação	Tarefa	Atividade de nível de folha. Não mais se decompõe.




**Figura 29 – UML versus BPMN**

**Fonte:** <https://iblogdomoa.wordpress.com/2011/09/14/o-ponto-de-contato-entre-bpmn-e-uml/>


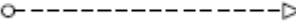

### Simbologia BPMN

Abaixo temos quatro categorias básicas de elementos com uma tabela explicando cada componente.



## Objetos de Fluxo

Objeto	Descrição	Figura
Evento	É algo que acontece durante um processo do negócio. Estes eventos afetam o fluxo do processo e têm geralmente uma causa (trigger) ou um impacto (result). Há três tipos de eventos, baseados sobre quando afetam o fluxo: Start, Intermediate e End	
Atividade	É um termo genérico para um trabalho executado. Os tipos de atividades são: Tarefas e sub-processos. O sub-processo é distinguido por uma pequena cruz no centro inferior da figura.	
Gateway	É usado para controlar a divergência e a convergência da sequência de um fluxo. Assim, determinará decisões tradicionais, como juntar ou dividir trajetos.	



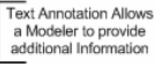
## Objetos de Conexão

Objeto	Descrição	Figura
Fluxo de sequência	É usado para mostrar a ordem (sequência) com que as atividades serão executadas em um processo	
Fluxo de mensagem	É usado para mostrar o fluxo das mensagens entre dois participantes diferentes que os emitem e recebem.	
Associação	É usada para associar dados, texto, e outros artefatos com os objetos de fluxo. As associações são usadas para mostrar as entradas e as saídas das atividades	

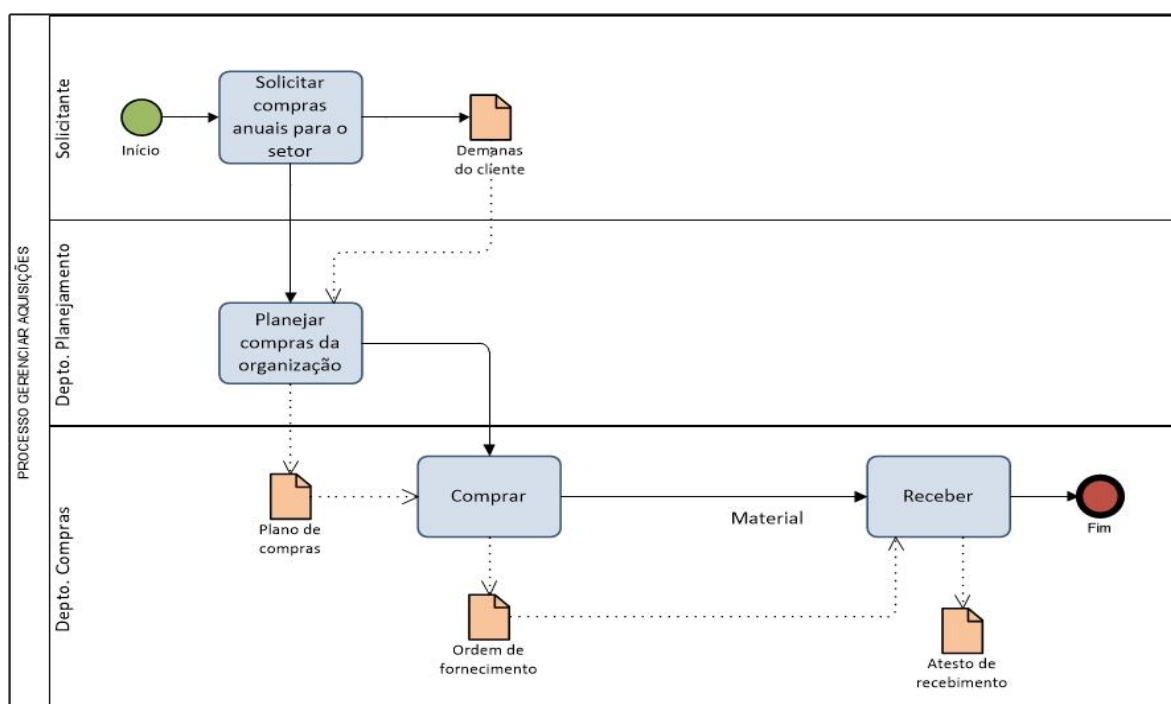
**Swinlanes:** Servem basicamente para organizar e separar as atividades em categorias.

Objeto	Descrição	Figura
Pool	Um <i>pool</i> representa um participante em um processo. Ele atua como um container gráfico para dividir um conjunto de atividades de outros <i>pools</i> , geralmente no contexto de situações de B2B.	
Lane	Uma <i>lane</i> é uma subdivisão dentro de um <i>pool</i> usado para organizar e categorizar as atividades.	

**Artefatos:** Indica entradas e saídas geradas pelas atividades de um processo

Objeto	Descrição	Figura
Objetos de dados	O objeto de dado é um mecanismo para mostrar como os dados são requeridos ou produzidos por atividades. São conectados às atividades com as associações.	
Grupo	Um grupo é representado por um retângulo e pode ser usado para finalidades de documentação ou de análise.	
Anotações	As anotações são mecanismos para fornecer informações adicionais para o leitor de um diagrama BPMN	

Na figura abaixo temos um BPD – Business Process Diagram que demonstra num exemplo simples a modelagem de um processo de aquisição de materiais de uma organização.



**Figura 30 – BPD – Business Process Diagram**

**Fonte:** <http://www.tiespecialistas.com.br/2013/01/modelagem-de-processos-notacoes/>

## Ferramentas BPMN

	ADONIS	A principal tarefa de ADONIS é melhorar continuamente Processo e O Desempenho Empresarial nas organizações através da construção de um sistema integrado de gestão e prestação de informações transparentes para a tomada de decisão.
	ARIS Express	A principal tarefa de ADONIS é melhorar continuamente Processo e O Desempenho Empresarial nas organizações através da construção de um sistema integrado de gestão e prestação de informações transparentes para a tomada de decisão.
	BizAgi	Bizagi Business Process Management (BPM) soluções torna a modelagem, execução e melhoria dos processos de negócio mais fácil para todos, não importa se você é uma pequena organização ou uma grande corporação.
	Visio	MS Visio é para você que está capturando os processos de negócios existentes, analisando a sua cadeia de fornecimento, ou monitorando o desempenho do processo de negócio.
	LucidChart	LucidChart é uma aplicação on-line de diagramação que suporta fluxogramas, BPMN, UML, ERD e muitos outros tipos de diagramas. Ele também vem com colaboração em tempo real, sem emenda - Visio importação / exportação e fácil integração com Google Apps.

**Figura 31 – Ferramentas BPMN**

**Fonte:** <http://www.pmgacademy.com/pt/biblioteca-digital/ferramentas/ferramentas-para-gerenciamento-de-processos-de-negocio-bpm?limitstart=0>

## SoaML

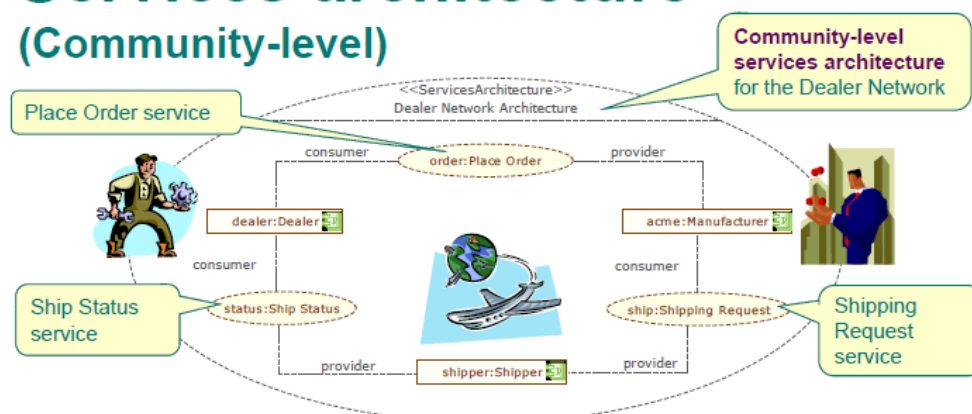
Define um perfil (profile UML) de estereótipos para modelar serviços do ponto de vista do arquiteto.

### Profile UML permite a representação de

- Provedores e consumidores de serviço
- Mensagens trocadas entre eles
- Interfaces dos serviços
- Orquestração e coreografia de serviços
- Contratos e políticas entre provedores e consumidores
- Arquiteturas de serviços

No exemplo abaixo temos um diagrama utilizando o estereótipo <<ServicesArchitecture>> modelando a arquitetura geral em alto nível de como os participantes trabalham juntos para o propósito de prestação e utilização serviços expressos.

## Services architecture (Community-level)



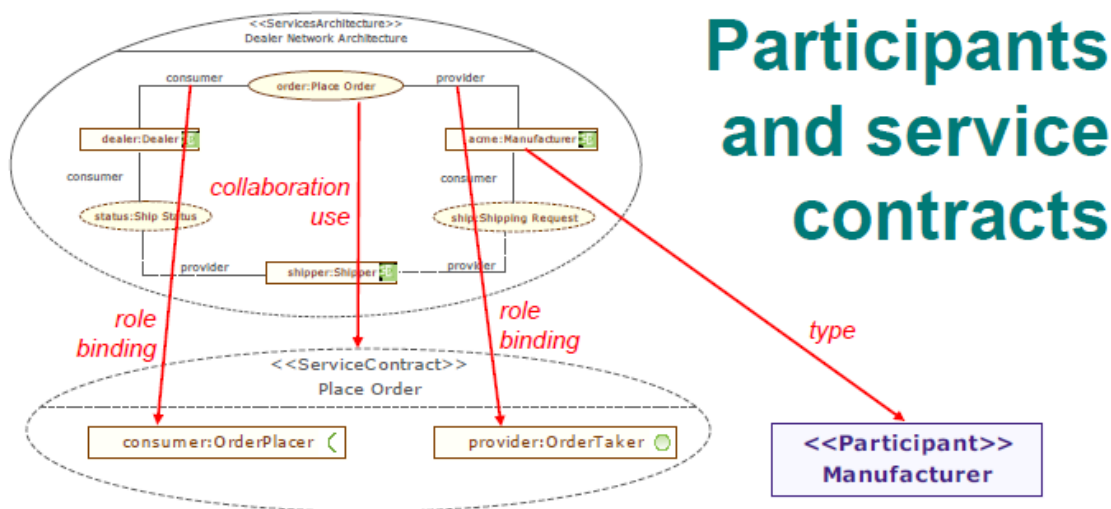
### Services architecture:

- High level description of how **participants** work together for a purpose by providing and using services expressed as **service contracts**.
- UML collaboration stereotyped «ServicesArchitecture».

**Figura 32 – Diagrama SoaML**



Abaixo podemos identificar o uso do estereótipo <<ServiceContract>> onde temos a especificação do serviço Place order (Pedido de Encomenda) definindo os papéis que cada participante desempenha e a interface de uso do serviço.



### Service contract:

- Service specifications that define the **roles** each participant plays in the service and the **interfaces** they implement to play that role.
- UML collaboration stereotyped «ServiceContract».

### Participant:

- Represent logical or real people or organizational units that participate in services architectures and/or business processes.
- UML class stereotyped «Participant».

**Figura 33 – Diagrama SoaML**  
Fonte: SoaML Tutorial, SSAIE 2010

## Ferramentas SoaML

Ferramenta	Fabricante	Permite geração de código a partir de modelos?	Gratuita?
Modelio Designer (Free) Edition	Softteam	Não	Sim
Modelio Enterprise Edition	Softteam	Sim	Não
Magic Draw (com CameoSOA)	No Magic	Não	Não
Magic Draw com Model Pro Gold (com CameoSOA, ModelPro e profiles)	Respectivamente, No Magic e ModelDriven.org	Sim	Não
Rational Software Architect	IBM	Sim	Não
Rational Software Modeler	IBM	Sim	Não
Sparx Systems' Enterprise Architect	Sparx System's	Sim	Não

**Figura 34 – Ferramentas SoaML**  
Fonte: SoaML Tutorial, SSAIE 2010

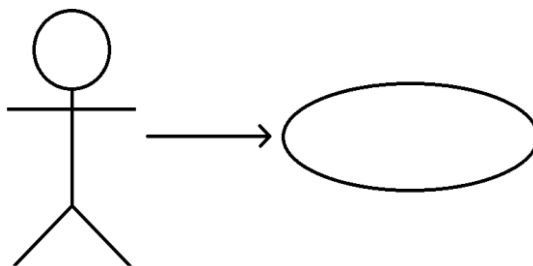


## Modelagem de Requisitos

A modelagem de requisitos é o primeiro e mais importante passo no processo de modelagem de sistemas. Conhecer os requisitos e descrevê-los de forma clara e completa é fundamental para o sucesso das fases posteriores que resultarão na criação do software.

A maior dificuldade dessa fase é que a maioria dos requisitos não estará bem definida no início do projeto e isso é fator importante de risco. As metodologias de desenvolvimento tradicionais acreditam que é possível levantar todos os requisitos em uma fase. Já as metodologias de desenvolvimento ágeis aceitam que os requisitos mudam. Nesse tipo de processo de desenvolvimento, os requisitos são capturados para promover uma estimativa e dar início ao primeiro ciclo de construção e entregas. A especificação de requisitos, neste caso, é tratada como um artefato que deve ser apenas bom o suficiente para iniciar a codificação.

A linguagem UML pode ser utilizada para modelar requisitos em qualquer tipo de metodologia de desenvolvimento de software. Os diagramas podem ser complementados de descrições textuais com informações adicionais.



**Figura 35 - Ator versus Caso de Uso**

Uma boa Modelagem de Requisitos deve ser capaz de mostrar as conexões entre pessoas, conceitos e objetos. Mostrar uma visão global em alto nível. Descrever as funções que abrangem o requisito, pessoas e objetos físicos para o entendimento do analista.

Através de Diagramas de Casos de Uso a modelagem de requisitos pode ser construída. Os diagramas podem ser em formatos de gráficos e seguidos da versão descritiva é mais completa. Os principais componentes de um diagrama de caso de uso são: Caso de Uso (especifica uma funcionalidade do sistema), Ator (entidade externa que interage com os casos de uso) e Sistema (conjunto de casos de uso).

### Como identificar um Ator?

- É quem utiliza a funcionalidade!
- É quem vai precisar de suporte do sistema para realizar o trabalho!
- É quem precisa manter ou administrar o sistema!
- São equipamentos que irão manipular alguma função do sistema!
- São outros sistemas que vão interagir com o sistema que está sendo modelado!

### Como identificar um caso de uso?

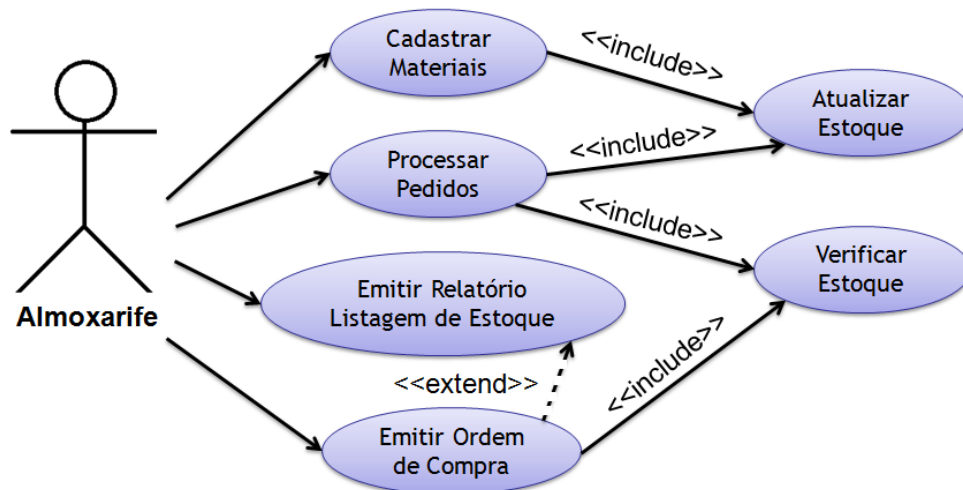
- É o que o ator precisa fazer no sistema!
- São funcionalidades que serão utilizadas por um ator!
- São funcionalidades que precisam chamar outros sistemas!

Vamos exercitar a Modelagem de Requisitos através de um exemplo parecido com situações reais que serão encontradas nas organizações. Você é o Arquiteto e deve modelar o seguinte requisito:

### Requisito RF001:

João trabalha no setor de almoxarifado e tem a função de cadastrar materiais de consumo da empresa. Além disso, ele atende aos pedidos dos funcionários quando eles precisam de algum material de escritório. Uma vez por mês ele repõe alguns itens e para isso ele precisa emitir um relatório constando da quantidade em estoque de cada material cadastrado.

Com base na estória do João que trabalha no almoxarifado, você criou o seguinte Diagrama de Caso de Uso:



**Figura 36 - Diagrama de Caso de Uso - Almoxarifado**

Para especificar de forma mais completa o requisito acima, você complementou a modelagem com o modelo descritivo de Caso de Uso. Este modelo possui algumas propriedades que devem ser preenchidas indicando todas as características importantes do caso de uso para que ele seja construído.

- ✓ Identificador
  - Código único para referenciar o requisito na documentação do sistema
- ✓ Descrição
  - Breve resumo sobre o funcionamento e objetivo do caso de uso
- ✓ Importância
  - Pode ser uma classificação que diz a importância do caso de uso para o sistema (Baixa, Média, Alta)

- ✓ Ator Principal
  - Entidade externa (usuário, sistema, equipamentos) que irá disparar a execução do caso de uso
- ✓ Atores Secundários
  - Outras entidades externas que também podem iniciar o caso de uso
- ✓ Pré-Condições
  - Condições que devem ser satisfeitas antes de iniciar o caso de uso
- ✓ Fluxo Principal
  - Sequência de ações realizadas pelo ator e/ou sistema de forma a concluir com sucesso o caso de uso
- ✓ Fluxos Alternativos
  - Sequência de ações alternativas ações realizadas pelo ator e/ou sistema que podem também levar o caso de uso ao fim normal
- ✓ Fluxos de Exceção
  - Sequência de ações realizadas pelo ator e/ou sistema que levam o caso de uso para um estado de interrupção.
- ✓ Pós-Condições
  - Estado que o sistema deve apresentar após o fim normal do caso de uso
- ✓ Regras de Negócio
  - Regras referentes ao negócio que são relevantes para a implementação
- ✓ Notas de Implementação
  - Recomendações de implementação específicas do caso uso

Caso de Uso - Cadastrar Materiais	
Identificador	RF001
Descrição	Permitir ao Almojarife controlar os Materiais com as opções de inclusão, alteração, exclusão, consulta e atualização do estoque
Importância	Baixa
Ator Principal	Almojarife
Atores Secundários	Administrador do Sistema
Pré-condições	O sistema deve estar totalmente configurado; ID de usuário e senha devem ser obtidos
Fluxo Principal	<ol style="list-style-type: none"> <li>1. O almojarife efetua login no sistema</li> <li>2. O almojarife entra com seu ID de usuário</li> <li>3. O almojarife entra com sua senha</li> <li>4. O sistema carrega as funcionalidades de acordo com as permissões do usuário</li> <li>5. O almojarife seleciona "Cadastrar Materiais"</li> <li>6. O sistema apresenta a tela de cadastro</li> <li>7. O almojarife preenche todos os campos obrigatórios e seleciona "Gravar"</li> <li>8. O sistema exibe uma mensagem de confirmação indicando sucesso na operação</li> </ol>

**Figura 37 - Modelo Descritivo do Caso de Uso (Parte 1) - Almojarifado**

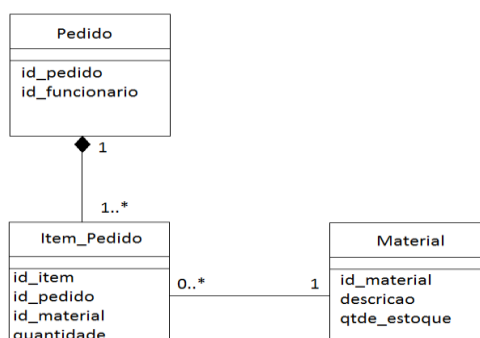
Caso de Uso - Cadastrar Materiais	
Fluxos Alternativos	<ol style="list-style-type: none"> <li>1. O almoxarife efetua login no sistema através de acesso biométrico</li> <li>2. O sistema carrega as funcionalidades de acordo com as permissões do usuário</li> <li>3. O almoxarife seleciona “Cadastrar Materiais=&gt;Importar dados via arquivo txt”</li> <li>4. O sistema apresenta uma tela solicitando que o usuário informe o diretório do arquivo</li> <li>5. O almoxarife seleciona a opção “Confirmar Importação”</li> <li>6. O sistema exibe uma mensagem de confirmação indicando sucesso na operação</li> </ol>
Fluxos de Exceção	<ol style="list-style-type: none"> <li>1. Se o usuário errar o procedimento de login por três vezes ele deve ser bloqueado</li> <li>2. O sistema exibe uma mensagem avisando o bloqueio</li> <li>3. O sistema exibe informações para contato com o Administrador do sistema</li> </ol>
Pós-condições	Após o fim normal do caso de uso o sistema deve exibir uma mensagem de sucesso e ter cadastrado o material, assim como ter atualizado a quantidade em estoque do mesmo
Regras de Negócio	RN1 - O código do material deve ser o mesmo do seu código de barras RN2 - A descrição do material deve ser preenchida conforme catálogo do fornecedor
Notas de Implementação	- Utilizar componente de importação de arquivos - Utilizar herança da classe genérica para cadastros básicos

**Figura 38 - Modelo Descritivo do Caso de Uso (Parte 2) - Almoxarifado**

## Modelagem de Domínio

O Modelo de Domínio é um modelo conceitual que descreve as diversas entidades presentes em um sistema e seus respectivos relacionamentos. Este artefato é normalmente criado em paralelo ou após a especificação das Estórias ou Casos de Uso. O Modelo de Domínio mostra como o sistema é estruturado em um conjunto de abstrações. Neste modelo estão representadas todas as entidades persistentes de um sistema, seus atributos e os relacionamentos entre elas. Ele é utilizado para auxiliar na validação dos requisitos funcionais. Tem também a característica de promover o entendimento e divulgação de conhecimento do sistema.

A figura a seguir mostra um Diagrama de Domínio simples representando o requisito funcional RF001 do Sistema de Gestão de Almoxarifado apresentado na seção anterior.



**Figura 39 - Modelo de Domínio - Sistema de Gestão de Almoxarifado**

Na fase de desenho é importante criar um Modelo de Domínio completo e que represente o conhecimento necessário para a implementação do negócio. O Modelo de Domínio é uma primeira versão do Diagrama de Classes e seu objetivo é identificar classes conceituais que representam de forma abstrata objetos do mundo real. Uma estratégia para identificar classes conceituais é analisar as descrições dos requisitos na procura por substantivos ou frases que estão no lugar de substantivos.

## Requisito RF001

João trabalha no setor de almoxarifado e tem a função de cadastrar **materiais** de consumo da empresa. Além disso, ele atende aos **pedidos** dos funcionários quando eles precisam de algum **material de escritório**. Uma vez por mês ele repõe alguns itens e para isso ele precisa emitir um relatório constando da quantidade em estoque de cada material cadastrado.

Analisando a descrição do requisito acima, podemos identificar algumas classes conceituais através da seleção de substantivos:

### ✓ Materiais

Material
id_material
descricao
qtde_estoque

### ✓ Pedidos

Pedido
id_pedido
id_funcionario

### ✓ Material de Escritório

Item_Pedido
id_item
id_pedido
id_material
quantidade

## Resumo

- ✓ A modelagem de requisitos é o primeiro e mais importante passo no processo de modelagem de sistemas
- ✓ Existem outras ferramentas específicas para modelagem de processos (BPMN) e arquiteturas orientadas a serviços (SoaML)
- ✓ Uma boa Modelagem de Requisitos deve ser capaz de mostrar as conexões entre pessoas, conceitos e objetos. Mostrar uma visão global em alto nível. Descrever as funções que abrangem o requisito, pessoas e objetos físicos para o entendimento do analista
- ✓ O Modelo de Domínio é um modelo conceitual que descreve as diversas entidades presentes em um sistema e seus respectivos relacionamentos

## Capítulo 6 - Estilos Arquiteturais

---

### Objetivos do Capítulo

- ✓ Introduzir o conceito de Estilos Arquiteturais
- ✓ Apresentar vantagens na utilização de Estilos Arquiteturais
- ✓ Mostrar uma visão geral dos estilos Arquiteturais existentes

### Introdução aos Estilos Arquiteturais

---

Um Estilo Arquitetural consiste de um tipo de padrão que determina restrições no nível de arquitetura onde essas restrições são aplicadas em elementos estruturais, tais como, componentes e módulos. Um estilo arquitetural define um conjunto de elementos que devem ser utilizados e como eles devem se comportar na estrutura definida. Módulos, componentes conectores e portas. Além disso, um estilo pode também definir responsabilidades para os elementos da estrutura arquitetural. Dentre as vantagens dos estilos arquiteturais, podem se destacar as seguintes:

- ✓ Estabelecem restrições que organizam a estrutura do sistema
- ✓ Comunica a intenção do design
- ✓ Permite melhor e maior reutilização
- ✓ Melhora a comunicação entre a equipe técnica
- ✓ Ajudam estrategicamente atender requisitos não-funcionais críticos

### Tipos de estilos arquiteturais

---

Nesta seção apresentamos alguns estilos arquiteturais, sua visão geral e suas vantagens e desvantagens.

#### **Big Ball Mud**

Falta de estrutura arquitetural aparente, crescimento descontrolado, código mal escrito. Impossível garantir qualidade!



**Figura 40 - Estilo Arquitetural - Grande bola de lama (Big Ball Mud)**

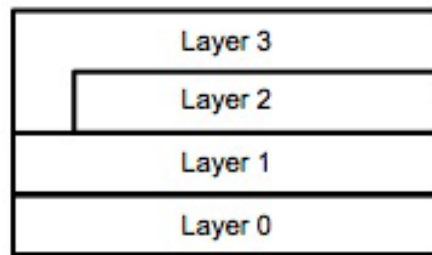
Fonte: <http://blogdozanei.wordpress.com/2012/05/04/futebol-jornalismo-e-estomago/>

#### **Layered Style**

Pilha ordenada de camadas que usam apenas camadas adjacentes. Há casos especiais onde o exemplo mostra a camada 3 acessando a camada 1.

- ✓ **Vantagens**
  - Facilidade de compreensão
  - Facilidade de manutenção
  - Desenvolvimento independente

- Facilidade de Reutilização
- ✓ **Desvantagens**
  - Duplicação de funcionalidade
  - Overhead de implementação e desempenho
  - Em alguns casos, é difícil estruturar um sistema através de camadas
    - Daí ocorre sua violação (implementação fora do padrão em camadas)

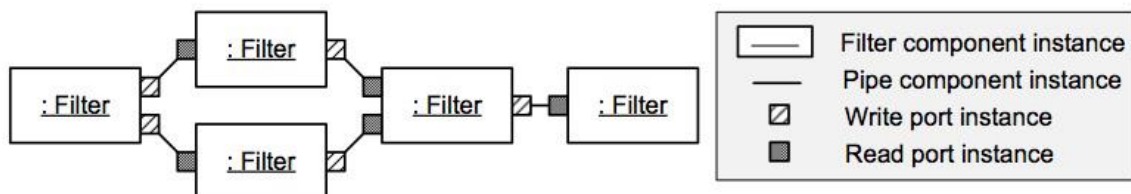


**Figura 41 - Estilo Arquitetural - Layered Style**

### **Piper-and-Filter**

Cada filtro processa incrementalmente sua entrada e escreve sua saída. Os filtros são independentes e somente interagem com outros através de suas vias e não podem compartilhar estados

- ✓ **Vantagens**
  - Encapsulamento
  - Alta Coesão
  - Reuso
  - Interação com filtros de forma limitada, logo, baixo acoplamento
- ✓ **Desvantagens**
  - Processamento em lotes
  - Pode ter baixa performance
  - Não indicado para aplicações interativas
  - Tratamento de erros é difícil e a segurança fica em jogo



**Figura 42 - Estilo Arquitetural - Piper-and-Filter**

### **Batch-Sequential**

Cada estágio lê completamente sua entrada e escreve sua saída de uma vez

- ✓ **Vantagens**
  - Útil para problemas que podem ser resolvidos em vários passos de forma sequencial



### ✓ Desvantagens

- Não permite processamento em paralelo
- Pode ter problemas com performance

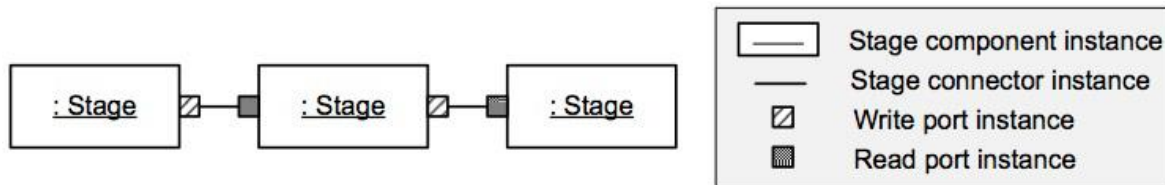


Figura 43 - Estilo Arquitetural - *Batch-Sequential*

### **Model-centered**

Os componentes dependem apenas do componente modelo e interagem através dele. Baixo acoplamento.

### ✓ Vantagens

- Manutenção facilitada
- A camada de Interface pode ser alterada facilmente
- Facilidade na Integração com outros sistemas

### ✓ Desvantagens

- Visões e controles interagem somente através do modelo

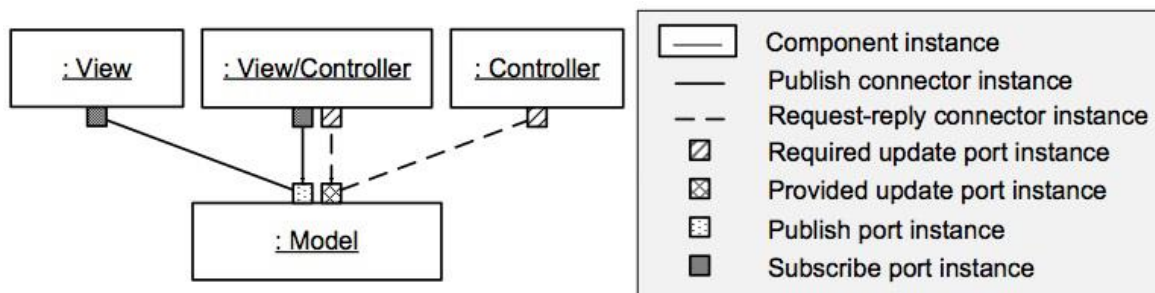


Figura 44 - Estilo Arquitetural - *Model-centered*

### **Publish-subscribe**

Também conhecido como baseado em eventos (*event-based*). Os componentes independentes publicam eventos e fazem "assinaturas" para acompanhar outros eventos.

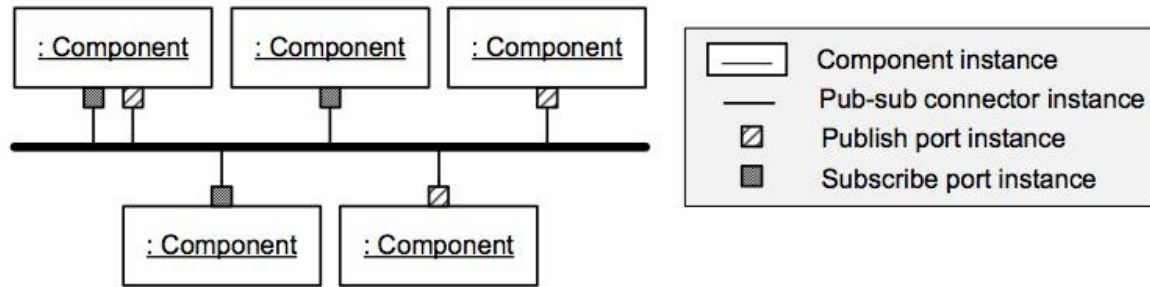
### ✓ Vantagens

- Relativamente fácil adicionar/remover e alterar componentes
- A independência dos componentes proporciona reusabilidade

### ✓ Desvantagens

- Não há garantia que algum componente irá responder há um evento
- Não há garantia que a ordem de resposta é a ideal
- O tráfego de eventos pode ser variável (desempenho pode ser afetado)





**Figura 45 - Estilo Arquitetural - Publish-subscribe**

### Client-server

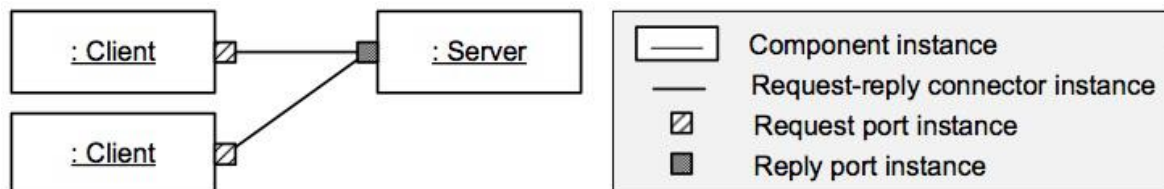
Clientes requisitam serviços aos servidores. Geralmente de modo síncrono e por meio de um conector de requisição e resposta

#### ✓ Vantagens

- É a base de muitas aplicações web atuais
- Separação de interesses
- Fácil expandir a infraestrutura
- Permite distribuir o processamento

#### ✓ Desvantagens

- Disponibilidade é um fator crítico
- O gerenciamento de redundância é complicado
- Servidores podem fazer pedidos a outros servidores, mas não a outros clientes
- O cliente precisa conhecer os serviços oferecidos pelo servidor
- O cliente precisa saber como contactar os servidores



**Figura 46 - Estilo Arquitetural - Client-server**

### Peer-to-Peer

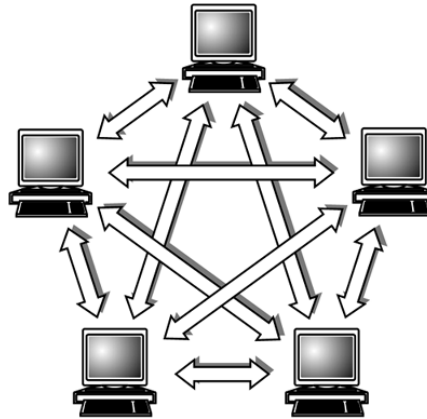
Cada nó pode requisitar ou fornecer serviços para outro nó

#### ✓ Vantagens

- Reusabilidade
- Reconfigurabilidade (modificabilidade)

#### ✓ Desvantagens

- Filtros independentes
- Processamento incremental



**Figura 47 - Estilo Arquitetural - Peer-to-Peer**

Fonte: <http://tekarts.com/wp-content/uploads/2012/02/peer-to-peer-network.gif>

### Map-reduce

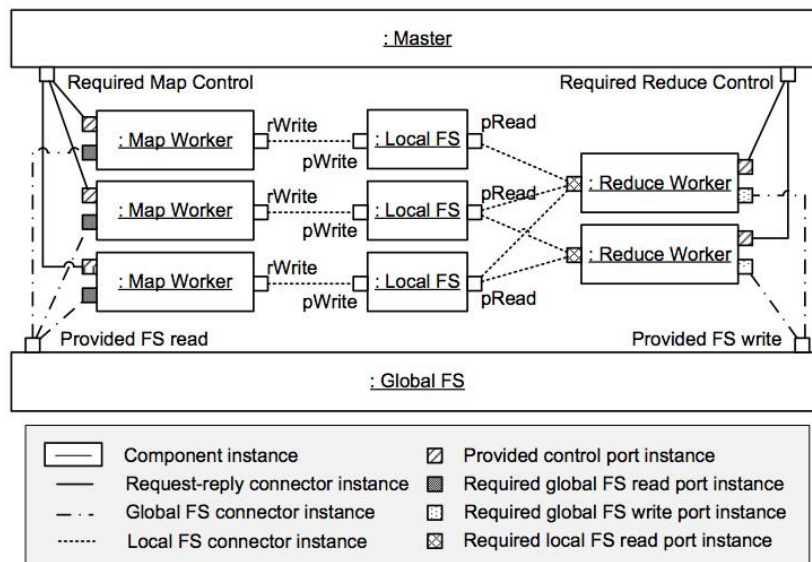
Grandes conjuntos de dados são processados de forma distribuída através de múltiplos computadores. Os resultados do processamento são combinados e agrupados formando a resposta

#### ✓ Vantagens

- Escalabilidade
- Performance
- Disponibilidade
- 

#### ✓ Desvantagens

- Complexidade
- O custo pode ser alto



**Figura 48 - Estilo Arquitetural - Map-reduce**

**São estilos do tipo alocação e geralmente discutidos por Engenheiros de Rede:**

**Estilo espelhado (*Mirrowed*):** O hardware é replicado e roda em paralelo para garantir disponibilidade

**Pilha (*Rack*):** Servidores são empilhados em uma estrutura (rack)

**Fazenda (*Farm*):**

- ✓ Vários computadores interconectados geograficamente no mesmo local
- ✓ Geralmente um conjunto de racks

### **Resumo**

- ✓ Todo estilo arquitetural impõe uma ou mais restrições
- ✓ Todo software possui uma arquitetura, seja ela boa ou não
- ✓ As restrições dos Estilos Arquiteturais organizam a estrutura

## Capítulo 7 - Padrões de Projeto

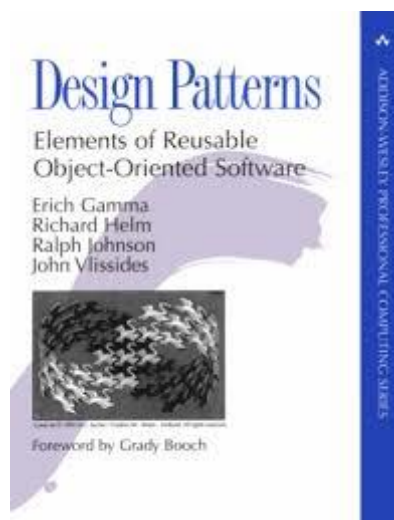
### Objetivos do Capítulo

- ✓ Definir Padrão de Desenho
- ✓ Apresentar a lista de Padrões de Desenho

### Introdução aos Padrões de Projeto

Uma definição formal para Padrões de Projeto (Design Patterns) é que: "Os padrões de projeto são descrições de objetos que se comunicam e classes que são customizadas para resolver um problema genérico de design em um contexto específico" **Gamma, Helm, Vlissides & Johnson (1995)**. Os quatro autores são conhecidos como "The **Gang of Four**", por isso, os Padrões são muitas vezes referenciado por "Padrões GoF".

Enquanto Padrões de Arquitetura aparecem no nível de Arquitetura do Sistema, os Padrões de Desenho aparecem no nível de Implementação e são expressos na forma de problema versus sugestão de solução. Ao longo dos anos, padrões foram surgindo e outros substituídos por padrões melhores. Muitas linguagens de programação atuais já incorporaram a implementação de alguns padrões, como por exemplo, o *Dot Net* da *Microsoft* com o componente de acesso a dados ADO.net que utiliza os padrões Adapter e Command. A figura abaixo apresenta um bom livro para consulta sobre Padrões de Projetos dos "The Gang of Four". Embora o livro seja antigo, ele ainda é referência para o assunto.



**Figura 49 - Livro Design Patterns: Elements of Reusable Object-Oriented Software**

**Fonte:** <http://www.pearsoned.co.uk/bookshop/detail.asp?item=171742>

Os elementos que descrevem os Padrões de Projeto são:

- ✓ Nome
- ✓ Problema
  - Quando aplicar o padrão
- ✓ Solução
  - Descrição que comunica o problema e como usar classes e objetos para resolver
- ✓ Consequências

- Custo benefício de se usar o padrão
- Impactos sobre o sistema (flexibilidade, extensibilidade, portabilidade e eficiência do sistema)

## Padrões de Projeto (*Design Patterns*)

Atualmente, existe uma lista contendo 23 Padrões de Projeto. Eles são divididos em três grupos: Padrões de Criação (criação de classes e objetos), Padrões Estruturais (alteração da estrutura de um problema) e Padrões Comportamentais (controle do seu comportamento). A figura abaixo mostra a lista dos padrões classificados por tipo.

Propósito Padrões de Projeto		
Padrões de Criação	Padrões Estruturais	Padrões Comportamentais
<ul style="list-style-type: none"> <li>• Factory Method</li> <li>• Builder</li> <li>• <b>Abstract Factory</b></li> <li>• Prototype</li> <li>• <b>Singleton</b></li> </ul>	<ul style="list-style-type: none"> <li>• Class Adapter</li> <li>• Object Adapter</li> <li>• Bridge</li> <li>• Composite</li> <li>• Decorator</li> <li>• <b>Facade</b></li> <li>• Flyweight</li> <li>• <b>Proxy</b></li> </ul>	<ul style="list-style-type: none"> <li>• Interpreter Template</li> <li>• Method</li> <li>• Chain of Responsibility</li> <li>• <b>Command</b></li> <li>• Iterator</li> <li>• Mediator</li> <li>• Memento</li> <li>• <b>Observer</b></li> <li>• State</li> <li>• Strategy</li> <li>• Visitor</li> </ul>

**Figura 50 - Tabela de Padrões de Projeto**

Na figura acima foi destacado os padrões de desenho que foram discutidos nas aulas em vídeo. Basicamente dois padrões de criação (*Abstract Factory* e *Singleton*), dois estruturais (*Facade* e *Proxy*) e dois comportamentais (*Command* e *Observer*). Eles são padrões comuns e bastante utilizados em aplicações corporativas.

### **Factory Method**

“Definir uma interface para criar um objeto, mas deixar as subclasses decidirem que classe instanciar. O Factory Method permite adiar a instanciação para subclasses” [7]

### **Builder**

“Separar a construção de um objeto complexo de sua representação de modo que o mesmo processo de construção possa criar diferentes representações” [7]

### **Abstract Factory**

“Fornecer uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas” [7]

### **Prototype**

“Especificar tipos de objetos a serem criados usando uma instância protótipo e criar novos objetos pela cópia desse protótipo” [7]

**Singleton**

"Garantir que uma classe tenha somente uma instância e fornece um ponto global de acesso para a mesma" [7]

**Adapter**

"Converter a interface de uma classe em outra interface, esperada pelo cliente. O Adapter permite que interfaces incompatíveis trabalhem em conjunto – o que, de outra forma, seria impossível" [7]

**Brigde**

"Desacoplar uma abstração da sua implementação, de modo que as duas possam variar independentemente" [7]

**Composite**

"Compor objetos em estruturas de árvore para representar hierarquia partes-todo. Composite permite aos clientes tratarem de maneira uniforme objetos individuais e composições de objetos" [7]

**Decorator**

"Dinamicamente, agregar responsabilidades adicionais a objetos. Os Decorators fornecem uma alternativa flexível ao uso de subclasses para extensão de funcionalidades" [7]

**Facade**

"Fornecer uma interface unificada para um conjunto de interfaces em um subsistema. Facade define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado" [7]

**Flyweight**

"Usar compartilhamento para suportar eficientemente grandes quantidades de objetos de granularidade fina" [7]

**Proxy**

"Fornecer um substituto ou marcador da localização de outro objeto para controlar o acesso a esse objeto" [7]

**Interpreter**

"Dada uma linguagem, definir uma representação para sua gramática juntamente com um interpretador que usa a representação para interpretar sentenças dessa linguagem" [7]

**Method**

"Definir o esqueleto de um algoritmo em uma operação, postergando alguns passos para as subclasses. Template Method permite que subclasses redefinam certos passos de um algoritmo sem mudar a estrutura do mesmo" [7]

**Chain of Responsibility**

"Evitar o acoplamento do remetente de uma solicitação ao seu receptor, ao dar a mais de um objeto a oportunidade de tratar a solicitação. Encadear os objetos

receptores, passando a solicitação ao longo da cadeia até que um objeto a trate” [7]

**Command**

“Encapsular uma solicitação como objeto, desta forma permitindo parametrizar cliente com diferentes solicitações, enfileirar ou fazer o registro de solicitações e suportar operações que podem ser desfeitas” [7]

**Iterator**

“Fornecer um meio de acessar, sequencialmente, os elementos de um objeto agregado sem expor sua representação subjacente” [7]

**Mediator**

“Definir um objeto que encapsula a forma como um conjunto de objetos interage. O Mediator promove o acoplamento fraco ao evitar que os objetos se refiram uns aos outros explicitamente e permitir variar suas interações independentemente” [7]

**Memento**

“Sem violar o encapsulamento, capturar e externalizar um estado interno de um objeto, de maneira que o objeto possa ser restaurado para esse estado mais tarde” [7]

**Observer**

“Definir uma dependência um para muitos entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente” [7]

**State**

“Permite a um objeto alterar seu comportamento quando seu estado interno muda. O objeto parecerá ter mudado de classe” [7]

**Strategy**

“Definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis. Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam” [7]

**Visitor**

“Representar uma operação a ser executada nos elementos de uma estrutura de objetos. Visitor permite definir uma nova operação sem mudar as classes dos elementos sobre os quais opera” [7]

O conceito de cada padrão de desenho foi apresentado segundo o livro do “The Gang of Four”. Para obter um exemplo prático de cada um desses padrões, você pode visitar o blog de Marcos Brizenno que possui demonstrações bem didáticas sobre cada padrão: <http://brizenno.wordpress.com/category/padroes-de-projeto/>

**Resumo**

- ✓ Existem Padrões de Criação, Padrões Estruturais e Comportamentais
- ✓ Padrões de Projeto são utilizados no nível de implementação
- ✓ Os Padrões são soluções para problemas recorrentes em Orientação a Objetos



## Capítulo 8 - Modelagem Arquitetural

### Objetivos do Capítulo

- ✓ Apresentar os objetivos da Modelagem Arquitetural
- ✓ Apresentar os artefatos gerados no processo de Modelagem Arquitetural
- ✓ Apresentar o TOGAF – The Open Group Architecture Framework

### Introdução à Modelagem Arquitetural

A Modelagem Arquitetural é um conjunto de atividades relacionado à estruturação de uma aplicação. O processo engloba tanto a tomada de decisões a respeito desta estrutura quanto a sua documentação.

### No processo de Modelagem Arquitetural o Arquiteto deve considerar:

- ✓ Os requisitos arquiteturais do projeto
- ✓ O conhecimento técnico da equipe de desenvolvimento
- ✓ Os riscos técnicos conhecidos
- ✓ As ferramentas que serão utilizadas na fase de construção
- ✓ Os padrões de projeto que serão utilizados
- ✓ O estilo arquitetural a ser adotado
- ✓ A infraestrutura disponível para execução do sistema nos ambientes de desenvolvimento, homologação e produção



**Figura 51 - Arquiteto de Software - Modelagem Arquitetural**

**Fonte:** <http://empregopelomundo.com/2013/06/top-10-dos-melhores-trabalhos-de-2013-da-revista-forbes/>

**O processo de Modelagem Arquitetural pode ser descrito da seguinte forma:**

**Analisar:** Avalie os Requisitos Arquiteturais e os Riscos técnicos do projeto. Certifique-se de que os Requisitos Arquiteturais estão maduros o suficiente para permitir o início da Modelagem Arquitetural.

**Analisar:** Para cada Risco técnico conhecido, verifique a necessidade da realização de uma Prova de Conceito para provar ou refutar o risco.

**Aplicar:** Após a execução das Provas de Conceito (se existirem), crie a estrutura estática do sistema, documentando-a através de um Diagrama de Componentes. Leve em consideração o princípio da Separação de Responsabilidades e os Padrões de Desenho relevantes para o sistema. Crie Diagramas de Sequência, se necessário para esclarecer como os diversos Componentes interagem em cenários críticos.

**Documentar:** Selecione as principais tecnologias que devem ser utilizadas para a implementação do Desenho. Descreva sucintamente o motivo pelo qual cada tecnologia foi adotada e por que alternativas foram descartadas.

**Documentar:** Crie um Diagrama de Implantação que mostra como os Componentes e Tecnologias são mapeados para os elementos de infraestrutura.

**Verificar:** Repita iterativamente os passos anteriores até que a Arquitetura do Sistema aparente estar madura o suficiente.

### Artefatos da Modelagem Arquitetural

**A Modelagem Arquitetural geralmente produz dois artefatos:**

**Modelo Arquitetural:**

É um documento contendo uma descrição das principais decisões estruturais na forma de texto e diagramas UML.

**Arquitetura Executável:**

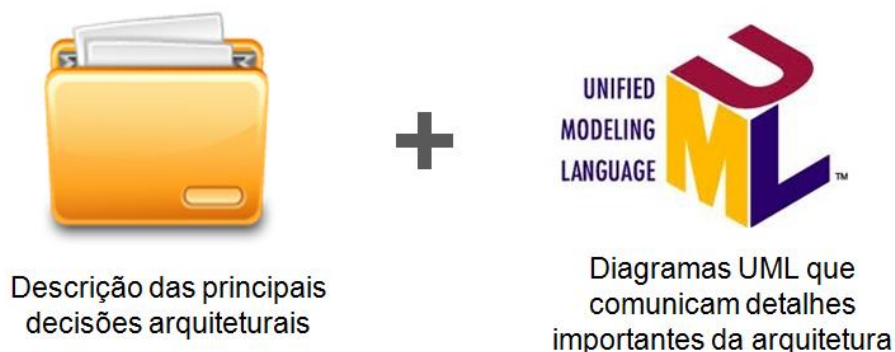
Consiste em uma versão inicial do sistema que geralmente é produzida pelo Arquiteto com o objetivo de garantir a estrutura definida para o software. A intenção não é implementar funcionalidades e sim iniciar corretamente a implementação que será continuada pela equipe de desenvolvimento. A Arquitetura Executável representa então um “esqueleto” do sistema, que deverá ser preenchido com funcionalidades à medida que o processo de desenvolvimento avança.



**Figura 52 - Artefatos da Modelagem Arquitetural**

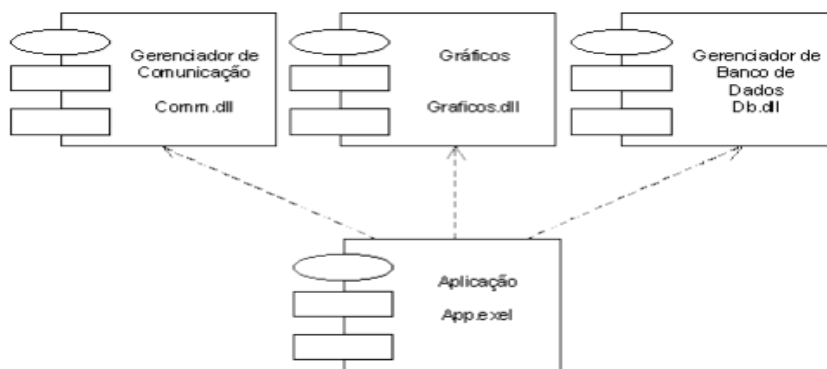
Além dos Diagramas da UML, o Modelo Arquitetural do sistema pode ser complementado com documentos textuais contendo informações adicionais sobre decisões de Arquitetura. É importante destacar que na maioria das vezes não é desejado e muito menos necessário criar todos os diagramas da UML para representar a Modelagem. Muitos diagramas são similares e a diferença está no nível de detalhe dos componentes modelados. Além disso, o custo para se produzir diagramas detalhados talvez não seja revertido em melhor

entendimento pela equipe de desenvolvimento, pode é causar confusão. O ideal é criar apenas os diagramas que são essenciais para comunicar a estrutura e comportamento de itens do software mais complexos.



**Figura 53 - Modelo Arquitetural**

Através do Diagrama de Componentes é possível descrever os componentes que irão compor o sistema, assim como seus relacionamentos. Componentes UML são unidades isoladas, autônomas e encapsuladas dentro de um Sistema de Software.

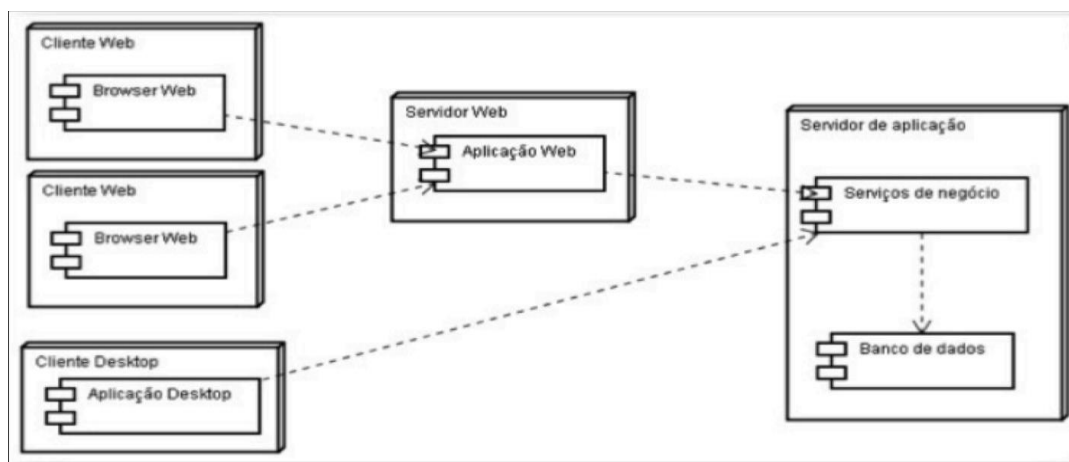


**Figura 54 - Diagrama de Componentes**

Fonte: <http://goo.gl/SI6VQr>

O Diagrama de Implantação tem a função de descrever os elementos de infraestrutura que serão necessários para implantação do sistema, assim como os Componentes que serão instalados. São detalhados o hardware no qual a aplicação será executada e o Software utilizado nas diversas camadas da aplicação.

Um Diagrama de Implantação contém Nós (Nodes), que representam ambientes de execução para Componentes de um sistema. Nós (Nodes) podem conter outros nós ou Componentes. Uma hierarquia de Nós pode ser usada para descrever os diversos elementos de um ambiente de execução, partindo do Hardware em direção ao Software.



**Figura 55 - Diagrama de Implantação**

**Fonte:** <http://www.klebermota.eti.br/2013/07/30/desenvolvimento-de-sofware-orientado-a-objetos/>

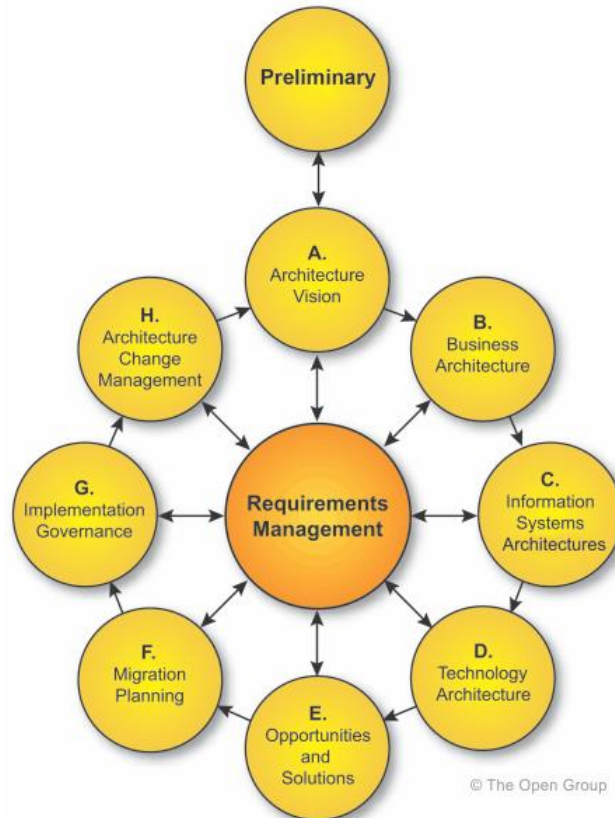
## TOGAF – The Open Group Architecture Framework

“É um framework que auxilia criar, detalhar, avaliar e construir uma arquitetura de TI correta para as organizações promovendo alinhamento entre as necessidades do negócio e o parque tecnológico. É um guia que auxilia a gestão de TI a promover a real evolução dos serviços de TI em detrimento às necessidades de negócio fornecendo um conjunto detalhado de métodos e ferramentas para o desenvolvimento de uma arquitetura corporativa.” [14].

**Atualmente na versão 9.1, publicada em Dezembro de 2011 compreendendo quatro tipos de arquitetura:**

- Negócios
- Dados
- Aplicações
- Tecnologia

O “coração” do TOGAF é o ADM (*Architecture Development Method*) que estabelece uma metodologia para o desenvolvimento e manutenção de uma Arquitetura Corporativa. A figura abaixo mostra como são organizadas as fases do ADM.



**Figura 56 - Diagrama de Implantação**

Fonte: <http://goo.gl/y0Nndd>

### **Por que a arquitetura corporativa deve ser considerada com assunto estratégico pelas empresas?**

- Ajuda a identificar lacunas entre o estado atual e o estado desejado pela organização.
- O TOGAF acelera o ciclo de desenvolvimento dessa arquitetura, fornecendo respostas para as perguntas o que, quem, quando, como e por quê.

Para obter o material completo sobre o TOGAF e ter acesso à descrição das fases para sua implantação e recomendações, consulte o site oficial em: <https://www.opengroup.org/togaf/>

### **Resumo**

- ✓ Processo de Modelagem Arquitetural consiste em um conjunto de práticas que tem por objetivo produzir o Modelo Arquitetural e a Arquitetura Executável
- ✓ Utilize os diagramas de implantação e componentes para comunicar o modelo de arquitetura para os interessados e para a equipe de desenvolvimento

## Referências bibliográficas

---

- [1] BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. *Software Architecture in Practice*. 2ª ed. Boston: Pearson Education, 2003.
- [2] PRESSMAN, Roger S. *Engenharia de software*. 6ª ed. Porto Alegre: Bookman, 2006.
- [3] UFPE. *Gestão de Riscos em Projetos de Software*. Disponível em: <<http://www.cin.ufpe.br/~processos/TAES3/slides-2010.2/Apresentacao-Riscos.pdf>>. Acesso em: 24 set. 2013.
- [4] MICROSOFT. *Estimativa*. Disponível em: <<http://msdn.microsoft.com/pt-br/library/hh765979.aspx>>. Acesso em: 24 set. 2013.
- [5] ALATS. *Estimativas*. <[http://www.slideshare.net/elias.nogueira/2-encontro-mensal-alats-estimativas?from\\_search=2](http://www.slideshare.net/elias.nogueira/2-encontro-mensal-alats-estimativas?from_search=2)>. Acesso em: 01 out. 2013.
- [6] OMG. *Unified Modeling Language*. Disponível em: <<http://www.omg.org/uml>>. Acesso em: 24 set. 2013.
- [7] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. Estados Unidos: Hardback, 1995. 416 p. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- [8] UFPE. *Padrões de Projetos GoF*. Disponível em: <[www.cin.ufpe.br/~if718/transparencias/pdf/05-padroesGoF.pdf](http://www.cin.ufpe.br/~if718/transparencias/pdf/05-padroesGoF.pdf)>. Acesso em: 24 set. 2013.
- [9] BEZERRA, Eduardo. *Princípios de Análise e Projeto de Sistemas com UML*. Campus, 2006.
- [10] George Fairbanks. *Just Enough Software Architecture: A Risk-Driven Approach*. Marshall & Brainerd, 2010.
- [11] BPMN ORG. *Business Process Modeling Notation (BPMN) Information*. <<http://www.bpmn.org>>. Acesso em: 25 fev. 2016.
- [12] IGRAFX. *BPMN: Business Process Modeling Notation*. <<http://www.es.igrafx.com/solutions/bpmn>>. Acesso em: 25 fev. 2016.
- [13] SHAPE. *SoaML Tutorial: Service Modelling with SoaML*. <[http://www.uio.no/studier/emner/matnat/ifi/INF5120/v11/div/SoaML\\_Tutorial.pdf](http://www.uio.no/studier/emner/matnat/ifi/INF5120/v11/div/SoaML_Tutorial.pdf)>. Acesso em: 25 fev. 2016.
- [14] TI ESPECIALISTAS. *Arquitetura Corporativa com TOGAF*. <<http://www.tiespecialistas.com.br/2013/02/arquitetura-corporativa-com-togaf/>>. Acesso em: 28 fev. 2016.

[15] IBM developerWorks. *TOGAF O que é e por quê?* <<https://www.ibm.com/developerworks/community/blogs/tlcbr/entry/togaf?lang=en>>. Acesso em: 28 fev. 2016.