ESTUDO DE VIABILIDADE DO USO DE ARQUITETURA ORIENTADA A MICROSSERVIÇOS PARA MAXIMIZAR O REAPROVEITAMENTO DE CÓDIGO

FEASIBILITY STUDY OF USING MICROSERVICES ORIENTED ARCHITECTURE TO MAXIMIZE CODE REUSE

Fábio Pereira de Souza[1], João Batista Freire Xavier[2], Max Leonardo Alves Diniz[3], Erwin Alexander Uhlmann[4]

RESUMO: O artigo “estudo de viabilidade do uso de arquitetura orientada a microsserviços para maximizar o reaproveitamento de código” apresenta uma discussão sobre a utilização de microsserviços e sua relevância para as empresas de desenvolvimento de *software*. O objetivo desse estudo é indicar as principais características de uma arquitetura que permita a maior reutilização possível de código de programação. A metodologia empregada no artigo compreendeu pesquisas sobre as práticas e os elementos tecnológicos disponíveis no mercado de desenvolvimento de sistemas, como a Engenharia de Software, SOA, HTTP, REST, JSON, Desenho Arquitetural e aspectos de Segurança, apresentando também uma métrica de viabilidade baseada em quantidade de sistemas e de funcionalidades para quantificar uma probabilidade de reaproveitamento de código.

PALAVRAS-CHAVE: Arquitetura. Serviços. Microsserviços. Integração. Reaproveitamento. Produtividade.

ABSTRACT: *The article "feasibility study of using micro-services oriented architecture to maximize code reuse" presents a discussion on the use of micro-services and their relevance for software development companies. The objective of this study is to indicate the main characteristics of an architecture that allows the greatest possible reuse of programming code The methodology employed in the article understood practices research and technological elements available in the market of systems development, like Software Engineering, SOA, HTTP, REST, JSON, Architectural Design and security aspects, featuring also a viability metric based on quantity of systems and features to quantify a probability of code reuse.*

KEYWORDS: *Architecture. Services. Micro-services. Integration. Reuse. Productivity.*

[1] Graduando em Ciência da Computação - 8º semestre. UnG - Universidade Guarulhos. fabiosouza.web@gmail.com
[2] Graduando em Ciência da Computação - 8º semestre. UnG - Universidade Guarulhos. joao-freirexavier@hotmail.com
[3] Graduando em Ciência da Computação - 8º semestre. UnG - Universidade Guarulhos. maxladiniz@outlook.com
[4] Prof. Me. Orientador. UnG - Universidade Guarulhos. euhlmann@prof.ung.br

1. INTRODUÇÃO

No ramo de desenvolvimento de *softwares*, um fator importante é a entrega dentro do prazo estipulado. Algumas empresas têm dificuldade em se adequar a esses prazos sem que a equipe seja sobrecarregada ou tenha um gasto maior com a contratação de mais profissionais.

Na tentativa de solucionar esse problema, o desenvolvimento deste artigo vai estudar a viabilidade da utilização de um novo conceito de arquitetura de software baseada em microsserviços. De acordo com Bagio (2012), sistemas baseados em microsserviços são sistemas descentralizados em código e com dependências quebradas sem perda de integração entre si, o que possibilita a maximização da quantidade de código que possa ser reutilizado e, com isso, ganho de produtividade no desenvolvimento.

O estudo contou com análise de tecnologias envolvidas, apresentação da arquitetura, definição de uma métrica para auxiliar a empresa a definir se é viável ou não a implantação da arquitetura em determinado cenário com base no custo-benefício.

Como a arquitetura é uma especificação e exige implementação, não foi apontado quais tecnologias deveriam ser utilizadas, mantendo o foco apenas nos conceitos, o que torna possível a implementações em diferentes linguagens de programação.

A prática de reutilização de código costuma ser comum entre os times de desenvolvimento, principalmente em linguagens orientadas a objetos, onde o próprio conceito nasceu baseando-se no reaproveitamento. Rotinas utilizadas por mais de um consumidor são extraídas para um único local e, quando necessário, é feito uma referência para a origem. Este hábito resolve o problema, porém se restringe apenas à própria aplicação, não sendo possível reaproveitar funcionalidades que está implantada em outra aplicação. Na proposta, todas as funcionalidades são extraídas para serviços, ficando expostas para o consumo de clientes distintos.

A importância deste artigo está em cumprir o objetivo de aumentar a produtividade de uma empresa apenas com a mudança no padrão de desenvolvimento, sem a necessidade de aumentar o quadro de funcionários ou sobrecarregar o time de desenvolvimento.

1.1. ENGENHARIA DE SOFTWARE

A Engenharia de Software é uma área que estuda as fases do processo de desenvolvimento de sistemas a fim de otimiza-los. Para Sommerville (2007), é um ramo que estuda maneiras de otimizar a qualidade e um *software* de qualidade é aquele que é fácil de usar, funciona conforme esperado, é de fácil manutenção, possui integridade dos dados entre outras características ligadas a satisfação do usuário. De forma geral, os engenheiros de *software* adotam uma abordagem sistemática e organizada em seu trabalho, que é frequentemente, a maneira mais eficaz de produzir *software* de alta qualidade.

A Engenharia de Software procura selecionar o método mais apropriado para um conjunto de circunstâncias e uma abordagem mais criativa e menos formal pode ser mais eficaz em outras circunstâncias. Na concepção de Paula Filho (2000) trata-se de uma área abrangente, que envolve mapeamento de requisitos, documentação de processos, desenvolvimento, testes, implantação e manutenção de sistemas. O alvo desses padrões são aplicações como produto que precisam de agilidade no desenvolvimento e envolvem custos as empresas, não

sendo viável a aplicação dos conceitos em *softwares* pequenos sem caráter comercial.

1.2. ARQUITETURA DE SOFTWARE

Segundo Ghezzi (2000), a Arquitetura de Software é um conjunto de componentes relacionados que satisfaz requisitos funcionais e não funcionais de um sistema ou uma abstração situada entre o problema que pretende solucionar a implementação técnica. De acordo com essas duas definições, podemos dizer que arquitetura de software é um conjunto de elementos organizados de forma abstrata que define a estrutura de um *software* através de seus componentes e relacionamentos, não se preocupando em um primeiro momento com a implementação, mas sim com a organização estrutural do problema, afim de solucioná-lo de maneira mais eficiente.

A Arquitetura de Software funciona como um meio de definir e estruturar de maneira gráfica a melhor maneira de se desenvolver um sistema, podendo estar presente no contexto de uma única aplicação, assim como na integração de diversos sistemas de empresas diferentes. Para Garlan e Shaw (1994), o processo de definição de uma arquitetura consiste em um amplo estudo das necessidades e a análise crítica das opções levando em consideração as vantagens e desvantagens de cada uma em relação a outra para solucionar o problema de maneira que se obtenha o resultado esperado.

Em geral, serve como uma forma de planejar e estruturar a solução, sempre analisando o quão complexas as funcionalidades podem se tornar, evitando eventuais problemas desde o princípio. Um fator importante é a melhoria contínua, segundo Magalhães (2005), um sistema é dinâmico pois sofre diversas alterações através de manutenção corretiva, adequação de negócio, atendimento a atualizações de legislação ou modernização tecnológica. Essas mudanças, podem desviar o propósito inicial da arquitetura do momento em que foi construída, por esse motivo é importante a avaliação constante de o quanto a aplicação é atendida pela implementação arquitetural e quais os pontos negativos e positivos, a fim de buscar sua melhora constante.

1.3. SOA

A Arquitetura Orientada a Serviços, do inglês *Software Oriented Architecture* (SOA), é um estilo arquitetural que propõe a distribuição do negócio em serviços. “A SOA estabelece um modelo arquitetônico que visa aprimorar a eficiência, agilidade e a produtividade de uma empresa, posicionando os serviços como os principais meios para que a solução lógica seja representada” (ERL, 2009, p.24). Para os autores Mackenzie, Laskey, Mccabe, Brow e Metz (2006), o valor da SOA está no oferecimento de um padrão único para grandes sistemas que precisam estar integrados mesmo que isso não tenha sido planejado em um primeiro momento, o que é comum em organizações que estão com os processos em constante evolução.

2. MÉTODOS

2.1. HTTP

Kurose (2006) define o HTTP, que significa Hypertext Text Transfer Protocol (Protocolo de Transferência de Hipertexto), como um meio que possibilita a comunicação de dados entre o dispositivo do cliente e o servidor, permitindo que páginas HTML sejam renderizadas no navegador. A organização W3C, que regulamenta padrões para a criação de conteúdos para a Web, definiu oito métodos de comunicação HTTP, são eles: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS e CONNECT, cada qual com uma ação

específica de acordo com o recurso utilizado.

2.2. REST

Quatro destes métodos, o *Get, Put, Post* e *Delete*, que representam respectivamente as ações listar, atualizar, gravar e excluir, são utilizadas no REST, que segundo o idealizador, Dr. Fielding (2000), significa *Representational State Transfer* (Transferência de Estado Representativo) e define um estilo arquitetural para sistemas distribuídos mediante o uso de URLs através de métodos HTTP que requisitam recursos e que podem ser representados através de um documento HTML ou de dados estruturados como a XML - uma linguagem de marcação que descreve diversos tipos de dados -, e o JSON, servindo como uma maneira de se desenvolver serviços voltados para a Web guiado pelas práticas do protocolo HTTP.

2.3. JSON

O JSON, que significa *JavaScript Object Notation* (Notação de Objeto JavaScript), é definido por Deitel (2008) como um formato de transferência de dados baseado em texto, muito utilizado para representar objetos JavaScript - uma linguagem de programação interpretada em navegadores Web do lado do cliente - em formato *string* e transmiti-los por meio de uma rede de computadores. Comparado ao XML, que é uma linguagem de marcação para criação de documentos organizados hierarquicamente, o JSON tem algumas vantagens consideráveis como: Facilidade de leitura, rapidez da análise sintática, simplicidade para a criação de objetos e a possibilidade de transmissão e manipulação de dados via internet com eficiência, sendo então suportado por diversas linguagens de programação.

2.4. MÉTRICA DE VIABILIDADE

Um modelo matemático foi criado para demonstrar a viabilidade de uso de uma arquitetura orientada a microsserviços, nomeada Métrica de Viabilidade. Este cálculo visa quantificar para a empresa a possibilidade de reaproveitar as funcionalidades e justificar o uso da arquitetura, baseando-se principalmente em duas variáveis: quantidade de sistemas e de funcionalidades.

A fórmula a seguir apresenta esta métrica:

Fórmula 1 - Métrica de Viabilidade

$$P = \frac{A_i}{\sum_i^n A_{i+1}}$$

Onde:

P = Probabilidade de reutilização;
A = quantidade de funcionalidades do sistema i;
i = identificação do sistema;
n = quantidade total de sistemas.

A saída deste cálculo gera uma porcentagem que possibilita a empresa verificar a viabilidade da utilização da arquitetura, podendo até aplica-la à sua realidade organizacional, levando em conta fatores como tempo e custos (investimento).

3. DESENVOLVIMENTO

Este capitulo tem como objetivo o detalhamento da proposta arquitetural e a

aplicação da métrica de viabilidade com base em valores demonstrativos para exemplificar a utilização.

3.1. ANÁLISE

Durante o estudo foi feito um levantamento de requisitos de arquitetura de software, momento em que foram identificados dez itens importantes que devem ser atendidos pela proposta. A Tabela 1 a seguir apresenta e descreve esses requisitos seguidos por uma coluna que representa o impacto, totalizando 10 pontos, onde o maior número representa um caráter mais estratégico para a arquitetura, portanto, de maior importância.

Tabela 1 - Requisitos Fundamentais da Arquitetura

ID	Requisito	Impacto
1	Agilizar o processo de desenvolvimento	1,5
2	Proporcionar maior competitividade no mercado	0,5
3	Facilitar manutenção	1,5
4	Não afetar o processo de testes	0,5
5	Não limitar o desenvolvimento	1,0
6	Deve possuir disponibilidade	1,0
7	Possuir aspecto genérico	1,0
8	Garantir a segurança	0,5
9	Facilitar modernização das telas	1,0
10	Facilitar a integração entre os sistemas	1,5

Quando uma empresa apresenta ou segue um modelo de arquitetura monolítica onde todos os módulos são compostos em uma mesma unidade como, por exemplo uma aplicação financeira que possua os componentes Movimentação Financeira, Agendamentos e Manutenção de De-Para, todos acessando um único banco de dados, uma abordagem falha no quesito reaproveitamento.

Após uma análise bem definida, onde foi levantado os objetivos que se deseja atender e detalhado alguns aspectos mais técnicos, o próximo passo é introduzir os conceitos iniciais da proposta arquitetural, que será realizado no decorrer do próximo capitulo.

3.2. MICROSSERVIÇOS

A orientação a serviços é essencial para a arquitetura, mas a dúvida é se a sua utilização vai atender de fato todas as necessidades. Ao observar a evolução da arquitetura monolítica para a arquitetura orientada a serviços, o princípio foi descentralizar o código e quebrar as dependências entre os sistemas, sem que perdessem a integração entre si. Sendo assim, duas funcionalidades distintas não dependem uma da outra apesar de se comunicarem e, se houver indisponibilidade de uma, esta não afetaria as demais. O conceito de microsserviços seguiu essa tendência, descentralizando e quebrando ainda mais as dependências (BIAGIO, 2012).

3.3. SEGURANÇA

Quando o negócio da empresa está disponível através de serviços, deve-se tomar um cuidado especial com relação à segurança. O fato de que mais de um sistema vai consumir o serviço, não quer dizer que deve estar disponível para qualquer cliente que faça uma requisição.

Uma empresa que desenvolva uma API para o público em geral, como por exemplo, Google, *PayPal, Dropbox* entre outros, certamente deve tomar um cuidado muito maior, pois vai estar exposto a uma infinidade de requisições diárias, de várias pessoas ao redor do mundo, e apesar de não ser o caso da arquitetura, é um ponto que deve ser levado em consideração.

3.4. AUTENTICAÇÃO E AUTORIZAÇÃO

Os sistemas que implantarem a arquitetura, trafegarão dados críticos para as empresas por meio de serviços através da rede, por esse motivo, deve-se garantir que o usuário ou aplicação está de fato autorizado a visualizá-la. Para isso existe o conceito de autenticação, que faz a liberação do acesso a uma aplicação ou recurso. Já a autorização seria definir que uma vez autenticado, quais recursos o usuário teria acesso dentro da aplicação através de uma espécie de árvore de permissões ligando um usuário aos recursos que tem acesso (STALLINGS, 2008).

É importante salientar que independente do modelo de autenticação que será utilizado há diversas frentes em segurança que não podem ser deixadas de lado. Algumas delas estão além do escopo da arquitetura em si, como medidas envolvendo infraestrutura de rede e servidor por exemplo. Por motivos de escopo, não cabe aqui entrar em cada um desses assuntos, mas alguns deles que poderíamos citar é a utilização de protocolo HTTPs, restrição de acesso por IP, firewalls, restrição por protocolos de comunicação, MAC Address, entre outros.

3.5. HTTP BASIC

O HTTP Basic como o próprio nome indica é um padrão básico de autenticação HTTP quando um cliente faz uma requisição. Neste modelo os dados do cliente são enviados pelo próprio cabeçalho da requisição, codificado em Base64 (MACKENZIE, 2003). O padrão HTTP Basic não está ligado a como vai ser realizado a validação do acesso, mas sim, como essa informação vai trafegar até o servidor, onde por sua vez pode validá-lo através de uma base de dados ou outro recurso, como por exemplo o *Active Directory* do Windows. Uma vez validado é feito a resposta, podendo ser do *Status Code* 401, indicando que não está autorizado, ou 200, indicando que a requisição foi realizada com sucesso.

Este modelo de autenticação, apesar de simples pode ser útil, uma vez que é possível implantá-lo rapidamente se necessário, integrá-lo a outros modelos e obter bons níveis de segurança. Como por exemplo o fato de trafegar o usuário e senha pela rede, obviamente é um ponto negativo, pois esses dados podem ser interceptados e usados por alguém com más intenções, porém, podemos combiná-lo com o protocolo HTTPS que vai manter o meio de comunicação seguro evitando esse tipo de problema.

3.6. HTTP DIGEST

O HTTP Digest é semelhante ao modelo anterior: faz uma comunicação básica entre o cliente e o servidor enviando nome do usuário e senha junto da requisição. Contudo, diferente do HTTP Basic, que envia os dados diretamente, o HTTP Digest aplica uma criptografia MD5 na senha antes do envio (MACKENZIE, 2003). A saber, o MD5 é um mecanismo de *hash* unidirecional, ou seja, não pode ser transformado novamente no seu valor de origem. Para se fazer a validação de um código MD5 é necessário comparar os dois *hashs*: o de origem, enviado pelo cliente, e o original, armazenado previamente no servidor.

Assim como o modelo anterior, o HTTP Digest possui uma característica de simplicidade e de fácil implementação, além de possuir um recurso de segurança interessante que consiste na criptografia da senha antes dela ser enviada para o servidor, podendo ser útil na arquitetura para a proteção de alguns recursos.

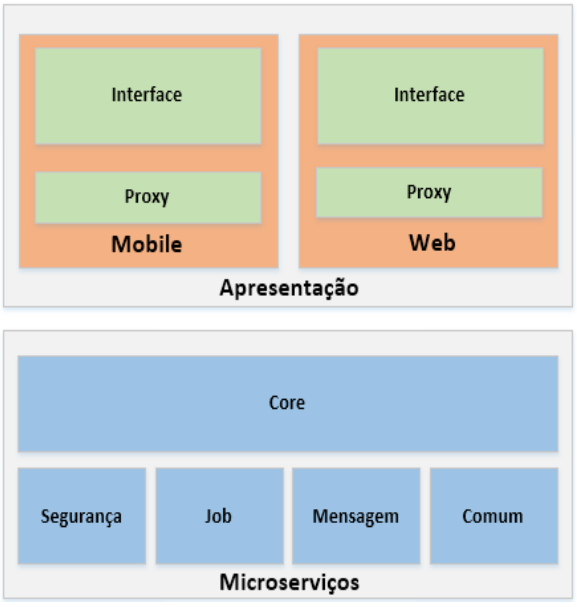
3.7. DESENHO ARQUITETURAL

No desenvolvimento do trabalho é perceptível a tendência de descentralização da lógica em fragmentos de *software* cada vez menores. Desde o modelo monolítico, com rotinas responsáveis por grandes fluxos de informações, passando pelo SOA, com a separação das regras de processo em serviços até chegar ao conceito de microsserviços quebrando esses processos em pedaços ainda menores.

Conforme a Figura 1 a arquitetura divide as aplicações em uma grade horizontal colocando todas os microsserviços de forma independente de um lado e as aplicações do outro, com o mínimo de código possível, sendo ideal apenas o código responsável pela construção das interfaces web e a camada de proxy que deve consumir os serviços.

A camada abaixo, de microsserviços é composta pelo *core* que contém toda a lógica e regra de negócio das aplicações, sendo de suma importância adotar uma característica mais genérica para garantir que as funcionalidades sejam reutilizadas por qualquer aplicação, mesmo que em um primeiro momento não haja nenhuma aplicação em vista para reutilização. Juntamente com o *core*, há outras camadas de microsserviços opcionais que auxiliam as funcionalidades principais, como segurança para autenticações e autorizações, os *jobs* para processamento em lote, mensageria para comunicar eventos e comum para guardar utilidades no geral.

Figura 1 - Desenho Arquitetural



3.8. APLICAÇÃO DA MÉTRICA DE VIABILIDADE

Para exemplificar e demonstrar o cálculo da Probabilidade de Reutilização foram criados dois cenários em que duas

empresas, X e Y, possuem uma série de sistemas a desenvolver.

No primeiro cenário, chamado de Caso 1, a empresa X irá desenvolver cinco sistemas, chamados de V1, W2, X3, Y4 e Z5, sendo que eles possuem 44, 52, 26, 68 e 18 funcionalidades, respectivamente. No segundo cenário, chamado de Caso 2, a empresa Y irá desenvolver nove sistemas, chamados de 1A, 2B, 3C, 4D, 5D, 6E, 7F, 8G e 9H, sendo que eles possuem 9, 10, 23, 19, 11, 8, 7, 13 e 5 funcionalidades, respectivamente.

A Tabela 2 demonstra o exemplo de aplicação do cálculo da Probabilidade de Reutilização (P, que representa em porcentagem o quanto se pode reaproveitar do sistema) na empresa X, do Esforço de Desenvolvimento (EsD, que representa a média de funcionalidades a serem de fato desenvolvidas) e da Economia de Desenvolvimento (EcD, que representa a média de funcionalidades reaproveitadas, ou seja, não precisariam ser desenvolvidas), além de especificar o sistema (S) e a quantidade de funcionalidades (F).

Tabela 2 - Cenário de aplicação da métrica de viabilidade: Caso 1

S	F	P	EsD	EcD
V1	44	26,83%	32	12
W2	52	33,33%	35	17
X3	26	14,29%	22	4
Y4	68	48,57%	35	33
Z5	18	9,47%	16	2
Total	**208**	**32,49%**	**140**	**68**

Neste cenário, a empresa X poderia obter até 32,49% de reaproveitamento das funcionalidades com a aplicação da arquitetura de microsserviços, e desenvolveria em média apenas 140 das 208 funcionalidades previstas na etapa de planejamento dos sistemas, ou seja, ela economizaria esforços em 68 funcionalidades, estas que antes demandariam mais tempo e custos de desenvolvimento.

A Tabela 3 demonstra o exemplo de aplicação do cálculo da Probabilidade de Reutilização na empresa Y. Neste cenário, a empresa Y obteria até 12,42% de reaproveitamento de código, e desenvolveria em média 92 das 105 funcionalidades previstas, o que significa economizar esforços de desenvolvimento em apenas 13 funcionalidades.

Tabela 3 - Cenário de aplicação da métrica de viabilidade: Caso 2

S	F	P	EsD	EcD
1A	9	11,54%	8	1
2B	10	12,99%	9	1

3C	23	13,61%	20	3
4D	19	10,98%	17	2
5D	11	14,47%	9	2
6E	8	10,13%	7	1
7F	7	8,75%	6	1
8G	13	16,67%	11	2
9H	5	6,17%	5	0
Total	**105**	**12,42%**	**92**	**13**

Para ambos os cenários ou estudos de caso se estabeleceu um critério que define que a utilização da arquitetura de microsserviços é viável caso o resultado total da Probabilidade de Reutilização seja maior ou igual a 15%. É importante enfatizar que essa métrica é representativa, utilizada como exemplo apenas para elucidar a comparação e facilitar a análise dos dados, sendo que esse valor poderia variar dependendo das necessidades da empresa, que poderá fazer um estudo de custos para chegar a uma métrica de probabilidade mais aproximada à sua realidade.

Nos estudos de caso apresentados constata-se que, quanto maior o número de funcionalidades condicionada ao número de sistemas, maiores são as chances de haver reaproveitamento de código ou ganho de desenvolvimento. Sendo assim, o Caso 1 é uma solução viável para utilização de uma arquitetura de microsserviços e o Caso 2, inviável, justamente por atingir menos de 15% de Probabilidade de Reutilização, conforme o critério que foi preestabelecido.

4. CONCLUSÃO

Este artigo possibilitou o estudo da viabilização de uma arquitetura de software orientada microsserviços, com o intuito de maximizar a quantidade de reutilização de código para a aumento produtividade no desenvolvimento de sistemas.

A utilização desta prática arquitetural proposta é justificada através da métrica de viabilidade apresentada, cujos resultados podem auxiliar as empresas a definirem se é ou não viável a implantação da arquitetura e estimar um custo-benefício. Por isso é importante destacar que a implantação dessa arquitetura é mais indicada a empresas que desenvolvem sistemas de ramos específicos para que os gastos financeiros e funcionais sejam vantajosos, já que a ideia principal é reaproveitar todas as funcionalidades desenvolvidas de modo que fiquem disponíveis através de serviços que permitam a sua utilização por diversos sistemas.

Para a criação de um modelo arquitetural robusto foi identificada e apresentada no artigo a importância da aplicação de ferramentas sólidas no mercado, como a Engenharia de Software, SOA, HTTP, REST, JSON e Desenho Arquitetural, além de aspectos de Segurança. Deste modo, conclui-se também que uma abordagem arquitetural de aplicações para reaproveitamento de código pode se entender para diversas áreas da computação aplicada.

5. REFERÊNCIAS

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. *Software Architecture in pratice*. 2ed. Boston: Pearson Addison Wesley, 1998.

BIAGIO, Luiz Arnaldo. *Planos de negócios:* estratégia para micro e pequenas empresas. 2. ed. Barueri, SP: Manole, 2012.

DEITEL, Paul; DEITEL, Harvey. *Ajax, Rich Applications e desenvolvimento web para programadores*. São Paulo: Pearson Pretince Hall, 2008.

ERL, Thomas. *SOA princípios de design de serviços*. São Paulo: Pearson Addison-Wesley, 2009.

FIELDING, Roy Thomas. *Architectural styles and the design of network-based software architectures. Doctor of Philosophy Dissertation*. Irvine: University of Californa, 2000.

FIELDING, Roy Thomas et al. *Hypertext transfer protocol: HTTP/1.1 specification*. Disponível em: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html. Acesso em: 23 abr. 2016.

GERLAN, David; SHAW, Mary. *An introduction to software architecture*. New Jersey: School of Computer Science Carnegie Mellon University Pittsburgh, 1994.

GHEEZZI, Carlo; JAZAYERI, Mehdi; MANDRIOLI, Dino. *Fundamentals of software engineering*. 2ed. New Jersey: Prentice Hall, 2002.

KUROSE, James; ROSS, Keith. *Redes de computadores e a internet:* uma abordagem top-down. 3. ed. São Paulo: Pearson Addison Wesley, 2006.

MACKENZIE, C. Matthew; LASKEY, Ken; MCCABE, Francis; BROW, Peter F.; METZ, Rebekah. *Modelo de referência para arquitetura orientada a serviços*. 1. ed. São Paulo: OASIS Open, 2006.

MACKEZIE, Duncan. *Aprenda visual basic .net em 21 Dias*. São Paulo: Pearson Education do Brazil, 2003.

MAGALHÃES, Dilmas Ribeiro. *Processo de melhoria contínua da arquitetura de software direcionado por indicadores de qualidade*. São Paulo: Instituto de Pesquisas Tecnológicas, 2005.

MEDEIROS, Ernani Sales de. *Desenvolvendo software com UML* 2.0. São Paulo: Pearson Makron Books, 2004.

METSKER, Steven John. *Padrões de projeto em Java*. Porto Alegre: Bookman, 2004.

PAULA FILHO, Wilson de Padua. *Engenharia de software:* fundamentos, métodos e padrões, 3. ed. Rio de Janeiro: LTC, 2000.

SANTOS, Wagner Roberto dos. *RESTful web services e a API JAX-RS*. São Paulo: Mundo J Magazine, 2009.

SOMMERVILLE, Ian. *Engenharia de software*. 8. ed. São Paulo: Pearson Addison-Wesley, 2007.

STALLINGS, Willian. *Criptografia e segurança de redes*. 4. ed. São Paulo: Pearson Prentice Hall, 2008.