UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ANÁLISE DO IMPACTO DAS PLATAFORMAS *PAY-AS-YOU-GO* DE COMPUTAÇÃO EM NUVEM NA CONSTRUÇÃO E PRECIFICAÇÃO DE SOFTWARE

DISSERTAÇÃO DE MESTRADO

Fernando Pires Barbosa

Santa Maria, RS, Brasil

2011

ANÁLISE DO IMPACTO DAS PLATAFORMAS *PAY-AS-YOU-GO* DE COMPUTAÇÃO EM NUVEM NA CONSTRUÇÃO E PRECIFICAÇÃO DE SOFTWARE

por

Fernando Pires Barbosa

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Mestre em Computação

Orientador: Profª.Drª. Andrea Schwertner Charão (UFSM)

Santa Maria, RS, Brasil

2011

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

ANÁLISE DO IMPACTO DAS PLATAFORMAS *PAY-AS-YOU-GO* DE COMPUTAÇÃO EM NUVEM NA CONSTRUÇÃO E PRECIFICAÇÃO DE SOFTWARE

elaborada por
Fernando Pires Barbosa

como requisito parcial para obtenção do grau de
Mestre em Computação

COMISSÃO EXAMINADORA:

Profª.Drª. Andrea Schwertner Charão (UFSM)
(Presidente/Orientador)

Prof. Dr. Mario Dantas (UFSC)

Profª. Drª. Iara Augustin (UFSM)

Santa Maria, 30 de Setembro de 2011.

AGRADECIMENTOS

À minha família, em especial aos meus pais, ao meu irmão e a todos aqueles que construíram a base de valores sobre a qual procuro conduzir minha vida. À Carol, minha eterna namorada e agora esposa, que adiciona uma pitada de paixão e amor nos nossos dias. Ao meu filho Felipe que, por uma dessas ironias do destino, nasceu justamente na semana em que finalizei o texto deste trabalho.

Aos meus amigos de infância e adolescência da grande metrópole São Sepé. À galera dos carnavais na praça, do futebol nas vilas, do Maju, da João XXIII, da Izolanda, da AABB, do Igaçu, dos bailes, das pescarias, churrascos, enfim. Depois da minha família, este é o porto seguro para onde sempre é possível voltar e reafirmar os valores. E aos novos amigos, de Santa Maria e região, que me dão o necessário complemento para a vida familiar e pessoal dos dias de hoje.

Aos professores, que desde o primário até para sempre, formaram e continuam formando a ponte através da qual o conhecimento é disseminado e ajuda a elevar a nossa sociedade.

A todos aqueles que fizeram parte da minha vida profissional, seja em Santa Maria ou nos lugares por onde passei. Todos, a sua maneira, contribuíram para o meu crescimento, especialmente as equipes dos projetos *sie* e *sim*, cujos caminhos e descaminhos se misturam com a minha história profissional até aqui.

À minha orientadora neste mestrado, que confiou no meu trabalho e conduziu com brilhante simplicidade o caminho desta dissertação, soltando e apertando as rédeas na medida certa para que fosse possível desenvolver o trabalho e construir este texto. A todos que me ajudaram durante o período de mestrado, seja revisando textos, sugerindo, criticando, apoiando ou dando dicas preciosas sobre o tal de *latex* e o até então desconhecido *java*.

Por último, uma deferência especial aos meus irmãos do *cdesp*. Não há dúvidas de que boa parte desta dissertação está ancorada nas noites em claro do final do século passado (realmente, a idade está ficando avançada).

“Essa audácia de buscar o novo
Sem pisar no rastro e reacender as brasas
É o contraponto de ter prenda e filhos
E ficar tordilho ao redor ’das casa’ ”
— CRISTIANO QUEVEDO

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

ANÁLISE DO IMPACTO DAS PLATAFORMAS *PAY-AS-YOU-GO* DE COMPUTAÇÃO EM NUVEM NA CONSTRUÇÃO E PRECIFICAÇÃO DE SOFTWARE

Autor: Fernando Pires Barbosa
Orientador: Profª.Drª. Andrea Schwertner Charão (UFSM)
Local e data da defesa: Santa Maria, 30 de Setembro de 2011.

Cloud Computing é um novo paradigma que está mudando a forma com que consumimos os recursos de TI. Uma das grandes mudanças está relacionada ao modelo de cobrança *pay-as-you-go*, em que se paga conforme o volume de recursos efetivamente consumidos. Há vários tipos de serviço oferecidos no ambiente *cloud* e a opção mais natural para os desenvolvedores de software são as Plataformas como Serviço (PaaS). Apesar de o tema *cloud computing* estar em evidência, há pouco ou nenhum estudo abordando o impacto deste modelo no desenvolvimento de software. O objetivo deste trabalho é contribuir para preencher esta lacuna e servir de base para que estudos mais detalhados sejam desenvolvidos. Para isto foi feita uma análise dos principais aspectos do desenvolvimento de software que serão afetados pelas plataformas *pay-as-you-go* de computação em nuvem, apontando itens específicos que sofrerão mudanças e evidenciando algumas destas mudanças através de um estudo de caso. Foram constatadas mudanças significativas em áreas relacionadas a precificação de software, *benchmarks* de desempenho, estimativas de desenvolvimento e engenharia de requisitos, sendo que esta última foi a que ficou mais evidente a partir do estudo de caso. Estas mudanças estão relacionadas a uma nova realidade em que o consumo racional dos recursos de TI tem impacto direto no preço do software, o que também deve gerar mudanças na forma como é tratada a otimização de código. O campo para novos estudos é vasto e este trabalho contribui para apontar alguns dos caminhos a seguir.

Palavras-chave: Computação em nuvem, precificação de software, plataformas *cloud*, *pay-as-you-go*.

ABSTRACT

Master's Dissertation
Post-Graduate Program in Informatics
Universidade Federal de Santa Maria

AN ANALYSIS OF THE IMPACT OF PAY-AS-YOU-GO CLOUD PLATAFORMS IN THE SOFTWARE DEVELOPMENT AND PRICING

Author: Fernando Pires Barbosa
Advisor: Prof[a].Dr[a]. Andrea Schwertner Charão (UFSM)

Cloud computing is a new paradigm that is changing the way we consume IT resources. One of the major changes is related to the *pay-as-you-go* pricing model. There are several kinds of cloud services being offered and the best option for software developers are Platform as a Service (*PaaS*). Even with *cloud computing* beeing a buzzword nowadays, there are few studies being developed to cover the way it will impact the software development. This thesis intents to contribute to fill this gap and provide the basis for new detailed studies about this subject. It presents an analysis about the main aspects in software delevopment that must be afected by the *pay-as-you-go* cloud model and highlights some of theses changes trough a case study. We noticed significant changes in the following areas: software pricing, performance benchmarks, software development estimation and requirements engineering. These changes are related to a new paradigm in which the IT resources used by software systems will directly impact the software price at all. This should also generate changes in how we deal with software optimization improvement. There is a large research field to be explored and this thesis contributes to point ou some directions.

Keywords: Cloud Computing, Software Pricing, Cloud Platform, code improvement.

LISTA DE FIGURAS

2.1 Camadas de abstração de serviços *cloud*, com destaque para *PaaS* (BARBOSA; CHARãO, 2011). 18
2.2 Focos de algumas plataformas quanto ao modo de utilização(BARBOSA; CHARãO, 2011). 19
2.3 Tabela e gráfico com medições do impacto financeiro da CPU no *AppEngine* 24

3.1 Comparativo: preço tradicional *vs.* preço c/ plataformas *pay-as-you-go* em nuvem. 31
3.2 Modelos baseados no COCOMO. As datas indicam o ano em que o primeiro artigo foi publicado (BOEHM; VALERDI, 2008). 32
3.3 Faixas de valor para os atributos de Plataforma do COCOMO II (BOHEM, 2000). 33
3.4 Características de qualidade da ISO9126 e ISO25010. Destaque p/ item eficência. 36
3.5 Evolução do levantamento de requisitos. Apresentação do SEI sobre qualidade de software (SEI, 2004). 37
3.6 Exemplo de formulário para controlar o atendimento aos critérios de qualidade relacionados a *Eficiência*(SEI/PSM, 2004). 38
3.7 Novo problema a ser respondido pelo processo de levantamento de requisitos. 39
3.8 Nova preocupação aos responsáveis pelo desenvolvimento. 39
3.9 Tipos de *benchmarks* SPEC disponíveis (SPEC, 2011). 41
3.10 Trecho de publicações de resultado feitas pela IBM e pela Cisco para o *benchmark SPECJEnterprise2010*. 43
3.11 Mudanças geradas pelas plataformas *pay-as-you-go* nos aspectos de desenvolvimento de software. 45
3.12 Trabalhos futuros relacionados às mundanças causadas no desenvolvimento de software pelas plataformas *pay-as-you-go*. 46

4.1 *Log* de chamadas RPC do SIE ordenado pelo tempo total de resposta (destaque para os métodos *IConsultaLocal.ConsultaAcervoBib* e *ISGCA.GetRotulo*). 49
4.2 *Log* de chamadas RPC do SIE ordenado pelo tempo total de resposta (destaque para os métodos *IConsultaLocal.ConsultaAcervoBib* e *ISGCA.GetRotulo*). 50
4.3 Tempo médio normalizado, retirando os 10% mais rápidos e 10% mais lentos (destaque: *IConsultaLocal.ConsultaAcervoBib* e *ISGCA.GetRotulo*). 50
4.4 Tempo total normalizado, retirando os 10% mais rápidos e 10% mais lentos (destaque: métodos *IConsultaLocal.ConsultaAcervoBib* e *ISGCA.GetRotulo*). 51
4.5 Os 12 métodos RPC com maior tempo de resposta total. 52
4.6 Exemplos de pontos de otimização encontrados no monitoramento do ERP SIE. 54
4.7 Métodos monitorados no SIE que possuem comportamento semelhante à medição *AppEngine* da seção 2.5. 55
4.8 Relação do número de *milisegundos* que corresponderiam ao custo de U$0,01 na plataforma *AppEngine*. 56

4.9 Projeção do preço total de uso em um dos Servidores de Aplicação do SIE, considerando que cada 328.500 *msec* de tempo de resposta incrementariam o custo em U$0,01 56
4.10 Resumo de métodos otimizáveis através de uma estrutura de *cache* simples . . 57
4.11 Os 10 usuários com maior somatório do tempo de resposta e a simulação do preço atribuído a cada um em função dos recursos consumidos 58
4.12 As 10 aplicações com maior somatório do tempo de resposta e a simulação do preço de cada uma em função dos recursos consumidos 59
4.13 Os 10 *Centros de Custo* com maior somatório do tempo de resposta e a simulação do preço apropriado para cada um 60
4.14 Esquema para o desenvolvimento de software em PaaS com modelo *pay-as-you-go* 62

LISTA DE TABELAS

2.1 Classificação dos tipos de serviço oferecidos em *cloud computing* 17
2.2 Resumo dos desafios relacionados a *cloud* computing 18
2.3 Modelo de cobrança e preços da plataforma *force.com*(SALESFORCE, 2010) 20
2.4 Modelo de cobrança e preços da plataforma *Heroku*(HEROKU, 2010) 20
2.5 Modelo de cobrança e preços da plataforma *MS AppFabric*(MICROSOFT, 2010) 21
2.6 Modelo de cobrança e preços da plataforma *Google AppEngine* (GOOGLE, 2010) 21
2.7 Quadro comparativo (resumo): *force.com, heroku, AppFabric e AppEngine* . 22
2.8 Analogia das plataformas com o modelo *pay-as-you-go* de energia elétrica .. 22
2.9 Resumo do modelos de cobrança e preço dos fornecedores *cloud IaaS* (SHANG et al., 2010) 23

3.1 Resumo de itens a considerar para determinar o preço de software (DAKIN, 1995). 29
3.2 Itens de estimativa COCOMO II. Destaque para itens relacionados a Plataforma. 32
3.3 Proposta de "novo tipo de software"p/ COCOMO 81 (GUHA; AL-DABASS, 2010). 34

LISTA DE ABREVIATURAS E SIGLAS

SDK	Software Development Kit
ERP	Enterprise Resource Planning
SIE	Sistema de Informações para o Ensino
SEI	Software Engineering Institute
IaaS	Infrastructure as a Service
DaaS	Data as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
SDK	Software Development Kit
JPA	Java Persistence API
PSM	Practical Software and Systems Measurement
MIPS	Million instructrions per second
Mflops	Million floating-point operations per second
RISC	Reduced instruction-set computing
CISC	Complex instruction-set computing
JMS	Java Message System
J2EE 5	Java Enterprise Edition 5.0
API	Application Programming Interface

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Contexto e Motivação 13
1.2 Objetivos 14
1.3 Organização do Texto 14

2 *CLOUD COMPUTING* E AS PLATAFORMAS *PAY-AS-YOU-GO* 16
2.1 *Cloud Computing* 16
2.2 Tipos de serviço e principais desafios 17
2.3 Plataformas e os modelos de cobrança 19
2.4 Tendência para os modelos de cobrança 22
2.5 Impacto financeiro de otimizações no *AppEngine* 23
2.6 Novo cenário gerado pelas plataformas *pay-as-you-go* 24
2.6.1 Incorporação do custo do hardware ao preço do software 24
2.6.2 Mudanças no desenvolvimento de software 25

3 PRECIFICAÇÃO E IMPACTO NO DESENVOLVIMENTO DE SOFTWARE .. 28
3.1 Precificação de software 28
3.1.1 Modelos atuais 28
3.1.2 Mudanças com as plataformas *pay-as-you-go* 30
3.2 Estimativas desenvolvimento (COCOMO) 31
3.2.1 Modelos atuais 31
3.2.2 Mudanças com as plataformas *pay-as-you-go* 33
3.3 Engenharia de software e Requisitos (ISO/IEC 25010) 35
3.3.1 Modelos atuais 35
3.3.2 Mudanças com as plataformas *pay-as-you-go* 38
3.4 Simulação de desempenho e *benchmarks* (SPEC) 39
3.4.1 Modelos atuais 40
3.4.2 Mudanças com as plataformas *pay-as-you-go* 42
3.5 Resumo e novos campos de pesquisa 44

4 ESTUDO DE CASO - ERP SIE 47
4.1 O estudo de caso e a nuvem *pay-as-you-go* 47
4.2 Implementação e medição 48
4.3 Análise dos Resultados 51
4.4 Simulação de Preço 54
4.5 Novas possibilidades para ERPs 57
4.6 Discussão 61

5 CONCLUSÃO E TRABALHOS FUTUROS 64

REFERÊNCIAS 67

1 INTRODUÇÃO

1.1 Contexto e Motivação

Cloud computing é um novo paradigma de computação baseado em economia de escala, que prevê a existência de uma infra-estrutura dinamicamente escalável, onde os recursos necessários são alocados e fornecidos sob demanda através da Internet (FOSTER et al., 2008). Nos últimos anos este novo paradigma vem ganhando espaço tanto na comunidade científica quanto no mercado de TI e grandes empresas como a *Amazon* (AMAZON, 2010), *Google* (GOOGLE, 2010) e *Microsoft* (MICROSOFT, 2010) vem apostando neste novo paradigma e desenvolvendo a infra-estrutura necessária para competir no mercado.

Um dos atrativos relacionados a *cloud computing* é o seu modelo de cobrança, também chamado de *pay-as-you-go*. No modelo *pay-as-you-go* os recursos de TI são oferecidos de forma ilimitada e os usuários pagam um valor proporcional ao volume de utilização real que tiverem durante um determinado período. Este mecanismo é bastante semelhante ao que estamos habituados a utilizar para o consumo de energia elétrica, em que a cobrança é feita mensalmente com base no número de KW consumidos.

Há diferentes tipos de serviço sendo oferecidos no ambiente *cloud* (HAMID R MOTAHARI-NEZHAD BRYAN STEPHENSON, 2009), e os principais são: Infra-estrutura como Serviço (*IaaS*, do inglês *Infrastructure as a Service*), Dados como Serviço (*DaaS*, do inglês *Data as a Service*), Plataforma como Serviço (*PaaS*, do inglês *Platform as a Service*) e Software como Serviço (*SaaS*, do inglês *Software as a Service*).

Os fornecedores de serviços do tipo *IaaS* já apresentam uma certa padronização no seu modelo de cobrança, que está dentro dos conceitos *pay-as-you-go* e apresenta valores na casa de centavos de dólar para itens como GB de dados transferidos e uso de CPUs (SHANG et al., 2010). Muitos estudos vem sendo feitos na área de *IaaS*, principalmente com o objetivo de otimizar a alocação dos recursos, o que é um aspecto fundamental em um ambiente baseado em economia de escala.

Entretanto, ainda há poucos trabalhos que avaliam o impacto deste novo modelo de cobrança para os desenvolvedores de software. As plataformas atuais já estão trabalhando, em maior ou menor intensidade, com o modelo *pay-as-you-go* e ainda não há uma estudo conclusivo que aponte a necessidade ou não de mudanças nas práticas de desenvolvimento de software tradicionais que são utilizadas atualmente.

Este trabalho avalia algumas das principais práticas relacionadas ao desenvolvimento de software, analisando-as sob o prisma das mudanças geradas pelo mecanismo de cobrança *pay-as-you-go*, apontando aspectos que precisarão ser revistos e evidenciando a necessidade de mudanças através de um estudo de caso com o ERP SIE [1] . Dentre os aspectos avaliados e que sofrerão mudanças encontram-se práticas relacionadas à precificação de software, estimativas de desenvolvimento, *benchmarks* de desempenho e engenharia de software e requisitos, sendo que este último ficou bastante evidenciado a partir do estudo de caso. Ainda há muitos estudos a serem feitos nesta área e este trabalho é uma contribuição para apoiar novas pesquisas sobre o tema.

1.2 Objetivos

Os objetivos deste trabalho são:

- Analisar o tema *cloud computing* sob uma ótica ainda pouco abordada pela comunidade científica: o seu impacto no desenvolvimento de software.
- Verificar em que nível o modelo de cobrança *pay-as-you-go* vem sendo aplicado pelas plataformas *cloud*.
- Identificar os aspectos do desenvolvimento de software que podem ser afetados pelas plataformas *pay-as-you-go*.
- Analisar estes aspectos e indicar quais pontos específicos necessitarão mudanças.
- Evidenciar as necessidades de mudança através de um estudo de caso.

1.3 Organização do Texto

Este trabalho está organizado da seguinte forma: o capítulo 2 apresenta os conceitos e desafios relacionados a *cloud computing* e os principais tipos de serviço oferecidos neste modelo, com destaque especial para as palataformas como serviço (*PaaS*). Também é feita uma análise das plataformas disponíveis atualmente, do novo cenário gerado pelas plataformas *pay-as-you-go* e do impacto que elas podem vir a causar no desenvolvimento de software.

O capítulo 3 avalia as práticas atuais de desenvolvimento de software, indicando pontos específicos que serão afetados e quais mudanças deverão ocorrer dentro deste novo cenário,

[1] O SIE é um ERP voltado para a administração de universidades utilizado cerca de 20 instituições brasileiras, dentre elas a própria Universidade Federal de Santa Maria (UFSM).

em que se paga pelos recursos de hardware conforme a sua real utilização. A avaliação inclui aspectos de precificação de software, estimativas de desenvolvimento, engenharia de requisitos e ferramentas para *benchmark* de desempenho.

O capítulo 4 é composto por um estudo de caso com o ERP SIE, realizando estimativas de preço e comprovando algumas das mudanças apontadas. O aspecto em que as mudanças ficam mais evidentes é a Engenharia de Requisitos, para a qual é apresentado um esquema que pode ser utilizado como base para a construção de novas práticas aderentes ao modelo *pay-as-you-go*. Ainda no capítulo 4 são discutidas novas oportunidades relacionadas ao uso de ERPs no modelo *pay-as-you-go*.

O capítulo 5 apresenta as conclusões e trabalhos futuros, resumindo o resultado das análises e as principais contribuições.

2 *CLOUD COMPUTING* E AS PLATAFORMAS *PAY-AS-YOU-GO*

A computação em nuvem (ou, em inglês, *cloud computing*) vem sendo objeto de vários estudos e muito se tem escrito e discutido sobre este tema, tanto na comunidade cientifícia como no mercado de TI. Uma das principais novidades relacionadas ao ambiente *cloud* é o seu modelo de negócio baseado em economia de escala. Vários tipos de serviço vêm sendo oferecidos no modelo *cloud*, entre eles as Plataformas como Serviço.

Este capítulo faz uma revisão dos conceitos de *cloud computing*, apresentando as suas origens em *grid computing* e os principais tipos de serviços oferecidos. Dentre os tipos de serviços, as *PaaS* surgem como alternativa natural para os desenvolvedores que desejam distribuir software em um ambiente *cloud*, por isso o capítulo também apresenta uma análise das plataformas *cloud* disponíveis atualmente, juntamente com seus modelos de cobrança. No final do capítulo, a seção 2.6 faz uma análise do potencial de impacto que as plataformas *pay-as-you-go* podem vir a causar no desenvolvimento de software.

2.1 *Cloud Computing*

A divulgação cada vez maior do termo *cloud computing* vem chamando a atenção dos profissionais de TI com a promessa de ser um novo paradigma (BUYYA; YEO; VENUGOPAL, 2008) que irá mudar a forma como os recursos de TI são comercializados. Várias definições já foram propostas para o termo *cloud computing*, sendo que uma das mais abrangentes é reproduzida a seguir:

"Cloud Computing é um paradigma de computação em larga escala que possui foco em proporcionar economia de escala, em que um conjunto abstrato, virtualizado, dinamicamente escalável de poder de processamento, armazenamento, plataformas e serviços são disponibilizados sob demanda para clientes externos através da Internet." (FOSTER et al., 2008)

Apesar de nova, a ideia de possibilitar o acesso a recursos de TI nestes moldes não é inédita, uma vez que este mesmo conceito está na origem da criação dos ambientes de *grid computing* (FOSTER et al., 2008). O termo *grid computing*, aliás, é uma analogia com as grades de fornecimento de energia (BUYYA; YEO; VENUGOPAL, 2008). Os ambientes em *grid* tiveram seu desenvolvimento durante a década de noventa, mas a expectativa original não foi cumprida totalmente e os ambientes em *grid* acabaram evoluindo segundo as necessidades das instituições científicas, que precisam de altas capacidades de desempenho computacional para realizar

determinados tipos de experimentos. Estas instituições encontraram no modelo de *grid* uma forma de compartilhar entre si os recursos que cada uma possui a sua disposição.

Cloud computing utiliza alguns dos conceitos e também algumas das tecnologias originadas durante o desenvolvimento dos ambientes *grid* (FOSTER et al., 2008). O contexto tecnológico disponível durante o período de desenvolvimento dos dois modelos pode ter dado origem a uma diferença que pode estar sendo a responsável por impulsionar o modelo *cloud* na direção de atender as expectativas surgidas durante o desenvolvimento dos ambientes *grid*. Esta diferença está relacionada ao **modelo de negócio**.

Enquanto *grid* está baseado no compartilhamento de recursos para benefício mútuo, o modelo de negócio de *cloud* está baseado na venda e utilização sob demanda destes mesmos recursos. A ideia é fornecer recursos infinitos que podem ser utilizados pagando-se uma taxa conforme o volume de utilização. Este modelo é baseado em economia de escala. O fornecedor da *cloud* investe na construção de uma infra-estrutura computacional que proporcione a ilusão de um volume infinito de recursos e os comercializa para um número de clientes alto o suficiente para tornar a operação rentável.

O modelo já vem sendo utilizado atualmente e há diferentes fornecedores e tipos de serviços disponíveis no mercado, indicando que as ideias propostas já estão em prática, ainda que não tenham alcançado todo o potencial de crescimento disponível.

2.2 Tipos de serviço e principais desafios

Com a diversidade de serviços oferecidos pelos fornecedores, várias classificações têm sido propostas para esclarecer as diferenças entre eles. Uma delas está resumida na tabela 2.1, adaptada de (HAMID R MOTAHARI-NEZHAD BRYAN STEPHENSON, 2009). Esta classificação identifica quatro tipos de serviços: *Infrastructure as a Service* (IaaS), *Data as a Service* (DaaS), *Platform as a Service* (PaaS) e *Software as a Service* (SaaS).

Tipo	Resumo	Exemplo
Infrastructure as a Service ***(IaaS)***	Disponibilização de recursos de hardware, como espaço em disco e capacidade de processamento	*Amazon S3*
Data as a Service ***(DaaS)***	Um tipo especializado de armazenamento que envolve serviços de Banco de Dados	*Amazon SimpleDB , Google Big Table*
Platform as a Service ***(PaaS)***	Provê serviços e APIs para facilitar o desenvolvimento e distribuição de aplicações	*Google AppEngine, force.com, Microsoft Azure AppFabric*
Software as a Service ***(SaaS)***	Provê aplicações inteiras acessadas diretamente pela Internet, sem necessidade de downloads ou aquisição de licenças.	*GMail , Salesforce.com , Google Docs*

Tabela 2.1: Classificação dos tipos de serviço oferecidos em *cloud computing*

Um desenvolvedor de software que queira construir e distribuir aplicações no modelo *cloud* pode fazer isso tomando como base qualquer um dos diferentes tipos de serviço existentes. A figura 2.1 mostra como uma aplicação pode ser desenvolvida e distribuída considerando os diferentes tipos de serviço como camadas de abstração. É possível construir uma aplicação e hospedá-la em um fornecedor *IaaS*, responsabilizando-se por manter as devidas configurações dos recursos de hardware e software relacionados. Outra opção seria um fornecedor *DaaS*, onde podem ser utilizadas as facilidades de serviços de banco de dados. Entretanto, a opção mais natural seria a utilização de um fornecedor *PaaS*, já que além da infra-estrutura esta opção inclui serviços que facilitam a construção, distribuição e acompanhamento do software.

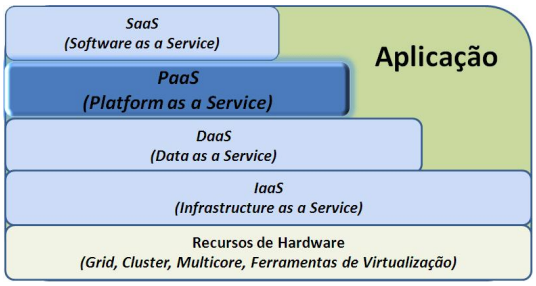


Figura 2.1: Camadas de abstração de serviços *cloud*, com destaque para *PaaS* (BARBOSA; CHARãO, 2011).

Para que os serviços *cloud* possam ser explorados em todo seu potencial, existem alguns desafios a serem vencidos. Dentre eles, destacam-se itens como segurança, escalabilidade e licenciamento (ARMBRUST et al., 2009), que estão resumidos na tabela 2.2.

Desafio	Resumo
Disponibilidade dos Serviços	Problemas do *data center*, problemas da aplicação, problemas com a Internet.
Data lock-in	Ficar preso a um Fornecedor (dados e/ou aplicações)
Segurança e Confidencialidade	Certeza de que os dados não serão acessados por terceiros. Questão mais cultural do que técnica.
Gargalos para transmissão de dados	A tecnologia para transmissão de dados tem evoluído mais lentamente que a tecnologia de armazenamento e processamento.
Escalabilidade e desempenho	"Virtualização" aplicada à capacidade de armazenamento. Forma como as aplicações irão escalar
Debug de erros	Como *debugar* erros visíveis apenas em larga escala?
Licenciamento	Os SLAs no modelo *Cloud* serão diferentes dos tradicionais que existem atualmente

Tabela 2.2: Resumo dos desafios relacionados a *cloud* computing

Tanto a comunidade científica quanto as empresas que fornecem soluções *cloud* vêm evoluindo nesses temas com uma velocidade considerável, o que pode estar sendo impulsionado pela própria disputa de mercado entre grandes empresas como *Amazon*, *Microsoft* e *Google*. Os desafios de licenciamento e escalabilidade possuem relação direta com o modelo de cobrança *pay-as-you-go* e tem destaque especial na análise apresentada na seção 2.3.

2.3 Plataformas e os modelos de cobrança

Como *PaaS* é uma opção natural para desenvolvedores que pretendem construir e distribuir software no modelo *cloud*, realizou-se um levantamento e uma breve análise das plataformas disponíveis. Em 2010 a *Wikipedia* listava as seguintes plataformas na categoria *cloud platforms* (WIKIPEDIA, 2010): *Amazon Simple Queue Service*, *Amazon SimpleDB*, *Amazon Web Services*, *AppScale*, *MS Azure Services Platform (AppFabric)*, *Caspio*, *Engine Yard*, *Force.com*, *Google App Engine*, *Sun Cloud*, *Heroku*, *Orange Scape*, *Rackspace Cloud*, *Vertebra*, *Visual WebGUI*. Atualmente, esta lista já encontra-se expandida.

Mesmo categorizadas pela *Wikipedia* como *cloud platform* e apresentando um comportamento básico que as coloca nesta categoria, elas possuem características específicas que as tornam diferentes entre si, o que pode ser identificado através das informações publicadas em seus respectivos *sites*. Uma das características em que a diferença é mais visível diz respeito ao propósito com que as plataformas são utilizadas pelos desenvolvedores. A tabela 2.2 ilustra esta diversidade de focos.

Foco da Plataforma	Nome/Fornecedor da Plataforma
Biblioteca de Serviços	Amazon Web Services (infraestrutura)
Criação Rápida de Formulários	Force.com, Caspio, Rollbase
Infra-Estrutura para Ruby on Rails	Heroku, Engine Yard
SDK para Programação e Publicação	Microsoft Azure AppFabric, Google App Engine

Figura 2.2: Focos de algumas plataformas quanto ao modo de utilização(BARBOSA; CHARÃO, 2011).

Visando manter uma representatividade mínima, a análise apresentada nesta seção selecionou fornecedores e ferramentas com diferentes focos, à exceção das plataformas classificadas como "Biblioteca de Serviços", por considerá-las mais próximas do modelo *IaaS* do que *PaaS*.

As plataformas *force.com*, *rollbase* e *caspio* são bastante semelhantes, permitindo criar formulários de forma rápida dentro de um padrão pré-estabelecido e com modelos de cobrança semelhantes. Para representá-las escolheu-se a *force.com* (SALESFORCE, 2010), que é uma das pioneiras não apenas em *PaaS*, mas também em *SaaS* com o *salesforce.com*. Também foi avaliada a plataforma *Heroku* (HEROKU, 2010) e as duas plataformas baseadas em SDKs: *AppFabric* (MICROSOFT, 2010) e *AppEngine* (GOOGLE, 2010).

A análise apresentada não é exaustiva, e dá uma atenção especial às características relacionadas ao modelo *pay-as-you-go*. O foco da avaliação está concentrado nos itens sobre escalabilidade e licenciamento (modelo de cobrança), tendo como pano de fundo uma analogia com

o modelo de consumo de energia elétrica.

A plataforma *force.com* possui a característica de escalabilidade automática, o que é importante no modelo *pay-as-you-go* na medida em que proporciona disponibilidade e transmite a sensação de "recursos infinitos". Mas o seu modelo de cobrança baseado em número de usuários e número de aplicações e tabelas está distante do modelo *pay-as-you-go* da energia elétrica, ficando mais próximo dos mecanismos tradicionais de software baseado em licença. A tabela 2.3 resume o modelo de preço da plataforma *force.com*.

Item	Free	Enterprise	Unlimited
Aplicações	1	10	*Free*
Usuários	100	$50/user/mês	$75/user/mês
Objetos/Tabelas	10	200	$2.000,00
Armazenamento	1 GB	*free*	*free*
Acesso a interfaces CRM	-	*free*	*free*
Acesso a celular	-	-	*free*
Suporte 24x7	-	-	*free*

Tabela 2.3: Modelo de cobrança e preços da plataforma *force.com*(SALESFORCE, 2010)

O modelo da plataforma *Heroku* apresenta algumas características *pay-as-you-go* onde é possível selecionar configurações de software e hardware diferentes para suportar as aplicações, inclusive através de funcionalidades específicas chamadas *add-ons*, conforme resume a tabela 2.4. Mas, além da necessidade de se contratar uma infra-estrutura pré-alocada, esta infra-estrutura não escala automaticamente, o que transfere para o desenvolvedor do software a responsabilidade de contratar mais hardware quando julgar que o desempenho da sua aplicação não está satisfatório.

Preços Heroku				
Banco de Dados			Dynos	
Tipo	Tamanho	Preço	Quantidade	Preço
Compartilhado	5MB	free	1	free
Compartilhado	20GB	$15/mês	2	$0,05/hora
Dedicado	2TB, 1 CPU	$200/mês	5	$0,20/hora
Dedicado	2TB, 5 CPU	$400/mês	10	$0,45/hora
Dedicado	2TB, 20 CPU	$1600/mês	20	$0,95/hora
Add-ons				
Tipo		Configuração	Preço	
Amanon RDS		-	*free*	
Cron		Execução diária	*free*	
Cron		Execução por hora	$3,00/mês	
SSL		Piggyback	*free*	
SSL		SNI	$5,00/mês	
SSL		Custom	$100,00/mês	
Domínio Customizado		Basic	*free*	
Domínio Customizado		Wild Card	$5,00/mês	

Tabela 2.4: Modelo de cobrança e preços da plataforma *Heroku*(HEROKU, 2010)

Já o *AppFabric* oferece um modelo mais próximo do *pay-as-you-go*, apresentando preços

na casa de centavos de dólar/hora para itens como GB de dados armazenados, GB de dados transferidos e horas de CPU utilizadas, resumido na tabela 2.5. As horas de CPU, entretanto, são faturadas com base na disponibilidade de uso e não na utilização real. Além disso, no *AppFabric* a escalabilidade é manual, o que transfere ao desenvolvedor a responsabilidade de contratar mais CPUs quando julgar necessário. Assim, apesar de praticar preços na casa de centavos de dólar/hora, estas duas limitações fazem com que o *AppFabric* não esteja tão próximo do modelo *pay-as-you-go* que estamos considerando neste trabalho.

Preços *Azure AppFabric*			
Item	**Unidade**	**Free**	**Adicional**
Largura de Banda de Entrada	Gigabyte	5 GB	$0,10
Largura de Banda de Saída	Gigabyte	5 GB	$0,15
CPU	Horas	25 horas	$0,12
Dados Armazenados	Gigabyte	500 MB	$0,15
Transações de Armazenamento	10.000 transações	10.000	$0,01
Banco de Dados Relacional	Gigabyte	-	$9,99

Tabela 2.5: Modelo de cobrança e preços da plataforma *MS AppFabric*(MICROSOFT, 2010)

O *AppEngine* é a plataforma que mais se aproxima do modelo de cobrança *pay-as-you-go* que utilizamos para consumir energia elétrica. Os preços, resumidos na tabela 2.6, são na casa de centavos de dólar/hora, mas sem as restrições existentes no *AppFabric*. No *AppEngine* também são cobrados GB armazenados, GB transferidos e horas de CPU, com a diferença de considerar somente as horas de CPU efetivamente utilizadas pela aplicação [1] . Somando isto à escalabilidade automática, que aloca (e desaloca) máquinas conforme a necessidade da aplicação, temos um modelo *pay-as-you-go* praticamente igual ao do fornecimento de energia elétrica.

Preços *AppEngine*			
Item	**Unidade**	**Free**	**Adicional**
Largura de Banda de Entrada	Gigabyte	10GB	$0,10
Largura de Banda de Saída	Gigabyte	10GB	$0,12
CPU	Horas	6,5 horas	$0,10
Dados Armazenados	Gigabyte	1GB	$0,15
E-mails enviados	Destinatário	-	$0,0001

Tabela 2.6: Modelo de cobrança e preços da plataforma *Google AppEngine* (GOOGLE, 2010)

[1]Em março de 2011 a Google anunciou mudanças no seu modelo de cobrança que passariam a vigorar em outubro. As mudanças afetam o cálculo referente a horas de CPU, que passam a considerar instâncias alocadas mas não alteram a sua essência, uma vez que as instâncias continuam sendo alocadas (e desalocadas) de forma automática. Até o momento da publicação desta dissertação estas mudanças ainda não estavam em vigor.

A tabela 2.7 resume os aspectos abordados nesta avaliação, adicionando as linguagens de programação suportadas e destacando os itens sobre o foco da plataforma, o seu modelo de cobrança e a forma com que implementa a escalabilidade.

Plataforma	Foco / Forma de Utilização	Modelo de Cobrança	Escalabilidade	Linguagem
Force.com	Construção de Formulários	Nº de usuários Nº de aplicações e tabelas	Automática	Appex
Heroku	Infra-estrutura p/ Aplicações Ruby on Rails	Infra-estrutura alocada + "add-ons"	Manual	Ruby on Rails
Micosoft Azure AppFabric	SDK para .net + infra-estrutura para hospedagem	Infra-estrutura alocada + "dados sob demanda"	Manual	.net framework
Google App Engine	SDK p/ Java e Python + infra-estrutura para hospedagem	Infra-estrutura e dados sob demanda	Automática	Java, Python

Tabela 2.7: Quadro comparativo (resumo): *force.com, heroku, AppFabric e AppEngine*

A partir da análise apresentada é possível fazer uma analogia com o modelo de distribuição e consumo de energia elétrica, identificando como cada uma das plataformas estaria posicionada em comparação com este modelo. A tabela 2.8 faz esta analogia hipotética.

Plataforma	Modelo de Cobrança	Analogia com Energia Elétrica
Force.com	Nº de usuários, Nº de aplicações e tabelas	Me diga **quantas pessoas** vão morar na sua casa e **quais os eletrodomésticos** que você tem e eu digo qual **o valor da sua conta**
Heroku	Infraesturutra alocada + *"add-ons"*	Fornecemos **geradores com várias configurações diferentes**. Você também pode **contratar serviços adicionais**, como rede estabilizada e no-breaks pagando um valor adicional por isso.
Azure AppFabric	Infraestrutura alocada + "dados sob demanda"	Temos **geradores de várias capacidades** a uma taxa fixa. **Além da taxa** cobramos **um valor proporcional** ao que você **utilizar**. Nos avise se você **precisar de mais energia do que a capacidade** do gerador, **alocamos um novo na hora**. Se quiser deixar de utilizá-lo nos avise para não cobrarmos.
App Engine	Infraestrutura e Dados "sob demanda"	Contrate o nosso serviço e comece a utilizar. **Utilize a energia livremente**. Nós vamos **medir a quantidade** que você utilizou **e enviar uma fatura** no final do mês

Tabela 2.8: Analogia das plataformas com o modelo *pay-as-you-go* de energia elétrica

2.4 Tendência para os modelos de cobrança

Aspectos de preço e licenciamento, assim como a escalabilidade estão entre os desafios apontados para *cloud computing* (ARMBRUST et al., 2009) e várias pesquisas vêm sendo realizadas nos últimos anos tratando desses temas. No caso dos serviços de infra-estrutura (*IaaS*), é possível observar uma convergência dos modelos de cobrança para a precificação na casa de centavos de dólar para itens como horas de CPU, GB armazenados por mês e GB de dados transferidos (com preços diferenciados para *download* e *upload*). Esta convergência pode ser constatada no site dos próprios fornecedores e também em alguns trabalhos científicos como o de SHANG et al. (2010), que ao propor um mercado de recursos de TI semelhante ao de *commodities* (com preços para o "mercado à vista"e "mercado futuro"), faz um resumo dos preços

praticados pelos principais fornecedores *IaaS*. Este resumo está reproduzido na tabela 2.9.

Item de Preço	Modelo de Cobrança	Amazon	Azure	Google	GoGrid	Rackspace
CPU/Processamento	Horas de CPU	$0,085/linux $0,12/windows	$0,12	$0,10	$0,10	$0,06
Storage/Armazenamento	GB/mês	$0,15	$0,15	$0,15 0,5GB free	$0,15 10GB free	$0,15
Upload de Dados	GB transferidos	$0,10	$0,10	$0,12	$free	$0,08
Download de Dados	GB transferidos	$0,17 $0,13 (se > 10TB)	$0,15	$0,10	$0,29	$0,22

Tabela 2.9: Resumo do modelos de cobrança e preço dos fornecedores *cloud IaaS* (SHANG et al., 2010)

A análise da seção 2.3 deixou claro que os modelos de preço das plataformas *cloud* ainda não apresentam o mesmo nível de convergência dos serviços de infra-estrutura (*IaaS*), com fornecedores apresentando políticas de preço bastante diferentes entre si. Entretanto, levando em consideração o conceito *pay-a-you-go* de *cloud computing* e a própria convergência dos modelos de cobrança *IaaS*, é bastante provável que os fornecedores *PaaS* sigam um caminho semelhante e adotem os mesmos itens de preço como base para o seu modelo de cobrança. Ainda que não esteja no nível de convergência dos serviços *IaaS*, a uniformização deste modelo já pode ser observada nas plataformas de grandes empresas como a Google (com o *AppEngine*) e a Microsoft (com o *AppFabric*), conforme demonstrado na seção 2.3.

2.5 Impacto financeiro de otimizações no *AppEngine*

Para testar o conceito *pay-as-you-go* foi realizado um experimento com a plataforma *Google AppEngine*: foi construída uma aplicação simples de consulta a base de dados para medir o impacto que uma otimização pontual de desempenho teria no preço final a ser pago pela aplicação, já que o item CPU é cobrado conforme as horas de uso. O *AppEngine* foi a plataforma escolhida por apresentar a estratégia de preço mais próxima do modelo *pay-as-you-go*.

A aplicação desenvolvida para o experimento foi uma operação bastante comum no desenvolvimento de software: a realização de uma consulta para recuperar todos os registros de uma determinada tabela do banco de dados. A aplicação foi feita utilizando o SDK Java do *Google AppEngine* e a tabela foi criada através das suas APIs, contendo 5 colunas e 100 registros.

A mesma aplicação foi construída de duas formas diferentes. A primeira realizava um acesso direto a tabela para recuperar todos os seus registros usando a especificação JPA. A segunda fazia este acesso uma única vez e guardava o resultado em uma *cache*, de forma que

acessos subsequentes consultam a *cache* ao invés de acessar a base de dados. Para esta implementação foi utilizada a especificação JCache. Os dados referentes ao uso de CPU foram monitorados através de uma ferramenta do *AppEngine*. Foram feitas 6.000 requisições em cada aplicação, acompanhando o consumo efetivo de CPU em intervalos pré-determinados e o resultado está demonstrado na tabela e no gráfico da figura 2.3.

Tabela de Medições

Nro. de Requisições	Aplicação Monitorada	Chamadas de Acesso a Dados	Tempo de CPU (Total)	Tempo de CPU (Datastore)	Custo em US	Custo em RS
1000	JCACHE	328	0,02	0	0,002	0,0034
1000	JPA	1639	0,1	0,07	0,01	0,017
2000	JCACHE	328	0,02	0	0,002	0,0034
2000	JPA	3025	0,18	0,15	0,018	0,0306
4000	JCACHE	328	0,03	0	0,003	0,0051
4000	JPA	5797	0,34	0,3	0,034	0,0578
6000	JCACHE	328	0,04	0	0,004	0,0068
6000	JPA	8489	0,5	0,44	0,05	0,085

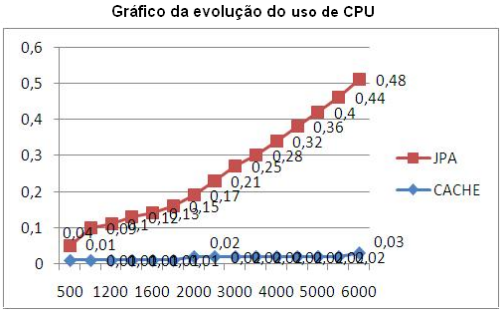


Figura 2.3: Tabela e gráfico com medições do impacto financeiro da CPU no *AppEngine*

Analisando a tabela percebe-se que as chamadas de acesso a dados permanecem praticamente constantes para a aplicação com JCache, enquanto aumentam significativamente para a aplicação JPA. Isto se explica justamente pelo uso da *cache*. Também se nota que o tempo de CPU consumido exclusivamente para acesso a dados (*Datastore*) é significativo se comparado com o tempo total. No gráfico fica claro o impacto do custo nas aplicações. Enquanto o consumo de CPU da aplicação JCache aumenta em torno de 0,01 unidade a cada 2.500 requisições, na aplicação JPA a evolução é de 0,01 unidade a cada 125 requisições (a leitura do gráfico mostra uma evolução média de 0,04 unidades a cada 500 requisições). Ou seja, utilizar *cache* nestas condições chega a ser 20 vezes mais barato do que realizar um acesso direto a base de dados.

2.6 Novo cenário gerado pelas plataformas *pay-as-you-go*

A partir da avaliação das *PaaS* e do experimento realizado é possível projetar um cenário onde os desenvolvedores de software podem construir os seus sistemas e colocá-los a disposição dos usuários em uma plataforma *cloud* com modelo *pay-as-you-go*, onde o preço é proporcional ao volume de recursos utilizado. Nesta hipótese, quanto maior a utilização do sistema maior será o preço a ser pago ao fornecedor.

2.6.1 Incorporação do custo do hardware ao preço do software

Uma das questões que surgem neste cenário é: quem irá pagar pelos recursos da plataforma que serão consumidos pelo sistema, o desenvolvedor/distribuidor do sistema ou o cliente?

Em sistemas tradicionais esta conta é geralmente paga pelo cliente. Seja em simples aplicativos de escritório ou em sistemas ERP (*Enterprise Resource Planning*) de grande porte, os custos de infra-estrutura sempre correm por conta do usuário do software. No caso dos sistemas ERP, a experiência acumulada pelo autor desta dissertação em mais de 10 anos de atividades relacionadas ao desenvolvimento, manutenção e distribuição do ERP SIE indica que o procedimento é o seguinte: o distribuidor do ERP oferece as licenças do sistema e um pacote de treinamentos, especificando ao cliente os recursos de hardware que devem ser adquiridos para suportar a operação do sistema. Entretanto, as plataformas *cloud* com modelo *pay-as-you-go* afetam este cenário, fortalecendo a possibilidade de incorporação do custo dos recursos de hardware ao preço final do software.

Por si só, a incorporação destes recursos ao preço final do software não é uma novidade, já que atualmente os sistemas podem ser hospedados em *Data Centers* e acessados pelos clientes através de uma conexão internet. A novidade agora é que somente os recursos efetivamente utilizados pelo sistema geram custos. No modelo anterior um desenvolvedor de software poderia contratar um *Data Center*, hospedar a sua aplicação e estabelecer um preço aos seus clientes. Conforme a utilização do sistema pelos usuários, os computadores contratados poderiam ficar ociosos ou sobrecarregados, mas o preço pago pelo desenvolvedor ao *Data Center* permanecia o mesmo. No novo modelo, o preço pago ao fornecedor da plataforma varia conforme a utilização do sistema pelos usuários e o consequente volume de recursos consumidos.

Por outro lado, o volume de recursos consumidos por um sistema não é função exclusiva da variação da demanda gerada por seus usuários, mas também do quanto o código-fonte está otimizado para consumir o menor volume de recursos possível. O experimento da seção 2.5 exemplifica esta situação. Neste cenário a construção de código otimizado ganha importância e alguns aspectos do desenvolvimento de software podem ser modificados.

2.6.2 Mudanças no desenvolvimento de software

A análise a seguir considera a hipótese de que i) o software é desenvolvido e distribuído através de uma plataforma *cloud*; ii) a plataforma utiliza o modelo de cobrança *pay-as-you-go*, baseado no volume de utilização de recursos; iii) os custos dos recursos consumidos pelo software são assumidos pelo seu desenvolver/distribuidor e incorporados ao preço final; e iv) o preço final não é uma função direta do volume de recursos consumidos.

A primeira mudança é também a mais evidente e está relacionada à própria existência de

funcionalidades que, por não estarem devidamente otimizadas consomem um volume de recursos maior do que poderiam. Enquanto no modelo tradicional o impacto desta falta de otimização dificilmente afetará o desenvolvedor do software, no modelo *pay-as-you-go* ela poderá consumir uma fatia do seu lucro, na medida em que ele pagará um valor maior ao fornecedor da plataforma.

O custo final de um código não otimizado irá depender do volume de utilização da funcionalidade propriamente dita, e este é um segundo aspecto em que o novo modelo de cobrança pode afetar o desenvolvimento de software. Embora se tenha a prática de identificar possíveis gargalos de desempenho e otimizá-los o máximo possível de forma a garantir um bom tempo de resposta, o foco desta otimização agora pode ser diferente. No novo modelo, além destas funcionalidades que são gargalos, a tendência é que também ganhem importância aquelas que são utilizadas de forma rotineira pelos usuários, mas que possuem uma frequência de uso significativa.

Dentro deste contexto, a avaliação do volume de uso de uma determinada funcionalidade pode se tornar uma prática recomendada durante o desenvolvimento da aplicação, como forma de identificar quais trechos de código devem ter um maior ou menor cuidado com otimização. Ainda assim, novos pontos de otimização poderão ser descobertos durante a utilização efetiva do sistema pelos usuários, uma vez que cada cliente pode utilizar o sistema de maneira diferente e dar preferência a outras funcionalidades que não aquelas previstas pelos desenvolvedores do software.

Outro aspecto recorrente, que se pode destacar a partir da experiência adquirida com o ERP SIE, é a solicitação, por parte do cliente, da adição de novas funcionalidades ou mesmo de aplicações ou sistemas inteiros à solução. O procedimento normal neste caso é fazer a estimativa do número de horas que serão necessárias e aprovar um orçamento com o cliente para que a funcionalidade seja desenvolvida. Com o modelo *pay-as-you-go*, outros dois aspectos podem ser afetados: i) como será preciso avaliar (e/ou implementar) o grau de otimização necessário a partir da expectativa do volume de uso das novas funcionalidades, é possível que os orçamentos para desenvolver a funcionalidade tenham seu preço elevado; ii) o aumento das funcionalidades do sistema tende a aumentar sua utilização como um todo, o que poderá implicar um maior consumo de recursos, refletindo no valor pago ao fornecedor da plataforma e com consequente redução no lucro do desenvolvedor/distribuidor do sistema.

Além de valorizar o trabalho relacionado à otimização de código, esta preocupação com o

uso racional de recursos durante a etapa de construção do software também pode ter impacto nas técnicas utilizadas para análise de requisitos, uma vez que a questão envolvendo o volume de uso de uma determinada funcionalidade passa a ser um ponto importante a ser levantado já nas fases iniciais do desenvolvimento do software.

Outra questão diz respeito ao próprio foco da otimização de código. No modelo tradicional os problemas de otimização costumam ter o objetivo de minimizar tanto quanto possível o tempo de resposta de uma funcionalidade e aumentar o seu *throughput*. Com o modelo *pay-as-you-go*, o problema a ser resolvido pode vir a ser formulado de maneira diferente: com o objetivo de utilizar o mínimo de recursos possível para atingir um tempo de resposta aceitável para o usuário.

Os capítulos 3 e 4 avaliam em detalhe estas possibilidades de mudanças, cuja origem se dá justamente em função de dois conceitos básicos do modelo *pay-as-you-go*: pagar apenas pelos recursos utilizados e a sensação de "recursos infinitos" oferecida pelos fornecedores *PaaS*. No modelo tradicional se trabalha com uma limitação de hardware e se procura extrair dele o máximo possível. No novo modelo o hardware é ilimitado e deve-se utilizar somente o que for estritamente necessário.

3 PRECIFICAÇÃO E IMPACTO NO DESENVOLVIMENTO DE SOFTWARE

Conforme visto nas seções anteriores, já existem plataformas *cloud* oferecendo serviços no modelo *pay-as-you-go* e o novo cenário gerado por estas plataformas pode ter impacto em algumas das práticas relacionadas à construção de software.

Este capítulo faz uma revisão das práticas atuais de desenvolvimento de software que serão afetadas, indicando i) os pontos específicos onde estas mudanças irão ocorrer, ii) quais serão estas mudanças e iii) as novas oportunidades de pesquisa que deverão surgir em função disso. A análise é feita tendo como pano de fundo a questão envolvendo o novo modelo de cobrança, segundo o qual se paga pelos recursos de hardware consumidos conforme a sua real utilização.

São avaliados neste capítulo os próprios modelos de precificação de software existentes, as técnicas para realizar estimativas de tempo e custo de desenvolvimento, práticas relacionadas à engenharia de requisitos e ferramentas para *benchmark* e simulação de desempenho. Ao final do capítulo apresenta-se um quadro comparativo, resumindo as mudanças identificadas e as novas oportunidades de pesquisa.

3.1 Precificação de software

O debate sobre o problema da precificação de software não é novo e vem sendo abordado já há alguns anos sob vários aspectos. Trabalhos da década de 90 já faziam levantamentos sobre como estabelecer um preço justo para o software (DAKIN, 1995) e mais recentemente novas questões vêm sendo abordadas, inclusive fazendo comparativos entre o modelo tradicional de licença de uso perpétua com o novo modelo de software como serviço (SaaS), que vem ganhando força nos últimos anos (KAMDAR; ORSONI, 2009).

3.1.1 Modelos atuais

Trabalhos com as mais variadas análises e propostas vêm sendo desenvolvidos sobre precificação de software, dentre eles:

- Modelos que levam em consideração o valor adicionado ao negócio do cliente (KAMDAR; ORSONI, 2009), que por sua vez estão baseados nas teorias tradicionais de marketing e economia (GRANGER, 1977; WILSON, 1972; DORWARD, 1987);

- Utilização de modelos baseados nas estratégias de negociação de ações em bolsas de va-

lores (QIN; RU-XIANG, 2010; QIN; YADI, 2010; QIN; RUXIANG, 2010; CAI; CHEN, 2008);

- Aplicação de modelos que levam em consideração a técnica de contabilidade de custos como apoio para formação de preço (ZHENG et al., 2006);

- Estudos para incorporar a sensibilidade de preço dos clientes ao processo de desenvolvimento de software, de forma a identificar módulos ou funcionalidades que devem ser priorizados durante a fase de construção do software (HARMON; RAFFO; FAULK, 2003);

Embora haja vários estudos sobre estratégias de precificação, a formação de preço para software possui alguns elementos básicos que podem ser aplicados a todos os modelos. Em um editorial de 1995 (DAKIN, 1995) a revista da IEEE faz uma análise dos itens envolvidos no estabelecimento do preço de um software, que estão resumidos na tabela 3.1.

Item	**Descrição**
1. Benefício (R$) para os clientes	Realizar uma estimativa do valor aproximado que o software irá gerar para um determinado o cliente.
2. R$ Unitário < R$ Benefício	Estabelecer um preço que seja menor que esta estimativa de "benefício percebido", de forma que o cliente tenha uma percepção de lucro ao adquirir o software.
3. Potencial de Receita (R$ Unitário * Potencial de venda)	Realizar uma estimativa do potencial de venda do software e multiplicar o preço unitário por esta estimativa com o objetivo de obter o preço total do software.
4. Pontencial de venda descontado a valor presente[2]	As vendas não ocorrerão todas ao mesmo tempo, então é necessário elaborar um fluxo de caixa com as estimativas de venda nos períodos futuros.
5. Estimar o custo de denvolvimento	Os custos para desenvolvimento do software devem ser estimados e então deduzidos do preço final de forma a encontrar o resultado.
6. Estimar reparos e manutenção	Também deve ser previsto um valor para realizar reparos e manutenção, especialmente em softwares que acompanham períodos de garantia.
7. Deduzir despesas de comercialização e distribuição	Além dos custos para desenvolver, também devem ser estimadas despesas para realizar a divulgação, comercialização e distribuição do software.
8. Perdas com pirataria	A pirataria afeta o volume de vendas do software e não deve ser deixada de lado.

Tabela 3.1: Resumo de itens a considerar para determinar o preço de software (DAKIN, 1995).

3.1.2 Mudanças com as plataformas *pay-as-you-go*

Alguns dos itens da tabela 3.1 são afetados pelo modelo baseado em plataformas *pay-as-you-go* em nuvem e um deles é o *8. Perdas com pirataria*. Como neste modelo os softwares ficam hospedados no fornecedor da plataforma, a cópia ilegal perde espaço, o que por si só já é uma mudança. Outro reflexo nas estratégias de preço está relacionado às licenças de uso perpétuas. Com a possibilidade de os recursos de hardware serem incorporados à estrutura de custo dos desenvolvedores, a tendência é que o mecanismo de licença perpétua caia em desuso, visto que se o software for utilizado *ad-eternum* isto implicará em custos *ad-eternum* para o fornecedor, o que não é um modelo sustentável. Esta questão envolvendo a mudança nos mecanismos de licenças de uso já vem sendo analisada em trabalhos que discutem o modelo de software como serviço (SaaS) (KAMDAR; ORSONI, 2009), independente de sua relação com as plataformas *pay-as-you-go*.

Embora as questões relacionadas à pirataria e aos mecanismos de licença devam originar mudanças nas estratégias de preço dos fornecedores de software, elas não chegam a afetar a base sobre a qual o desenvolvimento e precificação de software estão apoiados atualmente. A análise do capítulo 2, entretanto, indica que há mudanças mais significativas que estão por vir, e a primeira delas está relacionada ao item *3. Potencial de Receita*.

As plataformas *pay-as-you-go* em nuvem fornecem recursos de hardware infinitos e cobram um valor final que varia conforme o volume de recursos utilizados. No modelo tradicional, os custos com hardware são bancados pelos clientes, mas no modelo em nuvem a tendência é que estes custos passem a onerar o desenvolvedor/fornecedor do software. Com isto, parte da receita adquirida com a comercialização e uso do software estará comprometida para pagar o fornecedor da plataforma em que o software estiver hospedado. Assim, dentro do modelo *pay-as-you-go*, o estabelecimento do valor final de um software passa a ser dependente não apenas do seu potencial de geração de receita, mas também do volume de recursos de hardware que ele irá consumir para que esta receita seja gerada. A figura 3.1 resume esta mudança.

Neste cenário, há também uma mudança na forma como os recursos de hardware são vistos durante o processo de desenvolvimento. Atualmente, as estimativas de hardware possuem um papel acessório na construção de software, normalmente relacionado a limitações e/ou restrições que indicam uma configuração mínima na qual o software deve ser executado. No modelo *pay-as-you-go*, o software sempre deverá ser projetado para executar com o mínimo possível de hardware, já que isto afeta diretamente o lucro do seu desenvolvedor/fornecedor.

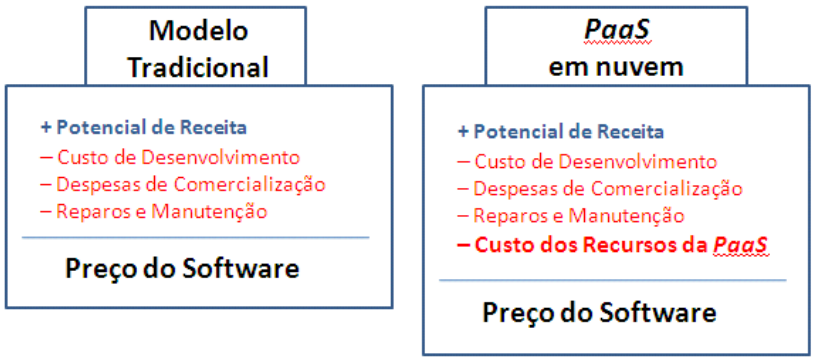


Figura 3.1: Comparativo: preço tradicional *vs.* preço c/ plataformas *pay-as-you-go* em nuvem.

Esta realidade aponta para mudanças que irão impactar no item *5. Estimar o custo de desenvolvimento* da tabela 3.1. Tanto os modelos de estimativa quanto os processos e *frameworks* de desenvolvimento de software atuais podem não estar preparados para lidar com esta nova importância adquirida pela estimativa de hardware, o que é natural considerando o papel acessório que esta atividade desempenha atualmente. Assim, os modelos atuais precisarão ser revisados para incorporar esta nova realidade. A amplitude desta mudança é avaliada a seguir.

3.2 Estimativas desenvolvimento (COCOMO)

A estimativa de custo para desenvolvimento de software teve seus estudos iniciados na década de 60 e desde então vem sendo amplamente debatida na comunidade científica. No final da década de 70 e início da década de 80 foram propostos vários modelos de estimativa, que gradualmente foram adaptados e melhorados, em um processo que continua até os dias de hoje.

3.2.1 Modelos atuais

Um dos modelos mais utilizados atualmente é o COCOMO II (Constructive Cost Model) (BOEHM; VALERDI, 2008), que tem sua origem no COCOMO 81 e possui diversas ramificações e especializações como o COSYSMO (Constructive Systems Engineering Cost Model) e o COCOTS (Commercial-off-the-Shelf Cost Models). A figura 3.2 resume os modelos construídos a partir do COCOMO.

No COCOMO II, as estimativas são feitas em função do número de homens-hora necessários para desenvolver o sistema, que é calculado com base em 22 itens de influência, dos quais 5 são de escala (com influência exponencial) e 17 são direcionadores de custo (com influência linear). Os direcionadores de custo ainda são divididos em 4 tipos diferentes: produto, equipe/pessoal, plataforma e projeto, conforme a tabela 3.2.

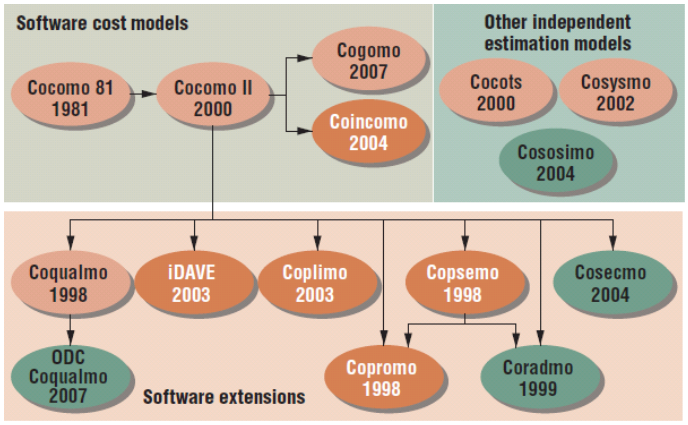


Figura 3.2: Modelos baseados no COCOMO. As datas indicam o ano em que o primeiro artigo foi publicado (BOEHM; VALERDI, 2008).

II-ItensDeEstimativaDestaquePlataforma.PNG

	Tipo do Atributo	Direcionador de Custo	Taxas					
			Very Low	Low	Nominal	High	Very High	Extra High
Direcionadores de Custo (influência linear)	Produto	Required software reliability	x	x	x	x	x	
		Database size		x	x	x	x	
		Product Complexity	x	x	x	x	x	x
		Developed for Reusability		x	x	x	x	x
		Documentation match to lifecycle needs	x	x	x	x	x	
	Plataforma	Time Constraint			x	x	x	x
		Storage Constraint			x	x	x	x
		Platform Volatility		x	x	x	x	
	Pessoal/Equipe	Analyst capability	x	x	x	x	x	
		Programmer Capability	x	x	x	x	x	
		Personnel Continuity	x	x	x	x	x	
		Application Experience	x	x	x	x	x	
		Platform Experience	x	x	x	x	x	
		Language and Toolset Experience	x	x	x	x	x	
	Projeto	Use of Software Tools	x	x	x	x	x	
		Multisite Development	x	x	x	x	x	x
		Required Development Schedule	x	x	x	x	x	x
Fatores de Escala (Influencia Exponencial)		Precedentedness	x	x	x	x	x	x
		Development Flexibility	x	x	x	x	x	x
		Architecture / Risk Resolution	x	x	x	x	x	x
		Team Cohesion	x	x	x	x	x	x
		Process Maturity	x	x	x	x	x	x

Tabela 3.2: Itens de estimativa COCOMO II. Destaque para itens relacionados a Plataforma.

Segundo a metodologia do COCOMO II, cada um dos itens da tabela 3.2 possui uma determinada influência no resultado final da estimativa e, quanto maior for a classificação de cada atributo, maior será o tempo estimado para desenvolver o software. O atributo *Required software reliability*, por exemplo, significa o nível de tolerância a falhas do software e pode variar de *Very Low* até *Extra High*, onde *Very Low* significa que a consequência de um erro é apenas o incômodo de ter de corrigi-lo, *High* significa que podem haver perdas financeiras ou grande inconveniência para os usuários e *Very High* significa que pode haver perda de vida humana.

Os atributos do tipo **plataforma** do COCOMO II estão relacionados ao ambiente de hardware e software no qual o sistema em desenvolvimento será executado. Estes são os atributos que serão mais afetados pelas plataformas *pay-as-you-go* em nuvem.

Time Constraint está relacionado ao percentual do tempo total de execução disponível que será utilizado pelo sistema e varia de *Nominal* até *Extra High*. O segundo atributo de plataforma, *Storage Constraint* está relacionado às restrições de armazenamento do software (qual o % de disco que será utilizado pela aplicação em comparação com o volume total disponível para uso). Já o terceiro e último atributo, *Platform Volatility* está relacionado com a frequência de alterações/mudanças que afetam a plataforma para a qual o software está sendo construído (se o software em construção é um sistema operacional, então a plataforma é o hardware. Se o sistema em construção for um editor de texto em rede, então a plataforma inclui os computadores, a própria rede que os interliga e os repositórios distribuídos). A figura 3.3 mostra como os valores para estes atributos devem ser ranqueados de acordo com o manual de uso do modelo COCOMO II (BOHEM, 2000).

Execution Time Constraint (TIME)

TIME Descriptors:			≤ 50% use of available execution time	70% use of available execution time	85% use of available execution time	95% use of available execution time
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.11	1.29	1.63

Main Storage Constraint (STOR)

STOR Descriptors:			≤ 50% use of available storage	70% use of available storage	85% use of available storage	95% use of available storage
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.05	1.17	1.46

Platform Volatility (PVOL)

PVOL Descriptors:		Major change every 12 mo.; Minor change every 1 mo.	Major: 6 mo.; Minor: 2 wk.	Major: 2 mo.;Minor: 1 wk.	Major: 2 wk.;Minor: 2 days	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.87	1.00	1.15	1.30	n/a

Figura 3.3: Faixas de valor para os atributos de Plataforma do COCOMO II (BOHEM, 2000).

3.2.2 Mudanças com as plataformas *pay-as-you-go*

Nos últimos anos começaram a surgir alguns trabalhos abordando o impacto do tema *cloud computing* nos modelos de desenvolvimento. Este é o caso de um artigo de 2010 (GUHA; AL-DABASS, 2010) que, ao analisar o impacto da Web 2.0 e dos ambientes *cloud* no desenvolvimento de software, propõe um ajuste no item *Software Development Mode* do COCOMO 81. O *Software Development Mode* é um item relacionado ao tipo de software que será desenvolvido e que, em sua versão tradicional possui três opções: *Organic*, *Semi-detached* e *Embedded*. A proposta do artigo é adicionar uma nova classificação chamada *Cloud Computing* conforme demonstra a tabela 3.3.

A tese do artigo é que esta nova classificação seria mais complexa que as demais e contaria

SW Proj.	a	b	c	d
Organic	2.4	1.05	2.5	.38
Semi-Detached	3.0	1.12	2.5	.35
Embedded	3.6	1.2	2.5	.32
Cloud Computing	**4**	**1.2**	**2.5**	**.3**

Tabela 3.3: Proposta de "novo tipo de software"p/ COCOMO 81 (GUHA; AL-DABASS, 2010).

para aumentar as estimativas de tempo. Em contrapartida, o desenvolvimento para ambientes *cloud* iria proporcionar maior índice de reusabilidade, o que diminuiria as linhas de código e o tempo total de desenvolvimento.

Esta tese é bastante simplista e insuficiente para abordar as mudanças originadas pelas plataformas *pay-as-you-go*. Ressalte-se, inicialmente, que a dependência do COCOMO 81 em relação ao número de linhas de código e a complexidade cada vez maior dos ambientes de desenvolvimento foram alguns dos motivos que levaram os pesquisadores a substituí-lo pelo COCOMO II. Além de propor um modelo para solucionar um problema que já foi contornado pelo COCOMO II (a complexidade dos ambientes de desenvolvimento), o artigo não aborda nenhum aspecto específico do modelo *pay-as-you-go* em nuvem, como por exemplo o conceito de "recursos infinitos"e a incorporação destes recursos ao preço final do software.

O capítulo 2 apresentou o modelo *cloud computing* e como ele lida com o conceito de "recursos infinitos", demonstrando que já existem fornecedores de plataforma em nuvem trabalhando com este conceito e oferecendo soluções *pay-as-you-go* para os desenvolvedores de software. Neste cenário, os itens *Execution Time* e *Storage Constraint* do COCOMO II precisam ser revistos, já que tanto os recursos de CPU quanto os de armazenamento estarão sempre disponíveis.

Por outro lado, conforme detalhado na seção 3.1, o uso destes recursos passará a ter impacto no preço final do software, o que traz um novo problema a ser considerado nas estimativas de desenvolvimento: como saber antecipadamente o volume de recursos que será consumido por uma determinada aplicação? E quão otimizado deverá ser o código da aplicação para que o volume de recursos consumido por ela não comprometa o preço final do software? É preciso avaliar os modelos e processos de engenharia de software atuais para verificar se eles dão o devido suporte para que estas questões sejam respondidas.

3.3 Engenharia de software e Requisitos (ISO/IEC 25010)

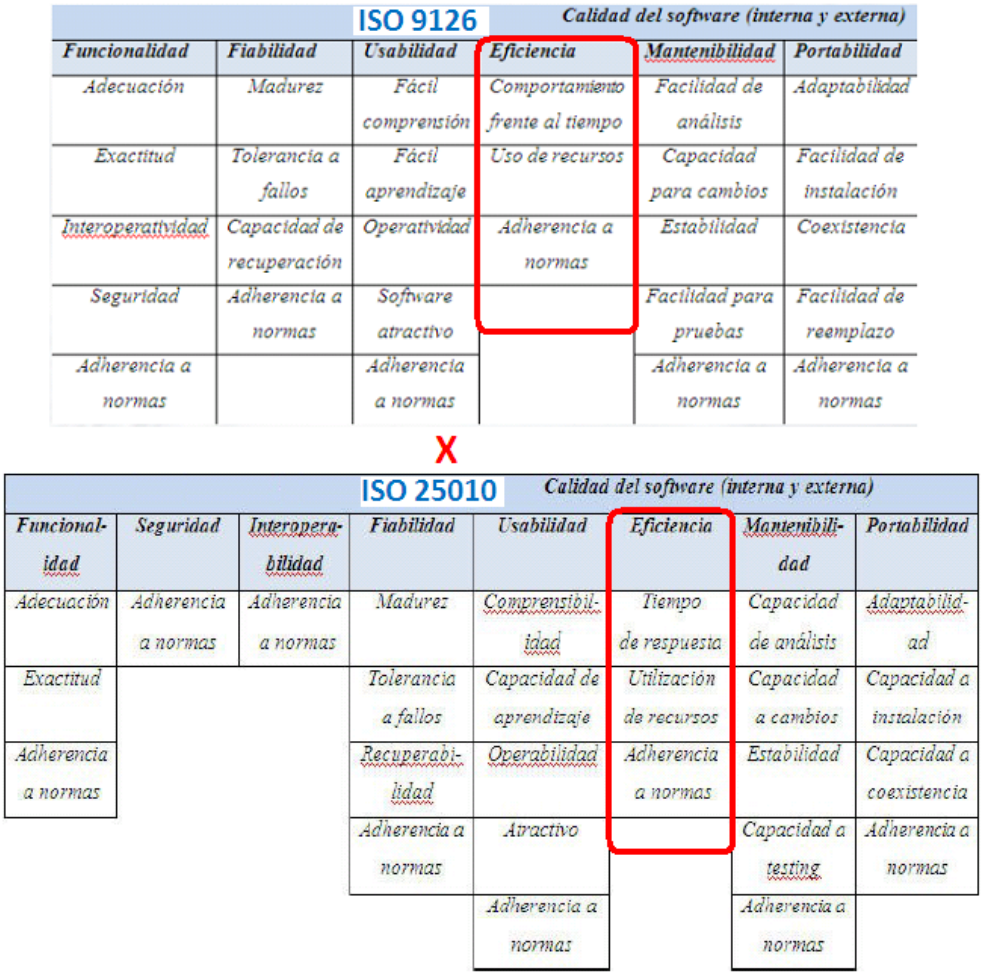
Um dos aspectos mais importantes do desenvolvimento de software é a engenharia de requisitos, que de uma forma geral é reconhecida pela literatura como um dos pontos mais críticos e complexos do processo de desenvolvimento de software (PANDEY; SUMAN; RAMANI, 2010). Entende-se a procedência desta preocupação, já que são os requisitos que irão determinar como o sistema irá se comportar e quais funcionalidades ele irá conter. A construção dos requisitos do sistema é um processo sistemático que começa logo nas primeiras fases do desenvolvimento e continua durante boa parte do tempo com o intuito de representar da melhor forma possível todas as funcionalidades que o software irá conter.

3.3.1 Modelos atuais

Os requisitos são normalmente classificados de duas formas: *Requisitos Funcionais* e *Requisitos Não-Funcionais* (JALOTE, 1991). Os requisitos *funcionais* são aqueles que descrevem as ações que serão realizadas pelo sistema independentemente do ambiente em que ele será executado, desconsiderando portanto quaisquer particularidades ou restrições referentes ao meio físico (hardware, rede, etc.). Os requisitos *não-funcionais* são os que descrevem justamente o ambiente físico e eventuais restrições de implementação, desempenho, plataforma, disponibilidade, etc. Um documento de especificação de requisitos (tambem chamado SRS - *Software Requirement Specification*) contempla tanto os requisitos *funcionais* quanto os *não-funcionais*, relacionando-os de tal forma que seja possível identificar como as funcionalidades do software se comportam dentro do ambiente no qual ele será executado. Assim, é comum associar aos requisitos atributos como: tempo de resposta, disponibilidade, portabilidade, capacidade e tempo de *recovery*, entre outros (PANDEY; SUMAN; RAMANI, 2010).

Sendo um tema importante na Engenharia de Software, os requisitos também são objeto de estudo da área de qualidade. A norma ISO/IEC 9126 (ISO, 2001) estabeleceu os padrões de qualidade para software e sua versão mais recente é a ISO/IEC 25010 (ISO, 2011). A nova norma é uma revisão da ISO/IEC 9126 e, mesmo acrescentando alguns itens, não mudou a estrutura básica original (AL-QUTAISH, 2009), conforme pode ser visto na figura 3.4.

O trecho em destaque na figura 3.4 está relacionado à característica de *eficiência*, que é definida pela norma ISO 9126 como: "*a capacidade do produto de software de apresentar desempenho apropriado, relativo à quantidade de recursos usados, sob condições pré-estabelecidas*". A *Eficiência* ainda é dividida em três sub-características): i) *Comportamento em Relação ao*



ISO 9126 — Calidad del software (interna y externa)					
Funcionalidad	**Fiabilidad**	**Usabilidad**	**Eficiencia**	**Mantenibilidad**	**Portabilidad**
Adecuación	Madurez	Fácil comprensión	Comportamiento frente al tiempo	Facilidad de análisis	Adaptabilidad
Exactitud	Tolerancia a fallos	Fácil aprendizaje	Uso de recursos	Capacidad para cambios	Facilidad de instalación
Interoperatividad	Capacidad de recuperación	Operatividad	Adherencia a normas	Estabilidad	Coexistencia
Seguridad	Adherencia a normas	Software atractivo		Facilidad para pruebas	Facilidad de reemplazo
Adherencia a normas		Adherencia a normas		Adherencia a normas	Adherencia a normas

X

ISO 25010 — Calidad del software (interna y externa)							
Funcional-idad	**Seguridad**	**Interopera-bilidad**	**Fiabilidad**	**Usabilidad**	**Eficiencia**	**Mantenibili-dad**	**Portabilidad**
Adecuación	Adherencia a normas	Adherencia a normas	Madurez	Comprensibil-idad	Tiempo de respuesta	Capacidad de análisis	Adaptabilid-ad
Exactitud			Tolerancia a fallos	Capacidad de aprendizaje	Utilización de recursos	Capacidad a cambios	Capacidad a instalación
Adherencia a normas			Recuperabi-lidad	Operabilidad	Adherencia a normas	Estabilidad	Capacidad a coexistencia
			Adherencia a normas	Atractivo		Capacidad a testing	Adherencia a normas
				Adherencia a normas		Adherencia a normas	

Figura 3.4: Características de qualidade da ISO9126 e ISO25010. Destaque p/ item eficência.

Tempo: tempos de resposta adequados; ii) *Utilização de Recursos*: utilizar tipos e quantidades apropriadas de recursos e iii) *Conformidade*: estar de acordo com normas e padrões relacionados a eficiência.

Assim, em Engenharia de Software, as questões relacionadas a desempenho e consumo de recursos são tratadas como requisitos *não-funcionais* e estão diretamente relacionadas com atributos da qualidade de software, mais especificamente no que diz respeito às suas características de *eficiência*. Esta eficiência é medida a partir de um critério objetivo, estabelecido previamente e que deverá ser obedecido pelo software. Um exemplo de critério objetivo pode ser o seguinte: *"em um servidor com 1GB de memória e dois processadores de 2GHz operando com uma carga de 400 usuários simultâneos, 95% das requisições devem retornar em até 5 segundos e 90% das requisições em até 3 segundos, com um volume de ocupação máximo de CPU e memória igual a 50%."*

Em uma apresentação (SEI, 2004) sobre qualidade de software realizada no SEPG 2004 Europa [1] , o SEI mostra um diagrama que exemplifica como as necessidades dos usuários e partes interessadas (*stakeholders*) vão sendo identificadas e detalhadas até que se tenha uma visão de quais são os requisitos *funcionais* e *não-funcionais* do software, que em última instância estarão relacionados com os atributos da qualidade. O diagrama da figura 3.5 é parte desta apresentação e resume esta visão unificada.

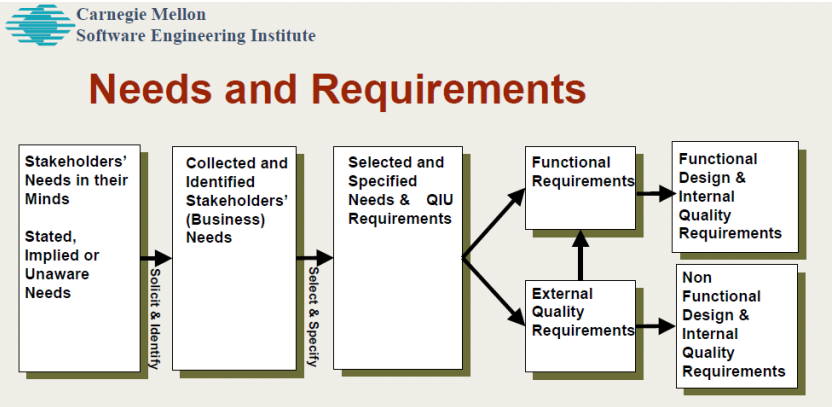


Figura 3.5: Evolução do levantamento de requisitos. Apresentação do SEI sobre qualidade de software (SEI, 2004).

Com os requisitos *não-funcionais* identificados, contendo critérios de *eficiência* bem definidos, é possível realizar medições para verificar se o software construído atende ou não a estes critérios, e é desta forma que as questões relacionadas ao consumo de recursos são tratadas durante o processo de desenvolvimento de software. A figura 3.6 é parte de uma apresentação(SEI/PSM, 2004) feita pelo SEI em um dos eventos do PSM (PSM, 2011), que é um projeto patrocinado pelo Departamento de Defesa dos EUA que tem por objetivo desenvolver práticas relacionadas a medição de qualidade em software. Ele demonstra um exemplo de formulário que pode ser utilizado para controlar se o software atende ou não aos requisitos de *eficiência*.

O capítulo 2 e a seção 2.6 apresentaram um novo cenário que está sendo criado pelas plataformas *pay-as-you-go*, onde o volume de recursos disponíveis é infinito e a sua utilização é cobrada sob demanda, de forma proporcional ao que está efetivamente sendo utilizado. A seção 3.1 apresenta os argumentos indicando que esta abordagem deverá ter impacto no preço final do software.

[1] O SEPG (SEI/SEPG, 2011) é uma das principais conferências da área de Engenharia de Software.

Internal quality measurement category				
CHARACTERISTIC	SUBCHARACTERISTIC	MEASURES	REQUIRED LEVEL	ASSESSMENT ACTUAL RESULT
Functionality	Suitability			
	Accuracy			
	Interoperability			
	Security			
	Compliance			
Reliability	Maturity			
	Fault tolerance			
	Recoverability (data, process,			
	Compliance			
Usability	Understandability			
	Learnability			
	Operability			
	Attractiveness			
	Compliance			
Efficiency	Time behaviour			
	Resource utilisation			
	Compliance			
Maintainability	Analyzability			
	Changeability			
	Stability			
	Testability			
	Compliance			

Figura 3.6: Exemplo de formulário para controlar o atendimento aos critérios de qualidade relacionados a *Eficiência*(SEI/PSM, 2004).

3.3.2 Mudanças com as plataformas *pay-as-you-go*

Em uma abordagem voltada para critérios relacionados à qualidade de software, as questões relacionadas ao consumo de recursos são tratadas como requisitos *não funcionais* Mas, em um cenário onde o volume de recursos consumidos afeta diretamente o lucro do desenvolvedor/fornecedor do software, o tratamento dado ao consumo de recursos ganha contornos mais estratégicos e o próprio foco do levantamento de requisitos deve sofrer alterações.

A abordagem tradicional induz o processo de levantamento de requisitos a identificar as funcionalidades que poderão ter um grande número de acessos simultâneos. Isto é feito para que os responsáveis por projetar e codificar o sistema deem a estas funcionalidades um tratamento especial, com o objetivo de garantir um bom tempo de resposta dentro de determinadas configurações de hardware, conforme orientam os critérios de qualidade.

Dentro da nova realidade, há um outro aspecto que ganha importância durante o levantamento de requisitos: estimar o volume total de recursos que será consumido pelo software. A figura 3.7 ilustra esta diferença. A otimização de um sistema para que ele tenha um consumo racional de recursos é uma tarefa bastante diferente do que a de prepará-lo para operar dentro de critérios de qualidade como "tempo de resposta".

Assim, há novas informações que precisam ser identificadas durante o processo de levantamento de requisitos. Estas novas informações têm por objetivo responder às seguintes per-

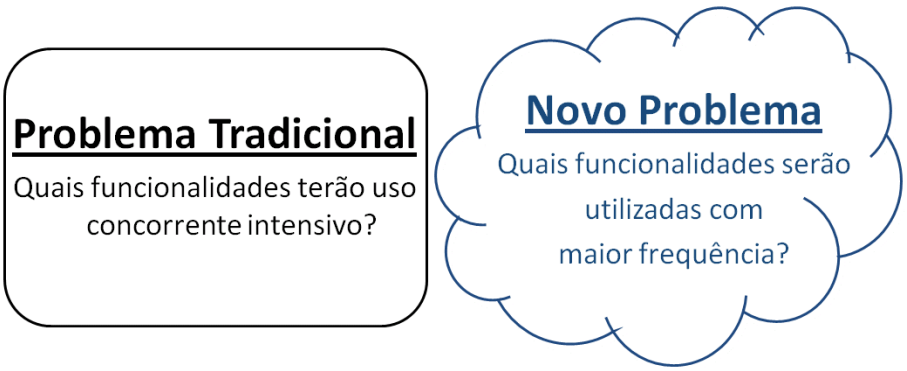


Figura 3.7: Novo problema a ser respondido pelo processo de levantamento de requisitos.

guntas: Quantos usuários irão utilizar uma determinada funcionalidade do sistema? Com que frequência estes usuários irão acessá-la? A resposta a estas perguntas irá indicar quais funcionalidades devem ter a sua otimização priorizada como forma de racionalizar o consumo de recursos e reduzir o valor pago ao fornecedor da plataforma, o que adiciona uma nova preocupação aos responsáveis por projetar e codificar o sistema, ilustrada na figura 3.8.

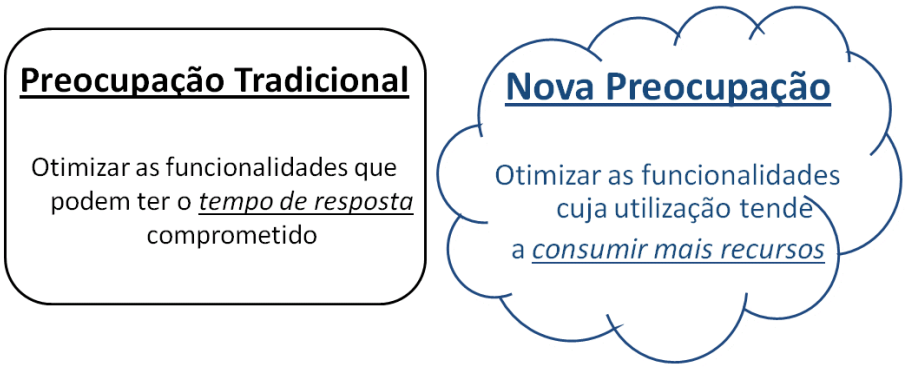


Figura 3.8: Nova preocupação aos responsáveis pelo desenvolvimento.

Este novo foco para o consumo de recursos aponta para um outro aspecto que precisa ser avaliado, que diz respeito às ferramentas disponíveis atualmente para realizar simulações e *benchmark* de desempenho. A próxima seção avalia como estas ferramentas serão afetadas.

3.4 Simulação de desempenho e *benchmarks* (SPEC)

A avaliação de desempenho dos sistemas de computador é, há vários anos, objeto de estudo da comunidade científica, que vem desenvolvendo e adaptando ferramentas e *benchmarks* com a finalidade de comparar vários aspectos relacionados à capacidade de processamento dos sistemas.

3.4.1 Modelos atuais

Até o final da década de 80 os principais instrumentos de medição eram MIPS e Mflops. Embora continue em uso atualmente, esta abordagem perdeu espaço com as diferenças de arquitetura entre RISC e CISC e também pelo fato de serem medições restritas à velocidade da CPU, desconsiderando aspectos importantes como uso de memória, disco e tempo de resposta de um sistema como um todo (GILADI; AHITAV, 1995).

Diante deste problema e também da necessidade de se obter dados confiáveis sobre a real capacidade de desempenho dos sistemas de computadores disponíveis, foi fundada em 1988 a *Standard Performance Evaluation Corporation* (*SPEC*) (SPEC, 2011). A SPEC começou com um pequeno número de fornecedores de hardware e atualmente é uma organização com mais de 60 membros que divulga, trimestralmente, centenas de resultados de medições de desempenho.

A SPEC funciona como uma entidade que desenvolve *benchmarks* para serem utilizados como ferramenta para medição de desempenho. Estes *benchmarks* são algoritmos ou sistemas completos desenvolvidos em uma determinada linguagem que podem ser comprados pelos fornecedores que desejam submeter as suas soluções a avaliações de desempenho.

O processo para divulgar o resultado de um *benchmark* SPEC segue algumas regras para garantir a confiabilidade das informações. Após comprar o *benchmark*, cada fornecedor realiza os testes no ambiente em que deseja realizar a medição e, ao reportar o resultado para a SPEC, informa todos os parâmetros, especificações, softwares e configurações que utilizou, de forma que a SPEC seja capaz de reproduzir os testes com o intuito de garantir a sua autenticidade. Após todas as conferências, o resultado pode ser divulgado e é possível então fazer um comparativo entre diversos fornecedores.

Os *benchmarks* oferecidos pela SPEC cobrem vários tipos de medição, cada uma delas com enfoque e objetivos diferentes, que vão desde desempenho de CPU e estações de trabalho gráficas até servidores web, aplicações Java, consumo de energia e servidores de e-mail. A figura 3.9 mostra como estão agrupados os *benchmarks* SPEC disponíveis atualmente.

Dentre os tipos de *benchmarks*, os que possuem uma relação mais próxima com o desenvolvimento de software são os relacionados a aplicações *Java* cliente/servidor. O *SPECjvm2008* e o *SPECjms2007* são menos aderentes que os demais. Enquanto o primeiro mede o desempenho da máquina virtual *Java*, o segundo é muito específico, ficando restrio à medição de uma solução *JMS*. Os que mais se adaptam são o *SPECJEnterprise2010* e o *SPECjbb2005*, sendo que o primeiro é mais representativo que o segundo. O *SPECjbb2005* mede o desempenho de

Tipo do Benchmark	Benchmarks disponíveis
CPU	SPEC CPU2006 SPEC CPUv6
Estações de Trabalho e Estações Gráficas	SPECviewperf® 11 SPECapcSM for 3ds Max™ 2011 SPECapcSM for Lightwave 3D® 9.6 SPECapcSM for Maya® 2009 SPECapcSM for Pro/ENGINEER™ Wildfire 2.0 SPECapcSM for Solid Edge V19™ SPECapcSM for SolidWorks 2007™ SPECapcSM for UGS NX 4
Computação de alto de desempenho,OpenMP e MPI	SPEC MPI2007 SPEC OMP2001
Java Cliente/Servidor	SPECjbb2005 SPECjEnterprise2010 SPECjvm2008 SPECjms2007
Servidores de e-mail	SPECmail2009 SPECmail2001
Sistemas de arquivo em rede	SPECsfs2008

Tipo do Benchmark	Benchmarks disponíveis
SIP - Session Initiation Protocol	SPECsip
Consumo de energia	SPECpower_ssj2008 SPECvirt_sc2010 (em desenvolvimento) SPECweb2009 (em desenvolvimento)
SOA	SOA
Virtualização	SPECvirt_sc2010
Servidores Web	SPECweb2009 SPECweb2005

Figura 3.9: Tipos de *benchmarks* SPEC disponíveis (SPEC, 2011).

uma aplicação cliente/servidor típica em *java*, mas não inclui aspectos *enterprise*, como por exemplo o acesso a banco de dados. Já o *SPECJEnterprise2010* foi desenvolvido, como o próprio nome diz, para cobrir todos os aspectos relacionados a uma aplicação *enterprise* típica, medindo o desempenho de soluções que implementam a especificação J2EE 5, a especificação *Java* voltada para desenvolvimento corporativo e que é bastante utilizada pelos desenvolvedores de software em geral, incluindo aí o Servidor Web, o Servidor de Aplicação, o Banco de Dados e os próprios serviços *JMS*.

O *SPECJEnterprise2010* (SPECJENTERPRISE, 2010) foi criado para substituir o *SPECjAppServer2004*, que possuía a mesma finalidade mas já estava obsoleto por não considerar tecnologias que foram sendo adicionadas na especificação JEE [2] ao longo dos anos, como o serviço de mensagens por exemplo.

A unidade de medida do *SPECJEnterprise2010* é o EjOPS - *Enterprise jAppServer Operations per Second*. Antigamente, além do EjOPS, também era fornecida uma informação sobre a relação preço/performance (preço/EjOPS), mas isto foi descontinuado a partir da versão *SPECJAppServer2004* devido à dificuldade em calcular e garantir a informação correta, já que era preciso verificar todos os componentes de preço de cada um dos fornecedores que publicavam resultados. Embora a relação preço/EjOPS não seja divulgada pela SPEC, qualquer pessoa ou fornecedor tem a liberdade de fazer o cálculo e divulgar esta relação, já que a publicação do resultado inclui toda a especificação de software e hardware que foi utilizada para realizar a

[2] O *SPECJAppServer2004* atendia a especificação JEE 3.0, enquanto o *SPECJEnterprise2010* atende a JEE 5.0

medição. A única ressalva nesse caso é que as questões de preço não são homologadas pela SPEC, ficando a critério de cada um fazer as devidas conferências para verificar se a relação preço/EjOPS divulgada está correta.

A figura 3.10 mostra um exemplo resumido de como são publicados os resultados para o *SPECJEnterprise2010*. Conforme se pode observar, são divulgados detalhes tanto do hardware como dos softwares utilizados, incluindo também eventuais configurações específicas de software. A publicação da *Cisco*, por exemplo, utiliza produtos Oracle como o Servidor de Aplicação *WebLogic 10.3.4* e o SGBD *Database 11g* junto a equipamentos da própria *Cisco* com até 64 núcleos para produzir um resultado de 17.301,86 EjOPS. Já a publicação da *IBM* utiliza como Servidor de Aplicação o *IBM WebSphere V7* e o SGBD *IBM DB2 9.7* em conjunto com equipamentos IBM com até 8 núcelos e produz um resultado de 1.103,40 EjOPS. Conforme se pode deduzir, para calcular a relação preço/EjOPS basta verificar o preço de todos os softwares e equipamentos fornecidos por cada publicação e fazer um cálculo aritmético simples.

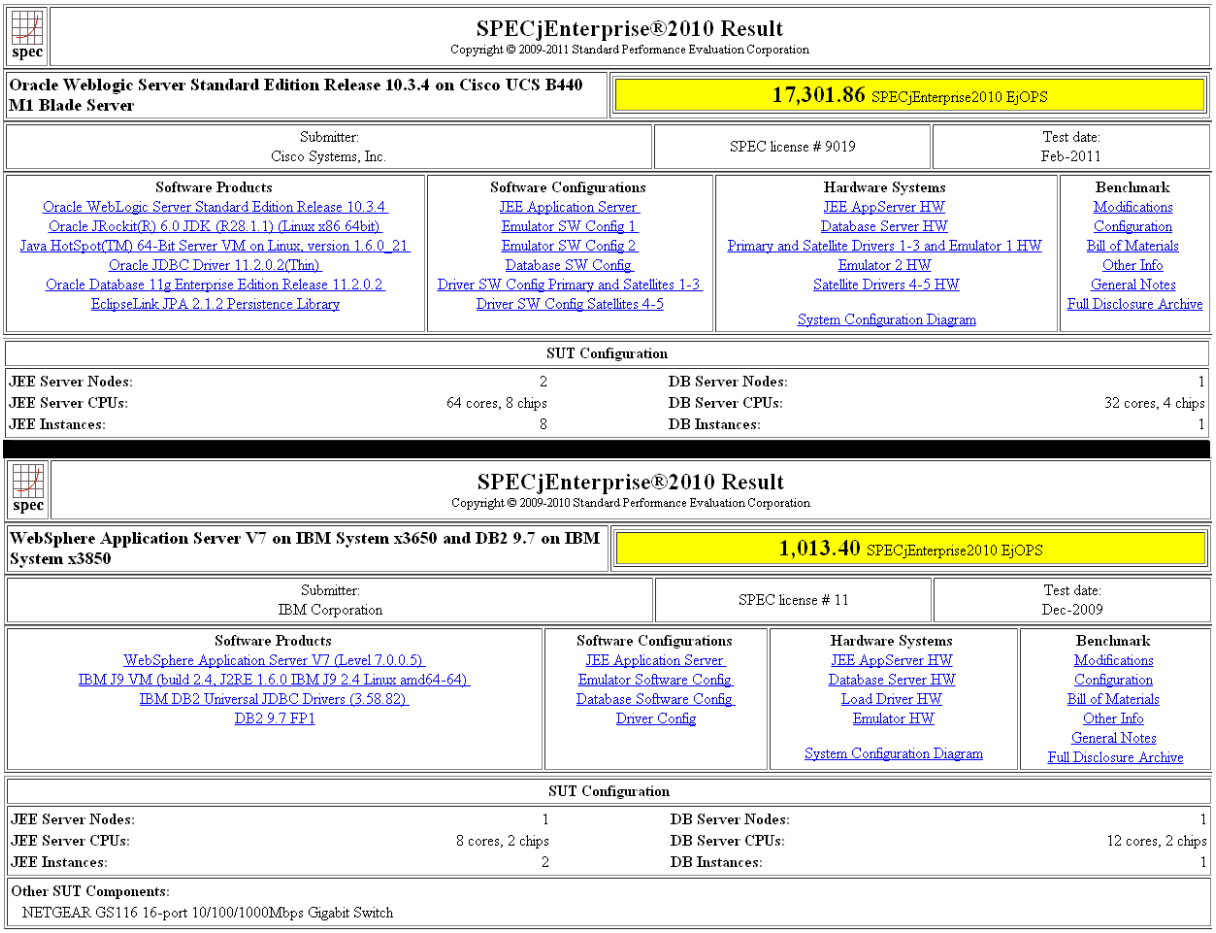
E como isto se relaciona com o desenvolvimento de software atual? A plataforma J2EE é uma especificação padrão que é implementada por diversos fornecedores. Então, as organizações que desenvolvem sistemas nesta plataforma orientam os seus clientes a comprarem os produtos que atendem a esta especificação. De forma análoga, também indicam uma configuração de hardware recomendada para que o sistema que está sendo entregue tenha um desempenho aceitável (conforme os parâmetros de qualidade apresentados na seção 3.3). O desenvolvedor do software até pode recomendar um determinado fornecedor de SGBD ou de Servidor de Aplicação, mas a decisão sobre quais produtos e quais equipamentos comprar fica a critério do cliente, que pode utilizar as medições EjOPS (ou, se for o caso, uma relação preço/EjOPS) como apoio para a sua tomada de decisão.

Mas, em um cenário onde tanto os produtos quanto os equipamentos são fornecidos sob demanda pelo fornecedor da plataforma *cloud* em um mecanismo de cobrança *pay-as-you-go*, como ficaria este modelo? Ele seria o mais adequado para os desenvolvedores de software?

3.4.2 Mudanças com as plataformas *pay-as-you-go*

As medições tradicionais, como as realizadas pelos *benchmarks* da SPEC, precisarão ser modificadas. A informação referente a EjOPS, por exemplo, é uma boa medida de desempenho mas não possui nenhuma relação de preço.

Além disso, a informação é divulgada tendo como base uma determinada configuração de



SPECjEnterprise®2010 Result

Copyright © 2009-2011 Standard Performance Evaluation Corporation

Oracle Weblogic Server Standard Edition Release 10.3.4 on Cisco UCS B440 M1 Blade Server

17,301.86 SPECjEnterprise2010 EjOPS

Submitter: Cisco Systems, Inc.	SPEC license # 9019	Test date: Feb-2011

Software Products	Software Configurations	Hardware Systems	Benchmark
Oracle WebLogic Server Standard Edition Release 10.3.4	JEE Application Server	JEE AppServer HW	Modifications
Oracle JRockit(R) 6.0 JDK (R28.1.1) (Linux x86 64bit)	Emulator SW Config 1	Database Server HW	Configuration
Java HotSpot(TM) 64-Bit Server VM on Linux, version 1.6.0_21	Emulator SW Config 2	Primary and Satellite Drivers 1-3 and Emulator 1 HW	Bill of Materials
Oracle JDBC Driver 11.2.0.2(Thin)	Database SW Config	Emulator 2 HW	Other Info
Oracle Database 11g Enterprise Edition Release 11.2.0.2	Driver SW Config Primary and Satellites 1-3	Satellite Drivers 4-5 HW	General Notes
EclipseLink JPA 2.1.2 Persistence Library	Driver SW Config Satellites 4-5	System Configuration Diagram	Full Disclosure Archive

SUT Configuration

JEE Server Nodes:	2	**DB Server Nodes:**	1
JEE Server CPUs:	64 cores, 8 chips	**DB Server CPUs:**	32 cores, 4 chips
JEE Instances:	8	**DB Instances:**	1

SPECjEnterprise®2010 Result

Copyright © 2009-2010 Standard Performance Evaluation Corporation

WebSphere Application Server V7 on IBM System x3650 and DB2 9.7 on IBM System x3850

1,013.40 SPECjEnterprise2010 EjOPS

Submitter: IBM Corporation	SPEC license # 11	Test date: Dec-2009

Software Products	Software Configurations	Hardware Systems	Benchmark
WebSphere Application Server V7 (Level 7.0.0.5)	JEE Application Server	JEE AppServer HW	Modifications
IBM J9 VM (build 2.4, J2RE 1.6.0 IBM J9 2.4 Linux amd64-64)	Emulator Software Config	Database Server HW	Configuration
IBM DB2 Universal JDBC Drivers (3.58.82)	Database Software Config	Load Driver HW	Bill of Materials
DB2 9.7 FP1	Driver Config	Emulator HW	Other Info
		System Configuration Diagram	General Notes
			Full Disclosure Archive

SUT Configuration

JEE Server Nodes:	1	**DB Server Nodes:**	1
JEE Server CPUs:	8 cores, 2 chips	**DB Server CPUs:**	12 cores, 2 chips
JEE Instances:	2	**DB Instances:**	1

Other SUT Components:
NETGEAR GS116 16-port 10/100/1000Mbps Gigabit Switch

Figura 3.10: Trecho de publicações de resultado feitas pela IBM e pela Cisco para o *benchmark SPECJEnterprise2010*.

hardware. No ambiente das plataformas *pay-as-you-go* estas configurações são transparentes para o desenvolvedor de software, que terá acesso a uma fatura mensal informando qual é o preço a ser pago pela infra-estrutura de hardware que foi consumida no período.

Medições com a característica de "*volume de processamento* x *configuração de hardware*"deixam de ter relevância em um ambiente *cloud pay-as-you-go*. Nestes ambientes, seriam necessárias medições que informassem o preço para se executar uma determinada carga de trabalho em um determinado fornecedor *cloud* e não qual a sua capacidade de processamento. Até porque, em tese, a capacidade de processamento do fornecedor *cloud* é infinita.

Há ainda um outro aspecto a ser considerado. De uma forma geral os *benchmarks* estão focados em capacidade de processamento. Mas, conforme demonstra a seção 2.4, a tendência é que os itens de preço das plataformas *cloud* incluam não apenas processamento, mas também armazenamento e transferência de dados. Então, estes itens também devem ser considerados nas medições futuras.

Em 2010 o Departamento de Ciência da Computação da Universidade de Duke (CS, 2010) publicou um artigo onde introduziu o problema relacionado a comparação de diferentes fornecedores *cloud* e apresentou o *CloudCmp* (LI et al., 2010), uma ferramenta para comparar o custo e o desempenho destes fornecedores. O trabalho é bastante detalhado e além de propor algumas métricas para serem utilizadas, também apresenta medições (tanto de preço como de desempenho) que foram realizadas com alguns dos principais fornecedores *cloud* disponíveis atualmente: *Amazon*, *Microsoft*, *Google* e *Rackspace*.

Embora não tenha sido focado exclusivamente em plataformas [3] , o trabalho desenvolvido pelos autores do *CloudCmp* dá uma boa ideia de como deverão ser construídos os *benchmarks* dentro da nova realidade proporcionada pelas plataformas *pay-as-you-go* em nuvem. Dentre as medições identificadas e demonstradas pelo trabalho estão itens como: *latência*, *tempo de resposta* e, principalmente, *custo vs. desempenho*. As medições foram feitas com aplicações de três tipos: uso intensivo de CPU, uso intensivo de memória e uso intensivo de I/O. Também foram feitas medições com base no TPC-W, um *benchmark* padrão para *web services* transacionais.

Embora as medições do *CloudCmp* possam ser utilizadas para se ter uma ideia inicial de custo/desempenho dos fornecedores *cloud*, como o próprio artigo cita em seus trabalhos futuros, há ainda um grande horizonte de mudanças a serem desenvolvidas nesta área. O artigo cita como trabalho futuro a utilização dos dados do *CloudCmp* para predizer o custo/desempenho de aplicações específicas. Isto incluiria a adaptação de vários dos *benchmarks* existentes atualmente (como os da SPEC, por exemplo) e também a construção de modelos que possibilitem aos desenvolvedores de software realizar estimativas de custo dos recursos de hardware logo no início do processo de desenvolvimento, como forma de auxiliar no estabelecimento do preço final do software.

3.5 Resumo e novos campos de pesquisa

As plataformas *pay-as-you-go* afetarão vários aspectos do desenvolvimento de software. A análise apresentada neste capítulo destacou as estratégias de *precificação*, os modelos para elaboração de *estimativas* (com o COCOMO), as práticas de engenharia de software (em especial o levantamento de requisitos) e os *benchmarks* de desempenho.

Todas estas mudanças estão relacionadas ao novo modelo de cobrança, que considera o

[3] Apenas no caso da *Google* foi utilizada efetivamente uma *PaaS* (o *AppEngine*), enquanto nos demais foram utilizados serviços do tipo *DaaS* ou *IaaS*.

volume de recursos efetivamente consumidos, o que afeta aspectos estratégicos como a própria precificação de software. O quadro da figura 3.11 resume as mudanças analisadas nesta seção.

Item	**Descrição**	**Como é Atualmente**	**O que irá mudar com as plataformas *pas-as-you-go***
Precificação	A determinação de um preço para o software.	Existem várias estratégias, em geral divididas em duas linhas: *1-licença de uso perpétua* e *2-aluguel*. Em qualquer das estratégias é definido um preço com base no potencial de venda e de geração de receita do software.	O mecanismo de licença perpétua tende a desaparecer. Após calculado o potencial de geração de receita, será diminuído desse total a estimativa de consumo de recursos.
Estimativas de Desenvolvimento (COCOMO)	Estimativas de tempo e custo para o desenvolvimento de um software.	São realizadas estimativas de tempo que levam em consideração vários fatores. A principal metodologia utilizada é o COCOMO. Os recursos de hardware são tratados em 2 fatores principais: *Restrições de CPU* e *Restrições de Armazenamento*. Ambos são medidos com base no % de recursos disponíveis que será utilizado, variando de 50% a 95%.	A medição com base em % de recursos uitilizados deixa de fazer sentido em uma plataforma *pay-as-you-go*, onde os recursos são infinitos. Assim, os itens *Restrições de CPU* e *Restrições de Armazenamento* do COCOMO precisarão ser adaptados.
Engenharia de SW e Requisitos (ISO/IEC 25010)	Processos para desenvolvimento de software e a sua relação com os recursos de hardware	São levantados requisitos *funcionais* e *não-funcionais* que irão nortear o desenvolvimento do software. Os requisitos de hardware (*não-funcionais*) são tratados como critérios de qualidade, com níveis pré-estabelecidos que precisam ser atingidos (ISO/IEC 25010). Em geral, não não há preocupação com o consumo de recursos durante a fase de levantamento de requisitos. Foco atual: --> Quais funcionalidades terão uso concorrente intensivo? --> Otimizar aquelas cujo tempo de resposta poderá ser comprometido	O consumo de hardware passa a ter importância diferente, além do atendimento a critérios de qualidade, já que possui influência no preço final do software. Será preciso identificar, logo no início, o volume de utilização do sistema pelos usuários para que seja possível estimar o consumo de recursos. Esta estimativa provavelmente passará a ser feita em termos financeiros. Novo foco: Quais funcionalidades serão usadas c/ maior frequência? Otimizar aquelas cujo uso tende a consumir mais recursos.
***Benchmarks* de desempenho**	Ferramentas para comparar o desempenho de soluções de TI.	Existem *benchmarks* para vários tipos de software, sendo que a SPEC atua fortemente nesta área. Em geral, os *benchmarks* estão focados em medir o desempenho de uma determinada configuração de hardware quando submetida a execução de um tipo de software específico. Um exemplo é o *SPECJEnterprise2010*, relacionado a aplicações java enterprise, medido através de *EjOPS*. A partir de medições como essa os desenvolvedores de software podem orientar seus clientes sobre o hardware adequado para suportar a sua aplicação.	A informação sobre o preço dos recursos consumidos passa a ser tão ou mais importante que as medições baseadas em "transações por segundo". Já há alguns estudos preliminares nesta área e as informações sobre preço dos recursos consumidos precisarão ser adicionadas aos *benchmarks* e/ou novos *benchmarks* precisarão ser construídos com itens de medição diferentes.

Figura 3.11: Mudanças geradas pelas plataformas *pay-as-you-go* nos aspectos de desenvolvimento de software.

As mudanças geradas pelas plataformas *pay-as-you-go* também geram novas oportunidades de pesquisa. Há vários pontos relacionados ao desenvolvimento de software que ainda não foram abordados pela comunidade científica e o quadro da figura 3.12 faz um resumo das novas pesquisas que poderão ser realizadas dentro dos aspectos abordados na análise de que trata este capítulo.

Item	Descrição	Novas Oportunidades de Pesquisa
Precificação	A determinação de um preço para o software.	Construir modelos que levem em consideração o volume de recursos consumidos pela aplicação
Estimativas de Desenvolvimento (COCOMO)	Estimativas de tempo e custo para o desenvolvimento de um software.	Realizar novas experiências que levem em consideração critérios diferentes. Por exemplo: --> % de funcionalidades que precisam ser otimizadas --> % de funcionalidades com uso intensivo --> % da receita financeira estimada que pode ser comprometido c/ consumo de recursos
Engenharia de SW e Requisitos (ISO/IEC 25010)	Processos para desenvolvimento de software e a sua relação com os recursos de hardware	Modelos que adicionem ao processo de levantamento de requisitos informações sobre o volume de uso de cada funcionalidade.
***Benchmarks* de desempenho**	Ferramentas para comparar o desempenho de soluções de TI.	Novos *benchmarks* SPEC com medição baseada em preço final para execução de uma determinada carga de trabalho. Modelos que estimem o custo dos recursos de hardware com base em determinadas variáveis de entrada (por exemplo as do *IBM Function Points*)

Figura 3.12: Trabalhos futuros relacionados às mundanças causadas no desenvolvimento de software pelas plataformas *pay-as-you-go*.

4 ESTUDO DE CASO - ERP SIE

Este capítulo apresenta um estudo de caso para verificar o impacto real que pode ser causado pelo novo cenário gerado pelas plataformas *pay-as-you-go* de computação em nuvem. Este estudo considera especificamente as situações identificadas nos capítulos 2 e 3, que apontam para a necessidade de um consumo racional de recursos de hardware como forma de aumentar o potencial de lucro dos desenvolvedores/fornecedores de software.

O objetivo do estudo é, a partir de simulações do consumo de recursos, verificar como o novo cenário proporcionado pelos modelos de cobrança *pay-as-you-go* poderia afetar um software construído sem a preocupação com o consumo racional de recursos.

O caso apresentado utiliza o sistema SIE, cuja abrangência e alto nível de utilização na UFSM permitem uma medição bastante diversificada. O SIE foi desenvolvido a partir de meados de 1998 utilizando a linguagem Delphi e uma arquitetura multi-camadas onde as regras de negócio são processadas através de chamadas RPC em um ou mais servidores de aplicação centralizados (BARBOSA, 2000). O ERP completo possui cerca de 2.500 tabelas acessadas por mais de 4.000 aplicações que trabalham de forma integrada para gerenciar diferentes áreas da universidade, tais como: controle acadêmico (graduação, pós-graduação e ensino médio), controle de estoque, gestão de compras e contratos, empenhos, recursos humanos, projetos de pesquisa, biblioteca, protocolo, frota, entre outros.

4.1 O estudo de caso e a nuvem *pay-as-you-go*

Um cenário ideal para realizar todas as simulações envolvendo uma plataforma *pay-as-you-go* exigiria que o SIE estivesse hospedado em uma plataforma em nuvem. Este cenário, entretanto, seria inviável dentro do escopo deste trabalho, pois i) o sistema como um todo precisaria ser desenvolvido utilizando a arquitetura e/ou linguagem de programação específica da plataforma e ii) ainda que fosse possível adaptar o sistema para estas características seria necessário efetivamente hospedá-lo na plataforma e fazer com que todos os funcionários da UFSM (ou boa parte deles, para garantir uma representatividade suficiente) se conectassem ao sistema através da plataforma, e não através da infra-estrutura de hardware da UFSM.

Como este cenário é inviável, optou-se por realizar uma simulação do ambiente *pay-as-you-go* em nuvem com base na utilização da própria infra-estrutura já instalada e em uso pela UFSM. Esta abordagem não afeta os usuários do sistema nem a própria UFSM, pois não há necessidade

de contratar um fornecedor *cloud* e as simulações podem ser feitas através de um monitoramento que seja transparente ao usuário final do sistema. A realização destas simulações está divida em dois passos principais:

- medir (ou estimar) o volume de recursos consumido pelas funcionalidades do sistema;
- estabelecer uma relação de preço desta estimativa tomando como base as informações de preço coletadas na experiência com o *AppEngine*, relatada na seção 2.5.

Uma medição real e exata dos recursos efetivamente utilizados deveria conter estruturas de monitoramento que levam em consideração todos os itens que são precificados pelas plataformas *pay-as-you-go*, o que no caso do *AppEngine* inclui itens como: GB transferidos, GB armazenados e consumo de CPU. Realizar o monitoramento neste nível de detalhe não seria viável dentro do escopo deste trabalho, então optou-se por monitorar o *tempo de resposta* de cada funcionalidade do sistema, estabelecendo a seguir uma relação deste tempo de resposta com o consumo de recursos no *AppEngine* e por consequência com o preço que deveria ser pago ao fornecedor da plataforma.

Em um ERP como o SIE o tempo de resposta pode ser afetado por questões como: o consumo de CPU pelo servidor de aplicação, o volume de dados transferidos entre o servidor de aplicação e as telas do sistema, o consumo de CPU e o volume de acesso a disco do servidor de banco de dados e o próprio volume de informações armazenadas na base de dados. Apesar de todos estes aspectos estarem relacionados com o modelo de cobrança *pay-as-you-go* e de o tempo de resposta já ter sido utilizado em pesquisas relacionadas a desempenho (DING; HILLSTON; LAURENSON, 2010), a medição do tempo de resposta não substitui por completo o monitoramento detalhado dos recursos. Entretanto para o objetivo deste trabalho o *tempo de resposta* pode ser utilizado, sem grandes prejuízos, como item de análise.

4.2 Implementação e medição

Para realizar as medições foi implementada uma mudança no código-fonte do ERP com o objetivo de registrar o *log* de todas chamadas RPC feitas para a camada servidora do sistema. O mecanismo de *log* registra o nome do método RPC que está sendo chamado, o usuário e a aplicação que deram origem à chamada e o tempo que cada chamada levou para ser processada. A nova implementação foi aplicada no ambiente de produção [1] da UFSM e o monitoramento

[1] A infra-estrutura de computadores onde ficam hospedados os sistemas acessados pelos funcionários da UFSM no seu dia-a-dia.

ficou ativo em um dos servidores de aplicação durante o período de 20 minutos.

O período de 20 minutos foi suficiente para registrar dados referentes a mais de 35.000 chamadas RPC ao servidor, realizadas por um total de 58 usuários, que utilizaram 62 aplicações diferentes do sistema, resultando em chamadas a 602 métodos RPC diferentes. A partir desta base foi possível obter resultados como o ilustrado na figura 4.1, que relaciona os métodos RPC ordenados pelo tempo total de resposta. Nesta figura, pode-se verificar que o método *IConsultaLocal.ConsultaAcervoBib* é o que teve o maior tempo de resposta total (385.336 *msec*) e foi chamado 90 vezes durante o período de monitoramento. Ao mesmo tempo, o método *ISGCA.GetRotulo* foi chamado 6.252 vezes com um tempo de resposta total de 71.225 *msec*.

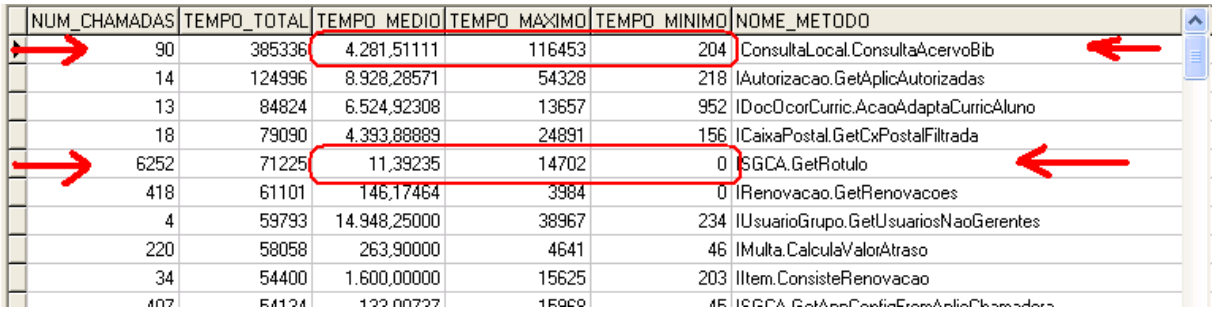
NUM_CHAMADAS	TEMPO_TOTAL	TEMPO_MEDIO	TEMPO_MAXIMO	TEMPO_MINIMO	NOME_METODO
90	385336	4.281,51111	116453	204	IConsultaLocal.ConsultaAcervoBib
14	124996	8.928,28571	54328	218	IAutorizacao.GetAplicAutorizadas
13	84824	6.524,92308	13657	952	IDocOcorCurric.AcaoAdaptaCurricAluno
18	79090	4.393,88889	24891	156	ICaixaPostal.GetCxPostalFiltrada
6252	71225	11,39235	14702	0	ISGCA.GetRotulo
418	61101	146,17464	3984	0	IRenovacao.GetRenovacoes
4	59793	14.948,25000	38967	234	IUsuarioGrupo.GetUsuariosNaoGerentes
220	58058	263,90000	4641	46	IMulta.CalculaValorAtraso
34	54400	1.600,00000	15625	203	IItem.ConsisteRenovacao
407	54134	133,00737	15968	45	ISGCA.GetAppConfigFromAplicChamadora
38	51406	1.352,78947	11469	15	IConfiguracao.CheckConfiguracoes

Figura 4.1: *Log* de chamadas RPC do SIE ordenado pelo tempo total de resposta (destaque para os métodos *IConsultaLocal.ConsultaAcervoBib* e *ISGCA.GetRotulo*).

Com o registro de todas as chamadas RPC e de seu tempo de resposta, é possível calcular o tempo médio de resposta de cada método dividindo o somatório total dos seus tempos de resposta pelo número de chamadas realizadas. Nesta aritmética o tempo médio de resposta do método *IConsultaLocal.ConsultaAcervoBib* fica em 4.281,51 *msec*, enquanto no método *ISGCA.GetRotulo* este tempo é de 11,39 *msec*

Uma análise mais detalhada da figura 4.2 evidencia uma possível distorção nas medições realizadas. O método *IConsultaLocal.ConsultaAcervoBib* teve o seu tempo de resposta máximo em 116.453 *msec* e o mínimo em 204 *msec*, enquanto o método *ISGCA.GetRotulo* teve o tempo máximo de 14.702 *msec* e o mínimo de 0 *msec*. Considerando que o tempo total de resposta do primeiro foi de 385.336 *msec* e o do segundo foi de 71.225 *msec*, a chamada mais demorada de cada um dos métodos foi responsável, respectivamente, por 30,22% e 20,64% do tempo total de resposta de todas as suas chamadas.

Esta discrepância entre os valores registrados pode ser explicada de diferentes formas. O método *ISGCA.GetRotulo* é implementado através de uma estrutura de *cache* genérica que utiliza semáforos para controlar a consistência da *cache* nos momentos em que os dados estão



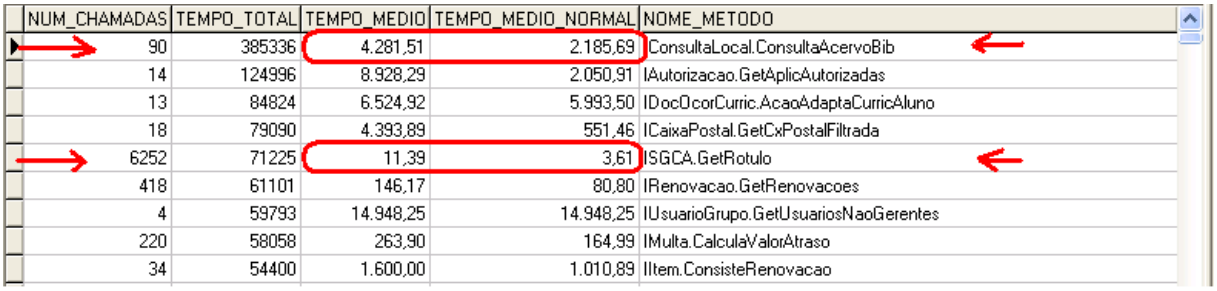
NUM_CHAMADAS	TEMPO_TOTAL	TEMPO_MEDIO	TEMPO_MAXIMO	TEMPO_MINIMO	NOME_METODO
90	385336	4.281,51111	116453	204	IConsultaLocal.ConsultaAcervoBib
14	124996	8.928,28571	54328	218	IAutorizacao.GetAplicAutorizadas
13	84824	6.524,92308	13657	952	IDocOcorCurric.AcaoAdaptaCurricAluno
18	79090	4.393,88889	24891	156	ICaixaPostal.GetCxPostalFiltrada
6252	71225	11,39235	14702	0	ISGCA.GetRotulo
418	61101	146,17464	3984	0	IRenovacao.GetRenovacoes
4	59793	14.948,25000	38967	234	IUsuarioGrupo.GetUsuariosNaoGerentes
220	58058	263,90000	4641	46	IMulta.CalculaValorAtraso
34	54400	1.600,00000	15625	203	IItem.ConsisteRenovacao

Figura 4.2: *Log* de chamadas RPC do SIE ordenado pelo tempo total de resposta (destaque para os métodos *IConsultaLocal.ConsultaAcervoBib* e *ISGCA.GetRotulo*).

sendo carregados ou atualizados. Assim, os tempos de resposta muito altos podem estar relacionados a condições de corrida onde a requisição estava esperando a liberação do semáforo.

Já no caso do método *IConsultaLocal.ConsultaAcervoBib* a situação é diferente. Ele não usa nenhuma estrutura de *cache*, fazendo um acesso direto a base de dados para consultar o acervo de exemplares da biblioteca. Neste caso, a discrepância entre os tempos de resposta pode estar relacionada a pelo menos duas situações: i) um parâmetro de pesquisa que resulta em um volume muito grande (ou muito pequeno) de resultados e ii) uma eventual espera devido a condições de corrida e/ou algoritmos específicos implementados pelo próprio banco de dados.

Em qualquer um dos casos existe uma possibilidade de que, pelo menos os tempos de resposta maiores não tenham relação direta com o volume de recursos consumidos. Então, embora não seja estritamente necessário para o escopo deste trabalho, optou-se por realizar uma normalização dos dados registrados, com o objetivo de minimizar os efeitos desta hipótese no resultado final das medições. Assim, para calcular o tempo médio de resposta de cada método foram excluídas do cálculo as chamadas 10% mais lentas e as 10% mais rápidas [2] . Os novos tempos médios obtidos estão demonstrados na figura 4.3



NUM_CHAMADAS	TEMPO_TOTAL	TEMPO_MEDIO	TEMPO_MEDIO_NORMAL	NOME_METODO
90	385336	4.281,51	2.185,69	IConsultaLocal.ConsultaAcervoBib
14	124996	8.928,29	2.050,91	IAutorizacao.GetAplicAutorizadas
13	84824	6.524,92	5.993,50	IDocOcorCurric.AcaoAdaptaCurricAluno
18	79090	4.393,89	551,46	ICaixaPostal.GetCxPostalFiltrada
6252	71225	11,39	3,61	ISGCA.GetRotulo
418	61101	146,17	80,80	IRenovacao.GetRenovacoes
4	59793	14.948,25	14.948,25	IUsuarioGrupo.GetUsuariosNaoGerentes
220	58058	263,90	164,99	IMulta.CalculaValorAtraso
34	54400	1.600,00	1.010,89	IItem.ConsisteRenovacao

Figura 4.3: Tempo médio normalizado, retirando os 10% mais rápidos e 10% mais lentos (destaque: *IConsultaLocal.ConsultaAcervoBib* e *ISGCA.GetRotulo*).

[2]Poderia ser utilizada uma distribuição normal, mas como não há necessidade de precisão estatística, optou-se por esta simplificação.

A partir dos novos tempos médios, também foi calculado o novo tempo total de resposta. Este cálculo foi feito multiplicando o tempo médio normalizado pelo número total de chamadas, o que resultou nos dados apresentados na figura 4.4. Assim, o tempo de resposta total do método *IConsultaLocal.ConsultaAcervoBib* passou de 385.336 *msec* para 196.712 *msec* e o do método *ISGCA.GetRotulo* passou de 71.225 *msec* para 22.563 *msec*.

NUM_CHA	TEMPO_TOTAL	TEMPO_TOTAL_NORMAL	TEMPO_MEDIO	TEMPO_MEDIO_NORMAL	NOME_METODO
90	385336	196.712,11	4.281,51	2.185,69	IConsultaLocal.ConsultaAcervoBib
14	124996	28.712,73	8.928,29	2.050,91	IAutorizacao.GetAplicAutorizadas
13	84824	77.915,50	6.524,92	5.993,50	IDocOcorCurric.AcaoAdaptaCurricAluno
18	79090	9.926,31	4.393,89	551,46	ICaixaPostal.GetCxPostalFiltrada
6252	71225	22.563,96	11,39	3,61	ISGCA.GetRotulo
418	61101	33.776,41	146,17	80,80	IRenovacao.GetRenovacoes
4	59793	59.793,00	14.948,25	14.948,25	IUsuarioGrupo.GetUsuariosNaoGerentes
220	58058	36.297,49	263,90	164,99	IMulta.CalculaValorAtraso
34	54400	34.370,22	1.600,00	1.010,89	IItem.ConsisteRenovacao

Figura 4.4: Tempo total normalizado, retirando os 10% mais rápidos e 10% mais lentos (destaque: métodos *IConsultaLocal.ConsultaAcervoBib* e *ISGCA.GetRotulo*).

A normalização do tempo médio de resposta também afeta o tempo de resposta total. O tempo medido durante os 20 minutos de monitoramento do SIE foi de 2.010.011 *msec* e o tempo total normalizado, com o descarte dos 10% mais lentos e 10% mais rápidos, é de 1.283.591 *msec*.

4.3 Análise dos Resultados

A análise dos resultados obtidos demonstra algumas situações peculiares, que são discutidas a seguir. O quadro da figura 4.5 contém os 12 métodos RPC que tiveram o maior somatório de tempos de resposta. O quadro mostra o número de chamadas feitas para cada método, os tempos de resposta efetivamente medidos (campo *Somatório*) e os tempos de resposta normalizados considerando o descarte dos 10% mais lentos e 10% mais rápidos (campo *Somatório Normalizado)*. A proporção destes 12 métodos (de um total de 602 monitorados) em relação ao tempo total de resposta é bastante significativa, ficando em 43,53% do tempo normalizado.

Avaliando o quadro da figura 4.5 o método que mais se destaca em relação aos tempos de resposta é o *IConsultaLocal.ConsultaAcervoBib*, que foi chamado 90 vezes e é responsável por 15,33% do tempo normalizado. A otimização do método está fora do escopo deste trabalho, mas uma análise da sua implementação é relevante para ajudar a entender os tipos de problema que podem surgir ao se realizar uma avaliação de desempenho que leve em consideração o consumo total de recursos de um determinado sistema.

	Nome do Método RPC	Nro. de Chamadas	Somatório		Somatório Normalizado	
			Tempo	%	Tempo	%
1	IConsultaLocal.ConsultaAcervoBib	90	385.336,0 msec	19,17%	196.712,1 msec	15,33%
2	IAutorizacao.GetAplicAutorizadas	14	124.996,0 msec	6,22%	28.712,0 msec	2,24%
3	IDocOcorCurric.AcaoAdaptaCurricAluno	13	84.824,0 msec	4,22%	77.915,5 msec	6,07%
4	ICaixaPostal.GetCxPostalFiltrada	18	79.090,0 msec	3,93%	9.926,3 msec	0,77%
5	ISGCA.GetRotulo	6252	71.225,0 msec	3,54%	22.564,0 msec	1,76%
6	IRenovacao.GetRenovacoes	418	61.101,0 msec	3,04%	33.776,4 msec	2,63%
7	IUsuarioGrupo.GetUsuariosNaoGerentes	4	59.793,0 msec	2,97%	14.948,3 msec	1,16%
8	IMulta.CalculaValorAtraso	220	58.058,0 msec	2,89%	36.297,5 msec	2,83%
9	IItem.ConsisteRenovacao	34	54.400,0 msec	2,71%	34.370,2 msec	2,68%
10	ISGCA.GetAppConfigFromAplicChamadora	407	54.134,0 msec	2,69%	35.356,2 msec	2,75%
11	IConfiguracao.CheckConfiguracoes	38	51.406,0 msec	2,56%	33.559,2 msec	2,61%
12	ITipoDocPessoa.GetTipoDocPorTipoPessoa	80	42.852,0 msec	2,13%	34.652,0 msec	2,70%
				56,08%		43,53%

Figura 4.5: Os 12 métodos RPC com maior tempo de resposta total.

O método *IConsultaLocal.ConsultaAcervoBib* faz uma pesquisa no acervo de exemplares de uma (ou mais) biblioteca(s) e pode ser executado com vários parâmetros de pesquisa diferentes. Cada parâmetro de pesquisa resulta em condições diferentes que são aplicadas na busca. Então, um trabalho de otimização deste método implicaria em i) coletar os parâmetros utilizados em cada chamada, ii) identificar quais parâmetros são utilizados com maior frequência pelos usuários e iii) avaliar quais combinações de parâmetros possuem maior impacto no tempo de resposta considerando o desempenho e o volume de utilização. Uma tarefa nada simples, que envolve um esforço significativo de análise.

O método *IAutorizacao.GetAplicAutorizadas* (responsável por 2,24% do tempo) apresenta uma característica diferente do anterior. Ele se refere a uma funcionalidade executada sempre que um usuário se conecta no sistema, recuperando uma lista das aplicações nas quais o usuário tem permissão de acesso. Assim, ele recebe um único parâmetro de execução, que é o identificador do próprio usuário que está dando origem a chamada. A lista de autorizações é uma informação que não muda com muita frequência, então uma eventual otimização nesta funcionalidade poderia envolver a utilização de um mecanismo de *cache* no servidor de aplicações. Entretanto, por se tratar de dados referentes a autorizações de acesso, o nível de consistência desta *cache* deveria ser próximo do tempo real para evitar que um usuário tenha acesso a funções cuja permissão já lhe foi revogada.

O método *ISGCA.GetRotulo*, por sua vez, possui uma outra característica. A sua implementação já utiliza um mecanismo de *cache*, mas devido ao grande número de chamadas (6.252)

ele acaba consumindo um percentual significativo (1,76%) do tempo de reposta. Uma eventual otimização neste caso poderia passar por uma revisão no código-fonte das aplicações que o chamam, de forma que elas próprias mantivessem uma cópia local das informações, minimizando o volume excessivo de chamadas para o servidor de aplicação. Isto poderia ser feito sem prejuízos pelo fato de se tratar de i) um conjunto pequeno de dados, com cerca de 3.500 registros e ii) uma informação que, caso esteja inconsistente, não provocaria perdas significativas para o uso do sistema.

Um outro caso é o do método *ISGCA.GetAppConfigFromAplicChamadora*, que recupera configurações gerais sobre a tela que está sendo aberta pelo usuário, tais como o título da janela e os nomes dos campos de tela. Trata-se de um método que faz parte da arquitetura do sistema e é chamado com alta frequência por estar presente na implementação padrão de todas as aplicações do sistema. A implementação de uma *cache* no servidor de aplicações poderia ser utilizada como estratégia para reduzir o consumo de recursos. Nesta mesma situação se encaixam também outros métodos que não estão entre os 12 com maior tempo de resposta, mas que também contribuem para o consumo geral de recursos do sistema: *IParInstituicao.GetRecords* e *ITabEstrutura.GetItensTabela*, respectivamente o 19º e o 37º na lista, que juntos contribuiram com 1,45% do tempo total de resposta normalizado. Somando este número com os 2,75% gastos pelo método *ISGCA.GetAppConfigFromAplicChamadora* se chega a 4,20% do tempo de resposta total.

Haveria ainda outros casos, como o dos métodos *ITipoDocPessoa.GetTipoDocPorTipoDocPessoa* (80 chamadas e 2,70% do tempo total) e *IConfiguracao.GetConfiguracao* (213 chamadas e 0,60% do tempo), que são funcionalidades específicas de um determinado sistema que também poderiam ser otimizadas através da implementação de um mecanismo de *cache* no servidor de aplicação.

Uma avaliação exaustiva de todas as possibilidades de otimização seria um trabalho interessante, mas os exemplos acima, resumidos na figura 4.6, são suficientes para os objetivos deste estudo de caso: demonstrar a diversidade de pontos de otimização existentes em um sistema como o ERP SIE e vislumbrar, em um caso prático, o quanto estas otimizações "deixadas de lado"podem representar em termos de desperdício no uso de recursos.

Método RPC	Nro. de Chamadas	Tempo de Resposta (normalizado)	Abrangência	Otimização
IConsultaLocal.ConsultaAcervoBib	90	15,33%	Sistema Específico (Biblioteca)	Complexa: envolve várias combinações de parâmetros e uma lógica diferente para cada combinação
IAutorizacao.GetAplicAutorizadas	14	2,24%	Sistema Específico (Controle de Acesso)	Simples: *cache* no servidor de aplicações (consistência forte)
ISGCA.GetRotulo	6.252	1,76%	Arquitetura	Trabalhosa: revisar todas aplicações que chamam. Possibilidade de utilizar uma cópia local dos dados (consistência muito fraca)
ISGCA.GetAppConfigFromAplicChamadora	407	2,75%	Arquitetura	Simples: *cache* no servidor de aplicações (consistência fraca)
IParInstituicao.GetRecords	702	0,85%		
ITabEstrutura.GetItensTabela	250	0,60%		
ITipoDocPessoa.GetTipoDocPorTipoPessoa	80	2,70%	Sistema Específico (Cadastro Único)	
IConfiguracao.GetConfiguracao	689	0,60%	Sistema Específico (Biblioteca)	

Figura 4.6: Exemplos de pontos de otimização encontrados no monitoramento do ERP SIE.

4.4 Simulação de Preço

As seções anteriores apresentaram um mapeamento do uso do ERP SIE durante um período de 20 minutos, destacando os métodos RPC que possuem um tempo de resposta total elevado e evidenciando oportunidades de otimização que poderiam ser implementadas com o objetivo de melhorar o desempenho do sistema.

Entretanto, o desempenho geral do sistema não vem sendo problema para a UFSM. O único ponto de otimização que realmente foi abordado (e melhorado) nos últimos anos está relacionado às funcionalidades utilizadas durante o período de matrícula dos alunos, que é o momento onde a carga de uso do sistema atinge o seu pico de demanda. Na prática, a investigação das oportunidades de melhoria de desempenho vem sendo deixadas de lado para que as equipes de desenvolvimento possam se concentrar em produzir novos módulos e funcionalidades.

A lógica utilizada pela UFSM para priorizar os esforços de sua equipe em relação ao SIE faz sentido: o benefício gerado por novos módulos e funcionalidades é maior que o benefício de, por exemplo, melhorar o tempo de resposta de 4 para 3 segundos, ou de 200 *msec* para 100 *msec*. Por outro lado, o prejuízo causado por uma sobrecarga no sistema durante o período de matrícula é muito grande, então se justifica o esforço da equipe de desenvolvimento em otimizar esta funcionalidade específica. Mas, no modelo *pay-as-you-go* o que mudaria nesta lógica? Uma visão financeira do consumo de recursos ajuda a responder esta pergunta.

A experiência relatada na seção 2.5 indica um parâmetro de custo para aplicações construídas na plataforma *Java* do *Google AppEngine*. Conforme os dados obtidos, o consumo de CPU

aumenta em média 0,01 unidade a cada 125 chamadas feitas a um método que recupera 100 linhas de uma tabela com 5 colunas. Considerando o preço de U$0,10 por hora de CPU, tem-se uma relação de U$0,01 a cada 1.250 chamadas a este método.

O quadro da figura 4.7 apresenta uma relação dos métodos RPC monitorados que possuem um comportamento semelhante ao da experiência da seção 2.5. As primeiras duas colunas representam, respectivamente, o número de registros e o número de colunas retornados por cada método. A terceira coluna ("Proporção") é o resultado da divisão do número de registros do método *AppEngine* pelo número de registros do método SIE. A linha em destaque é referente ao método *DescricoesClassif.GetRecords*, que possui o número de colunas mais próximo para efeitos de comparação (6, em comparação com as 5 do método *AppEngine*). As demais colunas são informações sobre o tempo de resposta obtidas durante o monitoramento de uso do SIE.

	Informações sobre o Método			Medições do Tempo de Resposta			Tempo Médio de Resposta		
	Nro. de registros do método	Colunas	Proporção (100 / nro. de registros do método)	Máximo	Mínimo	Total	Nro. de Chamadas	Tempo Médio (registrado)	Tempo Médio (normalizado)
ModalidadeCompra.GetRecords	17	14	5,9	124 msec	0 msec	402 msec	19	21,2 msec	15,4 msec
Biblioteca.GetRecords	15	19	6,7	31 msec	15 msec	276 msec	16	17,3 msec	15,3 msec
DescricoesClassif.GetRecords	5	6	20,0	31 msec	0 msec	139 msec	10	13,9 msec	13,1 msec
TabEstrutura.GetItensTabela	11	14	9,1	219 msec	0 msec	9.980 msec	250	39,9 msec	30,7 msec

Figura 4.7: Métodos monitorados no SIE que possuem comportamento semelhante à medição *AppEngine* da seção 2.5.

Fazendo uma analogia das informações da figura 4.7 com o método *AppEngine* medido na seção 2.5, é possível obter uma relação que informe quantos *milissegundos* de tempo de resposta correspondem a U$0,01. O quadro da figura 4.8 demonstra como ficaria esta relação. As primeiras duas colunas são o tempo médio obtido durante o monitoramento e as três colunas seguintes são referentes a uma estimativa do tempo médio que cada método levaria caso retornasse 100 registros ao invés do número que efetivamente está retornando.

Tomando o método *DescricoesClassif.GetRecords* como exemplo: ele tem um tempo médio de resposta de 13,9 *msec* e retorna, proporcionalmente, 20 vezes menos registros que o método *AppEngine* medido na seção 2.5. Se ele retornasse os mesmos 100 registros o seu tempo médio de resposta seria de 278 *msec*. Considerando o tempo normalizado (que descarta os 10% mais lentos e os 10% mais rápidos) o tempo de resposta médio ajustado seria de 262,8 *msec*. Através desta analogia chega-se à seguinte relação: cada 328.500 *msec* do tempo de resposta ajustado corresponde a U$0,01 no valor a ser pago pelo uso dos recursos da plataforma.

Com este número de referência, é possível fazer uma estimativa do preço total que seria

	Tempo Médio (monitorado)		Tempo Médio Ajustado (proporcional a 100 registros)			Tempo Médio Ajustado * 1.250 (App Engine: 1.250 chamadas = U$0,01)	
	Registrado	Normalizado	Proporção	Registrado	Normalizado	Registrado	Normalizado
ModalidadeCompra.GetRecords	21,2 msec	15,4 msec	5,9	124,5 msec	90,8 msec	155.572,8 msec	113.455,9 msec
Biblioteca.GetRecords	17,3 msec	15,3 msec	6,7	115,0 msec	101,8 msec	143.750,0 msec	127.250,0 msec
DescricoesClassif.GetRecords	13,9 msec	13,1 msec	20,0	278,0 msec	262,8 msec	347.500,0 msec	328.500,0 msec
TabEstrutura.GetItensTabela	39,9 msec	30,7 msec	9,1	362,9 msec	278,9 msec	453.636,4 msec	348.636,4 msec

Figura 4.8: Relação do número de *milisegundos* que corresponderiam ao custo de U$0,01 na plataforma *AppEngine*.

pago pelos 20 minutos durante os quais um dos servidores de aplicação do SIE foi monitorado. Para isto, basta dividir o total do tempo de resposta total (1.283.591 *msec*) por 328.500, o que dá o valor total de U$3,90. A figura 4.9 apresenta a projeção deste preço para um período de 1 hora, 1 mês e 1 ano, considerando 8 horas por dia e 21 dias por mês de uso.

20 min. (Tempo/328.500)	1 hora (20min x 3)	1 dia (1 hora x 8)	1 mês (1 dia x 21)	1 ano (1 mês x 12)
U$ 3,91	U$ 11,72	U$ 93,78	U$ 1.969,35	U$ 23.632,15

Figura 4.9: Projeção do preço total de uso em um dos Servidores de Aplicação do SIE, considerando que cada 328.500 *msec* de tempo de resposta incrementariam o custo em U$0,01.

Apesar de serem compostas a partir de uma lógica consistente, estas estimativas de preço não são exatas e o custo anual estimado de U$23.632,15 não pode ser considerado real. O valor serve apenas para se ter uma noção do volume de recursos financeiros que poderia estar envolvido em se tratando de um ERP com o volume de utilização do SIE. O objetivo deste estudo de caso é identificar pontos onde o desenvolvimento de software seria afetado pelas plataformas *pay-as-you-go* em nuvem e para isso as informações da figura 4.10 são reveladoras. Ela apresenta o preço (diário, mensal e anual) que os métodos analisados na seção 4.3 custariam para um fornecedor de software caso ele estivesse hospedando o SIE em uma plataforma com os parâmetros de preço considerados nesta análise.

Os métodos em destaque na figura são os que poderiam ser otimizados através da implementação de uma estrutura de *cache* simples. O preço total dos seis métodos que se encaixam nesta situação seria de U$2.302,56 por ano (U$191,88 por mês) .

As simulações de preço do *AppEngine* (seção 2.5) indicam que a utilização de uma estrutura de *cache* é 20 vezes mais barata do que o acesso direto a base de dados. Neste caso, a simples implementação de tais estruturas para os seis métodos em destaque na figura 4.10 reduziria o preço para U$115,13 por ano. Uma economia anual de U$2.187,43 por servidor de aplicação. A

Método RPC	Estratégia de Otimização	Tempo de Resposta (normalizado)	Preço				
			20 min. (Tempo/32.850)	1 hora (20min x 3)	1 dia (1 hora x 8)	1 mês (1 dia x 21)	1 ano (1 mês x 12)
IConsultaLocal.ConsultaAcervoBib		196.712 msec	U$ 0,599	U$ 1,796	U$ 14,37	U$ 301,80	U$ 3.621,66
IAutorizacao.GetAplicAutorizadas	Simples: *cache* no Servidor de Aplicações	28.712 msec	U$ 0,087	U$ 0,262	U$ 2,10	U$ 44,05	U$ 528,62
ISGCA.GetRotulo		22.564 msec	U$ 0,069	U$ 0,206	U$ 1,65	U$ 34,62	U$ 415,42
ISGCA.GetAppConfigFromAplicChamadora	Simples: *cache* no Servidor de Aplicações	35.356 msec	U$ 0,108	U$ 0,323	U$ 2,58	U$ 54,25	U$ 650,94
IParInstituicao.GetRecords		10.942 msec	U$ 0,033	U$ 0,100	U$ 0,80	U$ 16,79	U$ 201,45
ITabEstrutura.GetItensTabela		7.671 msec	U$ 0,023	U$ 0,070	U$ 0,56	U$ 11,77	U$ 141,23
ITipoDocPessoa.GetTipoDocPorTipoPessoa		34.652 msec	U$ 0,105	U$ 0,316	U$ 2,53	U$ 53,16	U$ 637,98
IConfiguracao.GetConfiguracao		7.732 msec	U$ 0,024	U$ 0,071	U$ 0,56	U$ 11,86	U$ 142,35
	Preço total dos métodos passíveis de otimização com *cache*		**U$ 0,38**	**U$ 1,14**	**U$ 9,14**	**U$ 191,88**	**U$ 2.302,56**

Figura 4.10: Resumo de métodos otimizáveis através de uma estrutura de *cache* simples.

UFSM utiliza cinco servidores de aplicação para o SIE. Se todos tiverem o mesmo perfil de uso, a economia anual chegaria a U$10.937,15. É bastante provável que, diante de números como este, a implementação destas otimizações ganhe importância e seja considerada uma prioridade.

4.5 Novas possibilidades para ERPs

Um dos grandes benefícios dos ERPs é a integração das informações disponíveis nos diversos setores de uma determinada instituição. De uma forma geral os ERPs podem cobrir aspectos que passam tanto pelas atividades fim da instituição como por atividades administrativas que suportam a sua operação. A integração proporcionada pelos ERPs facilita a obtenção de informações gerenciais, na medida em que diferentes módulos compartilham informações em comum. No caso do SIE, por exemplo, é possível identificar através de uma consulta simples à base de dados informações como: quais são os alunos de um determinado curso que participam de projetos de pesquisa, recebem bolsa de iniciação científica e retiraram livros na biblioteca nos últimos dois meses.

Esta mesma integração também pode ser observada no próprio cadastro dos usuários. Como o sistema é composto por vários módulos e o *login* é o mesmo para todas as suas aplicações, é bastante simples unificar as informações de uso do sistema de um determinado usuário.

Durante a implementação do mecanismo de *log* que seria utilizado no estudo de caso optou-se por registrar não apenas o tempo de resposta de cada chamada, mas também qual usuário estava originando a chamada e qual aplicação ele estava utilizando. Juntando esta medição detalhada do tempo de resposta com a relação de preço demonstrada na seção 4.4 é possível

atribuir a cada usuário um custo demandado por ele com a utilização do sistema. O quadro da figura 4.11 demonstra a relação obtida durante os 20 minutos de monitoramento do SIE. O usuário *bibweb*, que representa o maior custo, é um usuário genérico utilizado pelo módulo de Controle de Biblioteca do SIE para executar consultas anônimas feitas pela *web*. O próximo na relação dos que mais consumiram recursos é o usuário *Francisco* (U$0,66 no período de 20 minutos e U$3.979,65 em uma projeção anual), seguido pela *Edineia* (U$0,34 e U$2.085,67).

Usuário	Tempo de Resposta (normalizado)	Preço				
		20 min. (Tempo/328.500)	1 hora (20min x 3)	1 dia (1 hora x 8)	1 mês (1 dia x 21)	1 ano (1 mês x 12)
bibweb	460.015 msec	U$ 1,40	U$ 4,20	U$ 33,61	U$ 705,78	U$ 8.469,31
Francisco	216.156 msec	U$ 0,66	U$ 1,97	U$ 15,79	U$ 331,64	U$ 3.979,65
Edineia	113.284 msec	U$ 0,34	U$ 1,03	U$ 8,28	U$ 173,81	U$ 2.085,67
Felipe O.	51.052 msec	U$ 0,16	U$ 0,47	U$ 3,73	U$ 78,33	U$ 939,91
Felipe B.	43.427 msec	U$ 0,13	U$ 0,40	U$ 3,17	U$ 66,63	U$ 799,53
Paulo Vinicius	31.273 msec	U$ 0,095	U$ 0,29	U$ 2,28	U$ 47,98	U$ 575,76
Claudia Terezinha	28.774 msec	U$ 0,088	U$ 0,26	U$ 2,10	U$ 44,15	U$ 529,75
Gerson	23.591 msec	U$ 0,072	U$ 0,22	U$ 1,72	U$ 36,19	U$ 434,33
Maristela	19.617 msec	U$ 0,060	U$ 0,18	U$ 1,43	U$ 30,10	U$ 361,18
Jair	19.010 msec	U$ 0,058	U$ 0,17	U$ 1,39	U$ 29,17	U$ 349,98

Figura 4.11: Os 10 usuários com maior somatório do tempo de resposta e a simulação do preço atribuído a cada um em função dos recursos consumidos.

Além do consumo de cada usuário, é possível identificar as aplicações em que este consumo ocorreu. A maior parte dos recursos consumidos pelo usuário *Francisco* foi relacionada a aplicações do Módulo Acadêmico (*ACMAAdaptaCurric.exe* e *ACMOcorCurric.exe*), enquanto que as do usuário *Edineia* foram do Módulo de Controle de Acesso (*GCAMUsuario.exe* e *GCAMGUsuarios.exe*).

Utilizando as mesmas analogias de preço, também é possível identificar o consumo total de recursos de cada aplicação, que está representado na figura 4.12. A aplicação *ConsultaWebGenerica* é utilizada pela comunidade em geral para realizar consultas ao acervo bibliográfico. O volume de uso desta aplicação durante o período de monitoramento teve um custo equivalente a U$1,73 (U$10.441,88 em uma projeção anual).

Dentre as aplicações de uso restrito, a que mais consumiu recursos durante o monitoramento foi a *ACMAdaptaCurric.exe*, que faz parte do Módulo Acadêmico e é utilizada para fazer ajustes no currículo de um determinado aluno. Durante os 20 minutos de monitoramento esta aplica-

ção teria gerado um custo de U$0,389, que em uma projeção anual equivale a U$2.352,66. A próxima aplicação da lista é a *BibCRegistroMarc.exe*, utilizada para catalogar o acervo bibliográfico.

Aplicação	Tempo de Resposta (normalizado)	Preço				
		20 min. (Tempo/328.500)	1 hora (20min x 3)	1 dia (1 hora x 8)	1 mês (1 dia x 21)	1 ano (1 mês x 12)
ConsultaWebGenerica	567.156 msec	U$ 1,73	U$ 5,18	U$ 41,44	U$ 870,16	U$ 10.441,88
ACMAdaptaCurric.exe	127.786 msec	U$ 0,389	U$ 1,17	U$ 9,34	U$ 196,06	U$ 2.352,66
BibCRegistroMarc.exe	89.410 msec	U$ 0,272	U$ 0,817	U$ 6,53	U$ 137,18	U$ 1.646,12
GCAMGUsuarios.exe	71.502 msec	U$ 0,218	U$ 0,653	U$ 5,22	U$ 109,70	U$ 1.316,41
ACMOcorCurric.exe	66.649 msec	U$ 0,203	U$ 0,609	U$ 4,87	U$ 102,26	U$ 1.227,08
GCANavegacao.exe	59.546 msec	U$ 0,181	U$ 0,544	U$ 4,35	U$ 91,36	U$ 1.096,30
GCAMUsuario.exe	41.783 msec	U$ 0,127	U$ 0,382	U$ 3,05	U$ 64,10	U$ 769,26
EMEmpenho.exe	26.276 msec	U$ 0,080	U$ 0,240	U$ 1,92	U$ 40,31	U$ 483,77
ACMOfertaTurmasPorCurso.exe	23.596 msec	U$ 0,072	U$ 0,215	U$ 1,72	U$ 36,20	U$ 434,43
ACMBolsista.exe	21.629 msec	U$ 0,066	U$ 0,198	U$ 1,58	U$ 33,18	U$ 398,21

Figura 4.12: As 10 aplicações com maior somatório do tempo de resposta e a simulação do preço de cada uma em função dos recursos consumidos.

A informação de preço detalhada a nível de aplicação pode ser utilizada tanto pela instituição usuária do ERP quanto por um desenvolvedor que, eventualmente, esteja fornecendo a solução através de uma plataforma *pay-as-you-go*. No primeiro caso, ela permite identificar o custo da infra-estrutura de TI alocada para desempenhar atividades específicas. Através das aplicações da figura 4.12, por exemplo, pode-se identificar atividades específicas relacionadas ao acervo bibliográfico (aplicações *ConsultaWebGenerica* e *BibCRegistroMarc.exe*), à elaboração do currículo dos alunos (*ACMAdaptaCurric.exe*), ao controle de pagamentos (*EMEmpenho.exe*), à gestão de bolsistas (*ACMBolsistas.exe*), entre outras.

No segundo caso, de um desenvolvedor de software que esteja distribuindo a solução através de uma plataforma *pay-as-you-go*, as informações podem ser utilizadas para verificar como está a sua margem de lucro em um determinado módulo ou aplicação que compõe a solução e, se for o caso, revisar a sua política de preços ou mesmo modificar a estrutura de funcionamento do sistema para que as funcionalidades mais acessadas consumam um volume menor de recursos.

Há ainda um outro tipo de informação que pode ser obtida em função da característica de integração dos ERPs. No caso do SIE, por exemplo, um usuário do sistema está relacionado a um funcionário, que por sua vez está lotado em um departamento. Com isso é possível usar

as informações de preço para identificar o volume de recursos de infra-estrutura que está sendo consumido por cada *centro de custo* [3] da instituição. A figura 4.13 demonstra como ficaria esta informação tomando como base os dados monitorados do SIE. Esta informação foi obtida considerando a lotação de cada usuário no cadastro do SIE. Por exemplo: o usuário *Francisco* está lotado no *DERCA - Depto. de Controle e Registro Acadêmico*, então todos os recursos de infra-estrutura que ele demandar serão apropriados para o *DERCA*. Segundo as informações da figura 4.13, o DERCA consumiu U$0,842 durante os 20 minutos de monitoramento, com uma projeção anual de U$12.101,05.

Centro de Custo	Tempo de Resposta (normalizado)	Preço				
		20 min. (Tempo/328.500)	1 hora (20min x 3)	1 dia (1 hora x 8)	1 mês (1 dia x 21)	1 ano (1 mês x 12)
Outros (Anônimo)	657.274 msec	U$ 2,001	U$ 6,003	U$ 48,02	U$ 1.008,42	U$ 12.101,05
DERCA (Depto. de Registro Acadêmico)	276.613 msec	U$ 0,842	U$ 2,526	U$ 20,21	U$ 424,39	U$ 5.092,72
Biblioteca Central	56.068 msec	U$ 0,171	U$ 0,512	U$ 4,10	U$ 86,02	U$ 1.032,26
DEMAPA (Depto. de Material e Patrimônio)	52.599 msec	U$ 0,160	U$ 0,480	U$ 3,84	U$ 80,70	U$ 968,39
PRAE (Pró-Reitoria de Assuntos Estudantis)	47.397 msec	U$ 0,144	U$ 0,433	U$ 3,46	U$ 72,72	U$ 872,63
PRRH (Pró-Reitoria de Recursos Humanos)	36.782 msec	U$ 0,112	U$ 0,336	U$ 2,69	U$ 56,43	U$ 677,19
PROPLAN (Pró-Reitoria de Planejamento)	14.948 msec	U$ 0,046	U$ 0,137	U$ 1,09	U$ 22,93	U$ 275,20
PRA (Pró-Reitoria de Administração)	13.074 msec	U$ 0,040	U$ 0,119	U$ 0,96	U$ 20,06	U$ 240,71
Gabinete do Reitor	7.460 msec	U$ 0,023	U$ 0,068	U$ 0,55	U$ 11,45	U$ 137,34
DAG (Depto. de Arquivo Geral)	6.857 msec	U$ 0,021	U$ 0,063	U$ 0,50	U$ 10,52	U$ 126,24

Figura 4.13: Os 10 *Centros de Custo* com maior somatório do tempo de resposta e a simulação do preço apropriado para cada um.

Conforme se pode notar, há questões e oportunidades específicas relacionadas aos sistemas ERP. A apropriação de preço para aplicações e centros de custo pode ser expandida para considerar linhas de produtos ou processos de trabalho, de forma a incorporar o consumo dos recursos de TI como custos variáveis em um sistema de produção. Análises que levam em consideração este tipo de informação podem gerar mudanças na maneira de utilizar o software como estratégia para redução de custos. Metodologias de análise e otimização de processos são bastante difundidas atualmente e podem ser avaliadas como forma de identificar e eliminar desperdícios com o uso de software.

[3] Uma unidade usada para apropriar despesas, em geral relacionada a contabilidade de custos.

4.6 Discussão

Antes de mais nada é preciso ressaltar que as informações de preço apresentadas neste capítulo são fruto de uma simulação e não podem ser utilizadas como base para se inferir o volume financeiro real que seria gasto para hospedar o ERP SIE em uma plataforma *pay-as-you-go*. Isto porque as informações i) estão baseadas em um monitoramento de apenas 20 minutos e ii) fazem uso de uma projeção de preço baseada em uma analogia com o tempo de resposta, que apesar de ser suficiente para o escopo deste trabalho está longe de possuir a precisão dos estudos tradicionais de *benchmark*.

Entretanto, estas mesmas medições e projeções demonstram que as necessidades de mudança apontadas no capítulo 3 deverão mesmo acontecer, a começar pelas próprias ferramentas de *benchmark*. Instituições como a SPEC precisarão desenvolver ferramentas que levem em consideração o preço das plataformas *cloud*. Tais ferramentas serão necessárias não apenas para comparar diferentes fornecedores, mas também para que os desenvolvedores de software tenham uma expectativa real do volume financeiro que será consumido pelas suas aplicações.

O aspecto que ficou mais evidenciado no estudo de caso com o ERP SIE na UFSM diz respeito às práticas para levantamento de requisitos, tema que foi abordado na seção 3.3. O monitoramento dos tempos de resposta do SIE indicou vários pontos de otimização que poderiam ser facilmente atacados como forma de reduzir o consumo total de recursos. Pelo menos seis métodos RPC poderiam ser implementados com uma estrutura de *cache*, o que reduziria o custo total de uso do sistema em cerca de 9,2% (uma economia anual de U$2.187,43 por servidor de aplicação - U$10.937,15 considerando que a UFSM utiliza cinco servidores). Estas otimizações não são implementadas atualmente por não terem uma relação custo/benefício que justifique o esforço da equipe de desenvolvimento, que está concentrada em desenvolver novos módulos e funcionalidades.

O modelo *pay-as-you-go* acrescenta uma nova variável no processo de desenvolvimento de software, que se pode chamar de "*consumo racional de recursos*". As práticas de desenvolvimento de software atuais não estão preparadas para tratar os recursos de hardware com esta abordagem.

Conforme demonstra a análise da seção 3.3, os recursos de hardware são tratados pela Engenharia de Software como *requisitos não-funcionais*, que formam uma base para avaliação dos critérios de qualidade relacionados a *eficiência* conforme preveem as normas ISO/IEC 9126 e ISO/IEC 25010. Além do atendimento a padrões gerais, a norma prevê dois itens relacionados

a *eficiência*: *tempo de resposta* e *utilização de recursos*. Para atender estas duas características, o processo de desenvolvimento de software atual está focado em garantir um bom tempo de resposta para os usuários do sistema, tomando o cuidado de, para isso, não utilizar os recursos disponíveis em níveis que cheguem próximo ao seu limite de esgotamento.

É esta abordagem que cria situações como as que foram reveladas neste estudo de caso, onde: i) pontos que não estão afetando diretamente o tempo de resposta não são priorizados com esforços de otimização e ii) podem dar origem a um consumo de recursos representativo no preço final a ser cobrado em um modelo *pay-as-you-go*.

Talvez a norma ISO/IEC em si não precise ser modificada, mas a forma como ela é interpretada e implementada pelos desenvolvedores de software pode precisar de ajustes para se adequar ao modelo *pay-as-you-go* e evitar situações como as demonstradas neste estudo de caso, que implicam em desperdício de recursos que podem ter impactos financeiros significativos. Neste contexto, o esquema da figura 4.14 pode servir de base para novos estudos que tenham por objetivo adaptar o desenvolvimento de software a esta nova realidade.

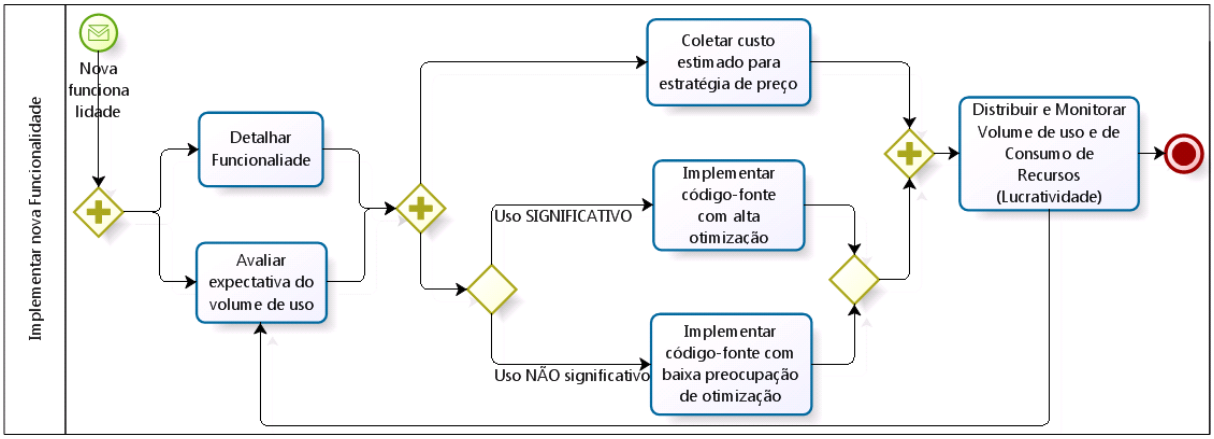


Figura 4.14: Esquema para o desenvolvimento de software em PaaS com modelo *pay-as-you-go*

A ideia proposta na figura 4.14 é que, ao mesmo tempo em que se está detalhando uma funcionalidade do sistema seja feita também uma avaliação da expectativa do volume de uso da funcionalidade. Com base nesta avaliação e nas próprias características da funcionalidade seria possível estimar o volume de recursos que será consumido e com isso decidir se ele é significativo a ponto de ser construído com a máxima otimização possível. Neste mesmo momento, caso o desenvolvedor de software pretenda distribuir a sua solução para terceiros, ele pode coletar informações do custo estimado de hardware para embasar a sua estratégia de formação de preço (o preço final cobrado deverá ser suficiente para cobrir todas as despesas de hardware e gerar uma margem de lucro satisfatória).

Todo este processo de estimativa precisaria ser suportado por uma arquitetura que permita o monitoramento constante do volume de uso real do sistema. Esta arquitetura é importante para que o desenvolvedor possa verificar periodicamente como está o consumo de recursos de cada funcionalidade e, se for o caso, implementar as otimizações necessárias, revisar a estrutura do software e/ou rever as suas políticas de preço.

5 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou uma visão sobre *cloud computing* que ainda vem sendo pouco abordada pela comunidade científica: o seu impacto nas questões relacionadas ao desenvolvimento de software. São apresentados os conceitos de *Cloud Computing*, os tipos de serviço oferecidos e o papel das plataformas (*PaaS*) no desenvolvimento de software para o modelo *cloud*.

As Plataformas como Serviço (*PaaS*) são a solução natural para os desenvolvedores que planejam distribuir seus softwares no ambiente *cloud* e já existem plataformas disponíveis no mercado que adotam, em diferentes níveis, o modelo *pay-as-you-go*. O modelo *pay-as-you-go* é o mesmo utilizado atualmente por diversos serviços de infra-estrutura do nosso cotidiano, como água e energia elétrica, em que se tem a ilusão de "recursos infinitos"e se paga conforme o volume de recursos que for efetivamente utilizado. No caso das *PaaS*, a tendência é a padronização dos seus modelos de preço, com valores na casa de centavos de dólar para itens como capacidade de processamento (CPU), GB armazenados e GB transferidos, semelhante ao que vem sendo observado nos serviços *IaaS* oferecidos por empresas como a *Amazon*.

Durante a análise das plataformas foram feitos experimentos para simular o impacto financeiro que uma otimização de código poderia ter no preço final pago ao fornecedor da plataforma. A partir destes experimentos foram identificados aspectos do desenvolvimento de software que poderiam sofrer mudanças em função do novo modelo *pay-as-you-go*, que deverá ganhar força nos próximos anos na esteira do crescimento dos ambientes *cloud*. Estas mudanças estão relacionadas justamente ao modelo de cobrança *pay-a-you-go*, que deverá exigir dos desenvolvedores de software uma abordagem diferente em relação ao consumo dos recursos de hardware, já que neste modelo o volume de recursos consumidos deverá ter influência direta no preço final do software.

Os aspectos identificados a partir das observações iniciais foram objeto de uma análise criteriosa com o objetivo de identificar de forma específica os pontos em que as plataformas *pay-as-you-go* deverão gerar um impacto suficiente para que novas pesquisas sejam desenvolvidas sobre o tema. Os aspectos em que esta análise de impacto foi realizada estão realacionados à *precificação de software*, *estimativas de desenvolvimento* (com o COCOMO), *engenharia de SW e requisitos* (com a norma ISO/IEC 25010) e *benchmarks de desempenho* (a partir dos *benchmarks* da SPEC).

Dentre os aspectos analisados, o da *engenharia de software e requisitos* foi aquele aquele cujos impactos foram mais evidenciados pelo estudo de caso, onde foi possível observar que os métodos tradicionais podem levar a um desperdício no consumo de recursos de hardware, que por sua vez tem impacto direto no preço do software. Há ainda outros aspectos que poderão ser afetados e não foram avaliados neste trabalho, como por exemplo o uso das plataformas *pay-as-you-go* durante o próprio processo de desenvolvimento de software. A própria análise realizada por este trabalho não é definitiva e novos estudos podem ser desenvolvidos nesta área, levando em consideração principalmente a temática do "consumo racional de recursos".

A utilização de estudos de caso para observar o impacto das mudanças também se mostra importante, pois além de ser útil para confirmar (ou desconfirmar) as hipóteses de mudança, também pode indicar novas oportunidades de pesquisa, como foi o caso do estudo com o ERP SIE, que identificou oportunidades específicas relacionadas a sistemas do tipo ERP.

O trabalho também produziu um artigo científíco que foi aceito na *XXXVII Conferencia Latinoamericana de Informática* (CLEI 2011, a ser realizado no Equador). O artigo enviado e aprovado pela comissão do evento é uma versão resumida deste trabalho, e seu texto contém uma versão bastante incipiente do estudo de caso com o ERP SIE e apresenta somente o potencial de impacto no desenvolvimento de software, sem a análise criteriosa que foi apresentada no capítulo 3.

As principais contribuições deste trabalho são:

- Uma análise do ambiente *cloud* e das plataformas *pay-as-you-go* sob uma ótica que vem sendo pouco abordada pela comunidade científica: a do desenvolvimento e precificação de software, indicando onde e como os aspectos analisados serão afetados, o que inclui os seguintes itens: *Precificação*, *Estimativas de Desenvolvimento*, *Engenharia de SW e Requisitos* e *Benchmarks de Desempenho*;

- Um estudo de caso que evidencia:

  Os desperdícios de recursos de hardware em sistemas que não foram construídos com a preocupação do consumo racional de recursos e o potencial de impacto financeiro gerado por estes desperdícios.

  A necessidade de mudança nos processos de levantamento de requisitos para evitar estas situações.

- A identificação, a partir do estudo de caso, de oportunidades específicas para sistemas do tipo ERP, que poderão:

  No caso dos clientes: detalhar gastos com recursos de TI em um nível relacionado às atividades da instituição e dos seus centros de custo.

  No caso dos fornecedores: avaliar o lucro (ou prejuízo) de cada módulo/funcionalidade do sistema.

Como trabalhos futuros, pode-se citar:

- Novos estudos para identificar outros aspectos do desenvolvimento de software que não foram abordados por este trabalho, dentre eles:

  A manutenção e adição de novas funcionalidades a softwares já existentes.

  A utilização do modelo *pay-as-you-go* durante o próprio processo de desenvolvimento de software, o que inclui o consumo de recursos na fase de testes e simulação.

- A construção e/ou adaptação de frameworks de desenvolvimento que deem suporte ao monitoramento contínuo e detalhado dos recursos de hardware consumidos.

- A construção de modelos de preço que levem em consideração o fato de os recursos de hardware estarem incorporados ao preço do software.

- Um estudo para calcular a estimativa real de custo que se teria para distribuir um ERP como o SIE em uma plataforma *pay-as-you-go*.

- A revisão e adaptação das práticas e modelos atuais relacionadas ao desenvolvimento de software, em especial:

  O método COCOMO, especificamente na forma de medição dos recursos de hardware: *Restrições de CPU* e *Restrições de Armazenamento*.

  As estratégias e modelos relacionados ao levantamento de *requisitos não-funcionais*, utilizados atualmente para identificar os requisitos de hardware.

  Os *benchmarks* de desempenho, especificamente nas questões relacionadas a preço.

REFERÊNCIAS

AL-QUTAISH, R. An Investigation of the Weaknesses of the ISO 9126 International Standard. In: COMPUTER AND ELECTRICAL ENGINEERING, 2009. ICCEE '09. SECOND INTERNATIONAL CONFERENCE ON, 2009. **Anais...** [S.l.: s.n.], 2009. v.1, p.275 –279.

AMAZON. **Amazon S3**. Acessado em Dez/2010, http://aws.amazon.com/s3.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R. H.; KONWINSKI, A.; LEE, G.; PATTERSON, D. A.; RABKIN, A.; STOICA, I.; ZAHARIA, M. **Above the Clouds**: a berkeley view of cloud computing. Berkeley, California, USA: [s.n.], 2009.

BARBOSA, F. P. **Projeto e Implementação de Um Framework para Desenvolvimento de Aplicações em Três Camadas**. Santa Maria: Curso de Ciência da Computação. Universidade Federal de Santa Maria., 2000.

BARBOSA, F. P.; CHARãO, A. Uma análise do impacto das plataformas pay-as-you-go de computação em nuvem no desenvolvimento e precificação de software. In: XXXVII CONFERêNCIA LATINOAMERICANA DE INFORMáTICA (XXXVII CLEI), 2011. **Proceedings...** [S.l.: s.n.], 2011.

BOEHM, B.; VALERDI, R. Achievements and Challenges in Cocomo-Based Software Resource Estimation. **Software, IEEE**, [S.l.], v.25, n.5, p.74 –83, sept.-oct. 2008.

BOHEM, D. **COCOMO II - Model Definition Manual, version 1.4**. USA: [s.n.], 2000.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S. Market-Oriented Cloud Computing: vision, hype, and reality for delivering it services as computing utilities. In: HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS, 2008. HPCC '08. 10TH IEEE INTERNATIONAL CONFERENCE ON, 2008. **Anais...** [S.l.: s.n.], 2008. p.5 –13.

CAI, Y.; CHEN, W. Software Product Pricing Strategies Study Based on Dynamic Stochastic Wealth Model. In: WIRELESS COMMUNICATIONS, NETWORKING AND MOBILE COMPUTING, 2008. WICOM '08. 4TH INTERNATIONAL CONFERENCE ON, 2008. **Anais...** [S.l.: s.n.], 2008. p.1 –4.

CS, D. **The Department of Computer Science at Duke University**. Acessado em Jul/2011, http://www.cs.duke.edu/.

DAKIN, K. Establishing a fair price for software. **Software, IEEE**, [S.l.], v.12, n.6, p.105 –106, nov 1995.

DING, J.; HILLSTON, J.; LAURENSON, D. Evaluating the Response Time of Large Scale Content Adaptation Systems Using Performance Evaluation Process Algebra. In: COMMUNICATIONS (ICC), 2010 IEEE INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. p.1 –5.

The Pricing Decision: economic theory and business theory. [S.l.]: Harper e Row Ltd., London, 1987.

FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. Cloud Computing and Grid Computing 360-Degree Compared. In: GRID COMPUTING ENVIRONMENTS WORKSHOP, 2008. GCE '08, 2008. **Anais...** [S.l.: s.n.], 2008. p.1 –10.

GILADI, R.; AHITAV, N. SPEC as a performance evaluation measure. **Computer**, [S.l.], v.28, n.8, p.33 –42, aug 1995.

GOOGLE. **AppEngine**. Acessado em Dez/2010, http://code.google.com/appengine/docs/billing.html.

Pricing, Principles and Practice. [S.l.]: Heinemann Educational, London, 1977.

GUHA, R.; AL-DABASS, D. Impact of Web 2.0 and Cloud Computing Platform on Software Engineering. In: ELECTRONIC SYSTEM DESIGN (ISED), 2010 INTERNATIONAL SYMPOSIUM ON, 2010. **Anais...** [S.l.: s.n.], 2010. p.213 –218.

HAMID R MOTAHARI-NEZHAD BRYAN STEPHENSON, S. S. **Outsourcing Business to Cloud Computing Services**: opportunities and challenges. USA: [s.n.], 2009.

HARMON, R.; RAFFO, D.; FAULK, S. Incorporating price sensitivity measurement into the software engineering process. In: MANAGEMENT OF ENGINEERING AND TECHNOLOGY, 2003. PICMET '03. TECHNOLOGY MANAGEMENT FOR RESHAPING THE WORLD. PORTLAND INTERNATIONAL CONFERENCE ON, 2003. **Anais...** [S.l.: s.n.], 2003. p.316 – 323.

HEROKU. **Heroku**. Acessado em Dez/2010, http://www.heroku.com/pricing.

ISO. **ISO/IEC 9126. Software engineering – Product quality**. [S.l.]: ISO, 2001.

ISO. **ISO/IEC 25010. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE)**. [S.l.]: ISO, 2011.

JALOTE, P. **An integrated approach to software engineering**. New York, NY, USA: Springer-Verlag New York, Inc., 1991.

KAMDAR, A.; ORSONI, A. Development of Value-Based Pricing Model for Software Services. In: COMPUTER MODELLING AND SIMULATION, 2009. UKSIM '09. 11TH INTERNATIONAL CONFERENCE ON, 2009. **Anais...** [S.l.: s.n.], 2009. p.299 –304.

LI, A.; YANG, X.; KANDULA, S.; ZHANG, M. CloudCmp: comparing public cloud providers. In: INTERNET MEASUREMENT, 10., 2010, New York, NY, USA. **Proceedings...** ACM, 2010. p.1–14. (IMC '10).

MICROSOFT. **Azure AppFabric**. Acessado em Dez/2010, http://www.microsoft.com/windowsazure/appfabric/overview/.

PANDEY, D.; SUMAN, U.; RAMANI, A. An Effective Requirement Engineering Process Model for Software Development and Requirements Management. In: ADVANCES IN RECENT TECHNOLOGIES IN COMMUNICATION AND COMPUTING (ARTCOM), 2010 INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. p.287 –291.

PSM. **Practical Software and Systems Measurement (PSM)**. Acessado em Jul/2011, http://www.psmsc.com.

QIN, W.; RU-XIANG, W. Research of military software pricing based on binomial tree method. In: COMPUTER SCIENCE AND INFORMATION TECHNOLOGY (ICCSIT), 2010 3RD IEEE INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. v.9, p.628 –632.

QIN, W.; RUXIANG, W. Study and application of military software pricing based on option pricing. In: SOFTWARE TECHNOLOGY AND ENGINEERING (ICSTE), 2010 2ND INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. v.2, p.V2–335 –V2–339.

QIN, W.; YADI, Z. The study of military software pricing based on option pricing models. In: ADVANCED COMPUTER THEORY AND ENGINEERING (ICACTE), 2010 3RD INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. v.6, p.V6–163 –V6–167.

SALESFORCE. **Salesforce platform**. Acessado em Dez/2010, http://www.salesforce.com/platform/platform-edition/.

SEI. **Measuring Software Product Quality**: the iso 25000 series and cmmi. Acessado em Jul/2011, http://www.sei.cmu.edu/library/assets/esepg.pdf.

SEI/PSM. **Software Quality Requirements and Evaluation, the ISO 25000 Series**. Acessado em Jul/2011, http://www.psmsc.com/Downloads/TWGFeb04/04ZubrowISO25000SWQualityMeasurement.pdf.

SEI/SEPG. **SEPG Conference Series**. Acessado em Jul/2011, http://www.sei.cmu.edu/sepg/.

SHANG, S.; JIANG, J.; WU, Y.; YANG, G.; ZHENG, W. A Knowledge-based Continuous Double Auction Model for Cloud Market. In: SEMANTICS KNOWLEDGE AND GRID (SKG), 2010 SIXTH INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. p.129 –134.

SPEC. **Standard Performance Evaluation Corporation**. Acessado em Jul/2011, http://www.spec.org/.

SPECJENTERPRISE. **SPECjEnterprise2010**. Acessado em Jul/2011, http://www.spec.org/jEnterprise2010/.

WIKIPEDIA. **Wikipedia Categories**: cloud platform. Acessado em Dez/2010, http://en.wikipedia.org/wiki/Category:Cloud_platforms.

The Marketing of Professional Services. [S.l.]: McGraw-Hill, London, 1972.

ZHENG, Y.; CAO, R.; SUN, W.; ZHANG, K.; JIANG, Z. Practical Application of FDC in Software Service Pricing. In: BUSINESS ENGINEERING, 2006. ICEBE '06. IEEE INTERNATIONAL CONFERENCE ON, 2006. **Anais...** [S.l.: s.n.], 2006. p.352 –357.