DISSERTAÇÃO DE MESTRADO

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM

CIÊNCIA DA COMPUTAÇÃO

"Catálogo de Padrões para o Desenvolvimento de Software como um Serviço Multi-Tenant"

ALUNO: Bruno Dias Leite
ORIENTADOR: Prof. Dr. Rosângela A. D. Penteado

São Carlos
Setembro/2014

CAIXA POSTAL 676
FONE/FAX: (16) 3351-8233
13565-905 - SÃO CARLOS - SP
BRASIL

UNIVERSIDADE FEDERAL DE SÃO CARLOS

DEPARTAMENTO DE COMPUTAÇÃO

BRUNO DIAS LEITE

CATÁLOGO DE PADRÕES PARA O DESENVOLVIMENTO DE SOFTWARE COMO UM SERVIÇO MULTI-TENANT

Dissertação apresentado ao Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de mestre em Computação

Orientação: Prof. Dra. **Rosângela Ap. Dellosso Penteado**

SÃO CARLOS – SP

2015

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

L533cp

Leite, Bruno Dias.
Catálogo de padrões para o desenvolvimento de software como um serviço *multi-tenant* / Bruno Dias Leite. -- São Carlos : UFSCar, 2015.
128 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2014.

1. Software. 2. Serviços de software. 3. Linha de produtos de software. 4. Arquitetura de software. 5. Customização. 6. Reuso. I. Título.

CDD: 005.3 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Catálogo de Padrões para o Desenvolvimento de Software Como um Serviço Multi-Tenant"

Bruno Dias Leite

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Profa. Dra. Rosângela Ap. Dellosso Penteado
(Orientadora - DC/UFSCar)

Profa. Dra. Maria Istela Cagnin Machado
(UFMS)

Prof. Dr. Antonio Francisco do Prado
(DC/UFSCar)

São Carlos
Outubro/2014

AGRADECIMENTOS

Agradeço à professora Drª. Rosângela, minha orientadora, por todo carinho, dedicação, paciência e amizade durante todos esses anos de convivência. Sou grato por sua confiança e participação, que foram fundamentais para realização deste trabalho.

Agradeço aos meus pais, que ensinaram-me desde cedo a importância dos estudos em minha vida, conduzindo-me para o caminho do conhecimento. Agradeço também pelo apoio moral e financeiro nos momentos difíceis.

Agradeço aos amigos que apoiaram meu trabalho, discutindo soluções, ajudando a conhecer o estado da arte em computação ou simplesmente proporcionando momentos de descontração.

Agradeço a Mariele por ter me incentivado a ingressar no mestrado e ter me apoiado em todos os momentos.

Agradeço a COSS Consulting e ao FIT-Instituto de Tecnologia por acreditarem na minha pesquisa, permitindo ausentar-me do local de trabalho para assistir as aulas, sem detrimento financeiro.

Enfim, agradeço a todos que contribuíram direta ou indiretamente para a realização deste trabalho.

RESUMO

Softwares como um Serviço (SaaS) representam uma forma de distribuição de software sob demanda e acessíveis via Internet. O desenvolvimento de SaaS possibilita aos consumidores do serviço (*tenants*) se beneficiarem do baixo custo de implantação e manutenção. A arquitetura *Multi-Tenant* (MT) é frequentemente utilizada no desenvolvimento de SaaS, pois permite o compartilhamento de recursos de software e hardware, proporcionando a redução de custos ao consumidor e ao fornecedor do serviço. Um desafio que o desenvolvedor de SaaS tem é o de manter o compartilhamento da instância do software em cenários com *tenants* que possuem necessidades divergentes. Neste projeto, para tratar a variabilidade dos *tenants* em SaaS MT foram criados alguns padrões, com base em técnicas de linhas de produtos de software, para customização de interfaces de usuários, dados, processos e permissões. Um *framework* denominado FoSaaS (*Framework of Software as a Service*) também foi desenvolvido para apoiar o uso dos padrões criados fornecendo apoio aos desenvolvedores na codificação de SaaS MT. Além de permitir a customização de SaaS MT, experimentos realizados neste projeto indicaram que os padrões melhoram o reúso, a qualidade e a produtividade dos SaaS MT desenvolvidos. A aplicabilidade dos padrões foi evidenciada por meio da realização de projetos reais no desenvolvimento de aplicações comerciais.

Palavras-chave: Software como um Serviço (SaaS), Linhas de Produtos de Software (LPS), arquitetura *Multi-Tenant*, gerenciamento de variabilidades, customização de software, reúso.

ABSTRACT

Software as a Service (SaaS) represent a form of software distribution on demand and accessible via the Internet. The development of SaaS enables service users (tenants) to benefit from the low cost of deployment and maintenance. The Multi-Tenant (MT) architecture is often used in the development of SaaS, it allows the sharing of software and hardware resources, providing cost savings to the costumers and the service provider. A challenge that the SaaS developer has is to keep sharing the instance of the software in scenarios with tenants who have differing needs. In this project, to address the variability of tenants in SaaS MT were set some patterns, based on Software Product Lines techniques for customizing user interfaces, data, processes and permissions. A framework called FoSaaS (Framework of Software as a Service) has also been developed to support the use of patterns, providing support to developers in SaaS MT coding. In addition to allowing the customization of SaaS MT, experiments in this project indicated that patterns improve reuse, quality and productivity in developed SaaS MT. The applicability of the patterns was evidenced through real projects in developing commercial applications.

Keywords: Software as a Service (SaaS), Software Product Lines (SPL), Multi-Tenant architecture, variability management, software customization, reuse.

LISTA DE FIGURAS

LISTA DE TABELAS E QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ASP - Provedores de Serviço de Aplicação (*Application Service Providers*)

BaaS – Negócio como Serviço (*Business as a Service*)

CI – Critérios de Inclusão

CE – Critério de Exclusão

DaaS - [Desenvolvimento, Banco de Dados, Área de Trabalho] como um Serviço ([*Development, Database, Desktop*] *as a Service*)

DDM - Desenvolvimento Dirigido por Modelos

DNS - *Domain Name System*

DSSA - *Domain-Specific Software Architecture*

IaaS – Infraestrutura como um Serviço (*Infraestructure as a Service*)

FaaS – Arcabouço como Serviço (*Framework as a Service*)

FAST - *Family-Oriented Abstration, Specification and Translation*

FODA - *Feature-Oriented Domain Analysis*

JVM – Máquina virtual Java (*Java Virtual Machine)*

HaaS – Hardware como um Serviço (*Hardware as a Service*)

MVO - Modelo de Variabilidade Ortogonal

MT – *Multi-Tenant*

LPS – Linha de Produtos de Software

SaaS – Software como um Serviço (*Software as a Service*)

PaaS – Plataforma como um Serviço (*Platform as a Service*)

PICOC - *Population, Intervention, Comparison, Outcome, Context*

PuLSE - *Product Line Software Engineering*

QP – Questão de Pesquisa

RS - Revisão Sistemática

URL - *Uniforme Resource Locator*

XaaS – Qualquer coisa como Serviço (*Everything as a Service*)

SUMÁRIO

Capítulo 1
INTRODUÇÃO

1.1 Contextualização

As constantes manutenções aplicadas em um software é uma das principais preocupações de desenvolvedores, que buscam soluções para produzir softwares com qualidade, reúso e maior produtividade. Softwares como um Serviço (*Software as a Service* - SaaS) é um modelo de distribuição de software em larga escala, com enfoque no baixo custo de implantação e manutenção mais eficiente (Nitu, 2009).

Os SaaS possibilitaram às pequenas empresas contratar sistemas de informação robustos, mediante o pagamento de uma taxa mensal de acordo com sua demanda (Taurion, 2009). A adoção de SaaS proporciona uma série de vantagens aos consumidores do serviço, conhecidos como *tenants*. Dentre essas vantagens pode-se citar como as principais: a redução dos custos de implantação, manutenção e evolução do software (Han, 2010). Do ponto de vista do fornecedor do serviço, a redução dos custos operacionais favorece a oferta do SaaS para uma demanda maior dos *tenants*.

Em um cenário em que existem muitos *tenants* do serviço, cada *tenant* necessita de uma instância única da aplicação, sendo que uma instância representa o software compilado em execução. Esse modelo, chamado de *single-tenant,* não favorece a redução dos recursos providos pelo fornecedor do serviço. Isto acontece, pois os recursos são dedicados exclusivamente para cada *tenant*, mesmo quando não estão em uso (Weissman e Bobrowski, 2009). Uma solução para reduzir os custos de execução dos SaaS, quando

considerados recursos como energia e hardware é a utilização da arquitetura *Multi-Tenant* (MT) (Turner, 2003; Chong, 2006), uma arquitetura que possibilita o compartilhamento da instância do SaaS por mais de um *tenant* simultaneamente. O compartilhamento da instância entre múltiplos *tenant*, permite ao fornecedor do SaaS minimizar os esforços despendidos em manutenções corretivas, adaptativas e evolutivas, uma vez que apenas um código-fonte e uma instalação do software precisam ser manipuladas.

Embora os benefícios da arquitetura MT justifiquem o crescimento do mercado de SaaS, ainda existem diversos desafios em termos segurança, desempenho, escalabilidade e customização de *tenants* a ser resolvidos (Han, 2010). Os problemas associados a arquitetura MT são causados pela variação dos requisitos funcionais e não-funcionais dos *tenants*. Para que os SaaS MT possam atender a diversidade de requisitos dos *tenants* é necessário que o software seja customizável (Zhang *et al.*, 2007; Mietzner *et al.*, 2009). A variabilidade dos *tenants* pode ser considerada em termos de interfaces com usuários (*User Interfaces* - UI), dados, processos e permissões (Kitano *et al.*, 2010; Li *et al.*, 2011, 2009; Mietzner*et al*, 2009; Sun *et al.*, 2008; Ruehl e Andelfinger, 2011; Yao *et al.*, 2011; Zainuddin e Gonzalez, 2011). Esse trabalho de mestrado tratou os problemas associados a variação dos requisitos funcionais dos tenants. O tratamento dos requisitos não-funcionais será abordado em trabalhos futuros.

Northop (2008) afirmou que o conceito de Linhas de Produtos de Software (LPS) é uma forma promissora de reúso, pois tem como objetivo promover a geração de produtos específicos com base na reutilização de uma infraestrutura central (Clements e Northop, 2002). A manipulação e gerenciamento das variabilidades em LPS têm técnicas e métodos capazes de tratar LPS com alta variabilidade (Pohl *et al.*, 2005). Visando criar padrões para a customização dos *tenants* em SaaS MT, este trabalho propõe aplicar técnicas e métodos de LPS para tratar a variabilidade dos *tenants*.

Diferente de LPS, a variabilidade dos *tenants* deve ser manipulada em tempo de execução, uma vez que a instância do SaaS deve reunir todas as variabilidades desejadas por um conjunto de *tenants*. A vantagem desta proposta em relação ao desenvolvimento tradicional de SaaS é que permite manipular com reúso as funcionalidades similares dos consumidores do serviço.

1.2 Motivação e Objetivo

Cherobino (2007) afirma que pessoas e empresas têm utilizado cada vez mais serviços de Internet. Essa informação pode ser confirmada pela pesquisa realizada pela *International Data Corporation* (IDC) em 2012 (IDC, 2013), que apontou que os serviços de nuvem geraram 28 bilhões de dólares em lucro para companhias e que em 2017 o valor gerado será de 76,1 bilhões de dólares.

Softwares na forma de serviço têm acompanhado essa tendência mundial. Um estudo realizado pela Gartner Group em 2011, mostrou que no ano de 2010 as vendas de SaaS atingiram os 9 milhões de dólares e 10,7 milhões em 2011, indicando crescimento de 16,2%. Esse mesmo estudo indicou que a porcentagem de software comercializado na forma de serviço representava 10% do total em 2010 e que em 2014 poderia chegar aos 16%.

Semelhante aos SaaS, Linhas de produtos de software têm sido utilizadas por grandes corporações a fim de reduzir os custos de produção e aumentar a produtividade. A Hewlett-Packard (HP) aumentou em 600% a produtividade dos sistemas de impressoras, já a Motorola apresentou crescimento de 400% na produtividade dos sistemas de celulares. Ambas utilizaram LPS (Bass *et al.*, 2003). A alta efetividade de LPS no desenvolvimento de software e o crescimento dos SaaS influenciaram para a criação desta proposta.

Além desses indicativos apresentados, alguns dos motivos que também influenciaram a investigação do tema deste projeto de mestrado foram:

- **Ausência de abordagens para criar SaaS MT customizáveis com foco na implementação:** um mapeamento sistemático realizado por este autor indicou a ausência dessas abordagens. Alguns estudos envolvem abordagens que relacionam técnicas de LPS e SaaS MT, contudo os resultados obtidos estavam voltados apenas para gestão e configuração dos *tenants* em SaaS MT*;* Embora os trabalhos obtidos não tratassem a implementação das variabilidades, a configuração dos *tenants* proposto por esse trabalho foi baseado nos estudos apresentados por Julia Schroeter *et al.* (2012) e Ralph Mietzner *et al.* (2009).
- **Existência de poucos padrões para desenvolvimento de SaaS MT customizáveis**: uma revisão sistemática dos padrões para desenvolvimento de SaaS MT resultou em seis padrões voltados para customização dos *tenants* (Khan *et al.*, 2011). Os padrões

obtidos não apresentavam a descrição formal necessária para que fossem utilizados em outros estudos, pois nenhum dos padrões descrevia a estrutura ou exemplos de código;

- **Experiência profissional do autor em desenvolvimento de SaaS MT:** o autor desta dissertação possui três anos de experiência profissional no desenvolvimento de SaaS, o que possibilitou evidenciar na prática, os desafios enfrentados pelo uso da arquitetura MT. Por meio das experiências profissionais, o autor observou que o desenvolvimento dos SaaS MT era considerado *ad-hoc*, uma vez que cada funcionalidade era desenvolvida específica por *tenant*. Dessa forma, a evolução do software não estava baseada em reúso, tornando o código-fonte complexo e com baixa manutenibilidade.

Os objetivos propostos para este trabalho de mestrado estão relacionados ao contexto e motivação apresentados:

- **Criar padrões para o desenvolvimento de SaaS MT customizáveis:** desenvolver um catálogo de padrões utilizando a descrição formal apresentada por Gamma *et al.* (1995). Os padrões devem tratar da customização de interfaces de usuário (UI), dados, processos e permissões dos *tenants.* Os padrões devem prover mais produtividade ao desenvolvimento e melhorar a qualidade do código-fonte em comparação com o desenvolvimento *ad-hoc* de SaaS MT.
- **Criar um *framework* para apoiar a implementação de SaaS MT:** desenvolver um *framework* de aplicação que incorpore os padrões propostos por este estudo e auxilie o desenvolvedor de software a criar SaaS MT com melhor produtividade, reúso e qualidade.
- **Realizar um estudo de aplicabilidade da proposta:** aplicar os padrões criados nesta proposta no desenvolvimento de SaaS MT com *tenants* que apresentem requisitos variantes. Dessa forma, a proposta é avaliada em cenários reais, sendo que o SaaS MT deve atender aos requisitos de todos *tenants* simultaneamente. Esse estudo é importante para auxiliar o refinamento dos padrões e avaliar a efetividade da proposta.
- **Avaliar a proposta por meio de experimentos controlados**: experimentos são realizados no ambiente de trabalho do autor e com alunos de pós-graduação da Universidade Federal de São Carlos. Os experimentos são realizados de acordo com

o proposto pelo método *Goal/Question/Metric* (Basili *et al.*, 1994), e visam a avaliar os padrões desenvolvidos por este projeto de mestrado, em relação à qualidade, ao nível de reúso e à produtividade no desenvolvimento de SaaS MT.

1.3 Organização

Esta dissertação está organizada em sete capítulos. O primeiro capítulo apresentou a contextualização da proposta, a motivação e os objetivos pretendidos. No Capítulo 2 são apresentados os principais conceitos utilizados para o desenvolvimento deste trabalho: SaaS, computação em nuvem e LPS. No Capítulo 3 é apresentada a teoria de padrões para desenvolvimento de software e um mapeamento sistemático realizado sobre o uso de padrões para o desenvolvimento de SaaS MT. No Capítulo 4 é elaborado um catálogo de padrões criados para manipular a variabilidade dos *tenants*, em relação as UI, aos dados, processos e permissões. No Capítulo 5 é descrito o *Framework of Software as a Service* (FoSaaS), uma proposta de um artefato de software criado para auxiliar a adoção dos padrões no desenvolvimento de SaaS MT. No Capítulo 6 é realizada a avaliação dos padrões e do FoSaaS por meio de estudos de aplicabilidade e experimentos controlados. No último capítulo são discutidas algumas considerações sobre o projeto realizado e também são apresentadas as suas limitações e alguns dos trabalhos futuros que podem ser realizados a fim de complementar esta proposta.

Capítulo 2
FUNDAMENTAÇÃO TEÓRICA E COMPUTACIONAL

2.1 Considerações Iniciais

A computação em nuvem e a Internet formam o núcleo da nova geração de tecnologia da informação. As facilidades de acesso aos serviços oferecidos pela Internet possibilitam organizações usar softwares instalados fora das suas propriedades e ainda sim serem confiáveis. A expansão da computação em nuvem fez surgir um novo modelo de distribuição de software, voltado para exploração da economia em escala. Esse modelo, chamado Software como um Serviço (SaaS) (Nitu, 2009), é a nova forma de distribuir software, com enfoque no baixo custo e atendimento sob demanda, semelhante a assinatura dos serviços de telefonia e televisão.

Neste capítulo são apresentados os principais conceitos para estabelecer a fundamentação teórica, visando ajudar o entendimento da proposta de mestrado. Na Seção 2.2 são apresentados os conceitos de SaaS, mostrando os benefícios e problemas associados a esse modelo de distribuição de software. Um exemplo simplificado de SaaS também é apresentado. Na Seção 2.3 são apresentados os conceitos de Linha de Produtos de Software, com ênfase nas técnicas para o seu desenvolvimento. Na Seção 2.4 são expostas as considerações finais sobre a fundamentação teórica.

2.2 Software como um Serviço

SaaS é um modelo de distribuição, no qual os softwares são instalados em servidores do fornecedor de serviço e acessados via Internet (Nitu, 2009). A distribuição de software na forma de serviço, normalmente envolve alguma forma de pagamento mensal, sendo disponibilizado e utilizado por meio de navegadores web (Han, 2010). Assim, um SaaS deve ser mantido e executado no domínio do fornecedor de serviço (Tao e Liao, 2008).

No modelo de distribuição SaaS, o fornecedor de serviço pode atender diversos clientes ao mesmo tempo, utilizando a mesma infraestrutura de software e hardware. Os clientes, também chamados de inquilinos – *tenants*, beneficiam-se do baixo investimento inicial, sem a necessidade de adquirir hardware para executar o software. Outro benefício é a liberdade de uso segundo a demanda, além de redirecionar a responsabilidade de manutenção, gerenciamento de software e hardware para o fornecedor de serviço (Han, 2010). Essas vantagens, incentivaram muitas organizações a converterem seus softwares para o modelo SaaS (Ma, 2007), por exemplo: Lojas Virtuais do UOL, Shopify, Basecamp, Granatum e Salesforce.

2.2.1 Computação em nuvem e o surgimento dos SaaS

A computação em nuvem tem como princípio, o conceito de permitir que consumidores de software adquiram recursos computacionais sob demanda e o pagamento desse serviço seja realizado segundo o volume de utilização (Armbrust *et al.*, 2009). Vaquero (2008) afirma que nuvens são grandes repositórios de recurso virtualizados (plataforma de desenvolvimentos, hardware ou serviços), facilmente acessíveis. Nesse contexto, foram definidos diferentes tipos de taxonomia para identificar os modelos oferecidos segundo o pagamento por uso. Muitas taxonomias foram criadas sob a perspectiva de negócios (Rimal; Choi; Lumb, 2009) e surgiram muitos termos para identificar os serviço: HaaS (Hardware como Serviço - *Hardware as a Service*), PaaS (Plataforma como Serviço - *Platform as a Service*), DaaS ([Desenvolvimento, Banco de Dados, Área de Trabalho] como Serviço - [Development, Database, Desktop] as a Service), IaaS (Infraestrutura como Serviço - *Infrastructure as a Service*), BaaS (Negócios como Serviço - *Business as a Service*), FaaS (Arcabouços como Serviço -

Framework as a Service), e o termo XaaS (Qualquer Coisa como Serviço - *Everything as a Service*), esse último conceito agrupa os serviços de forma genérica.

Na literatura, esses modelos são frequentemente agrupados em três categorias principais: SaaS, IaaS e PaaS. A PaaS é a distribuição de sistemas operacionais e plataformas de desenvolvimento acessíveis via Internet, sem a necessidade de download ou instalação; A IaaS é caraterizada pela terceirização da infraestrutura computacional, como servidores, armazenamento de dados e componentes de redes (Banerjee *et al.*, 2011). Os SaaS tratam de software acessíveis via Internet e pagos segundo a demanda de uso.

Uma antologia, proposta por Youseff, Butrico e Da Silva (2008), apresenta a relação entre os serviços oferecidos pela computação em nuvem. A Figura 2.1 ilustra essa ontologia.

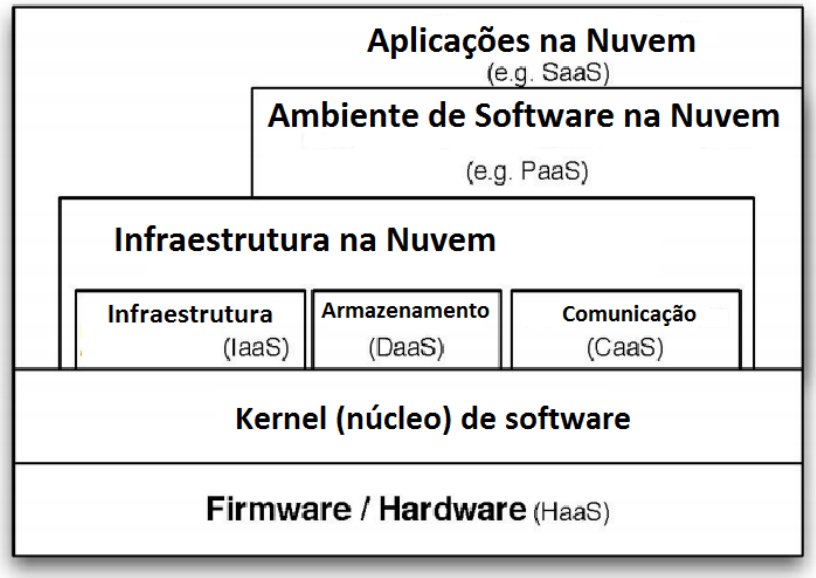


Figura 2.1: Ontologia da computação em nuvem (adaptado de Youseff, Butrico e Da Silva (2008)).

Esses modelos de serviço surgiram da evolução dos Fornecedores de Serviço de Aplicação (*Application Service Providers* - ASP) (Nitu, 2009). Esses fornecedores eram responsáveis por hospedar, gerenciar e executar softwares empresariais. O modelo ASP tornou-se comum em empresas que visavam reduzir custos de infraestrutura, terceirizando o controle de seus recursos computacionais. Os SaaS são uma extensão do modelo de negócio ASP, porém no modelo SaaS, a aplicação pertence ao fornecedor de

serviço, que é responsável pelo gerenciamento, desenvolvimento e evolução do software. Desse modo, os consumidores do serviço são apenas proprietários dos dados que inserem na aplicação (Nitu, 2009). Enquanto que no modelo tradicional de desenvolvimento de software, a aplicação também é de propriedade do cliente.
Em um SaaS as seguintes propriedades são desejáveis (Wei *et al.*, 2010):

- Multiclientes: No modelo de licença tradicional, os clientes têm total propriedade da aplicação. Isto ocorre, pois a relação entre os clientes e o software é um-para-um. No modelo SaaS, embora clientes usem o software com um serviço, eles não são proprietários do software. Assim, a relação entre os clientes e o software é muitos-para-um.
- Uso segundo a demanda: Os clientes não precisam comprar a licença do software. O cliente paga apenas pelo serviço segundo a sua demanda, semelhante a uma assinatura. O serviço pode ser estendido de acordo com a necessidade do cliente. Do mesmo modo, pode ser reduzido ou cancelado quando o cliente julgar necessário.
- Disponível a qualquer momento e lugar: No modelo de distribuição SaaS, o serviço de software está em ambiente de computação em nuvem, Dessa forma, o SaaS pode ser acessado sem restrição de tempo e lugar, sempre que o serviço de Internet estiver disponível.
- Implantação distribuída e mecanismos de gestão colaborativa: No modelo SaaS, o software é implantado em uma rede distribuída e heterogênea. Assim, o sistema de banco de dados – SGBD pode estar executando em um servidor enquanto a aplicação roda em outro servidor. O mecanismo de gestão colaborativa é usado para garantir a confiabilidade de serviço e a consistência do processamento de dados.

2.2.2 Arquitetura *Multi-Tenant*

Multi-tenant (MT) é a arquitetura de software que possibilita o compartilhamento de instâncias da aplicação, ou seja, os dados relativos a um *tenant* são visualizados apenas por ele e inacessível a outros *tenants*, enquanto a instância da aplicação é compartilhada entre todos (Mietzner, 2009).

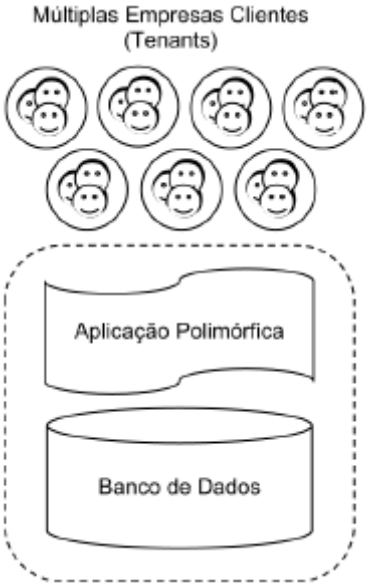


Figura 2.2: Representação de SaaS que utiliza a arquitetura MT (Adaptada Weissman e Bobrowski (2009)).

Na Figura 2.2 é apresentada uma ilustração abstrata da arquitetura MT, em que diversos *tenants* compartilha a mesma instância da aplicação. Para o correto funcionamento do software, a instância do SaaS deve atender cada *tenant* segundo suas necessidades. Com a arquitetura MT, o banco de dados deve ser desenvolvido de modo que os dados de um *tenant* não sejam acessíveis aos demais *tenants* (Chong, 2006). Dois métodos foram desenvolvidos para essa finalidade e são apresentados a seguir (Mietzner, 2009):

- **Particionamento de dados:** em um banco de dados relacional, criar a entidade `Tenant` e adicionar a chave estrangeira em todas entidades que necessitam ter os dados compartilhados entre os *tenants*. A entidade `Tenant` conterá o registro de todos os *tenants* da aplicação. Assim, a cada novo registro adicionado nas entidades compartilhadas, deverá ser informado a chave estrangeira `tenant_id`, como referência ao identificador do *tenant* a qual o dado pertence.

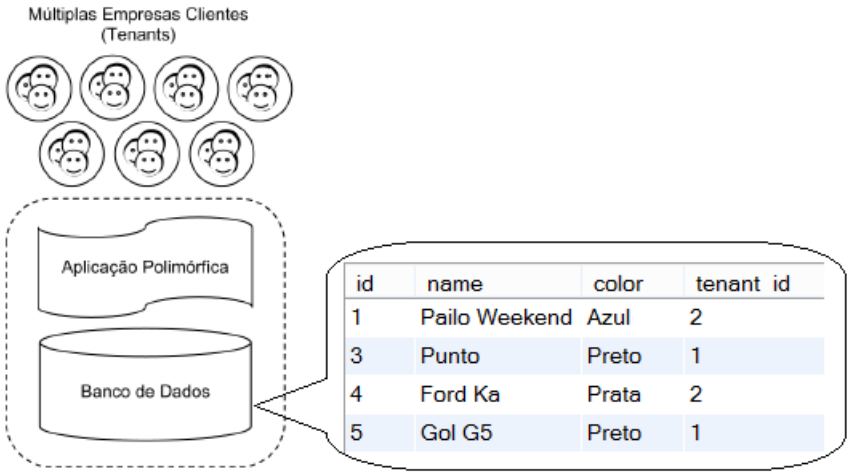


Figura 2.3: Mecanismo usado em aplicações MT para particionamento de dados entre os tenants.

Para exemplificar o método, considere um SaaS que gerencia registros de carros. Nesse método, a tabela que armazena os registros (`name` e `color`), também mantém o identificador do *tenant* (`tenant_id`) a qual o dado pertence. Na Figura 2.3 está ilustrada a visualização da tabela `Carro` em banco de dados relacional. Nesse exemplo, a tabela armazena quatros registros, dois pertencentes ao *tenant* com identificador 1 e dois registros cadastrados pelo *tenant* com identificador 2. Para que a aplicação apresente apenas os dados referentes ao *tenant* que faz uso do sistema, é necessário que todas as consultas sobre o banco de dados, contenha a restrição com o filtro por *tenant.* Para representar esse cenário, imagine um cliente, representado pelo *tenant* com identificador 1, que deseja listar todos os carros cadastrados no sistema. Para que a aplicação se comporte corretamente, isto é, apresentando apenas os registros de carros pertencentes ao *tenant* 1, a consulta é dada pela seguinte regra:

```
SELECT id, name, color
FROM car
WHERE tenant_id = 1;
```

Caso essa consulta fosse realizada sobre a tabela representada pela Figura 2.3, o resultado seria: Palio e Gol G5, pois esses registros pertencem ao *tenant* representado pelo identificador 1.

- **Múltiplos bancos de dados:** Nesse segundo método, os dados de cada *tenant* são armazenados em bancos de dados separados. Assim, os *tenants* compartilham a instância do software, porém cada *tenant* possui sua instância da base de dados. Na Figura 2.4 é ilustrado um cenário com esse método e com três *tenants*.

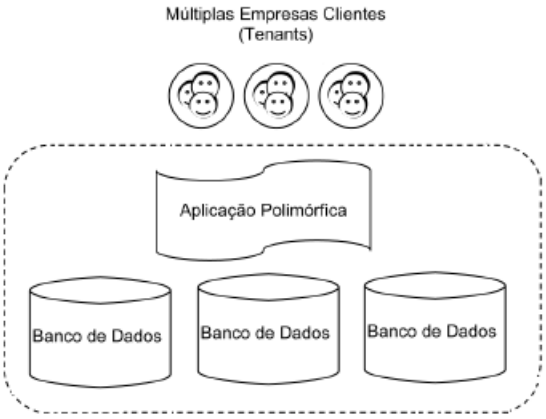


Figura 2.4: Representação de uma aplicação MT que utiliza múltiplos bancos de dados.

Do ponto de vista do fornecedor do SaaS, utilizar um banco de dados único e compartilhado é mais benéfico. Isto ocorre, pois apenas uma base de dados precisa ser configurada, mantida e tratada quanto aos aspectos de segurança e *backups* (Mietzner, 2009). Contudo, há casos que se faz necessário a opção de múltiplos bancos de dados. *Tenants* que necessitam do sigilo de informações (ex. instituições financeiras) devem evitar bancos compartilhados devido à fragilidade de acesso aos dados.

2.2.3 Classificação dos SaaS

Como visto neste capítulo, os SaaS podem ser *single-tenant* ou MT, por isso alguns autores classificam os SaaS segundo níveis de maturidade, criados de acordo com as propriedades de um SaaS. Zainuddin e Staples (2011) definiram cinco níveis de maturidade, analisando a configuração, o apoio a MT e a escalabilidade. Nas Figuras 2.5 a 2.8 são ilustrados os níveis de maturidade.

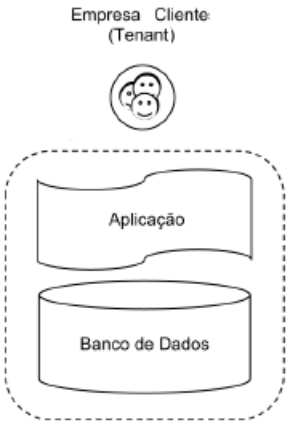


Figura 2.5: Aplicação SaaS - Maturidade nível 1 (Zainuddin e Staples, 2011).

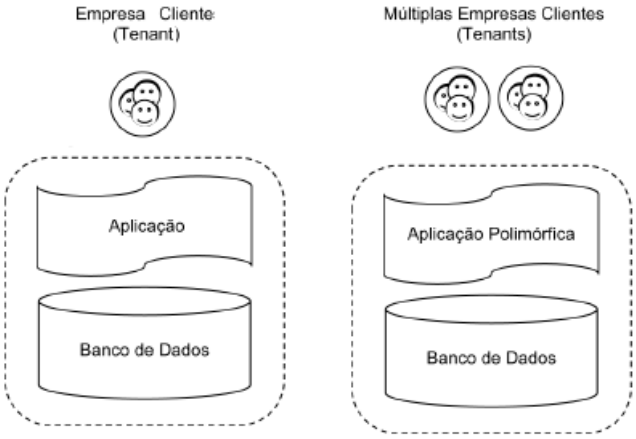


Figura 2.6: Aplicação SaaS - Maturidade nível 2 (Zainuddin e Staples, 2011).

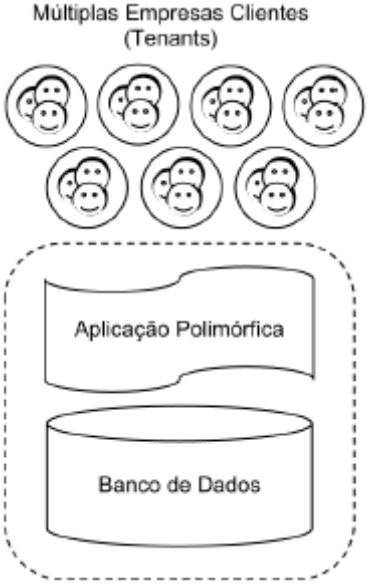


Figura 2.7: Aplicação SaaS - Maturidade nível 3 e 4 (Zainuddin e Staples, 2011).

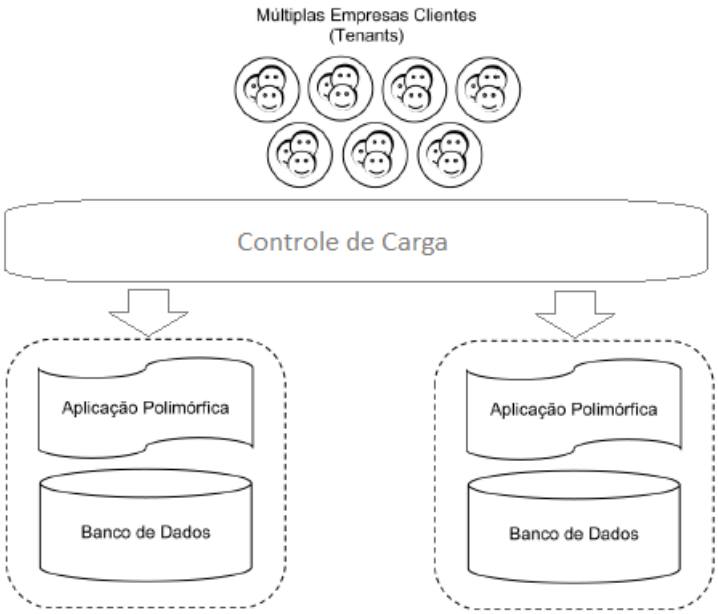


Figura 2.8: Aplicação SaaS - Maturidade nível 5 (Zainuddin e Staples, 2011).

No nível de maturidade 1 – Figura 2.5, o SaaS é personalizado para cada *tenant* e não existem opções de configurações pelo consumidor do serviço. Os SaaS do nível 1 não utilizam a arquitetura MT, por esse motivo são consideradas *single-tenant*, sendo que cada cliente utiliza sua própria instância do software.

No nível de maturidade 2 – Figura 2.6, alguns *tenants* utilizam a mesma instância do software, que apresenta algumas opções de configuração, tornando o SaaS mais genérico.

No nível 3 – Figura 2.7, o SaaS possui um grande número de opções de configuração, porém são acessíveis apenas ao fornecedor do serviço. Mesmo as configurações sendo definidas pelo fornecedor, o SaaS para estar no nível de maturidade 3, deve adotar a arquitetura MT.

A diferença entre o nível de maturidade 3 e o nível 4 – ambas Figura 2.7, é que a aplicação no nível 4, deve ser configurável pelo consumidor, além de usar a arquitetura MT.

O último nível da classificação de Zainuddin e Staples - Figura 2.8, define que o SaaS atende os requisitos do nível 4, além de possuir alto grau de escalabilidade. A escalabilidade pode ser obtida por meio de mecanismo de controle de carga, responsável por criar instâncias dinamicamente segundo a demanda.

2.3 Linha de Produtos de Software (LPS)

Uma LPS é definida como um conjunto de métodos e técnicas com o objetivo de facilitar e melhorar o processo de desenvolvimento de famílias de aplicações semelhantes (Clements e Northrop, 2002). O desenvolvimento de LPS, também conhecido por família de sistemas ou domínio de aplicação, é mais do que o reúso de software tradicional (Gomma, 2004). O reúso de software tradicional, normalmente está associado a repositórios de software, onde todo o código-fonte desenvolvido é armazenado. *Plug-ins*, bibliotecas ou *frameworks* de aplicação são considerados reúso oportunistas, pois a dificuldade para encontrar um código, às vezes, é menos oneroso do que reconstruí-lo (Bosch, 2000). O desenvolvimento *ad-hoc* não representa uma linha de produtos no sentido conceitual apresentado, pois o reúso em LPS é planejado, automatizado e sistemático (Bosch, 2000). Em LPS, o repositório que contém o núcleo de todas as aplicações da linha é chamado de ativo base e não armazena apenas código-fonte, mas também documentação, arquitetura, testes, padrões de projeto, cronograma e outros artefatos do desenvolvimento (Northrop, 2008).

No desenvolvimento tradicional de software - "*clone e own*", o produto que mais se assemelha ao software desejado, é copiado e retrabalhado para criar um novo produto. Dessa forma o enfoque está no produto individual e não na família de produtos de software. Uma LPS em comparação a abordagem "*clone* e *own*" apresenta as seguintes diferenças (SEI, 2003):

- Todos os artefatos de desenvolvimento são reusáveis e o ativo base normalmente inclui os artefatos de maior custo.
- O ativo base de uma LPS é criado com o objetivo de reúso e customização. Na abordagem *"clone e own"*, os softwares normalmente não apresentam mecanismos de customização, além dos utilizados pelo produto. Isso torna o esforço de customização na abordagem tradicional mais lento comparado a uma LPS;
- Na abordagem *"clone e own*", os produtos não possuem ligação, por isso é comum que a manutenção seja realizada em todos os softwares individualmente. Em uma LPS, as manutenções realizadas no ativo base, são aplicadas a todos os produtos que fazem parte da mesma família ou seja aqueles que utilizam o mesmo ativo base.

Bass *et al.* (2003) organizaram uma classificação de LPS, quanto aos benefícios da adoção, em três tipos – organizacionais, de Engenharia de Software e comerciais. Os benefícios organizacionais apresentam maior facilidade de entendimento do domínio do produto, maior qualidade e mais confiança do cliente. O benefício da Engenharia de Software tem o enfoque na reutilização dos artefatos, no melhor controle da qualidade do software e no estabelecimento de padrões de programação. Os benefícios comercias referem-se ao menor esforço de manutenção, testes e melhor planejamento.

Apesar dos benefícios de LPS para criação de família de produtos, a introdução de LPS em uma organização não é simples. Ela envolve muitas mudanças na forma de desenvolvimento e tempo para adaptação nos primeiros estágios (Long, 2002). Esses fatores podem trazer resistência por parte dos desenvolvedores e gerentes.

Cohen (2002) afirma que outro desafio é a necessidade de um especialista da linha de produto, responsável pelo domínio da aplicação e que compreenda as práticas de LPS. A evolução de uma LPS também é complexa, pois a dimensão do escopo pode comprometer a linha. Se o escopo da LPS for grande, a reutilização pode se tornar ineficaz, enquanto escopos pequenos, podem não justificar o uso da LPS. Por isso Buhne *et al.* (2003) discutem sobre a necessidade em se criar um plano sistemático e planejado para tornar a atividade de criação da LPS menos complexa.

2.3.1 Desenvolvimento e Abordagens para criação de LPS

O processo para desenvolvimento de uma LPS apresenta três atividades principais (Pohl *et al.*, 2005):

- **Engenharia de domínio:** atividade responsável pelo entendimento do domínio e o desenvolvimento da plataforma de artefatos reusáveis (ativo base), além da definição dos possíveis produtos da família.
- **Engenharia de aplicação:** atividade responsável pela criação dos produtos por meio do reúso do ativo base gerado na engenharia de domínio.
- **Gerenciamento/Evolução da LPS:** atividade responsável pelo ciclo de vida da LPS e pelo amadurecimento da linha de produtos.

Na Figura 2.9 nota-se que a engenharia de domínio gera os artefatos reusáveis (requisitos, modelos, código-fonte e teste) que serão usados para a criação dos produtos na engenharia da aplicação. O gerenciamento da LPS deve ser uma atividade constante do ciclo de vida da aplicação, cuidando dos riscos e do monitoramento da evolução da linha de produtos.

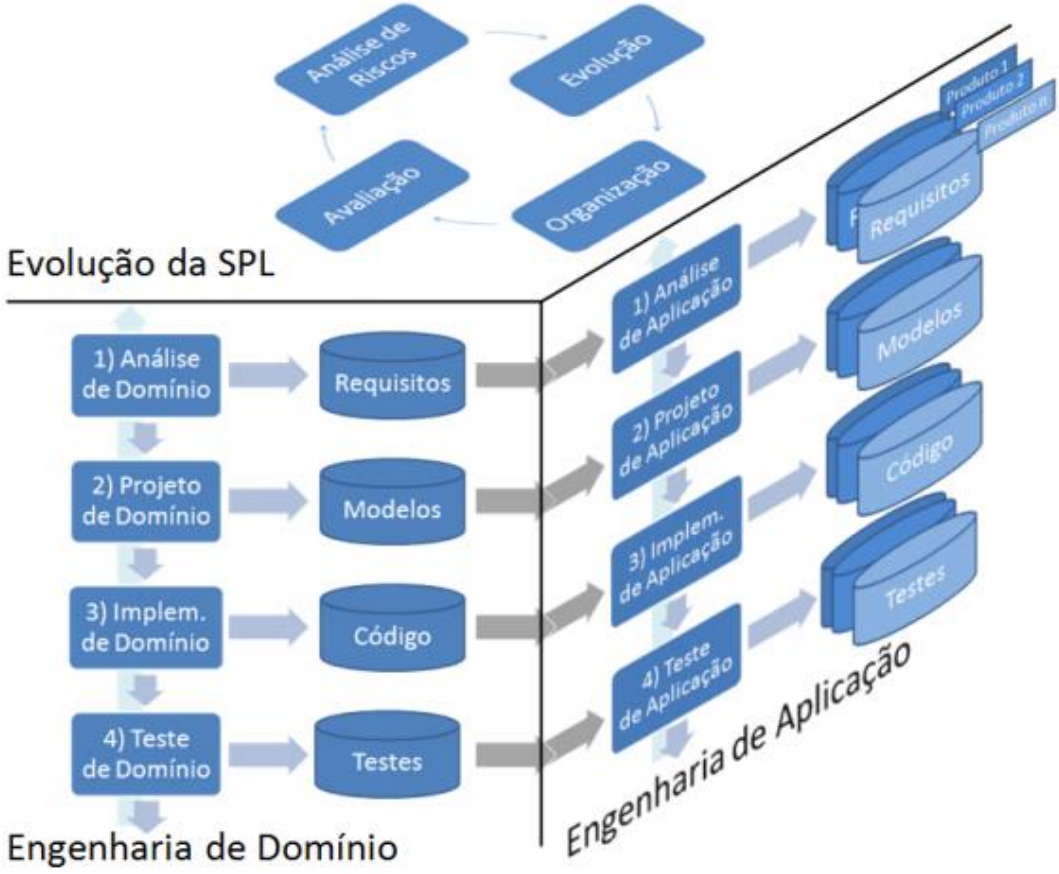


Figura 2.9: Processo de desenvolvimento de LPS (Adaptada Pohl *et at.* (2005)).

Chen *et al.* (2006) apresentam três abordagens principais para o desenvolvimento de Linhas de Produtos de Software – proativa, reativa e extrativa:

- **Proativa:** o ativo base é desenvolvido antes de existirem os produtos. A partir do planejamento inicial são considerados os produtos que poderão ser gerados pelo LPS.
- **Reativa:** o ativo base já existe, bem como uma versão da linha de produtos. No entanto, à medida que novos requisitos surgem é necessário a evolução da LPS por meio de incrementos.

- **Extrativa:** os produtos já existem, porém de forma individualizada. Para derivar uma versão inicial da LPS, os produtos devem ser analisados de modo que sejam extraídas as semelhanças e as variabilidades.

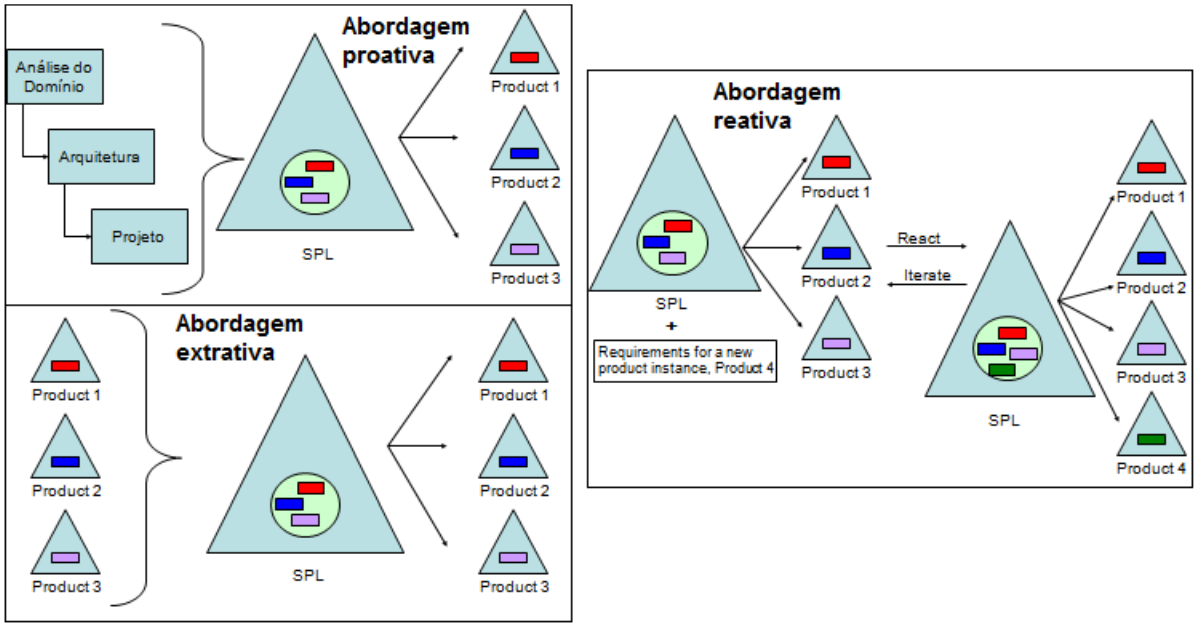


Figura 2.10: Abordagens para construção de LPS.

Na Figura 2.10 estão ilustradas as formas de desenvolvimento de LPS. A abordagem proativa pode utilizar as estratégias *big bang* ou incremental. Na estratégia *big bang* a LPS é adotada apenas para os novos produtos e a engenharia de domínio é realizada por completo antes que os produtos sejam desenvolvidos (Pohl *et al.*, 2005). Na estratégia incremental o ativo base não precisa estar finalizado para o início da engenharia de aplicação. Desta forma a evolução do ativo base e dos produtos ocorrem em paralelo.

O desenvolvimento da forma reativa pode ser dividido em três estratégias (Chen, 2006): baseada em infraestrutura, *branch-and-unit* e *bulkintegration*. A estratégia baseada em infraestrutura exige que novas funcionalidade sejam adicionadas primeiramente no ativo base e restringe que os produtos estejam desatualizados em relação à plataforma. As estratégias baseadas em *branch-and-unit*e *bulk integration* não restringem a desatualização entre ativo base e os produtos, porém na estratégia *branch-and-unit* é necessário que novas funcionalidades, presente nos produtos, sejam integradas à plataforma assim que o produto for lançado.

2.3.2 Variabilidade

Para facilitar a customização em massa, uma LPS deve fornecer meios para satisfazer as necessidades dos diferentes clientes da aplicação. Entende-se por variabilidade a capacidade de adaptar ou customizar um sistema (Pohl *et al.*, 2005).

Em uma LPS, as variabilidades são projetadas na arquitetura de referência, por meio dos pontos de variação e variantes (Jacobson *et al.*, 1997). Bachmann e Clements (2005) e Linden *et al.* (2007) definem variantes como uma opção dos artefatos gerados pela análise de domínio. As variantes são as possíveis instâncias em um ponto de variação. Enquanto que ponto de variação é definido como a diferença funcional de um elemento do ativo base. Esse ponto estabelece a seleção das possíveis variantes. Na Figura 2.11 é apresentado um exemplo de ponto de variação – "Tipo de Câmera" e as possíveis variantes que pode ser instânciada – "Câmera Preta e Branco" e "Colorido".

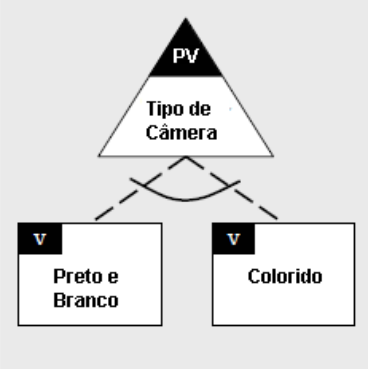


Figura 2.11: Exemplo de ponto variação e variantes (Adaptada de Pohl *et al.* (2005)).

As semelhanças e as variabilidades de uma LPS podem ser representadas em termos de *features*. As *features* podem ser definidas com uma funcionalidade visível do sistema ao usuário final (Kang *et al.*, 1900). Também pode ser definida como a unidade lógica de comportamento, constituída pelo agrupamento de requisitos funcionais e de qualidade (Bosch, 2000) ou ainda um requisito reutilizável ou característica de uma linha de produtos (Gomma, 2005).

Gomma (2005) define que a variabilidade em LPS pode ser expressa em nível de requisitos e projetos. O modelo de *features* é uma técnica bem sucedida para reúso de artefatos de software e tem como objetivo, representar os relacionamentos estruturais de um domínio de aplicação (Cohen e Northrop 1998, Kang *et al.* 1990).

As *features* são classificadas como: obrigatórias, opcionais ou alternativas (Czarnecki *et al.*, 2005) (Pohl *et al.*, 2005). A *feature* obrigatória é aquela que está presente em todos os produtos da LPS - representada por uma aresta com círculo preenchido na extremidade. A *feature* opcional é aquela que pode ou não estar nos produtos da LPS – representada por uma aresta com círculo vazio na extremidade. A feature alternativa é uma composição de *subfatures,* entre as quais deve-se escolher uma ou mais de acordo com a representação lógica associada (OR ou XOR) - sua representação é por meio de um arco. O arco pode ser preenchido (XOR), somente uma *subfeature* é necessária ou (OR) o qual permite a escolha de uma ou mais *subfeatures (*Deursen e Klint, 2001).

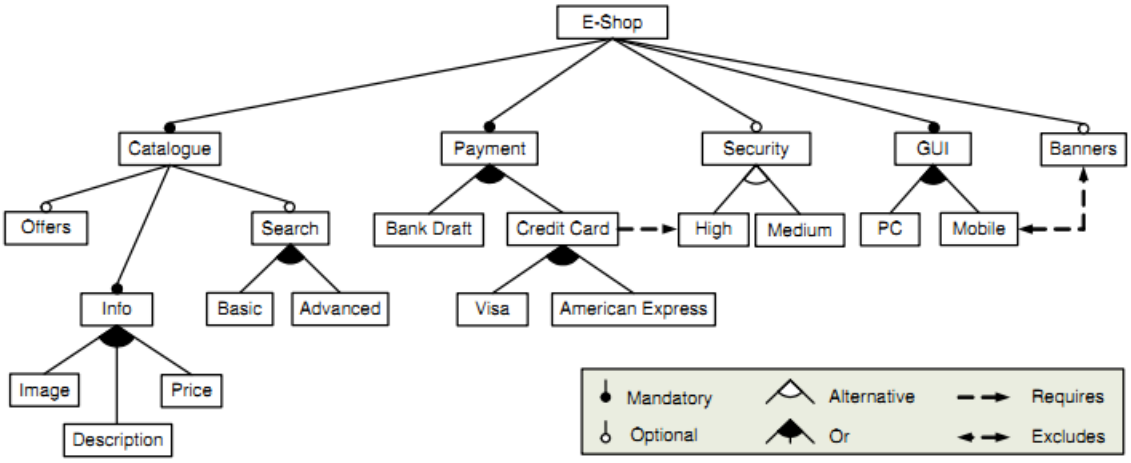


Figura 2.12: Exemplo de modelo de *features* para comércio eletrônico.

Na Figura 2.12 é apresentado um exemplo de modelo de *feature* para uma LPS de comercio eletrônico (*e-commerce*). A *feature* raiz representa toda a LPS. As *features* "*Catalogue*", "*Payment"*, "*GUI"* e "*Info"* são obrigatórias, portanto deve fazer parte de todos os produtos gerados pela LPS. A *feature* "*Search"* é formada por um grupo de *subfeatures* opcionais ("Basic" e "*Advanced"*), assim quando "*Search*" é usada, é necessário escolher uma ou as duas *subfeatures* ("*Basic"* ou *"Advanced"*). A *feature* "*Security"* representa um grupo de *subfeatures* alternativas, mutualmente exclusivas, sendo obrigatório a seleção da *feature* "*High*" ou "*Medium"*. No modelo ainda existem duas notações para representar as *features* que não possuem parentesco, chamadas de *cross-tree* (Deursen e Klint, 2001):

- ***Requires***: relação que representa a dependência de *features*. Uma *feature* "A" requer a *feature* "B", ou seja, implica que a existência de "A" no produto, obriga a existência de "B". Na Figura 2.15 essa relação é ilustrada pelas *features* "*CreditCard*" e "High Security". Quando a feature "*CreditCard*" é selecionada, necessariamente "*High Securty"* deve existir.
- ***Excludes***: relação que representa as *features* conflitantes. Quando uma *feature* "A" exclui a *feature* "B", elas não podem coexistir no produto. Na Figura 2.14 essa relação é representada pelas *features* "*GUI Mobile*" e "*Banners*". Assim, quando a interface da aplicação é feita para dispositivos móveis as propagandas não podem existir.

O modelo de *features* representa a modelagem da engenharia de requisitos, porém não apresenta o modo o qual as aplicações da LPS são geradas. Por isso deve existir um mapeamento entre modelo de *features* e os artefatos de implementação (classes, recursos, componentes, arquivos, entre outros). Esse mapeamento estabelece a relação "implica" entre os artefatos de implementação e as *features*. Por meio de um conjunto de regras, pode-se definir quais artefatos estarão presentes nos produtos criados pela LPS, verificando se as restrições presentes no modelo de *features* são respeitadas em cada produto gerado. (Cohen, 2002).

Em nível de projeto, as técnicas para modelagem da variabilidade incluem os modelos de variabilidade usando parametrização, informações ocultas e hierarquia (Gomma e Webber 2004). Na técnica de modelagem por parametrização, os valores associados aos parâmetros definem os componentes que fazem parte do produto gerado pela LPS. Em uma família de produtos, cada produto pode ter diferentes valores associados aos parâmetros. Existem quatro tipos diferentes de parâmetros (Gomma, 2005) - parâmetros de compilação, configuração (instalação), inicialização e tabelas de parâmetros. Muitos métodos para análise de LPS utilizam a parametrização. No método *Feature-Oriented Domain Analysis* (FODA) (Kang *et al.*, 1990), as *features* são classificadas em funcionais, não-funcionais ou parâmetro.

No método *Domain-Specific Software Architecture* (DSSA) (Hayes-Roth, 1995) os componentes genéricos são parametrizados para facilitar a customização de aplicações. O método *Family-OrientedAbstration, SpecificationandTranslation* (FAST) (Weiss and Lai, 1990) utiliza a parametrização como uma alternativa para obter a customização. Apesar de estar associado a diversos métodos, o gerenciamento de grande número de

parâmetros pode ser um problema, principalmente quando os parâmetros são conflitantes (Gomma, 2005).

A técnica de modelagem por ocultação da informação pode ser utilizada para obter a variabilidade em uma LPS. Diferentes versões dos componentes utilizam a mesma interface, porém com implementações distintas. Desse modo a variabilidade está oculta em cada versão do componente inserido no produto gerado pela LPS. Na variabilidade por informação oculta, as variantes são as versões do mesmo componente. Essa técnica tem como ponto negativo que as atualizações são limitadas às implementações do componente e não à interface (Gomma, 2005). Os principais métodos para análise de LPS que utilizam informação oculta são: FODA, FAST e DSSA.

Na técnica de modelagem por hierarquia, diferentes versões de uma classe, utiliza a hierarquia para herdar operações de uma superclasse, redefine-las ou estende-las. Esta técnica permite que a engenharia de aplicação selecione as variantes cujas funções podem variar. O método *Product Line Software Engineering* (PuLSE) (Bayer *et al.* 1999, DeBaud e Schmid 1999) utiliza essa técnica para modelar a variabilidade.

Gomma (2005) afirma que em uma LPS a combinação das três técnicas são necessárias para modelagem de variabilidade e o paradigma da orientação a objetos auxilia essa adoção. Porém, outras técnicas podem ser usadas a fim contribuir para a representação da variabilidade de uma LPS, como por exemplo: Padrões de Projeto (Gamma *et al.*, 1994); Arquitetura de Componentes (Roschelle *et al.*,2000); Programação Orientada a Aspectos (Ateyeh e Lockemann, 2006); e Desenvolvimento Dirigido por Modelos (Choquet e Corbière, 2006).

2.4 Considerações Finais

Neste capítulo foram apresentados os conceitos e as propriedades de SaaS e LPS, bem como algumas técnicas para a modelagem, gerenciamento e desenvolvimento das variabilidades em uma LPS. As técnicas apresentadas estão estabelecidas e fornecem mecanismos para o desenvolvimento de produtos com maior qualidade e produtividade, capazes de atender as diferentes necessidades dos clientes. Por essa razão, esse trabalho de mestrado utiliza as técnicas de LPS para manipular a variabilidade em SaaS MT, visando resolver o problema de customização dos *tenant.* No capítulo a seguir são apresentados os conceitos teóricos sobre padrões de projeto, que darão a fundamentação necessária para a proposta deste projeto.

Capítulo 3
PADRÕES DE PROJETO

3.1 Considerações Iniciais

Projetar software orientado a objeto não é simples. Apesar de projetistas experientes de software construírem bons modelos, desenvolvedores iniciantes cometem algumas falhas durante os primeiros desenvolvimentos. Esse fato ocorre pois um projeto de software deve ser capaz de resolver um problema específico e ao mesmo tempo ser genérico o suficiente para atender às manutenções corretivas, evolutivas e adaptativas. Padrões de projeto estão associados ao reúso de experiências. Um padrão utilizado para resolver um problema específico, torna o projeto orientado a objetos mais flexível, elegante e reusável (Gamma *et al.,* 1995). Padrões de projeto, também, tornam mais simples o reúso de modelos e arquitetura de software, auxiliando ao projetista a criar sistemas reusáveis, com maior manutenibilidade, documentação apropriada e fornecendo, de forma rápida e correta, uma solução (Gamma *et al.*, 1995).

Na Seção 3.2 são apresentados os conceitos de padrões de projeto e como eles devem ser descritos. Na Seção 3.3 é apresentado um mapeamento sistemático para obter os padrões, disponíveis na literatura científica, para o desenvolvimento de SaaS MT customizáveis. Na Seção 3.4 são expostas as considerações finais.

3.2 Padrões de Projeto

Em Engenharia de Software, padrões de projeto são uma solução geral e repetível para um problema que ocorre de forma recorrente em projetos de software. Os padrões são frequentemente confundidos com estruturas de dados, tais como listas, pilhas, entre outras que podem ser descritas por uma classe. Também não podem ter a complexidade de uma aplicação ou um subsistema. Um padrão de projeto deve apresentar a comunicação entre objetos e classes que são criadas para resolver um problema de projeto geral em um contexto particular (Gamma *et al.*, 1995).

Os padrões de projeto podem utilizar notações gráficas, contudo somente essas não são suficientes para descrever as decisões e as negociações entre os elementos do padrão. Um formato comum para descrição de padrões de forma consistente, visando facilitar a leitura, comparação e uso, é apresentado por meio do seguinte modelo (Zamani e Butler, 2009; Gamma *et al.*, 1995):

- **Nome e classificação:** descreve, sucintamente, a essência do padrão. A classificação pode ser associada a um conjunto de possíveis valores.
- **Propósito:** indica o problema a ser resolvido. Deve responder as seguintes questões – O que o padrão faz? Para qual problema particular esse padrão é indicado?
- **Também conhecido como:** outro nome para o padrão, caso exista.
- **Motivação/Cenário:** situação que ilustra o problema e como a estrutura de classes e objetos desse padrão resolvem o problema.
- **Aplicabilidade:** aponta as situações nas quais o padrão pode ser usado.
- **Estrutura:** representação gráfica das classes do padrão utilizando a notação da linguagem de modelagem.
- **Participantes:** objetos e classes que participam do padrão e quais a suas responsabilidades.
- **Colaborações:** indica como os participantes podem executar suas responsabilidades.
- **Consequências:** descreve como o padrão atinge seu objetivo e quais os resultados do uso do padrão.
- **Implementação:** deve responder as perguntas - Quais as técnicas e dicas devem ser conhecidas para implementar esse padrão? Existe algum problema associado a alguma linguagem de programação?

- **Exemplo de código:** fragmentos de código que ilustram como a solução é implementada.
- **Uso conhecido:** exemplo de uso em um sistema real. Deve ser descrito ao menos dois exemplos em diferentes domínios.
- **Padrões relacionados:** quais padrões de projeto são semelhantes e quais são as principais diferenças.

3.3 Padrões de Projeto para SaaS

O catálogo de padrões de projetos apresentado por Gamma *et al.* (1995) não está relacionado a um domínio específico. Para obter os padrões relativos ao domínio de SaaS foi realizado um mapeamento sistemático da literatura. O mapeamento teve a finalidade de avaliar o estado da arte de padrões de projeto para SaaS. A seguinte *string* de busca padrão foi aplicada nas bibliotecas digitais: ACM, IEEE e Science Direct e Scopus:

("saas" OR "software as a service" OR "software-as-a-service")
AND
("pattern" OR “patterns" OR "design pattern" OR "design patterns")

A aplicação da *string* de busca anteriormente citada, restringe os estudos que relacionam SaaS e padrões, independente da grafia. Embora o objetivo desse mapeamento sistemático seja identificar os estudos que abordam padrões para problema da variabilidade em aplicações SaaS *Multi-Tenant,* a *string* de busca criada não aplica essa restrição. O objetivo da flexibilidade na busca foi garantir que trabalhos que relacionam padrões para construir aplicações SaaS não fossem excluídos, uma vez que esses estudos poderiam auxiliar a pesquisa.

Para determinar os estudos primários relevantes, segundo as questões de pesquisa, foram definidos os critérios de inclusão (CI) e exclusão (CE):

- **CI1**: Estudos que relacionam padrões a construção de SaaS.
- **CI2**: Estudos que associam padrões para resolver o problema da variabilidade em SaaS MT.
- **CE1**: Estudos que não relacionam padrões a construção de SaaS.

- **CE2**: Estudos que associam padrões a SaaS, mas não apresentam descrição mínima necessária.
- **CE3**: Estudos com menos de 4 páginas, por se mostrarem irrelevantes no contexto desta pesquisa.
- **CE4**: Estudos em duplicidade.
- **CE5**: Estudos indisponíveis.

Por meio desse mapeamento sistemático foram encontrados 21 estudos, excluídos os artigos com menos de 4 páginas e em duplicidade sobraram 12 estudos. Nenhum artigo foi excluído pelas restrições CE2 e CE3. Após a leitura dos 12 estudos restantes foram extraídos 6 padrões distintos para desenvolvimento de SaaS e são apresentados a seguir:

3.3.1 *Parameter Pattern*

- **Motivação**: provedores de SaaS podem oferecer diferentes implementações de um mesmo serviço. São diferenciados com base nos valores dos parâmetros.
- **Aplicação**: esse padrão é aplicável em soluções que consideram as variabilidades dos parâmetros do sistema. Por meio da parametrização, os provedores do serviço podem planejar as variabilidades existentes no desenvolvimento do projeto. Por exemplo, pode ser aplicado para diferenciar modificações específicas por *tenant*, tais como, preferências de interface e campos dos formulários.
- **Exemplo**: sistemas com apoio a multilinguagem, podem basear-se na parametrização, para associar cada *tenant* a uma língua determinada.
- **Solução**: o armazenamento dos parâmetros específicos por *tenant* pode ser via arquivos XMLs ou registros na base de dados. Em geral, pode-se associar pares chave-valor para cada parâmetro.
- **Consequências**: provê a variabilidade em um mesmo código-fonte, contudo, apesar da flexibilidade aos *tenants*, o gerenciamento em um cenário complexo pode ser um problema em decorrência da quantidade de estruturas condicionais.

3.3.2 *Routing Pattern*

- **Motivação**: diferentes *tenants* podem compartilhar um mesmo requisito, contudo podem apresentar regras de negócio distintas. Esse padrão utiliza o roteamento de requisições segundo determinada regra de negócio ou requisito.
- **Aplicação:** para rotear requisições de diferentes origens, mudando o comportamento da aplicação. Essas mudanças podem ser feitas em tempo de execução, fornecendo maior flexibilidade, ou seja, pode ser utilizado para incluir ou excluir serviços e manipular exceções.
- **Exemplo:** em um sistema de compras via Internet, as formas de pagamento podem variar segundo a categoria do cliente. Esse mecanismo pode ser feito com base no redirecionamento dos serviços de pagamento, em consideração a classificação do cliente.
- **Solução:** existem diferentes formas de implementar o *Routing Pattern.* Algumas abordagens descrevem estruturas condicionais e outras a Programação Orientada a Aspectos (POA) para a interceptação de mensagens (Rahman *et al.*, 2010). Utilizando POA, *Web Services*(WS) são interceptados e então as regras de negócio são aplicadas a requisição.
- **Consequências:** permite separar regras de negócio do código-fonte, contudo, pode-se criar estruturas inconsistentes, tais como *loops* desnecessários. A escalabilidade também é um problema em um cenário com muitos *tenants,* devido ao aumento da complexidade.
- **Padrões relacionados**: os padrões *Proxy* e *Facade* são similares.

3.3.3 *Service Wrapping Pattern*

- **Motivação:** serviços em um SaaS podem ser exclusivos e incompatíveis. Isso pode ocorrer por causa de um problema técnico ou de uma determinada regra de negócio.
- **Aplicação:** nos casos de incompatibilidade de protocolos de comunicação, tipos de dados, formatos de mensagem e interfaces dos serviços que compõe a aplicação.
- **Exemplo:** Entende-se por incompatibilidade os serviços que estão baseados na comunicação assíncrona e outros na comunicação sincrônica. Isso acontece, porque em cada um dos casos, a requisição deve ser tratada segundo um algoritmo específico

O padrão é usado para enviar mensagens de notificações sobre pagamentos em um sistema de compras via Internet, podendo ser enviadas por meio de e-mails e SMS, contudo alguns clientes podem querer apenas uma das formas. Nesse caso, o padrão *Service Wrapping*, pode ser usado para esconder ou habilitar um dos serviços de notificação.

- **Solução:** para resolver o problema da incompatibilidade de serviços, pode-se usar serviços intermediários ou soluções que atuam como *middleware* entre interfaces incompatíveis, contendo a lógica de implementação necessária.
- **Consequências:** esconde a complexidade e simplifica a comunicação entre os serviços, contudo com o uso desse padrão há aumento da complexidade do sistema, pois aumenta a quantidade de serviços.
- **Padrões relacionados:** *Composite*, *Decorator*, *Wrapper*, *Proxy* e o *Adaptor* são similares.

3.3.4 *Variant/Template Pattern*

- **Motivação:** provedores de SaaS fornecem variantes estáticas aos *tenants* e esses podem configurá-las segundo suas necessidades.
- **Aplicação:** provedores podem permitir que os *tenants* configurem em tempo de execução o software, segundo um conjunto de variantes pré-definadas. Para isso, *templates* poderm ser selecionados pelos *tenants* para representar uma atividade dentro de um processo (Carraro e Chong, 2011). Regras de validação devem estar associadas aos *templates* para impedir incompatibilidades.
- **Exemplo:** em sistemas de gerenciamento de conteúdo, as interfaces com o usuário são associadas à *templates*, que gerenciam os padrões de cores e *layouts.*
- **Solução:** diferentes variantes podem estar disponíveis aos *tenants*, para que eles possam escolher a aparência, processos e campos de dados nos formulários. Essas informações podem ser armazenadas em arquivos de configuração e carregadas quando o *tenant* requisitar um serviço. Hierarquia, polimorfismo e abordagens de LPS podem ser usadas para gerar as variantes (Pohl, 2007).
- **Consequências:** não permite alta flexibilidade aos *tenants*, uma vez que as configurações estão baseadas em um conjunto pré-definido de variantes. A escalabilidade também é prejudicada, pois o gerenciamento de diferentes variantes

aumenta a complexidade. Em cenários com muitos *tenants*, as variantes devem ser gerenciadas por meio de uma plataforma flexível ou arquitetura na qual permiti o gerenciamento de diferentes variantes.

- **Padrões relacionados:** o *Parameter Pattern* pode ser usado na implementação desse padrão.

3.3.5 *Extension Points Pattern*

- **Motivação:** quando os *tenants* desejam implementar sua própria regra de negócio no serviço, o SaaS deve permitir uma forma de *upload* destas regras por meio de pontos de extensão.
- **Aplicação:** os *tenants* podem alterar o comportamento da aplicação em tempo de execução, por meio dos pontos de extensão. Desse modo os *tenants* podem modificar seus requisitos, processos ou até mesmo substituí-los por completo.
- **Exemplo:** o SAP *Process Integration* (PI) permiti que *tenants* realizem *upload* de códigos Java usando WS para extensão de suas regras de negócio (Jiang *et al.*, 2005) (Erradi *et al.*, 2006).
- **Solução:** *Web Services* atuam como pontos de extensão em uma arquitetura. Transformações utilizando XMLs e XSLTs são usadas para extensão de processos de negócio. Por meio desses métodos, os *tenants* podem adicionar modificações, utilizando ligações em *Web Service Definition Languagem* (WSDL).
- **Consequências:** esse padrão permiti a flexibilidade dos *tenants.* Contudo, oferece riscos devido a *loops* de execução inseridos inconsistentemente ou ameaças à segurança dos dados por meio de uma fonte maliciosa.
- **Padrões Relacionados**: similar ao padrão *Strategy*.

3.3.6 *Copy and Adapt Pattern*

- **Motivação:** quando a arquitetura MT torna-se complexa para o cenário ou quando os custos para sua manutenção são maiores que a separação em múltiplas instâncias.
- **Aplicação:** o padrão *Copy and Adapt* pode ser usado quando as técnicas de variabilidade não são suficientes ou quando a modificação da instância será prejudicial para outros *tenants*. Nesse padrão, os serviços podem ser modificados

livremente, fornecendo alta flexibilidade. Ideal para *tenants* que apresentam problemas com privacidade de dados.

- **Exemplo:** um SaaS para o gerenciamento de armazéns pode atender às necessidades gerais dos *tenants*, contudo este padrão deve ser usado quando um *tenant* necessita que seus dados não sejam compartilhados ou algum requisito interfira nos requisitos compartilhados.
- **Solução:** o SaaS é oferecido em instâncias separadas e com base de dados distintas. Desta forma, os serviços entre as instâncias se tornam independentes uns dos outros.
- **Consequências:** muito flexível para customização dos *tenants*, contudo este padrão não é escalável, pois os custos de execução são elevados. A replicação do código também aumenta os esforços de manutenção e diminui a produtividade.

3.4 Considerações Finais

Padrões de projeto são um importante artefato para apoiar a construção de software com elementos reusáveis, provendo maior extensibilidade e flexibilidade. A extensibilidade e a flexibilidade são importantes atributos em SaaS MT, uma vez que são alvos recorrentes de manutenções adaptativas e evolutivas oriundas dos diversos *tenants* que utilizam a aplicação. No domínio de SaaS é possível descrever seis possíveis padrões, como mostrados na Seção 3.3. Contudo, os padrões apresentados não fornecem detalhamento necessário para aplicação no desenvolvimento de SaaS. Faltam avaliações criteriosas da aplicabilidade desses padrões, bem como uma descrição mais formal para que os desenvolvedores de software sejam capazes de utilizá-los como *templates* para sua solução SaaS. No capítulo a seguir é apresentado um catálogo de padrões, propostos por este projeto de mestrado, para o desenvolvimento de SaaS MT customizáveis. Os padrões propostos visam resolver os problemas dos padrões existentes na literatura, com melhor manutenibilidade, escalabilidade e flexibilidade, com a descrição formal necessária para que sejam replicáveis.

Capítulo 4

CATÁLOGO DE PADRÕES PARA SAAS MULTI-TENANT CUSTOMIZÁVEIS

4.1 Considerações Iniciais

Padrões são importantes para manutenibilidade do software, pois proveem recursos para construção de software como elementos reusáveis, genéricos o suficiente para atender às manutenções corretivas, evolutivas e adaptativas. Contudo, os padrões devem ser formalmente descritos, de forma que os desenvolvedores possam replicá-los em seus trabalhos.

Esse capítulo representa o núcleo deste projeto de mestrado. Nele será apresentado o catálogo de padrões para desenvolvimento de SaaS Multi-Tenant (MT), capazes de manipular a customização de dados, processos, permissões e interfaces com usuários. Os padrões apresentados são descritos segundo o modelo de descrição de padrões (Zamani e Butler, 2009; Gamma *et al.*, 1995). Nas Seções 4.2 a 4.5 são descritos os padrões do catálogo. Na Seção 4.6 é realizado a comparação dos padrões obtidos pelo mapeamento da literatura (Capítulo 3), e os padrões propostos. Na Seção 4.7 são expostas as considerações finais.

4.2 Padrão Entidades MT

- **Nome:** Entidades *Multi-Tenant*
- **Classificação**: Em consideração a classificação de Gamma et al. (1995) este padrão é Criacional, pois está relacionado a criação dos objetos.
- **Propósito**: permite que as classes sejam identificadas como MT para obter o particionamento de dados. Indicado para auxiliar a implementação da arquitetura MT, com o compartilhamento da base de dados e resolução do *tenant* por meio do usuário autenticado.
- **Motivação**: em SaaS MT com compartilhamento da base de dados, os objetos persistidos não necessariamente pertencem ao mesmo *tenant.* Em um SaaS para gestão de comércio, os registros das vendas são gerados segundo as operações dos *tenants*, contudo a lista de produtos pode ser compartilhada entre eles. Nesse exemplo, algumas entidades do software são MT e outras não. Esse padrão foi criado para manipular as entidades MT e controlar em tempo de execução a propriedade dos dados.
- **Aplicabilidade**: o padrão Entidades MT deve ser usado quando deseja-se criar SaaS MT com compartilhamento da base de dados.
- **Estrutura**:

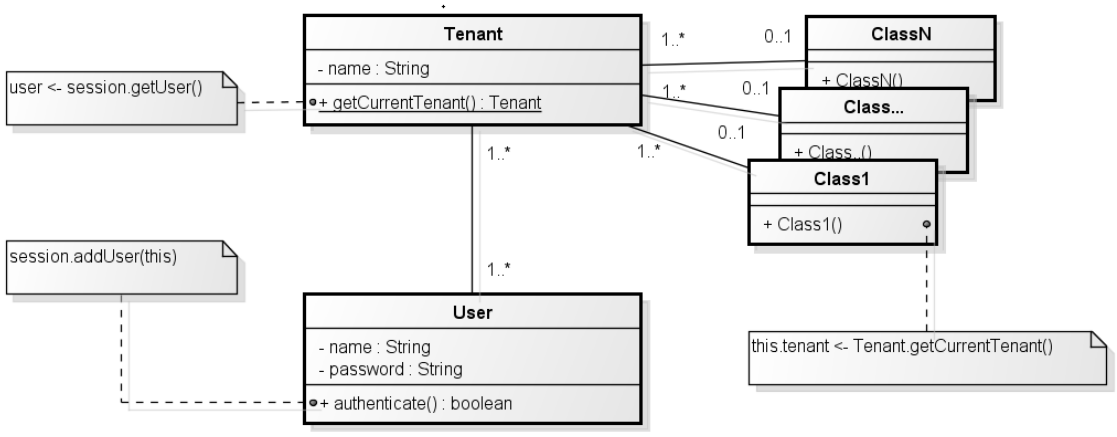


Figura 4.1: Estrutura do padrão de Entidades MT.

- **Participantes**: como apresentado na Figura 4.1, a classe `Tenant` define-los objetos que representam os consumidores do serviço e fornece o método estático `getCurrentTenant`, responsável pela resolução do *tenant* que faz uso do sistema.

Esse método utiliza informações do usuário inserido na sessão HTTP, após o processo de autenticação que acontece no método `authenticate` da classe `User`. As entidades do SaaS que são compartilhadas entre os *tenants* deve manter o relacionamento com a classe `Tenant` para indicar a propriedade do registro. Na Figura 4.1, as entidades do domínio do SaaS são identificadas pelo conjunto de classes: `Class1`, `Class...`, e `ClassN`.

- **Colaborações**: os usuários que utilizam o sistema devem ser autenticados para confirmar suas credenciais de acesso. O método `authenticate` realiza essa validação e adiciona a instância do usuário na sessão HTTP. Nos construtores das classes MT o atributo que indica a relação com a classe `Tenant` deve ser definido usando o valor do obtido pelo método `getCurrentTenant`. Nesse método é recuperado o usuário armazenado na sessão, por meio do relacionamento das classes `User` e `Tenant`, é possível recuperar o *tenant* da operação.
- **Consequências**: os benefícios deste padrão são: encapsulamento das estruturas necessárias para a arquitetura MT e redução do acoplamento entre as classes. Esse padrão ainda provê uma forma automatizada de definir a propriedade dos registros.

- **Implementação**:

```
1   class Tenant {
2       private String name
3
4       public static Tenant getCurrentTenant(){
5           return session?.getUser()?.tenant
6       }
7   }
8
9   class User {
10      private String name
11      private String password
12      private Tenant tenant
13
14      public boolean authenticate(){
15          if (!User.findByNameAndPassword(name, password)){
16              return false
17          }
18          session.addUser(this)
19          return true
20      }
21  }
22
23  class ClassN {
24      private Tenant tenant
25
26      public classN (){
27          this.tenant = Tenant.getCurrentTenant()
28      }
29  }
```

Figura 4.2: Exemplo de implementação do padrão Entidades MT em Groovy.

Na Figura 4.2, o método `authenticate`, linha 14, verifica por meio do método estático `findByNameAndPassword` da classe `User` se o usuário existe. Caso o usuário exista, a condição descrita na linha 15 retorna verdadeiro e o usuário é adicionado na sessão HTTP. A `ClassN` que representa uma entidade do domínio, deve sobrescrever o construtor padrão. No construtor criado, o atributo `tenant` deve ser definido como o valor obtido pelo método estático `getCurrentTenant`. Na linha 26 é apresentado como o construtor deve ser definido. Para evitar que o valor do atributo `tenant` seja atualizado, este deve ser privado e os métodos assessores não devem ser implementados.

- **Uso conhecido**: no Capítulo 6 é apresentado um estudo de aplicabilidade onde este padrão é usado.
- **Padrões relacionados**: a sessão HTTP pode ser implementada com o padrão *Session Facade*.

4.3 Padrão Campo Extra MT

- **Nome**: Campo Extra *Multi-Tenant.*
- **Classificação**: Em consideração a classificação de Gamma et al. (1995) este padrão é Estrutural, pois está relacionado a composição das classes e objetos.
- **Propósito**: permite que uma classe tenha atributos definidos pelos *tenants*. Indicados nos casos em que dois ou mais *tenants* apresentam requisitos divergentes e/ou conflitantes sobre as informações de formulários.
- **Motivação**: em SaaS MT, formulários comuns podem apresentar variantes de informações. Os *tenants* podem exigir a inclusão ou exclusão de campos durante todo ciclo de vida do software. Em um SaaS para gestão de comércio é possível realizar operações de venda de produtos. Neste processo deve ser informado o nome do cliente e produtos da venda. Contudo, *tenants* podem exigir informações adicionais, tais como o nome do vendedor. Parametrização poderia ser utilizado como solução, porém o número de parâmetros cresceria de modo linear ao número de campos extras, dificultado a manutenção do SaaS. A melhor solução, é um mecanismo capaz de criar, atualizar e apagar campos extras por *tenant.*
- **Aplicabilidade**: o padrão Campo Extra MT deve ser usado quando deseja-se criar atributos dinamicamente segundo as necessidades dos *tenants.*

- **Estrutura**:

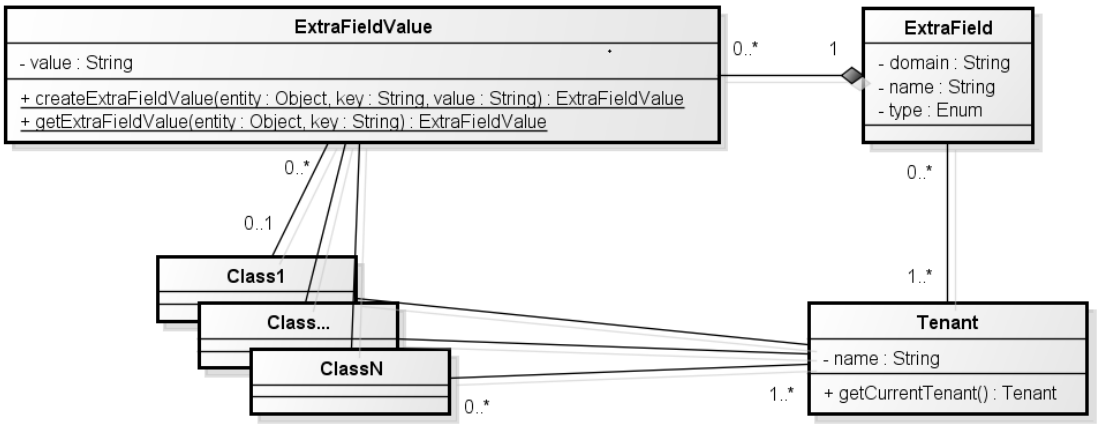


Figura 4.3: Estrutura do padrão de Campo Extra MT.

- **Participantes**: na Figura 4.3 é apresentado a estrutura do padrão Campo Extra MT. A classe `ExtraField` representa o campo extra da classe e possuem os atributos: `domain`, que identifica a entidade a qual o campo extra está associado; `name` que identifica o campo; e `type` que determina o tipo do atributo (ex: *String*, *int*, *float*, *Date*). A composição `ExtraFieldValue` representa os valores associados aos campos extra. As entidades do domínio (`Class1, Class..., ClassN`) devem manter uma relação um-para-muitos com `ExtraFieldValue`. Os métodos estáticos da classe `ExtraFieldValue` proveem uma interface para manipulação dos campos extra. A classe `Tenant` é usada para implementar o padrão Entidades MT.
- **Colaborações:** para as entidades que deseja-se ter atributos definidos em tempo de execução, deve-se associar a classe `ExtraFieldValue`. Nesta relação são mantidos os valores dos atributos criados dinamicamente. O método `createExtraFieldValue` é usado para atribuir o valor ao campo extra do objeto por meio de tuplas chave-valor. O método estático `getExtraFieldValue` é usado para recuperar o valor do atributo.
- **Consequências:** os benefícios de usar este padrão são: flexibilidade na customização de formulários, redução de parâmetros e estruturas *ad-hoc*. Associado ao padrão Entidades MT, as estruturas para tratamento da propriedade dos registros de campos extras são encapsuladas.

- **Implementação:**

```
1   class ExtraField {
2       private String name
3       private Enum type
4       private String domain
5       private Tenant tenant
6
7       public ExtraField (){
8           this.tenant = Tenant.getCurrentTenant()
9           //...
10      }
11      //...Getters e Setters
12  }
13
14  class ExtraFieldValue {
15      private String value
16      private ExtraField extraField
17      private Tenant tenant
18
19      public ExtraFieldValue (){
20          this.tenant = Tenant.getCurrentTenant()
21          //...
22      }
23
24      //...Getters e Setters
25
26      static ExtraFieldValue createExtraFielValue(Object entity, String key, String value){
27          private String domain = entity.class as String
28          private String name = key
29
30          //Recupera ou criar o campo extra
31          ExtraField ef = ExtraField.findOrCreateByNameAndDomainAndTenant(
32              name, domain, Tenant.getCurrentTenant())
33
34          //Cria o valor para o campo extra
35          ExtraFieldValue efv = new ExtraFieldValue(
36              extraField: ef, value: value)
37
38          return efv
39      }
40
41      static String getExtraFieldValue(Object entity, String key) {
42
43          for (ExtraFieldValue efv : entity.extraFieldValues) {
44              if (efv.extraField?.key == key) {
45                  return efv.value
46              }
47          }
48          return ""
49      }
50  }
51
52  class ClassN {
53      //...
54      private List<ExtraFieldValue> extraFieldValues
55      private Tenant tenant
56
57      public classN (){
58          this.tenant = Tenant.getCurrentTenant()
59          //...
60      }
61  }
```

Figura 4.4: Exemplo de implementação do padrão Campo Extra MT em Groovy.

Na Figura 4.4 é apresentado o trecho de código para manipulação de campos extra por meio desse padrão. Os atributos da classe `ExtraField`, linha 2 à linha 5,

definem o campo extra e indicam a qual entidade está associado. A classe `ExtraFieldValue` define por meio dos atributos `value`, linha 15, e `extraField`, linha 16, um valor para o campo extra criado. No método `createExtraFieldValue`, linha 26, devem ser fornecidos como parâmetros: o objeto que terá o campo extra, o nome do campo (`key`) e o valor (`value`). O primeiro passo do algoritmo deve avaliar se existe um objeto ExtraField, representado pelo par (`domain`, `key`). Essa avaliação é usada na linha 31 para criar ou recuperar o campo extra, evitando duplicações de registros. Após esse passo, é criado o objeto `ExtraFieldValue`. No método estático `getExtraFieldValue`, linha 41, o valor associado ao campo extra é recuperado por meio do par (`entity`, `key`). Assim como no método `createExtraFieldValue`, `entity` representa o objeto que recebe o campo extra. Para encontrar o valor do campo extra por meio dos parâmetros fornecidos, deve-se iterar a lista de `extraFieldValues` do objeto `entity`, procurando pela chave (`key`) que indica o nome do campo extra. Na Figura 4.5 é apresentado um trecho de código que faz uso desses métodos para manipular campos extras em tempo de execução.

```
1   //Lógica do SaaS
2   class Client {
3
4       //...
5       ClassN classN = new ClassN()
6
7       //Defini valor para o atributo extra
8       classN.extraFieldValues <<
9           ExtraFieldValue.createExtraFielValue(classN, "attribute1", "value1")
10
11      //Recupera o valor do atributo
12      String value = ExtraFieldValue.getExtraFieldValue(classN, "attribute1")
13  }
```

Figura 4.5: Exemplo de implementação do padrão Campo Extra MT em Groovy.

Neste exemplo, um objeto da `ClassN`, que possui uma lista de `ExtraFieldValues`, cria um novo campo extra, identificado por “atribute1” com valor “value1”. Na linha 8 o método `createExtraFieldValue` é invocado e o retorno é adicionado na lista de `extraFieldValues`, por meio do operador “<<”. Na linha 11 é apresentado um exemplo de código para recuperação do valor do campo extra “atributo1” por meio do método `getExtraFieldValue`.

- **Uso conhecido:** no Capítulo 6 é apresentado um estudo de aplicabilidade onde este padrão é usado.
- **Padrões relacionados:** os padrões Entidades MT, *Iterator* e *Entity Attribute Value Model* podem ser usados em sua implementação.

4.4 Padrão GUI MT

- **Nome**: *Graphical User Interface Multi-Tenant.*
- **Classificação**: estrutural
- **Propósito**: permite que os *tenants* personalizem elementos da interface. Indicado nos casos em que dois ou mais *tenants* apresentam requisitos divergentes e/ou conflitantes de UI.
- **Motivação**: em SaaS MT, os *tenants* podem exigir que elementos da UI sejam alterados para atender sua identidade visual. Cores, formas e disposição dos elementos são possíveis pontos de variação entre os *tenants.* Em um SaaS para gestão de comércio, a GUI para venda é composta por campos de entrada e botões de ações. Um *tenant* requisita que na parte superior, um logo de seu mercado esteja visível e a barra de navegação seja azul, para respeitar sua identidade visual. Esses requisitos são conflitantes com os demais *tenants*, uma vez que pertencem a outros pontos de venda e possuem outras identidades visuais. Parametrização poderia ser utilizado como solução, porém sempre que um novo *tenant* fosse inserido no SaaS, manutenções adaptativas deverão ser aplicadas. A melhor solução, é um mecanismo capaz de manipular os requisitos de UI dos *tenants.*
- **Aplicabilidade**: o padrão GUI MT deve ser usado quando deseja-se configurar as UI dos *tenants* e que elas sejam exibidas segundo o *tenant* que faz uso do SaaS.

- **Estrutura**:

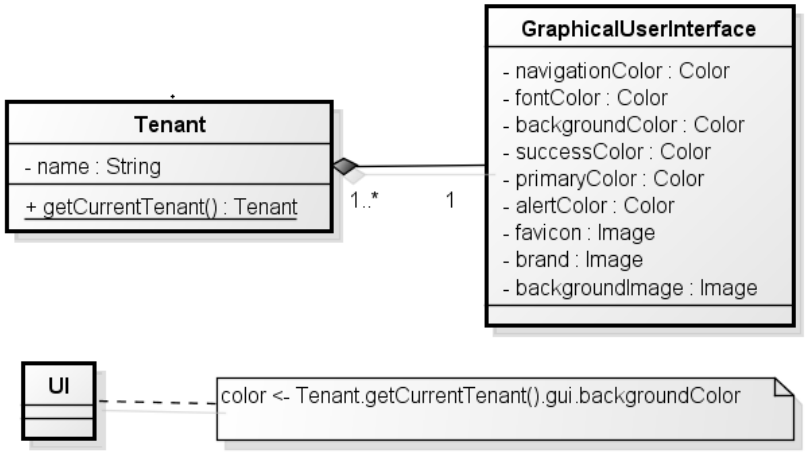


Figura 4.6: Estrutura do padrão GUI MT.

- **Participantes**: a classe `GraphicalUserInterface` é uma composição da classe `Tenant`, usada no padrão Entidades MT. `GraphicalUserInterface` contém os atributos que definem os elementos da interface. Esses atributos podem variar segundo as necessidades de customização de UI. A classe UI representa as estruturas de código, onde é necessário recuperar os valores dos elementos visuais, tais como as classes da camada de visão.
- **Colaborações**: no estágio de configuração dos *tenants*, os atributos que descrevem a variabilidade de UI devem ser definidos. Para cada instância da classe `Tenant`, uma instância da classe `GraphicalUserInterface` deve ser criada. Os elementos de UI devem se adaptar segundo os valores obtidos por esta relação em tempo de execução. A descoberta do *tenant* que faz uso do sistema é realizado por meio do método `getCurrentTenant` como apresentado no padrão Entidades MT.
- **Consequências:** os benefícios de usar este padrão são: flexibilidade na customização de UI, redução de parâmetros e estruturas *ad-hoc*. Associado ao padrão Entidades MT, as mudanças das UI são realizadas em tempo de execução segundo o *tenant* que utiliza o SaaS.

- **Implementação:**

```groovy
class Tenant {
    String name
    GraphicalUserInterface gui

    static Tenant getCurrentTenant(){
        return session?.getUser()?.tenant
    }
}

class GraphicalUserInterface {
    String navigationColor
    String fontColor
    String backgroundColor
    String primaryColor
    String successColor
    String infoColor
    String alertColor
    byte[] favicon
    byte[] brand
    byte[] backgroundImage
}
```

Figura 4.7: Exemplo de implementação do padrão GUI MT em Groovy.

```
<html>
    <head>
        <!-- header -->
        <style>
            .navigation {
                color: ${Tenant.getCurrentTenant()?.gui?.fontColor};
                background-color: ${Tenant.getCurrentTenant()?.gui?.navigationColor};
            }
            .body {
                color: ${Tenant.getCurrentTenant()?.gui?.fontColor};
                background-color: ${Tenant.getCurrentTenant()?.gui?.backgroundColor};
            }
            .btn-primary .label-primary .input-primary{
                background-color: ${Tenant.getCurrentTenant()?.gui?.primaryColor};
            }
            .btn-success .label-success .input-success {
                background-color: ${Tenant.getCurrentTenant()?.gui?.successColor};
            }
            .btn-info .label-info .input-info{
                background-color: ${Tenant.getCurrentTenant()?.gui?.infoColor};
            }
            .btn-alert .label-alert .input-alert{
                background-color: ${Tenant.getCurrentTenant()?.gui?.alertColor};
            }
        </style>
    </head>

    <body class="body">
        <nav class="navigation" >
            <img src="${createLink(controller:'graphicalUserInterface', action:'getFavicon')}" />
        </nav>
        <div>
            <!-- content -->
            <button type="submit" class="btn-primary"/>
            <input type="text" value="test" class="input-success"/>
            <label type="submit"class="alert-info">test</label>
        </div>
    </body>
</html>
```

Figura 4.8: Exemplo de implementação UI utilizando o padrão GUI MT em templates Groovy Server Page.

Na Figura 4.7 é apresentado a implementação das classes `Tenant` e `GraphicalUserInterface` na linguagem de programação Groovy. Na Figura 4.8 é apresentado como um elemento da camada de visão pode adaptar-se segundo os valores definidos no objeto `gui`, da classe `GraphicalUserInterface`. Nesse exemplo, os valores obtidos do objeto, linha 7, são aplicados na criação de classes *Cascade Style Sheet* (CSS). As classes CSS, criadas internamente no arquivo, podem ser aplicadas nos elementos de interface, tais como botões (`buttons`), entradas (`inputs`) e marcadores (`labels`) como apresentado entre as linhas 28 e 37 da Figura 4.8.

- **Uso conhecido:** no Capítulo 6 é apresentado um estudo de aplicabilidade onde este padrão é usado.
- **Padrões relacionados:** o padrão Entidades MT.

4.5 Padrão Catálogo de *Features* MT

- **Nome:** Catálogo de *Features Multi-Tenant***.**
- **Classificação**: Em consideração a classificação de Gamma et al. (1995) este padrão é comportamental, pois está relacionado a iteração entre objetos ou classes
- **Propósito**: permite que o provedor do serviço defina as *features* do SaaS, as quais os *tenants* terão acesso. Indicado nos casos em que dois ou mais *tenants* apresentam requisitos funcionais distintos e/ou conflitantes.
- **Motivação***:* definir o SaaS em termos de *features*, facilita a gestão da configuração e provê maior flexibilidade ao software para atender os requisitos gerais e específicos dos *tenants* (Mietzner *et al.*, 2009). Contudo, é importante criar mecanismos padronizados para configuração e execução dos *tenants*, segundo suas *features*. Em um SaaS MT para pontos de venda, as formas de pagamento podem ser: dinheiro, boleto ou cartão. Entretanto, alguns *tenants* podem utilizar cheque como forma de pagamento, enquanto outros não. Com o objetivo de tratar os requisitos variantes dos *tenants* esse padrão foi criado.

- **Aplicabilidade**: o padrão Catálogo de *Features* MT, pode ser usado para gerenciar o comportamento do SaaS, baseado no catálogo de *features*, a qual o *tenant* está associado.

- **Estrutura:**

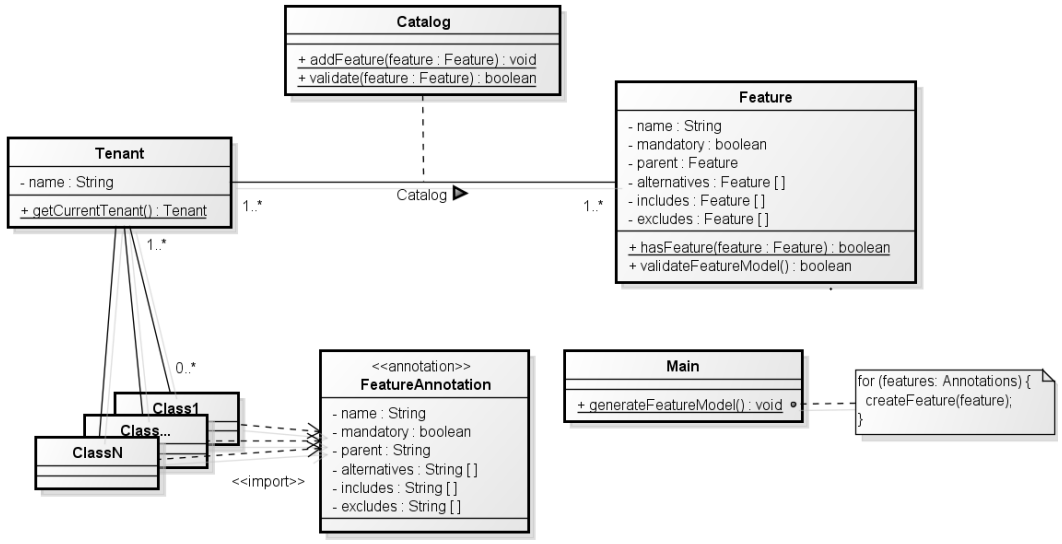


Figura 4.9: Estrutura do padrão Catálogo de Features MT.

- **Participantes:** as classes: `Class1`, `Class...` e `ClassN` representam as entidades do modelo do SaaS MT. A classe `Tenant` define os objetos que representam os consumidores do serviço e fornece o método estático `getCurrentTenant`, apresentado no padrão Entidades MT. `Catalog` é uma classe de associação, que relaciona os *tenants* e suas *features*. `Feature` é a classe que representa as características do SaaS. Os atributos da classe `Feature` descrevem os relacionamentos e restrições necessários para construção de um modelo de *feature*. A classe `Main` é responsável por gerar as *features* do SaaS na inicialização do sistema, por meio do método estático `generateFeatureModel`, que cria o modelo de *features* do SaaS. A classe `Feature` contém o método estático `hasFeature`, que analisa se a *feature* acessada está contida no catálogo do *tenant* que faz uso do SaaS. Esse método é o núcleo do padrão e será melhor detalhado no tópico de implementação.
- **Colaborações:** para que sejam consideradas *features*, as classes e métodos do modelo devem ser marcadas utilizando a anotação `FeatureAnnotation`. Os atributos da

anotação definem as restrições e relacionamentos entre as *features* do SaaS, que devem respeitar as regras do modelo de *features*. Na inicialização do SaaS, os artefatos anotados são capturados por meio de metadados, que analisam anotações sobre as classes e seus métodos. Cada *feature* obtida é validada por meio do método `validateFeatureModel`, que impede que as regras de formação do modelo sejam quebradas.

- **Consequências:** este padrão fornece uma forma fácil de manipular os requisitos específicos por *tenant*, reduzindo estruturas de código repetidas e provendo uma forma de organização e reúso de código. Isso é possível por meio da remoção de artefatos de código, específico por *tenant*, possibilitando que todas as variabilidades possam ser utilizadas por um ou mais *tenants* sem alterações do código-fonte.
- **Implementação:** na Figura 4.10 é apresentado um exemplo de criação do modelo de *features*, por meio do método `generateFeatureModel`, que busca as classes e métodos marcados pela anotação `FeatureAnnotation`.

```
class Main {

    //Representa o modelo de feature do SaaS
    List<Feature> featureModel = new ArrayList<Feature>()

    static public void generateFeatureModel() {
        for (class in DomainClasses) {
            if (class.featureAnnotation != null) {
                Feature feature = new Feature()
                feature.name = class.featureAnnotation.name
                feature.mandatory = class.featureAnnotation.mandatory
                feature.parent = Feature.findByName(class.featureAnnotation.parent)
                feature.alternatives << Feature.findAllByNameInList(class.featureAnnotation.alternatives)
                feature.includes << Feature.findAllByNameInList(class.featureAnnotation.includes)
                feature.excludes << Feature.findAllByNameInList(class.featureAnnotation.excludes)

                if (feature.validateFeatureModel() == false){
                    throw new Exception("A feature ${feature.name} viola as regras do modelo de feature")
                }
                featureModel << feature
            }

            for (method in class.methods) {
                if (method.featureAnnotation != null) {
                    Feature feature = new Feature()
                    feature.name = method.featureAnnotation.name
                    feature.mandatory = method.featureAnnotation.mandatory
                    feature.parent = Feature.findByName(method.featureAnnotation.parent)
                    feature.alternatives << Feature.findAllByNameInList(method.featureAnnotation.alternatives)
                    feature.includes << Feature.findAllByNameInList(method.featureAnnotation.includes)
                    feature.excludes << Feature.findAllByNameInList(method.featureAnnotation.excludes)

                    if (feature.validateFeatureModel() == false){
                        throw new Exception("A feature ${feature.name} viola as regras do modelo de feature")
                    }
                    featureModel << feature
                }
            }
        }
    }
}
```

Figura 4.10: Trecho de código para criação das features.

No trecho de código apresentado na Figura 4.10, percorre-se todas as classes do domínio da aplicação. Para cada classe encontrada é verificado se possui a anotação `FeatureAnnotation` e o mesmo se repete para seus métodos. Em ambos os casos são criados objetos do tipo `Feature`. Cada objeto criado, é validado se as regras de formação do modelo de *features* foram respeitadas, uma exceção será lançada, linhas 18 e 34, caso isso ocorra. Em caso de erro, o desenvolvedor de software deve corrigir as anotações para que as regras do modelo de *features* sejam respeitadas. Na Figura 4.11 é apresentado um trecho de código para o método `validadeFeatureModel`.

```
1   public boolean validateFeatureModel(){
2
3       if (this.includes.contains(this.excludes) ||
4           this.excludes.contains(this.includes) ||
5           this.includes*.parent == this.parent ||
6           this.excludes*.parent == this.parent ||
7           this.excludes*.mandatory == true ||
8           this.alternatives*.mandatory == true ||
9           this.alternatives*.parent != this.parent) {
10          return false
11      }
12      return true
13  }
```

Figura 4.11: Trecho de código para validação do modelo das features.

Esse método, da classe `Feature`, verifica as possíveis inconsistências causadas pela inserção da *feature* no modelo. Nas linhas 3, 4, 5 e 6 são verificados se as *features cross-tree* são classificadas como *includes* e *excludes* simultaneamente ou ainda se possuem parentesco com a *feature* a ser inserida. Nas linhas 7 e 8 são verificadas as condições em que a *feature* inserida nunca poderia ser usada, que ocorre quando as *features* classificadas como *excludes* ou alternativas são obrigatórias. A validação da linha 9, apenas certifica que *features* consideradas alternativas são *subfeatures* de uma mesma *feature.*

```
class TenantConf {

    //...
    Tenant tenant = new Tenant()
    Catalog catalog = new Catalog()
    tenant.catalog = catalog

    if (Catalog.validate(feature)) {
        tenant.catalog.addFeature(feature)
    }
}
```

Figura 4.12: Trecho de código para criação do catálogo de *features*.

Os objetos que representam os *tenants* da aplicação são associados às *features* por meio da classe `Catalog`. O catálogo de *features* deve ser construído de acordo com as necessidades do *tenant*. A criação do catálogo normalmente está associada ao estágio de configuração do software, contudo esse padrão não limita a atualização após a execução do software. No trecho de código apresentado na Figura 4.12, observa-se que antes de adicionar uma *feature* no catálogo, é necessário validá-la por meio do método `validate` da classe `Catalog`. Esse método verifica se a adição da *feature* condiz com o modelo de *features* do SaaS. O gerenciamento da execução do SaaS MT, é realizado pelo método estático `hasFeature` da classe `Feature`. Para isso, todos artefatos de implementação que estão associados a uma *feature* devem ser circundados com a condição `hasFeature(feature: Feature)`. Esse método recupera o *tenant* que faz uso do sistema, verificando o seu catálogo. Na Figura 4.13 é apresentado um exemplo de código para o método.

```
class Feature {
    private String name
    private boolean mandatory
    private Feature parent
    private List<Feature> alternatives
    private List<Feature> includes
    private List<Feature> excludes

    //Contrutores
    //Getter e Setters

    static public void hasFeature(feature) {
        Tenant currentTenant = Tenant.getCurrentTenant()
        if (currentTenant.catalog.findbyFeature(feature)){
            return true
        }
        return false
    }
}
```

Figura 4.13: Verificação se o *tenant* possui a *feature* em seu catálogo.

```
@Feature(
    name="Bank Check",
    mandatory=false,
    parent="Payment Method",
    alternatives=[],
    excludes=[],
    includes=["Client"]
)
class BankCheck {
    private Integer accountNumber
    private Client owner
    //...
}
//...lógica do SaaS
paymentMethodsAvailables = ["Cash", "Bill", "Credit Card"]

if (Feature.hasFeature("Bank Check")){
    paymentMethodsAvailables << "Bank Check"
}
```

Figura 4.14: Exemplo de manipulação do fluxo de execução do SaaS, por meio do método `hasFeature`.

Na Figura 4.14 é apresentado como a lógica do SaaS pode ser manipulada segundo o catálogo de *features*. Nesse exemplo, o SaaS provê as formas de pagamento: dinheiro, boleto e cartão a todos os *tenants*. Apenas os *tenants* que possuirem a *feature* "Bank Check" em seu catálogo terão o método disponível.

- **Uso conhecido:** no Capítulo 6 é apresentado um estudo de aplicabilidade onde este padrão é usado.
- **Padrões relacionados:** é usado o padrão Entidades MT, *builder* e Strategy

4.6 Padrões Relacionados

Nesta seção são apresentados os padrões relacionados ao desenvolvimento de SaaS MT. Esses padrões foram obtidos por meio de um mapeamento sistemático da literatura, apresentado em detalhes no Capítulo 3. Na Tabela 4.1 são descritos os padrões obtidos, indicando suas vantagens e desvantagens em relação à escalabilidade, complexidade, flexibilidade, extensibilidade, manutenibilidade e reusabilidade para manipular a variabilidade dos *tenants* em termos de interfaces com usuários, dados, processos, permissões.

Na Tabela 4.2 é apresentada a comparação dos padrões propostos com os padrões existentes na literatura que tratam do mesmo tipo de variabilidade, evidenciando as vantagens e desvantagens de cada um.

Tabela 4.1: Padrões para desenvolvimento de SaaS MT obtidos pelo mapeamento sistemático da literatura.

Nome Padrão	Função principal	Customizações	Vantagens	Desvantagens
Parameter Pattern	Diferenciar modificações específicas por tenant que possam ser tratadas com o mecanismo de chave/valor.	UI e dados	Baixa complexidade; Alta manutenibilidade para cenários simples; Alta reusabilidade.	Baixa manutenibilidade e escalabilidade para cenários complexos; Baixa extensibilidade; Baixa flexibilidade.
Routing Pattern	Isolar as regras de negócio do código-fonte por meio de serviços.	Processos	Alta extensibilidade; Alta manutenibilidade; Alta flexibilidade.	Alta complexidade; Baixa reusabilidade e escalabilidade para cenários com muitos tenants.
Service Wrapping Pattern	Tratar a incompatibilidade de protocolos de comunicação, tipos de dados, e interfaces dos serviços que compõe o SaaS.	Processos	Alta extensibilidade; Alta reusabilidade; Alta flexibilidade,	Alta complexidade; Baixa manutenibilidade; Baixa escalabilidade para cenários com muitos tenants.
Variant/Template Pattern	Permitir que as funcionalidades sejam definidas em tempo de execução, segundo um conjunto de variantes pré-definidos.	UI, dados e processos	Alta flexibilidade; Alta reusabilidade.	Alta complexidade; Baixa manutenibilidade e escalabilidade para cenários com muitos tenants.

Extension Points Pattern	Permitir que os tenants possam implementar suas próprias regras de negócio	Processos	Alta flexibilidade; Alta extensibilidade.	Alta complexidade; Baixa manutenibilidade; Ameaças à segurança.
Copy and Adapt Pattern	Utilizado quando as técnicas de variabilidade não são suficientes.	Processos, UI, dados e permissões	Alta reusabilidade; Baixa complexidade.	Baixa manutenibilidade; Baixa produtividade e escalabilidade para cenários com muitos tenants.

Tabela 4.2: Comparações dos padrões propostos em relação aos padrões da literatura.

Padrão proposto	**Padrão da literatura**	**Vantagens do padrão proposto**	**Desvantagens do padrão proposto**
Campo Extra MT/ GUI MT	*Parameter Pattern*	Melhor escalabilidade, manutenibilidade e flexibilidade para cenários com muitos *tenants*	Campo Extra MT e GUI MT são mais complexos para implementar
Campo Extra MT/ GUI MT	*Variant/Template Pattern*	Melhor escalabilidade e manutenibilidade para cenários com muitos *tenants*	Não há
Campo Extra MT/ GUI MT	*Copy and Adapt Pattern*	Melhor escalabilidade, produtividade e manutenibilidade para cenários com muitos *tenants*.	Campo Extra MT e GUI MT são mais complexos para implementar

Catálogo de *Features* MT	*Routing Pattern*	Melhor escalabilidade e reusabilidade para cenários com muitos *tenants*.	É mais complexo para implementar.
Catálogo de *Features* MT	*Service Wrapping Pattern*	Melhor manutenibilidade para cenários complexos com muitos *tenants*.	Menor extensibilidade.
Catálogo de *Features* MT	*Variant/Template Pattern*	Melhor escalabilidade e manutenibilidade para cenários com muitos *tenants*.	Menor extensibilidade.
Catálogo de *Features* MT	*Extension Points Pattern*	Melhor manutenibilidade e sem ameaças à segurança.	Menor extensibilidade.
Catálogo de *Features* MT	*Copy and Adapt Pattern*	Melhor escalabilidade, produtividade, manutenibilidade para cenários com muitos *tenants*.	É mais complexo para implementar; Menor extensibilidade.
Entidades MT	Permissões	Não foram obtidos padrões para tratar a variabilidade de permissão	

As principais desvantagens dos padrões propostos em relação aos padrões existentes na literatura são a complexidade de implementação e a menor extensibilidade. A complexidade de implementação é reduzida pelo uso do FoSaaS, que já implementa os padrões em sua estrutura, como apresentado em detalhes no Capitulo 5. A menor extensibilidade é causada pelo alto acoplamento das estruturas dos padrões propostos. Contudo, o acoplamento das classes foi necessário para aumentar a reusabilidade e manutenibilidade de SaaS com muitos *tenants*. Os padrões propostos têm diferencial quanto à escalabilidade com relação aos padrões existentes na literatura. Para cenários em que há *tenants* com requisitos conflitantes, os padrões da literatura se tornam problema em relação à manutenção, uma vez que se dedicam a tratar a individualidade dos *tenants* sem tratar o reúso das *features.* Os padrões propostos não se baseiam em estruturas específicas por *tenant,* cada requisito específico é tratado como *feature,* de modo que outros *tenants* possam reusa-las sem a edição do código-fonte.

4.7 Considerações Finais

Os padrões apresentados são dedicados a resolver os problemas associados a arquitetura MT, causados pela variação dos requisitos funcionais dos *tenants.* Em outras palavras, o tratamento das variabilidades por meio dos padrões está endereçado a resolver os problemas de customização dos *tenants* em um SaaS MT. Cada padrão está dedicado a resolver um tipo de variabilidade. Na Tabela 4.3 são relacionados o padrão ao tipo de variabilidade que ele trata.

Tabela 4.3: Padrões propostos e a variabilidade manipulada

Nome do padrão	Variabilidade tratada
Entidade MT	Permissão
Campo Extra MT	Dados
GUI MT	Interfaces de usuários
Catálogo de *Features* MT	Processos

Para facilitar a incorporação dos padrões aos SaaS, foi desenvolvido um *framework* de aplicação que implementa os padrões em sua estrutura. No próximo capítulo, essa ferramenta é descrita em detalhes.

Capítulo 5
FRAMEWORK DE APLICAÇÕES FOSAAS

5.1 Considerações Iniciais

No desenvolvimento de *Software as a Service Multi-tenant* (SaaS MT), desenvolvedores enfrentam problemas recorrentes com escalabilidade, segurança, performance e customização (Kitano *et al.*, 2010; Li *et al.*, 2011). Desenvolvedores inexperientes podem encontrar desafios para manipular corretamente a variabilidade, à medida que novos *tenants* são adicionados à aplicação. Esse problema pode ser atenuado por meio de ferramentas que deixam imperceptíveis as estruturas que gerenciam a variação dos requisitos não funcionais e funcionais dos *tenants.*

O *Framework of Software as a Service* (FoSaaS) foi construído para apoiar o desenvolvimento de SaaS MT, minimizando o problema da customização em aplicações MT e provendo soluções para tratar a variabilidade dos *tenants*. Com o objetivo de reduzir o esforço na construção de SaaS, o FoSaaS utiliza técnicas de reúso, como: os padrões propostos neste projeto de mestrado (Capítulo 4); padrões de projeto do Gamma *et al.* (1995); Linha de Produtos de Software (LPS); Desenvolvimento Dirigido por Modelos (*Model-Driven Development* - MDD); Desenvolvimento Dirigido por Comportamento (*Behavior-Driven Development* - BDD); e boas práticas de desenvolvimento. Com o uso do FoSaaS, os desenvolvedores de software têm um guia para a criação de SaaS MT extensíveis e flexíveis.

Para que outros estudos possam utilizar o FoSaaS como referência e auxiliar desenvolvedores a replicá-lo em outras linguagens, neste capítulo serão descritos a estrutura, implementação, e um guia para utilização do FoSaaS. Na Seção 5.2 é apresentada a estrutura do *framework* por meio de sua arquitetura e o diagrama de classes. Esses artefatos ilustram como os dados foram modelados e a iteração dos padrões criados por esse trabalho. Na Seção 5.3 são apresentados detalhes da implementação do FoSaaS, indicando como as técnicas de reúso foram aplicadas. Na Seção 5.4 é apresentado um guia do processo de desenvolvimento de SaaS MT com o FoSaaS. Na Seção 5.5 são expostas as considerações finais.

5.2 Estrutura

As estruturas principais do FoSaaS são baseadas no catálogo de padrões propostos por este projeto de mestrado. A descrição em alto-nível do FoSaaS é apresentada em detalhes nas seções a seguir, por meio de sua arquitetura e diagrama de classes.

5.2.1 Arquitetura

A arquitetura do FoSaaS está baseada na estrutura dos padrões Entidades MT, Campo Extra MT, GUI MT e Catálogo de Features MT, apresentados em detalhes no Capítulo 4. Esses padrões são propostos para obter a customização de interfaces, dados, processos e permissões do *tenants* em SaaS MT. Na Figura 5.1 é ilustrado como os padrões podem ser utilizados para manipular as personalizações.

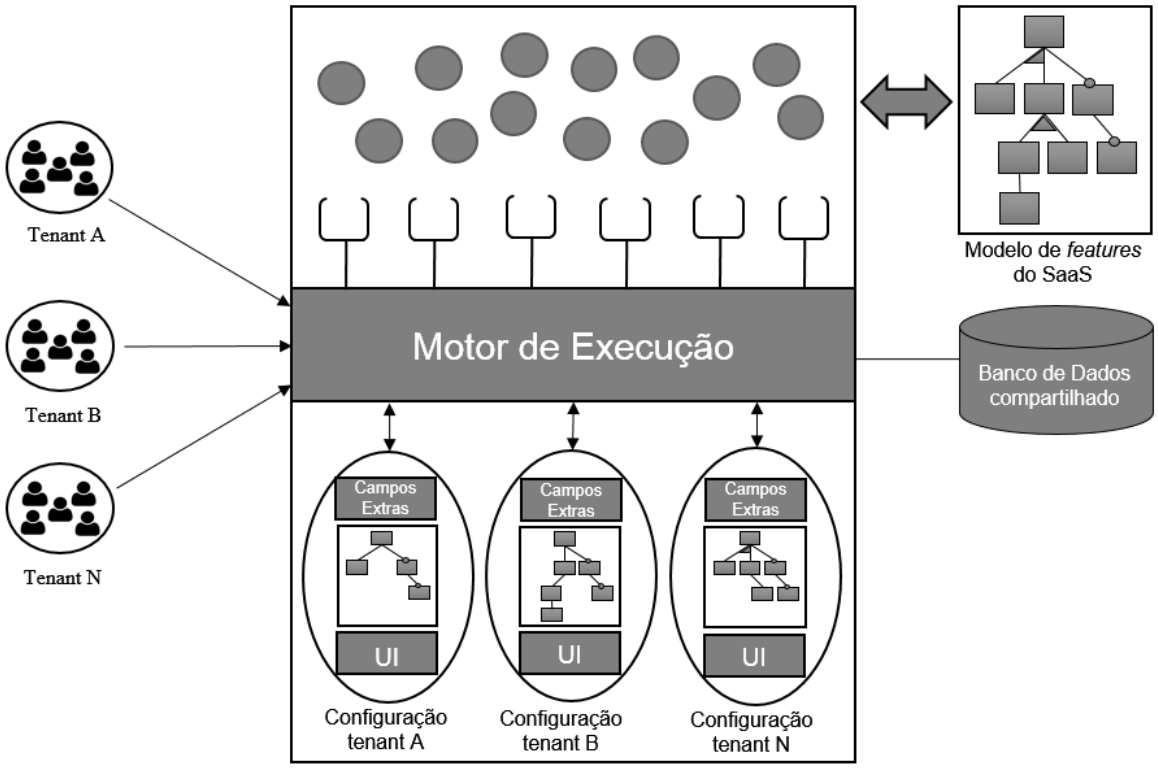


Figura 5.1 Arquitetura das interações dos componentes do FoSaaS.

A customização dos processos e permissões é realizada por meio do padrão Catálogo de *Features* MT. Esse padrão é usado para controlar a execução dos *tenants*, por meio de um conjunto de *features*. Na Figura 5.1, as *features* acessíveis ao *tenant* são representadas pela imagem central da configuração do *tenant*. Essas *features* são selecionadas no estágio de configuração do *tenant* e compõe um subconjunto do modelo de *features* do SaaS. Por meio dessa arquitetura, os *tenants* podem ser inseridos e excluídos do SaaS em tempo de execução, em outras palavras, não é necessário a recompilação do software. Para que isso fosse possível, o FoSaaS apresenta uma interface de usuário responsável por manipular os registros dos *tenants* ativos. Por meio dessa interface, o provedor do serviço é capaz de gerenciar as *features* de cada *tenant* e suas características visuais, apresentado em detalhes no padrão GUI MT (Seção 4.4).

O motor de execução contém os métodos `getCurrentTenant` e `hasFeature`, responsáveis por manipular a execução dos processos em tempos de execução, de acordo com o *tenant* que faz uso do SaaS. O funcionamento detalhado dos métodos é apresentado na Seção 4.5 do capítulo anterior.

A customização dos dados é realizada por meio do padrão Campo Extra MT. Esse padrão é utilizado para flexibilizar o conteúdo dos formulários de acordo com a necessidade

do *tenant.* Por meio de métodos definidos no padrão, é possível associar novos atributos a uma classe, deste modo os *tenants* podem incluir ou excluir informações de formulários durante todo ciclo de vida do software. Na Figura 5.1 os campos extras criados pelos *tenants* são representados pelo bloco superior da configuração do *tenant*. O motor de execução contém os métodos `createExtraFieldValue` e `getExtraFieldValue` responsáveis por manipular os campos de acordo com o *tenant* que faz uso do SaaS. O funcionamento detalhado dos métodos é apresentado na Seção 4.3 do capítulo anterior.

A customização das interfaces com usuário (*User Interfaces* - UI) é realizada por meio do padrão GUI MT. Esse padrão permite que os *tenants* configurem cores, formas e disposição dos elementos da interface de acordo com a sua identidade visual. Na Figura 5.1 as personalizações das UI são representadas pelo bloco inferior da configuração do *tenant*. O motor de execução contém os métodos que alteram os elementos da GUI de acordo com o *tenant* que faz uso do SaaS. O funcionamento detalhado dos métodos é apresentado na Seção 4.2 do capítulo anterior.

O padrão Entidades MT é utilizado para implementar a arquitetura MT, com o compartilhamento da base de dados e resolução do *tenant* por meio do usuário autenticado. O motor de execução contém o método `getCurrentTenant`, responsável pela resolução do *tenant* que faz uso do sistema e utilizado para persistir os dados indicando a propriedade do registro. O funcionamento detalhado do método é apresentado na Seção 4.1 do capítulo anterior.

No FoSaaS foram usados outros padrões amplamente utilizados na indústria de software, tais como *Singleton*, *Observer*, *Adaptor*, *Proxy*, *State* e o padrão composto *Model-View-Controller* (MVC), a maioria deles presentes na própria estrutura do *framework* utilizado para o desenvolvimento.

5.2.2 Diagrama de Classes

As classes que compõem a estrutura dos padrões propostos por este projeto de mestrado formam as principais classes do modelo do FoSaaS. Na Figura 5.2 é apresentado o diagrama de classes do FoSaaS, sem as classes das camadas *View* e *Controller*, omitidas para facilitar a visualização. A relação da estrutura dos padrões e do *framework* não é um-para-um, pois as classes `Role` e `PermissionGroup` foram inseridas ao modelo e não fazem parte de nenhum padrão específico. Na Tabela 5.1 é apresentada em detalhes cada

classe, indicando o padrão de origem, a principal função e a qual tipo de customização está dedicada.

Figura 5.2: Diagrama de classes que representa a estrutura do FoSaaS.

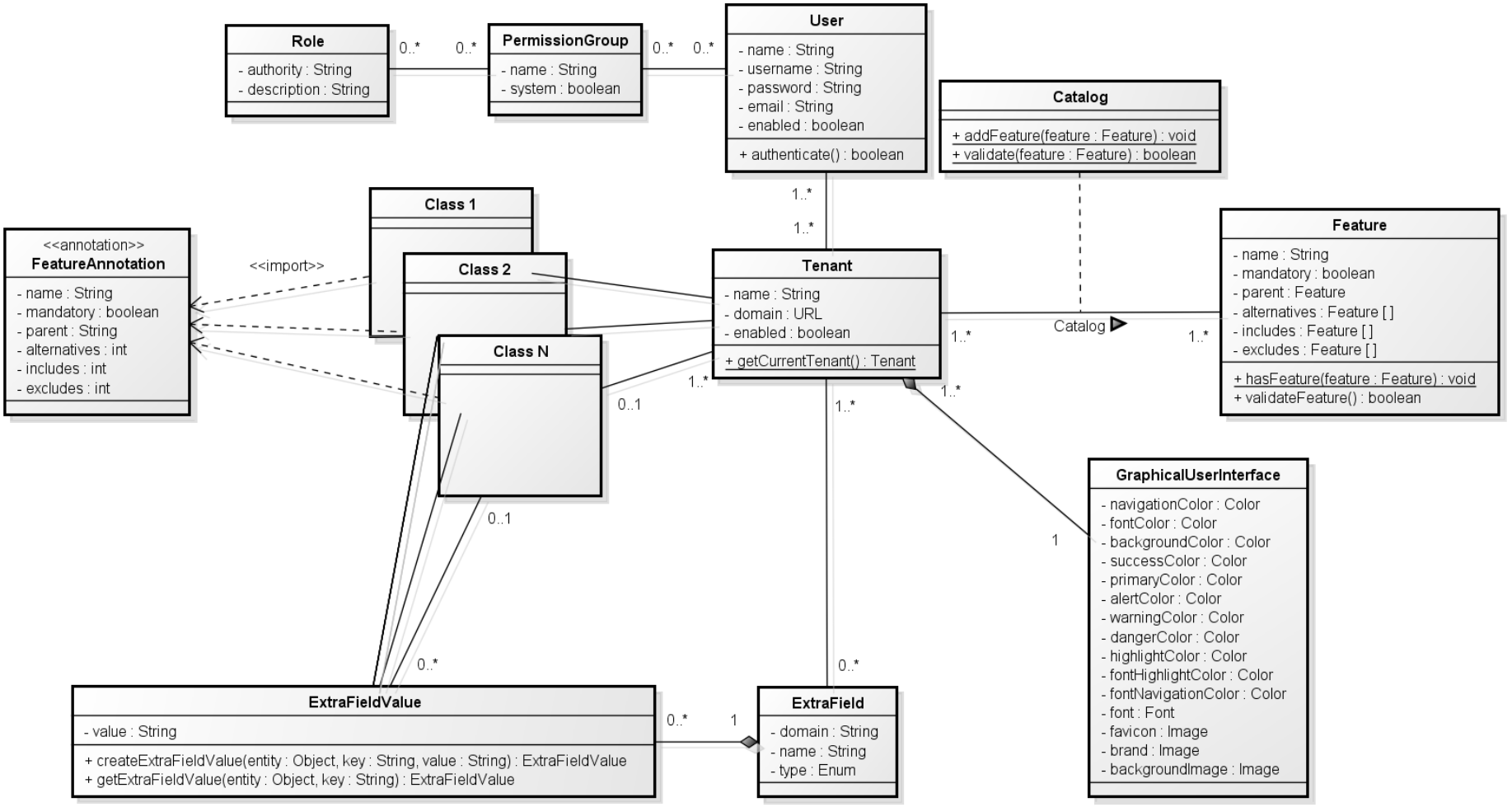


Tabela 5.1: Descrição das classes presentes no modelo do FoSaaS.

Classe	Padrão	Função principal	Customização
`Tenant`	Entidades MT, GUI MT, Campo Extra MT, Catálogo de *Feature* MT	Define os objetos que representam os consumidores do serviço e fornece o método estático `getCurrentTenant`, responsável pela resolução do *tenant* que faz uso do sistema.	Interfaces, dados, processos, permissões
`User`	Entidades MT	Define os objetos que representam os usuários do SaaS e fornece o método `authenticate`, responsável por adicionar a instância do usuário na sessão HTTP.	Dados, permissões
`Catalog`	Catálogo de Feature MT	Classe de associação que relaciona os *tenants* e suas *features*. Possui os métodos estáticos `addFeature` e `validate`, responsáveis por adicionar novas *features* ao *tenant*, verificando se nenhuma regra do modelo de *features* do SaaS foi violada.	Processos
`Feature`	Catálogo de Feature MT	Classe que representa as características do SaaS. Os atributos da classe `Feature` descrevem os relacionamentos e restrições necessários para construção de um modelo de *features*. O método `validateFeatureModel` impede que as regras de formação do modelo sejam violadas, enquanto o método estático `hasFeature`	Processos

		analisa se a *feature* acessada está contida no catálogo do *tenant* que faz uso do SaaS.	
`FeatureAnnotation`	Catálogo de Feature MT	Anotação para classes e métodos que define as restrições e relacionamentos entre as *features* do SaaS.	Processos
`GraphicalUserInterface`	GUI MT	Composição da classe `Tenant` que contém os atributos que definem os elementos da interface do usuário e representam a identidade visual do *tenant*. Cada instância da classe `Tenant` está relacionada a uma instância da classe `GraphicalUserInterface`.	Interfaces
`ExtraField`	Campo Extra MT	Define os campos extras dos formulários e possui atributos que identificam o tipo e a classe à qual o campo está associado.	Dados
`ExtraFieldValue`	Campo Extra MT	Composição que define os valores associados aos campos extras dos formulários. Os métodos estáticos `createExtraFieldValue` e `getExtraFieldValue` são usados para atribuir e recuperar os valores respectivamente.	Dados
`Class1, 2, N`	Entidades MT, Campo Extra MT, Catálogo de *Feature* MT	Representam as classes do domínio da aplicação. No padrão Entidades MT estão relacionadas a classe `Tenant` para indicar a propriedade do dado. No padrão Campo Extra MT estão relacionadas à classe `ExtraFieldValue`, para que novos atributos possam ser definidos pelo *tenant*. Por fim, no padrão Catálogo de *Features* MT, representam	Nenhuma

		as classes que podem ser marcadas com a anotação `FeatureAnnotation`.	
`PermissionGroup`	Nenhum	Define uma categoria de usuários que compartilham as mesmas permissões de acesso no SaaS.	Permissões
`Role`	Nenhum	Define as permissões de acesso (ex. cadastrar novos usuários, editar informações do *tenant*, listar produtos, etc). O atributo `authority` é usado para indicar o nome da permissão.	Permissões

5.3 Desenvolvimento do FoSaaS

Esta Seção é dedicada a apresentar os detalhes do processo de desenvolvimento do FoSaaS. O objetivo é auxiliar desenvolvedores e pesquisadores a replicarem o *framework* em seus trabalhos, apresentando as ferramentas utilizadas e como as técnicas de reúso foram aplicadas.

5.3.1 Ferramentas

O sistema utilizado para controle de versão do código fonte foi o GIT (2012), um sistema de código aberto e distribuído. O gerenciamento de incidentes e repositórios foi realizado por meio do Bitbucket (Atlassian, 2013), um sistema *web* de hospedagem de repositórios GIT. O FoSaaS foi implementado utilizando a linguagem de programação Groovy (Codehaus, 2013) e o *framework* de aplicações Grails (SpringSource, 2012). O Grails fornece o mecanismo de encapsulamento de softwares de propósito específico em *plug-ins* e a linguagem Groovy maior nível de abstração que a torna mais próxima da linguagem natural. Esse mecanismo auxilia a incorporação de código fonte de terceiros a aplicação. Para digitação do código fonte foi utilizado o SublimeText, um editor de texto compatível com a sintaxe Groovy. Foram utilizados os bancos de dados MySQL (2011) para desenvolvimento, por ser livre de licença e o H2 (H2database, 2014), um banco de dados relacional com armazenamento em memória para fins de teste. O desenvolvimento do FoSaaS foi guiado por comportamento (BDD) com auxílio do *plug-in* Spock (Spock, 2013), que permite definir cenários para execução de testes unitários. Na Tabela 5.2 é apresentado cada *plug-in* Grails utilizado no desenvolvimento do FoSaaS e sua principal função.

Tabela 5.2: Lista do plug-ins Grails utilizado no FoSaaS.

Nome	Função
Codenarc	Análise estática do código fonte visando encontrar falhas em potencial.
Spring-security-core	Fornece mecanismo de autenticação de usuários e encriptação de senhas.
Twitter-bootstrap	Fornece uma biblioteca CSS para criação das UI.
Grails-melody	Monitora recursos de hardware utilizados pelo SaaS e identifica problemas de performance.
Spock	Define cenários para execução de testes unitários
Code-coverage	Indica linhas e a porcentagem de código fonte cobertos pelos testes unitários.

Visando facilitar a incorporação do FoSaaS nos SaaS desenvolvidos em Grails, sua implementação foi encapsulada em um *plug-in.* O código fonte do FoSaaS é aberto e pode ser acessado por meio do endereço: https://bitbucket.org/brunodleite/frameworksaas

5.3.2 Gerador automático de código fonte

O FoSaaS, além de implementar em sua estrutura principal os padrões para desenvolvimento de SaaS MT também fornece o mecanismo de geração automática de código fonte. Baseado em MDD, o FoSaaS utiliza *templates*, um mecanismo de transformação responsável por gerar o código da aplicação a partir de um modelo. *Templates* de código são arquivos descritos em uma Linguagem Específica de Domínio (*Domain Specific Language* – DSL) que representam um meta-código para geração do código fonte em uma linguagem de programação específica. Os *templates* tornaram-se um importante artefato no MDD, pois a geração automática de código protege os desenvolvedores das complexidades da implementação (Lucrédio, 2009). O FoSaaS faz uso de *templates* visando a automatização do processo de desenvolvimento por meio de modelos genéricos e pré-definidos e esses são usados como gabarito para geração dos métodos Criar, Recuperar, Atualizar e Apagar (*Create, Retrieve, Update, Delete* - CRUD). Na transformação do modelo para código fonte, são considerados: atributos, relacionamentos, anotações e restrições descritas nas classes pertencentes ao modelo do SaaS. Nas Figuras 5.3 e 5.4 são

apresentados dois exemplos de *templates* do FoSaaS. O primeiro é utilizado para geração dos controladores, classes responsáveis por manipular requisições HTTP, bem como orquestrar o uso dos serviços que contém as regras de negócio. Na Figura 5.4 é apresentado um *template* para geração das UI, um arquivo HTML composto por formulários e botões.

```
<%=packageName ? "package ${packageName}\n\n" : ''%>
import grails.plugins.springsecurity.Secured
import org.springframework.dao.DataIntegrityViolationException

class ${className}Controller extends core.BaseController {

    static allowedMethods = [
        create: ['GET', 'POST'],
        edit:   ['GET', 'POST'],
        delete: ['GET', 'POST']
    ]

    // For automatic role creation
    static roles = ['SHOW', 'ADD', 'EDIT', 'DEL']

    @Secured(['ROLE_${className.toUpperCase()}_ADD'])
    def create = {
        log.info '* Method: create'
        log.debug "* Params: \${params}"

        switch (request.method) {
            case 'GET':
                [${propertyName}: new ${className}(params)]
                break

            case 'POST':
                def ${propertyName} = new ${className}(params)
                if (!${propertyName}.save(flush: true)) {
                    return render(view: 'create', model: [${propertyName}: ${propertyName}])
                }

                flashMessage('default.created.message', ${propertyName}.id)
                redirect(action: 'show', id: ${propertyName}.id)
                break
        }
    }
```

Figura 5.3: Trecho do template para geração dos controladores.

```
<%=packageName%>
<!doctype html>
<html>
    <head>
        <meta name="layout" content="main">
        <g:set var="entityName"
            value="\${message(code: '${domainClass.propertyName}.label', default: '${className}')}" />
        <g:set var="entitiesName"
            value="\${message(code: '${domainClass.propertyName}.label.plural', default: '${className}')}" />
        <title>
            <g:message code="default.create.label" args="[entityName]" />
        </title>
    </head>
    <body>
        <bootstrap:header title="\${message(code:'default.create.label', args:[entityName])}">
            <bootstrap:buttonList/>
        </bootstrap:header>

        <g:if test="\${flash.message}">
            <bootstrap:alert class="alert-info">\${flash.message}</bootstrap:alert>
        </g:if>

        <g:form class="form-horizontal" action="create" <%= multiPart ? ' enctype="multipart/form-data"' : '' %>>
            <bootstrap:form title="\${message(code:'${propertyName}.form.legend')}">
                <f:all bean="${propertyName}"/>
            </bootstrap:form>
            <bootstrap:actionButtons>
                <button type="submit" class="btn btn-primary">
                    <i class="glyphicon glyphicon-ok glyphicon glyphicon-white"></i>
                    <g:message code="default.button.create.label" default="Create" />
                </button>
            </bootstrap:actionButtons>
        </g:form>
    </body>
</html>
```

Figura 5.4: Template para geração da UI do método criar.

Nos *templates*, o código Groovy é identificado pelos marcadores `${}` que contém variáveis de metadados: `propertyName`, `packageName`, `className` e `domainClass`. Os valores das variáveis de metadado são atribuídos de acordo com a classe que está sendo transformada. Na Tabela 5.3 são descritas as variáveis de metadado, sua função no *template* e um exemplo. As demais informações não são interpretadas como variáveis e apareceram no formato original no código fonte gerado.

Tabela 5.3: Descrição dos metadados presentes nos templates do FoSaaS.

Metadado	Função	Exemplo
`propertyName`	Representa o identificador do atributo presente na classe transformada.	custoDoProduto
`className`	Representa o nome da classe transformada.	Produto
`packageName`	Nome do pacote que contém a classe transformada.	rfid.fit.org
`domainClass`	Concatenação de `packageName` e `className`.	rfid.fit.org.Produto

Embora o MDD não seja o foco deste projeto, a geração automática de código mostrou-se um importante artefato para reduzir a complexidade do desenvolvimento. Os benefícios do MDD para implementação dos FoSaaS foram: aumento da produtividade, redução do trabalho repetitivo e aumento da qualidade do software com a eliminação de trechos de código redundantes.

5.4 Diretrizes para desenvolvimento de SaaS utilizando o FoSaaS

Essa seção é voltada para desenvolvedores e engenheiros de software que desejam utilizar o FoSaaS no desenvolvimento de SaaS MT. O ciclo de vida dos SaaS, produzidos com apoio do FoSaaS, baseia-se nas atividades de desenvolvimento de LPS, visando o reúso do ativo base e gerenciamento da variabilidade. Contudo, o FoSaaS não produz produtos como em uma LPS, mas um SaaS MT composto por todos os possíveis produtos dela. Na Figura 5.5 são ilustradas as diretrizes do processo de desenvolvimento por meio de um diagrama de atividades.

Na Tabela 5.4 são descritas as atividades que compõem o processo de desenvolvimento do SaaS. Para cada atividade são apresentadas as entradas e saídas esperadas, as ferramentas necessárias e as ações que devem executadas. Ao final do processo espera-se gerar uma instância do software capaz de atender as necessidades de todos os *tenants* para o domínio do SaaS.

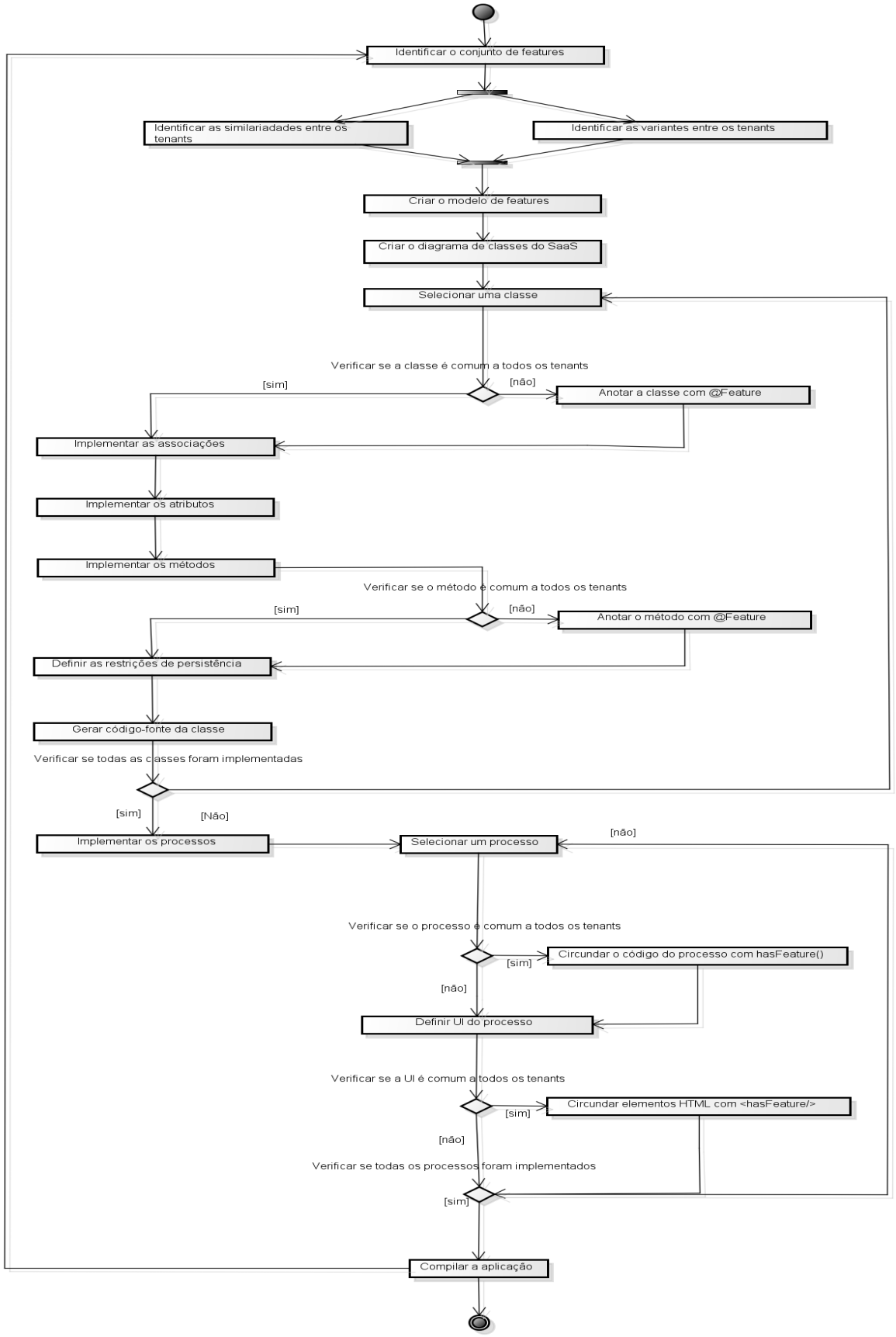


Figura 5.5: Diretrizes para desenvolvimento de SaaS MT com o auxílio do FoSaaS.

Tabela 5.4: Descrição das atividades das diretrizes de desenvolvimento com o FoSaaS.

Atividade	Entrada(s)	Saída(s)	Ações	Ferramenta(s)
Identificar o conjunto de *features* do SaaS	Nenhuma	Documento de requisitos de cada *tenant*	Levantar os requisitos de todos os *tenants* do SaaS	Questionários; Formulários; Entrevista
Identificar as similaridades entre os *tenants*	Documento de requisitos de cada *tenant* do SaaS	Definição do ativo base.	Realizar a análise de domínio e análise de aplicações para identificação das similaridades entre os *tenants*	Nenhuma
Identificar as variantes entre os *tenants*	Documento de requisitos de cada *tenant* do SaaS	Documento com as *features* variantes	Realizar a análise de domínio e análise de aplicações para identificação das *features* divergentes/conflitantes ente os *tenants*	Nenhuma
Criar o modelo de *features* do SaaS	Documento com as *features* comuns e variantes	Modelo de *features*	Construir o modelo de *features* de acordo as *features* obtidas pela análise de domínio e aplicações	Feature IDE
Criar o diagrama de classes do SaaS	Modelo de *features*	Diagrama de classes	Criar o diagrama de classes que compõe o modelo do SaaS	Astah community
Selecionar uma classe	Diagrama de classes	Uma classe	Selecionar uma classe que não tenha sido implementada	Nenhuma
Anotar a classe com `@Feature`	Resultado da análise	Declaração da classe com a anotação `@Feature`	Anotar a declaração da classe com `@Feature` e preencher os atributos da anotação de acordo com as restrições do modelo de *features* do SaaS	Editor de texto
Implementar as associações	Implementação da classe	Implementação da classe com as associações	Adicionar os atributos que representam os relacionamentos com as demais classes do modelo	Editor de texto
Implementar os atributos	Implementação da classe	Implementação da classe com os atributos	Adicionar os atributos da classe de acordo com o diagrama de classes	Editor de texto
Implementar os métodos	Implementação da classe	Implementação da classe com os métodos	Implementar os métodos de acordo com o diagrama de classes	Editor de texto

Anotar o método com `@Feature`	Resultado da análise	Declaração do método com a anotação `@Feature`	Anotar a declaração do método com `@Feature` e preencher os atributos da anotação de acordo com as restrições do modelo de *features* do SaaS	Editor de texto
Definir as restrições de persistência	Implementação da classe	Implementação da classe finalizada	Indicar por meio das `constraints` quais as restrições de valores para os atributos definidos na classe.	Editor de texto
Gerar código fonte da classe	Implementação finalizada	CRUD da classe implementada	Executar o comando de geração de código fonte indicando a classe que será transformada.	*Prompt* de comando
Implementar os processos do SaaS	Todas as classes implementadas	Processos implementados	Implementar os processos do SaaS, de acordo com as regras de negócio de cada *tenant*	Nenhuma
Selecionar um processo	Processos implementados	Um processo	Selecionar um processo que ainda não foi analisado	Nenhuma
Circundar o código fonte do processo com `hasFeature`	Resultado da análise	Código fonte do processo condicionalmente circundado	Adicionar a condição `hasFeature` entorno do código fonte do processo.	Editor de texto
Definir UI do processo	Código fonte do processo	Código fonte da UI do processo	Implementar os elementos que compõe a UI do processo de acordo com a necessidade dos *tenants*	Editor de texto
Circundar elementos HTML com `<hasFeature/>`	Código fonte da UI do processo	Código fonte da UI condicionalmente circundado	Adicionar o marcador `hasFeature` entorno do código fonte da UI do processo	Editor de texto
Compilar a aplicação	*Feature* implementada	Executável	Compilar o código-fonte da aplicação.	Compilador

5.5 Considerações Finais

O FoSaaS é uma ferramenta que implementa os padrões que manipulam as variabilidades em SaaS MT. Esse capítulo apresentou como foi realizado o desenvolvimento do FoSaaS, as técnicas utilizadas e um guia para que desenvolvedores e pesquisadores possam replicá-lo em seus projetos. Um estudo de aplicabilidade, apresentado no Capítulo 6, utilizou o FoSaaS para os desenvolvimentos do SaaS comerciais. Esse estudo evidenciou aumento do reúso, qualidade e produtividade. Para verificar os resultados evidenciados foi realizado uma avaliação por meio de experimentos controlados, apresentados em detalhes no próximo capítulo

Capítulo 6

AVALIAÇÃO

6.1 Considerações Iniciais

A avaliação de novas técnicas e produtos é importante para que desenvolvedores possam utilizá-las de forma segura e eficiente, com base em elementos concretos e dados confiáveis (Tichy, 1998). Neste capítulo é apresentada uma avaliação do catálogo de padrões para a construção de Software como um Serviço *Multi-Tenant* (SaaS MT), utilizando o framework FoSaaS (*Framework of Software as a Service*), proposto neste projeto de mestrado.

Na Seção 6.2 são apresentados dois estudos da aplicabilidade dos padrões e do FoSaaS. Na Seção 6.3 são apresentados os objetivos, questões e métricas elaboradas a partir do método *Goal/Question/Metric* (GQM) (Basili *et al.*, 1994) para a avaliação deste projeto por meio de experimentos controlados. Na Seção 6.4 são descritos os experimentos conduzidos para a obtenção dos resultados e na última seção são expostas as considerações finais.

6.2 Estudo da Aplicabilidade

Com o objetivo de avaliar na prática a proposta deste mestrado, foi realizado um estudo da aplicabilidade dos padrões por meio de dois projetos com clientes reais que foram desenvolvidos no FIT. Os desenvolvedores desses projetos são os participantes identificados como P1 a P6 descritos na Seção 6.4.2 deste capítulo

Em comum, os projetos utilizaram o FoSaaS para criar SaaS MT capazes de atender os requisitos funcionais de diversos *tenants* simultaneamente. Os *tenants* desses dois projetos foram: HP, IBM, Flextronics e DHL que apresentavam requisitos similares de software, porém com algumas particularidades. Na Seção 6.2.1 são apresentados com mais detalhes a motivação desses projetos, as *features* extraídas do levantamento de requisitos e como o FoSaaS foi utilizado no tratamento da variabilidade dos *tenants.*

6.2.1 Motivação

Produtos e ativos identificados por rádio frequência (*Radio Frequency Identification* - RFID) devem ser associados a um código eletrônico de produto (*Electronic Product Code* - EPC) (EPCGlobal, 2013). O EPC *Information System* (EPCIS) é um padrão desenvolvido pela GS1 e EPCGlobal (EPCGlobal, 2013), para permitir que as operações de negócio realizadas sobre itens com RFID sejam compartilhadas entre parceiros de negócio. Ao longo da cadeia de suprimentos, os itens produzidos e comercializados são movidos entre as unidades de negócio como, por exemplo, armazenamento, recebimento, expedição, etc. Os dados obtidos por meio de RFID, em cada operação de negócio, podem fornecer informações sobre a localização e situação dos itens em um determinado tempo e espaço. As informações obtidas por esses dados, permitem a rastreabilidade dos itens.

Para obter a rastreabilidade dos itens na cadeia de suprimentos, os projetos tiveram como objetivo criar um EPCIS como um serviço e um software complementar responsável por capturar as operações de negócio, denominado RFIDaaS (RFID *as a Service*). Em decorrência dos benefícios do modelo SaaS, esses projetos foram dedicados à disseminação da tecnologia RFID para pequenas e médias empresas. Contudo, a alta variabilidade das aplicações RFID precisou ser tratada de forma efetiva para que todos os benefícios da arquitetura MT pudessem ser obtidos. Os SaaS foram construídos para

atender as *features* apresentadas na Seção 6.2.2, por meio dos padrões elaborados neste projeto e o desenvolvimento foi auxiliado pelo FoSaaS.

6.2.2 Features dos *tenants*

Nas Tabela 6.1 e 6.2 são apresentadas as *features* identificadas do EPCIS e do RFIDaaS, respectivamente, para os *tenants* do serviço. *As features* foram extraídas no início do projeto por meio do levantamento de requisitos realizado com cada *tenant* do SaaS.

Tabela 6.1: Features do RFIDaaS

	Tenants			
Features	HP	Flextronics	IBM	DHL
Gerenciamento de usuários e permissões.	X	X	X	X
UI de acordo com sua identidade visual	X	X		
Cadastro de produtos	X	X	X	X
Cadastro de clientes	X			X
Cadastro de fornecedores		X		X
Histórico do item	X		X	
Localização do item por área	X		X	
Recebimento		X	X	
Montagem		X	X	
Empacotamentos (caixa ou palete)		X	X	
Empacotamento de produtos diferentes			X	
Armazenagem		X	X	X
Transferência entre áreas		X	X	X
Separação		X	X	X
Expedição		X	X	
Carregamento no caminhão				X
Entrega				X
Venda	X			
Reciclagem	X		X	
Comunicação com sistemas legados		X		X

A partir das Tabelas 6.1 e 6.2 é possível identificar que tanto para as *features* do RFIDaaS quanto do EPCIS, os *tenants* apresentam *features* comuns e divergentes. Por exemplo, para a *feature* "Cadastro de produtos" que faz parte de todos os *tenants* e a *feature* "Entrega", que é exclusiva do *tenant* DHL.

Tabela 6.2: Features do EPCIS

	Tenants			
Features	HP	Flextronics	IBM	DHL
Gerenciamento de usuários e permissões.	X	X	X	X
UI de acordo com sua identidade visual	X	X		
Relatório de operações de inventário		X	X	X
Relatório de operações de acondicionamento		X	X	
Relatório de operações de movimentação		X		X
Relatório de operações de negócio	X	X	X	
Comunicação com sistemas legados		X	X	

6.2.3 Desenvolvimento dos projetos utilizando o FoSaaS

O FoSaaS foi utilizado desde o início dos projetos, atuando como base para o desenvolvimento orientado ao catálogo dos padrões propostos. O padrão Entidades MT foi usado para o compartilhamento da base de dados para tratar da customização dos dados e permissões, de modo que um *tenant* não pudesse acessar os dados de outros *tenants*. O padrão Campo Extra MT foi aplicado na customização de formulários, utilizados nas operações de negócio. Embora nem todos os *tenants* exigissem uma identidade visual específica, o padrão GUI MT foi utilizado na customização das interfaces dos *tenants* HP e Flextronics. Cores, botões e imagens foram configurados de acordo com os requisitos desses *tenants*. A customização dos processos foi realizada por meio do padrão Catálogo de *Features* MT. Esse padrão foi amplamente utilizado no desenvolvimento dos SaaS e possibilitou gerenciar o comportamento dos softwares baseados no catálogo de *features* de cada *tenant.*

6.2.4 Resultados dos Estudos

A fim de avaliar os SaaS gerados por este estudo da aplicabilidade da proposta, em relação a capacidade de atender as necessidades de todos os *tenants* simultaneamente, foram definidos cenários de uso. Cada cenário descreve as regras de negócio para as companhias Flextronics e IBM. Os cenários simularam o comportamento real dos *tenants* em um processo de empacotamento. Nesse processo, os itens identificados por RFID devem ser associados a um recipiente (caixa ou palete) no qual são empacotados.

Tabela 6.3: Cenário A: Customização dos dados para o processo de empacotamento

Informações do formulário	***Tenants***	
	Flextronics	**IBM**
Padrão	X	X
Identificador do Pallet	X	
Part Number		X

A variabilidade dos requisitos da Flextronics e IBM, apresentada na Tabela 6.3, torna as informações do formulário de empacotamento distintas e devem ser tratadas individualmente. O padrão Campo Extra MT foi utilizado para resolução dos conflitos de requisitos, dessa forma as duas companhias são tratadas como *tenants* e os campos "Identificador do Palete" e "*Part Number"* são considerados campos extras da Flextronics e IBM, respectivamente.

Tabela 6.4: Cenário B: Customização do processo de empacotamento

Regras de negócio	***Tenants***	
	Flextronics	**Flextronics**
Associar os itens ao recipiente (padrão)	X	X
Permitir empacotamento de produtos iguais	X	X
Permitir empacotamento de produtos diferentes		X

Semelhante ao cenário A, os *tenants* considerados no cenário B (Tabela 6.4) apresentam requisitos divergentes. Entretanto, a variação dos requisitos é referente as regras de negócio que compõem o processo de empacotamento. Neste caso, o tratamento

da variabilidade foi realizado por meio do padrão Catálogo de *Features* MT, de modo que o catálogo de *features* do *tenant* IBM possui a *feature* "Empacotamento de produtos diferentes", enquanto o *tenant* Flextronics apenas a *feature* "Empacotamentos".

Outros cenários simularam o comportamento dos SaaS para customizações de UI e permissões, utilizando os padrões GUI MT e Entidades MT respectivamente. A análise das simulações evidenciou a viabilidade dos padrões no desenvolvimento de aplicações RFID como serviço. Os padrões utilizados no tratamento das customizações atenderam à variabilidade dos requisitos dos *tenants*, mantendo as vantagens de SaaS para distribuição de software RFID de baixo custo.

6.3 GQM

Um conjunto de avaliações foi aplicado com o intuito de avaliar a proposta deste projeto de mestrado. Essas avaliações foram planejadas com base no método GQM, que propõe a definição dos objetivos, questões e métricas das avaliações (Solingen e Berghout, 1999; Basili *et al.*, 1994). Na aplicação do GQM é gerado um modelo de interpretação das métricas, que permite ao responsável pela avaliação julgar os resultados obtidos e decidir quais ações devem ser tomadas a partir delas. Nesta avaliação, o objetivo foi verificar a capacidade dos padrões propostos (Capítulo 5) e inseridos no FoSaaS, para criar SaaS MT customizáveis com maior qualidade, reúso e produtividade. A seguir são apresentados os objetivos e as questões elaboradas.

Objetivos da avaliação (*Goals*):

A. Verificar o reúso obtido nos SaaS gerados com o auxílio do FoSaaS.
B. Avaliar a produtividade obtida no desenvolvimento de SaaS com o auxílio do FoSaaS.
C. Avaliar a qualidade obtida no desenvolvimento de SaaS com o auxílio do FoSaaS.

Objetivo A: Reúso.

Analisa o projeto de software com ênfase no código-fonte dos softwares desenvolvidos.

Com o propósito de reusabilidade.

Com respeito ao grau de reúso obtido nos SaaS desenvolvidos.

Do ponto de vista dos desenvolvedores de softwares.

No contexto dos SaaS desenvolvidos com os padrões de projetos propostos por este projeto de mestrado.

Objetivo B: Produtividade.

Analisa o processo de desenvolvimento de software.

Com o propósito de avaliação.

Com respeito ao tempo gasto no desenvolvimento dos artefatos de implementação.

Do ponto de vista dos desenvolvedores dos softwares

No contexto dos SaaS desenvolvidos com os padrões de projetos propostos por este projeto de mestrado.

Objetivo C: Qualidade.

Analisa o código-fonte dos SaaS desenvolvidos.

Com o propósito de avaliação.

Com respeito aos defeitos e linhas de código do SaaS desenvolvidos.

Do ponto de vista dos desenvolvedores dos softwares.

No contexto dos SaaS desenvolvidos com os padrões de projetos propostos por este projeto de mestrado.

A partir das questões e objetivos acima descritos, as seguintes métricas foram elaboradas:

Objetivo A:

Q1. Qual o reúso dos artefatos de implementação dos SaaS desenvolvidos com auxílio do FoSaaS?

M1. Número de métodos generalizados entre os *tenants* do SaaS desenvolvido com auxílio do FoSaaS.

M2. Número de métodos generalizados entre os *tenants* do SaaS desenvolvido sem auxílio do FoSaaS.

Q2. Qual é a porcentagem de reúso dos SaaS desenvolvidos com o FoSaaS?

M1. Porcentagem de *features* reutilizadas na implementação com FoSaaS.

M2. Porcentagem de *features* reutilizadas na implementação *ad-hoc* (sem auxílio do FoSaaS).

Objetivo B:

Q1. O tempo gasto no desenvolvimento dos SaaS com auxílio do FoSaaS é maior que o desenvolvimento *ad-hoc*?

M1. Tempo gasto no desenvolvimento de SaaS com auxílio do FoSaaS.

M2. Tempo gasto no desenvolvimento de SaaS sem auxílio do FoSaaS.

Objetivo C:

Q1. Qual é a qualidade dos SaaS gerados com auxílio do FoSaaS?

M1. Número de defeitos dos SaaS gerados com auxílio do FoSaaS.

M2. Número de defeitos dos SaaS gerados no desenvolvimento *ad-hoc*.

M3. Número de linhas de código (KLOC) dos SaaS desenvolvidos com auxílio FoSaaS.

M4. Número de linhas de código (KLOC) dos SaaS desenvolvidos *ad-hoc*.

Para que as métricas sejam avaliadas é necessário conhecer a forma que elas devem ser interpretadas, como apresentado na Tabela 6.5. O modelo de interpretação das métricas é composto pelas conclusões geradas a partir dos valores obtidos pelos experimentos e as ações que devem ser tomadas em caso de resultado negativo. Na primeira coluna da Tabela a sigla XQiMi refere-se: X ao objetivo. Qi à Questão i do objetivo X e Mi à métrica da Questão Qi do objetivo X. Por exemplo: AQ1M1 refere-se ao objetivo Reúso (A), Questão 1 (Qual o reúso dos artefatos de implementação dos SaaS desenvolvidos com auxílio do FoSaaS?) e Métrica 1 (Número de métodos generalizados entre os *tenants* do SaaS desenvolvido com auxílio do FoSaaS).

Tabela 6.5: Interpretação das métricas utilizadas para a avaliação deste projeto de mestrado.

Análise da Métrica	**Conclusão**	**Ação**
AQ1M1 > AQ1M2	O FoSaaS auxília o reúso de artefatos de implementação	Nada a fazer.
AQ1M1 <= AQ1M2	O FoSaaS não fornece grau satisfatório de reúso dos artefatos de implementação.	Melhorar a reusabilidade dos artefatos gerados pelo FoSaaS.
AQ2M1 > AQ2M2	Com auxílio do FoSaaS mais *features* são implementadas com reúso.	Nada a fazer.
AQ2M1 <= AQ2M2	Com auxílio do FoSaaS menos *features* são implementadas com reúso.	Melhorar os recursos do FoSaaS para reúso de *features*.
BQ1M1 < BQ1M2	O FoSaaS provê maior produtividade ao desenvolvimento de SaaS	Nada a fazer.
BQ1M1 >= BQ1M2	O FoSaaS provê menor produtividade ao desenvolvimento de SaaS	Analisar e corrigir possíveis complexidades do FoSaaS.
CQ1M1 < CQ1M2	O FoSaaS provê maior qualidade aos SaaS que o desenvolvimento *ad-hoc*	Nada a fazer.
CQ1M1 >= CQ1M2	O FoSaaS provê menor qualidade aos SaaS que o desenvolvimento *ad-hoc*	Corrigir os defeitos gerados pelo uso do FoSaaS.
CQ1M3 < CQ1M4	O FoSaaS provê menor complexidade aos SaaS	Nada a fazer.
CQ1M3 >= CQ1M4	O FoSaaS provê maior complexidade aos SaaS	Analisar e corrigir possíveis complexidades no FoSaaS e melhorar os mecanismos de reúso.

6.4 Experimentos

Experimentos provêm um método sistemático para controlar a avaliação de técnicas e ferramentas de software (Travassos *et al.*, 2002). Eles devem ser usados para obter evidências de uma pesquisa e testar as convicções em relação à realidade. Porém, seus resultados não devem ser vistos como provas, pois são válidos apenas para as condições nas quais foram analisadas (Basili *et al.*, 1986) (Travassos *et al.*, 2002).

Neste projeto de mestrado foram realizados experimentos para avaliar se os padrões para criação de SaaS MT auxiliam o desenvolvimento, provendo maior qualidade de software, reúso e aumento da produtividade. Esses experimentos foram definidos, planejados, executados e os dados analisados como proposto por Wholin et al (2000).

6.4.1 Experimento 1

O experimento foi realizado com alunos de pós-graduação da Universidade Federal de São Carlos (UFSCar). Primeiramente os participantes foram separados em dois grupos para desenvolverem duas aplicações. A primeira aplicação constou do desenvolvimento de um SaaS com auxílio do FoSaaS. Na segunda, constou do desenvolvimento de um SaaS similar, porém de maneira *ad-hoc*, ou seja sem o auxílio do FoSaaS ou outro *framework* similar. Para o desenvolvimento *ad-hoc* foram fornecidas todas as diretrizes do desenvolvimento, exceto os padrões para manipulação das variabilidades.

6.4.1.1 Planejamentos do Experimento 1

O experimento foi realizado no segundo semestre de 2013, com dez inscritos na disciplina Tópico em Engenharia de Software: Reúso de Software. O local do experimento foi um laboratório de computação do Departamento de Computação.

6.4.1.2 Hipóteses do Experimento 1

As hipóteses elaboradas referem-se ao reúso (Hr), qualidade (Hq) e produtividade (He), conforme apresentado na Tabela 6.6.

Tabela 6.6: Hipóteses do Experimento

Hipótese Hr	Os padrões inseridos no FoSaaS, para prover a customização em SaaS MT, aumentam o reúso em comparação ao desenvolvimento *ad-hoc*.
Hipótese Hq	Os padrões inseridos no FoSaaS, para prover a customização em SaaS MT, proveem mais qualidade ao software, em comparação ao desenvolvimento *ad-hoc*.
Hipótese He	Os padrões inseridos no FoSaaS, para prover a customização em SaaS MT, aumentam a produtividade em comparação ao desenvolvimento *ad-hoc*.

Os resultados obtidos pelos estudos da aplicabilidade (Seção 6.2) indicam aumento de produtividade, reúso e qualidade de software, contudo a avaliação desses resultados será realizada por meio deste experimento.

Os modos de desenvolvimento (com auxílio do FoSaaS e *ad-hoc*) podem ser considerados variáveis independentes, pois são manipuladas e controladas durante todo experimento. As variáveis dependentes, ou seja, as que estão sob análise são: reúso, qualidade e produtividade. Elas serão medidas utilizando as métricas apresentadas na Seção 6.3 - GQM.

6.4.1.3 Separação dos Grupos

Os participantes do experimento foram selecionados por meio de uma amostragem não-probabilística. Eles foram divididos em dois grupos e essa divisão ocorreu após responderem a um questionário sobre questões técnicas. A denominação usada para os grupos é a seguinte:

- G1: Grupo 1, formado por 5 alunos, participantes, de P1 à P5;
- G2: Grupo 2, formado por 5 alunos, participantes, de P6 à P10;

A distribuição dos grupos foi realizada com base nas respostas dos participantes ao questionário de conhecimento sobre conceitos teóricos envolvidos no experimento. O questionário foi realizado após o treinamento dos participantes e as questões foram auto-avaliativas sobre linguagens de programação, *frameworks* e a dificuldade encontrada na realização do treinamento. Os participantes responderam vinte e cinco questões sendo duas dissertativas e vinte e três alternativas, identificadas por um valor de 0 à 4 (0 – nenhum, 1 – pouco, 2 – bom, 3- alto, 4 – muito alto). Detalhes do questionário estão descritos no Apêndice A.

Os grupos foram organizados de modo que a média de conhecimento dos participantes fossem semelhantes. O gráfico da Figura 6.1 exibe a ordem e o grau de conhecimento de cada participante. O grupo 1 (G1) é formado pelos participantes P1 à P5, com média total de conhecimento igual a 10,8 enquanto o grupo 2 (G2), formado pelos participantes P6 à P10 possui a média de conhecimento igual a 10,6.

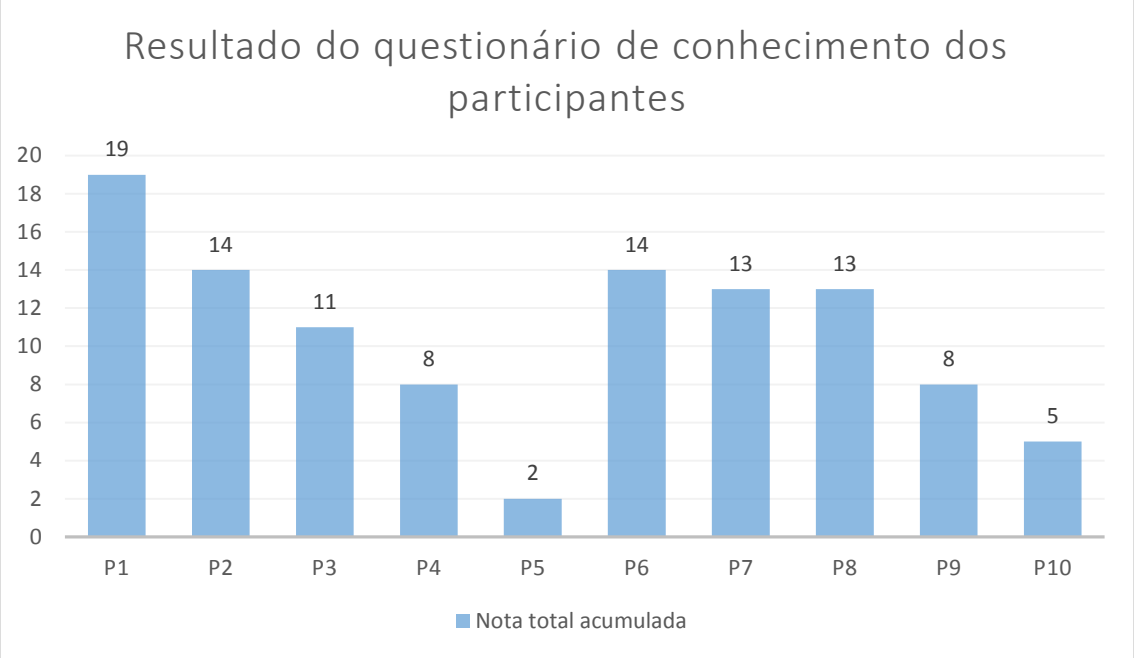


Figura 6.1: Conhecimento dos participantes do experimento 1.

6.4.1.4 Modelos e Instrumentação

Os documentos utilizados no experimento foram: formulário para execução e para especificação das aplicações a serem desenvolvidas (Apêndice B). O formulário de execução foi usado para medição do tempo e foi preenchido por cada participante segundo a etapa que realizava. Os participantes deveriam preencher seu nome, a identificação do experimento, o modo desenvolvido (FoSaaS x *ad-hoc*), considerações sobre fatos que ocorressem durante a execução e o horário de início e fim. Na primeira parte do experimento, os participantes de ambos os grupos desenvolveram um SaaS para gestão de compras e aluguéis de veículos e na segunda parte, um SaaS para gerenciamento de aluguéis e reformas de casas. Nas Figuras 6.2 e 6.3 estão ilustrados os diagramas de classes fornecidos aos participantes para a elaboração dos SaaS solicitados no experimento.

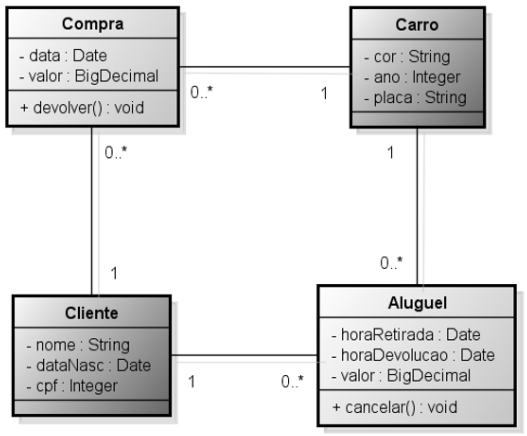


Figura 6.2: Diagrama de classe do SaaS para compra e aluguel de veículos.

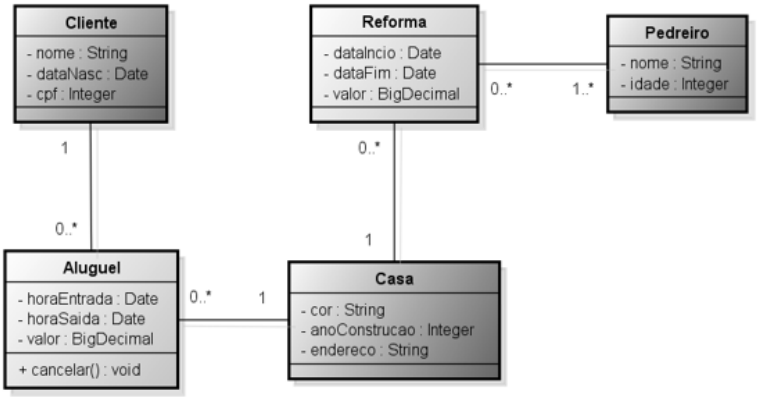


Figura 6.3: Diagrama de classe do SaaS para aluguel e reforma de casas.

Para avaliar a capacidade dos SaaS em manipular requisitos divergentes dos *tenants*, foi exigido aos participantes adicionarem dois *tenants* as aplicações. No primeiro SaaS, um *tenant* deveria ser capaz de alugar e comprar carros, enquanto o outro apenas alugar carros. Para o SaaS desenvolvido na segunda etapa do experimento, o primeiro *tenant* poderia alugar e reformar casas, enquanto o segundo apenas alugar casas.

6.4.1.5 Execução do Experimento 1

O experimento foi dividido igualmente em 3 fases: treinamento, piloto e execução. Na Tabela 6.7 estão descritas as fases do experimento.

Tabela 6.7: Fases do Experimento

Fases	**SaaS**	***Ad-hoc***	**FoSaaS**
Treinamento	Venda de produtos	Todos os participantes	Todos os participantes
Piloto	Hotel	Todos os participantes	Todos os participantes
Execução (Etapa 1)	A1- Aluguéis e vendas de carros	A1 – G1	A1 – G2
Execução (Etapa 2)	A2- Aluguéis e reformas de casas	A2 – G2	A2 – G1

Na primeira fase, os participantes receberam um treinamento de oito horas, com o objetivo de deixar todos com o mesmo conhecimento em programação web, Groovy & Grails, LPS e SaaS. Na fase piloto, os participantes desenvolveram um SaaS para gerenciamento de quartos de um hotel, simulando a execução do experimento e tiveram oportunidade de sanar eventuais dúvidas. Na fase de execução do experimento, os participantes foram separados em grupos, como apresentado na Seção 6.4.1.3. Na primeira etapa, os participantes do grupo G1 desenvolveram o SaaS de gestão de carros, com auxílio do FoSaaS, enquanto os participantes do grupo G2 desenvolveram o mesmo SaaS, porém de modo *ad-hoc*. Na segunda etapa, os participantes desenvolveram o SaaS para gestão de aluguéis e reformas de casas. Contudo, o modo de desenvolvimento foi invertido, os participantes do grupo G1 desenvolveram o SaaS no modo *ad-hoc* e os participantes do grupo G2 com auxílio do FoSaaS.

6.4.1.6 Dados Coletados do Experimento 1

Os dados do experimento foram coletados após sua execução e usados para responder às métricas do GQM. Os dados foram extraídos dos códigos-fontes gerados pelos participantes. O número de defeitos e KLOC foram obtidos com auxílio de ferramentas que automatizaram o processo, já as demais métricas foram calculadas manualmente por revisão manual de código. A ferramenta utilizada para contagem de linhas do código-fonte foi o software livre: CLOC (Sourceforge, 2014). Linhas em branco e comentários foram excluídos dessa contagem. O software livre Codenarc (Sourceforge, 2014) foi usado para análise estática do código-fonte. Essa ferramenta detecta

inconsistências, práticas ruins e problemas na sintaxe. O relatório gerado pela ferramenta classifica os defeitos com prioridade alta, média e baixa. A análise dinâmica dos defeitos, os seja, os defeitos gerados na execução dos SaaS foram capturados por meio de testes unitários e de integração com auxílio da ferramenta Spock, um *framework* de especificação de testes *open-source*. Os defeitos foram igualmente classificados: prioridade alta, média e baixa. Na Tabela 6.8 são apresentados os dados capturados pelo experimento com desenvolvimento *ad-hoc* e na Tabela 6.9, os dados do desenvolvimento com auxílio do FoSaaS.

Tabela 6.8: Dados do experimento *ad-hoc*

	P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**
KLOC	22611	21327	22721	22903	21920	22581	22610	21542	22609	22587
Tempo execução	0:42	0:31	0:41	0:44	0:41	0:32	0:41	0:50	0:42	0:40
N° defeitos prioridade alta	0	1	0	0	0	0	0	0	0	0
N° defeitos prioridade média	19	39	12	33	30	26	23	34	21	25
N° defeitos prioridade baixa	1	0	0	2	0	1	0	0	0	0
% *features* reusadas	40%	40%	20%	0%	0%	20%	20%	0%	0%	0%
N° métodos generalizados	3	2	2	0	0	2	0	0	0	0

Tabela 6.9: Dados do experimento com o FoSaaS

	P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**
KLOC	22380	21392	21920	22082	21324	22354	22374	21411	22380	22101
Tempo execução	0:36	0:28	0:35	0:37	0:42	0:29	0:27	0:30	0:32	0:42
N° defeitos prioridade alta	0	2	0	0	0	0	0	0	0	0
N° defeitos prioridade média	13	12	8	21	16	23	26	10	19	18
N° defeitos prioridade baixa	0	1	1	1	0	0	0	0	0	0
% *features* reusadas	100%	0%	80%	80%	80%	80%	100%	100%	80%	80%
N° métodos generalizados	20	0	20	20	20	20	20	20	20	20

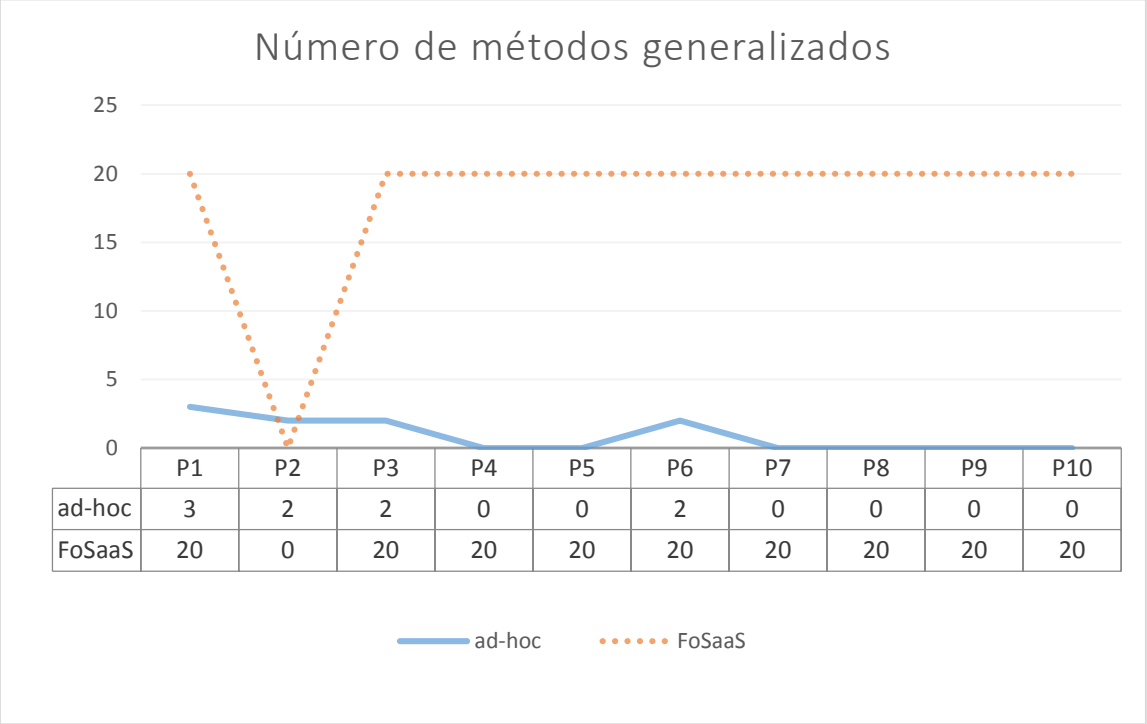
6.4.1.7 Análise dos Dados do Experimento 1

Os resultados das métricas propostas na Seção 6.3 são ilustrados de forma gráfica nas Figuras 6.4 à 6.8. Os dados obtidos do participante P2 foram excluídos, pois são considerados um *outlier*, devido a problemas encontrados na execução.

Na Figura 6.4, as linhas pontilhadas e contínuas representam as métricas AQ1M1 e AQ1M2, respectivamente, e são usadas para responder a questão Q1 do objetivo reúso: Qual o reúso dos artefatos de implementação dos SaaS desenvolvidos com auxílio do FoSaaS? A interpretação das métricas diz:

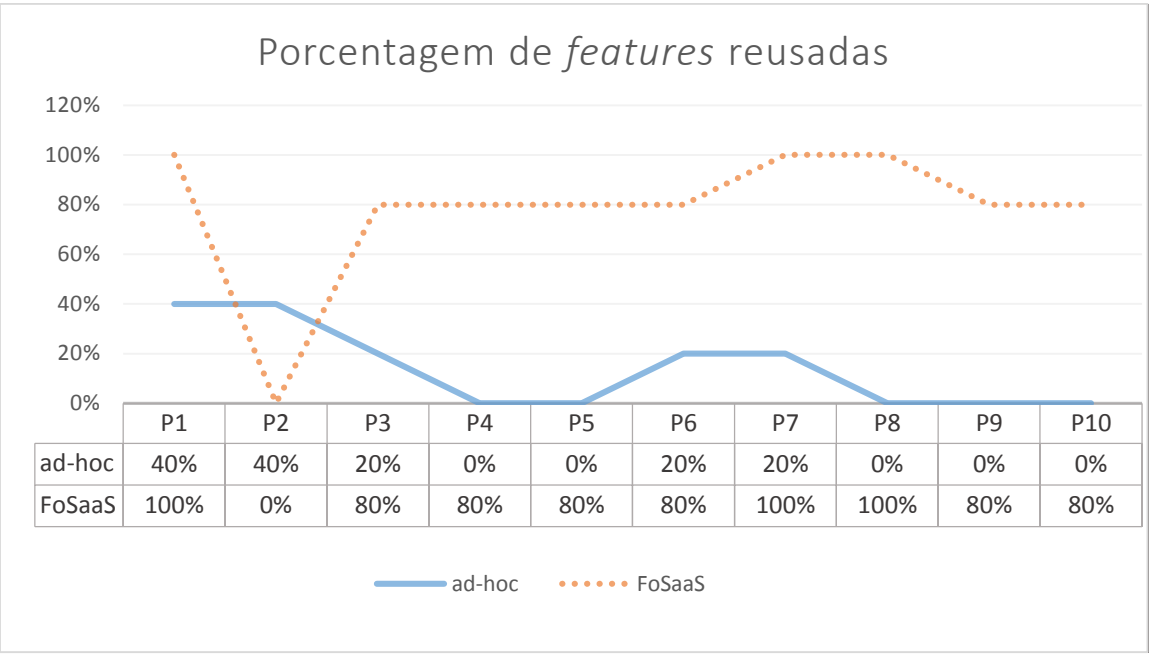
Para todo AQ1M1 é maior que AQ1M2, exceto para P2.

A relação não foi respeitada em P2, porém, pode ter sido causada por problemas técnicos que impediram a conclusão do experimento pelo participante. Com auxílio do FoSaaS os participantes obtiveram em média 96% a mais de métodos reusados.



	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
ad-hoc	3	2	2	0	0	2	0	0	0	0
FoSaaS	20	0	20	20	20	20	20	20	20	20

Figura 6.4: Número de métodos utilizados em mais de um *tenant* do SaaS.



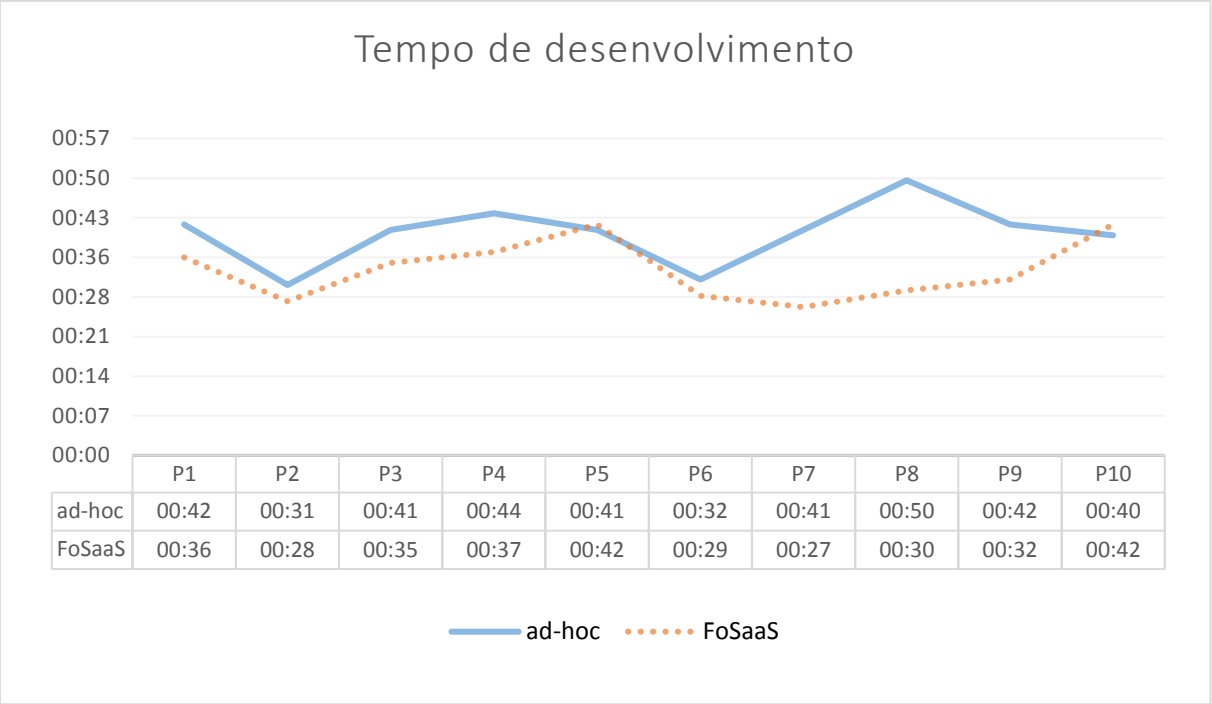
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
ad-hoc	40%	40%	20%	0%	0%	20%	20%	0%	0%	0%
FoSaaS	100%	0%	80%	80%	80%	80%	100%	100%	80%	80%

Figura 6.5: Porcentagem de features reusadas entre os tenants dos SaaS.

Na Figura 6.5 as linhas pontilhadas e contínuas representam as métricas AQ2M1 e AQ2M2, respectivamente, e são usadas para responder a questão Q2 do objetivo reúso: Qual é a porcentagem de reúso dos SaaS desenvolvidos com o FoSaaS? A interpretação das métricas diz:

Para todo AQ2M1 é maior que AQ2M2, exceto para P2.

A relação não foi respeitada em P2, porém, pode ter sido causada por problemas técnicos que impediram a conclusão do experimento pelo participante. Com auxílio do FoSaaS os participantes obtiveram em média 75,5% a mais de *feature* reusadas.



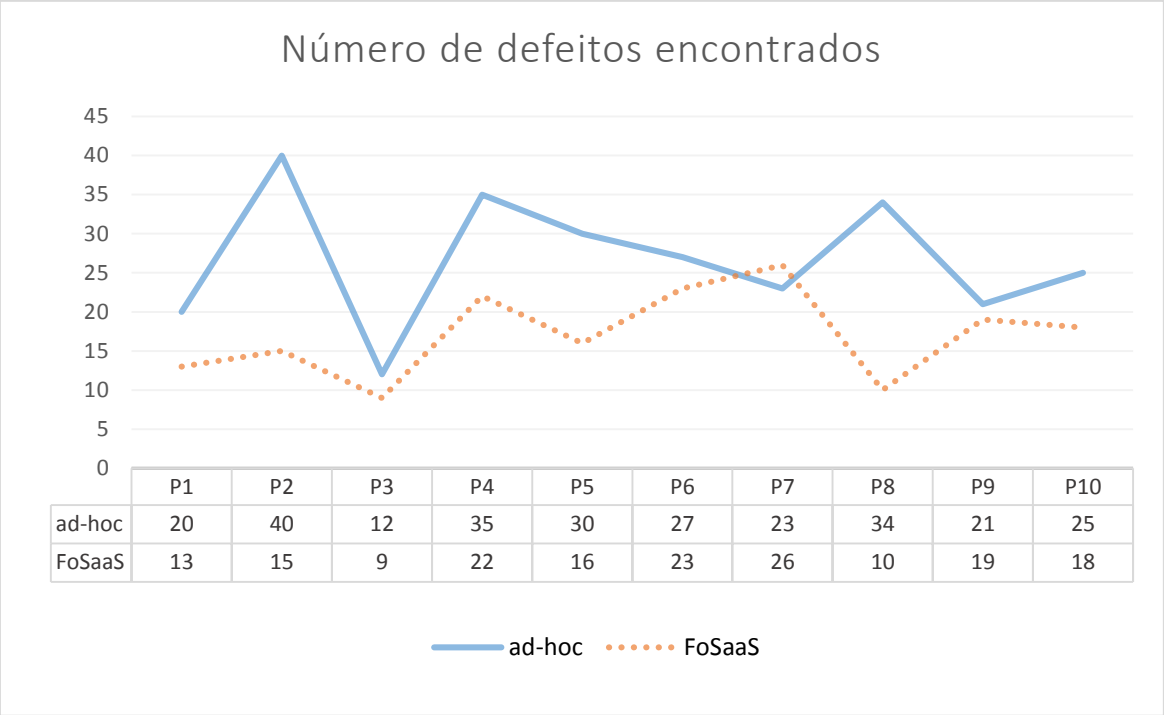
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
ad-hoc	00:42	00:31	00:41	00:44	00:41	00:32	00:41	00:50	00:42	00:40
FoSaaS	00:36	00:28	00:35	00:37	00:42	00:29	00:27	00:30	00:32	00:42

Figura 6.6: Tempo gasto no desenvolvimento dos SaaS.

Na Figura 6.6, as linhas pontilhadas e contínuas representam as métricas BQ1M1 e BQ1M2, respectivamente, e são usadas para responder a questão Q1 do objetivo produtividade: O tempo gasto no desenvolvimento dos SaaS com auxílio do FoSaaS é maior que o desenvolvimento *ad-hoc*? A interpretação das métricas diz:

Para todo BQ1M1 é menor que BQ1M2, exceto para P5 e P10.

A relação não foi respeitada em P5 e P10, porém com pouco diferença, que pode estar associado a conhecimento pré-existente dos participantes em desenvolvido de software. Em comum, esses participantes apresentam a pior média de conhecimento, como ilustrado na Figura 6.1 deste Capítulo. A proximidade dos tempos provavelmente foi influenciada pela duração do exercício que tinha o limite de uma hora. Com auxílio do FoSaaS, os participantes reduziram em média 5,5 minutos do tempo de execução para o contexto deste experimento.



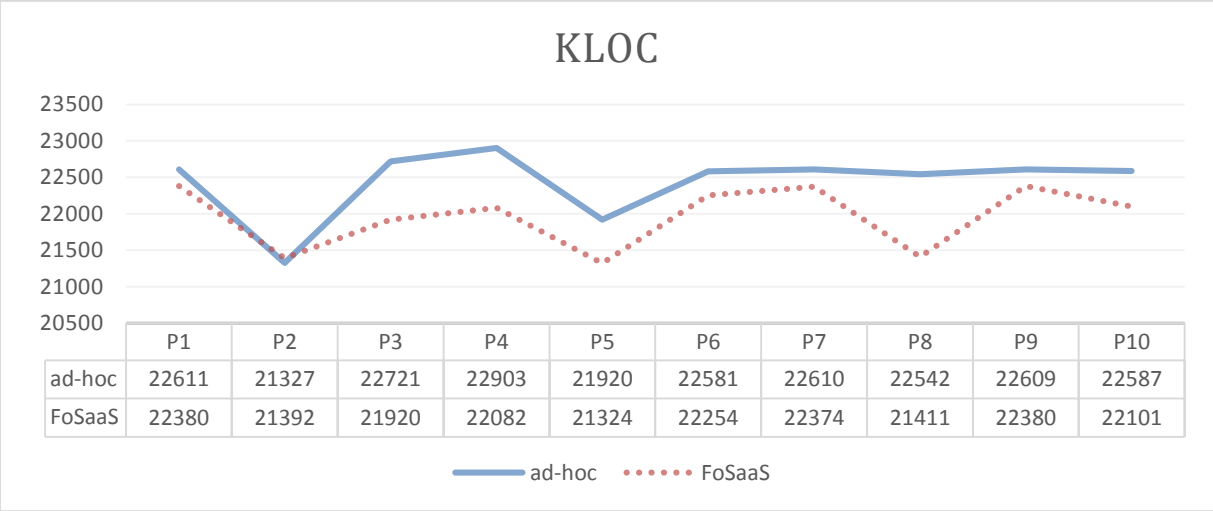
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
ad-hoc	20	40	12	35	30	27	23	34	21	25
FoSaaS	13	15	9	22	16	23	26	10	19	18

Figura 6.7: Número de defeitos encontrados nos SaaS.

Na Figura 6.7, as linhas pontilhadas e contínuas representam as métricas CQ1M1 e CQ1M2, respectivamente, e são usadas para responder a questão Q1 do objetivo qualidade: Qual é a qualidade dos SaaS gerados com auxílio do FoSaaS? A interpretação das métricas diz:

Para todo CQ1M1 é menor que CQ1M2, exceto para P7.

Com auxílio do FoSaaS os participantes reduziram em média 9,5 defeitos de código para o contexto desse experimento. A redução dos defeitos foi baixa, pois em média o próprio FoSaaS gerou 8,2 defeitos.



	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
ad-hoc	22611	21327	22721	22903	21920	22581	22610	22542	22609	22587
FoSaaS	22380	21392	21920	22082	21324	22254	22374	21411	22380	22101

Figura 6.8: Número de linhas de código.

Na Figura 6.8, as linhas pontilhadas e contínuas representam as métricas CQ1M3 e CQ1M4, respectivamente, e essas são usadas para responder a questão Q1 do objetivo qualidade. A interpretação das métricas diz:

Para todo CQ1M3 é menor que CQ1M4, exceto para P2.

A relação não foi respeitada em P2, porém, pode ter sido causada por problemas técnicos que impediram a conclusão do experimento pelo participante. Com auxílio do FoSaaS os participantes reduziram em média 539,7 linhas de código para o contexto do experimento.

6.4.1.8 Ameaça a validade do Experimento 1

Algumas ameaças à validade podem ser consideradas:

- **Participantes com pouca experiência em desenvolvimento de software:** os resultados obtidos podem ter sido influenciados pela falta de experiência dos alunos da pós-graduação em desenvolvimento de software. Essa ameaça foi contornada por meio do treinamento que buscou nivelar o conhecimento dos participantes do experimento.
- **Poucos participantes**: a amostra de participantes que realizaram o experimento foi pequena para que os resultados sejam considerados válidos.

- **Diferença de conhecimento entre os participantes:** a diferença de conhecimento entre os participantes do experimento podem ter gerado dados discrepantes. Esse ameaça foi contornada pela divisão dos grupos por níveis de conhecimento. Dessa forma, os grupos foram separados na tentativa de equilibrar o conhecimento.

6.4.2 Experimento 2

O segundo experimento foi realizado no segundo semestre de 2013 com profissionais do FIT - Instituto de Tecnologia, onde o autor deste projeto é colaborador. O objetivo do experimento é semelhante ao objetivo do experimento 1. Contudo esses participantes possuem mais experiência prática em desenvolvimento de software do que os alunos.

O objetivo deste experimento foi para avaliar se o nível de conhecimento dos participantes influencia no resultado obtido pelo experimento 1, realizado com estudantes de pós-graduação.

A condução do experimento, hipóteses e planejamento e documentos utilizados são iguais aos do experimento 1.

6.4.2.1 Separação dos Grupos

Os participantes do experimento 2 foram selecionados por meio de uma amostragem não-probabilística. Eles foram divididos em dois grupos e essa divisão ocorreu após os participantes responderem a um questionário no qual questões técnicas foram formuladas. A denominação usada para os grupos é a seguinte:

- G1: Grupo 1, formado por 3 alunos, participantes, de P1 à P3;
- G2: Grupo 2, formado por 3 alunos, participantes, de P4 à P6;

A distribuição dos grupos foi realizada com base no questionário de conhecimento, respondido pelos participantes antes da realização do experimento. Os participantes deveriam responder questões auto-avaliativas sobre seus conhecimentos nos conceitos teóricos associados a proposta, linguagens de programação, *frameworks* e a dificuldade encontrada na realização do treinamento.

Os grupos foram divididos de acordo com as respostas dos participantes às perguntas do questionário, visando o balanceamento do conhecimento entre os grupos. Na Figura 6.9 são apresentados a ordem e o grau de conhecimento de cada participante desse experimento.

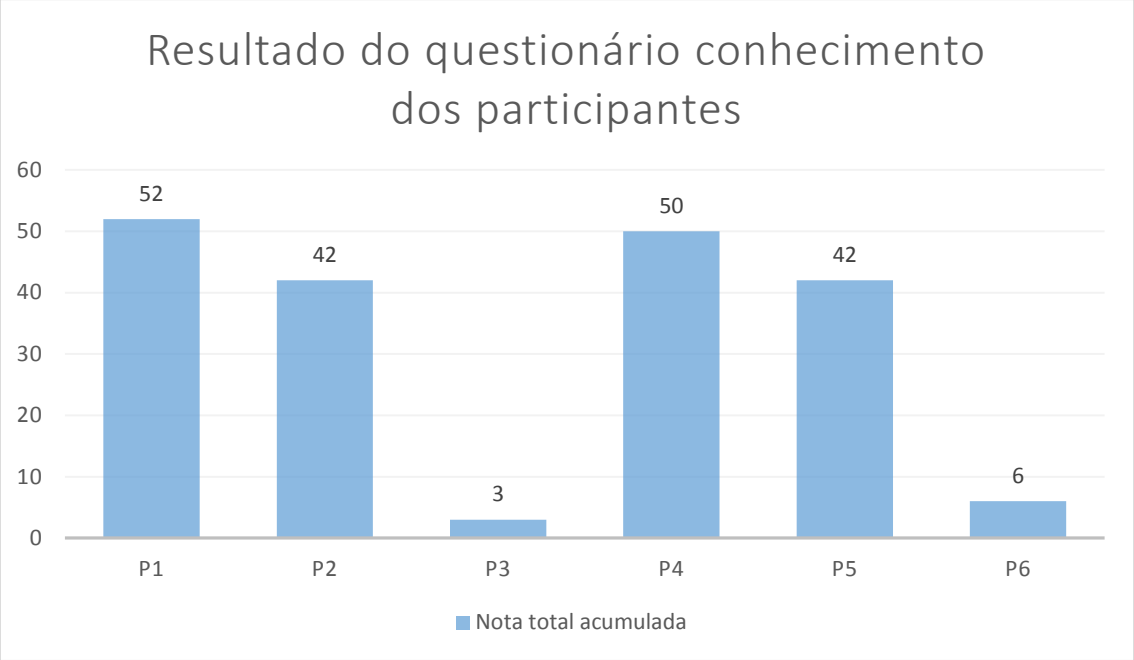


Figura 6.9: Conhecimentos dos participantes.

A separação dos grupos dos profissionais do FIT foi análoga a dos participantes de pós-graduação do experimento 1. O grupo 1 (G1), formado pelos participantes P1 à P3, tem média de conhecimento igual a 32,34 enquanto o grupo 2 (G2), com participantes de P4 à P6, tem média igual a 32,66.

6.4.2.2 Execução do Experimento 2

O experimento foi dividido igualmente em 3 fases: treinamento, piloto e execução. Na tabela 6.10 são apresentadas as descrições das fases.

Tabela 6.10: Fases do Experimento

Fases	**SaaS**	***Ad-hoc***	**FoSaaS**
Treinamento	Venda de produtos	Todos os participantes	Todos os participantes
Piloto	Hotel	Todos os participantes	Todos os participantes
Execução (Etapa 1)	A1- Aluguéis e vendas de carros	A1 – G1	A1 – G2
Execução (Etapa 2)	A2- Aluguéis e reformas de casas	A2 – G2	A2 – G1

Na primeira fase, os participantes receberam um treinamento de quatro horas, visando equilibrar os conhecimentos em programação web, Groovy & Grails, LPS e SaaS. O tempo de treinamento foi inferior ao do experimento 1 devido ao conhecimento pré-existente dos participantes deste experimento. A fase de piloto e execução foi semelhante ao experimento 1.

6.4.2.3 Dados Coletados do Experimento 2

Na Tabela 6.11 são apresentados os dados coletados após a execução do experimento com desenvolvimento *ad-hoc* e na Tabela 6.12, os dados do desenvolvimento com auxílio do FoSaaS. Esses dados foram extraídos a partir dos códigos-fontes gerados pelos participantes, semelhante ao experimento 1.

Tabela 6.11: Dados do experimento *ad-hoc*

	P1	**P2**	**P3**	**P4**	**P5**	**P6**
KLOC	22812	22190	23043	22002	22375	22329
Tempo execução	0:39	0:41	0:58	0:47	0:43	0:39
N° defeitos prioridade alta	0	0	0	0	0	0
N° defeitos prioridade média	18	17	29	17	19	23
N° defeitos prioridade baixa	0	0	2	0	0	0
% *features* reusadas	50%	20%	0%	20%	20%	20%
N° métodos generalizados	5	2	0	3	1	1

Tabela 6.12: Dados do experimento com o FoSaaS

	P1	**P2**	**P3**	**P4**	**P5**	**P6**
KLOC	20915	21219	21820	21828	22199	22096
Tempo execução	0:26	0:31	0:45	0:25	0:27	0:35
N° defeitos prioridade alta	0	0	0	0	0	0
N° defeitos prioridade média	9	11	21	8	9	12
N° defeitos prioridade baixa	0	0	0	0	1	0
% *features* reusadas	100%	100%	100%	100%	100%	100%
N° métodos generalizados	20	20	20	20	20	20

6.4.2.4 Análise dos Dados do Experimento 2

Os resultados das métricas propostas na Seção 6.3 são ilustrados por meio dos gráficos exibidos nas Figuras 6.10 à 6.14. Na Figura 6.10, as linhas pontilhadas e contínuas representam as métricas AQ1M1 e AQ1M2 respectivamente e são usadas para responder a questão Q1 do objetivo reúso: Qual o reúso dos artefatos de implementação dos SaaS desenvolvidos com auxílio do FoSaaS? A interpretação das métricas diz:

Para todo AQ1M1 é maior que AQ1M2.

Com auxílio do FoSaaS os participantes obtiveram em média 90% a mais de métodos reusados. O participante P3, que apresentou baixo nível de conhecimento e sem auxílio do FoSaaS não foi capaz de reusar nenhum método. Contudo, com auxílio do FoSaaS reusou 100% dos métodos. Essa mesma associação pode ser feita com os participantes P2, P5 e P6. Ambos possuem menor conhecimento e não foram capazes de reusar tantos métodos sem auxílio do FoSaaS. Isso indica que o FoSaaS auxilia o reúso, mesmo para desenvolvedores com pouca experiência.

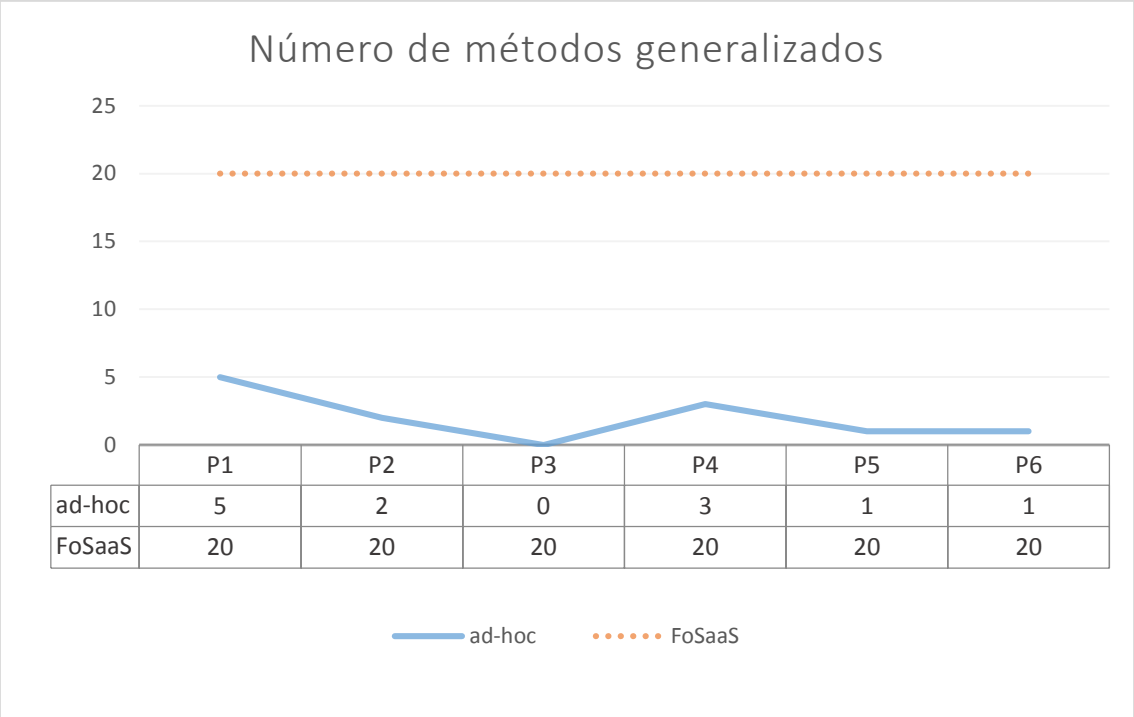


Figura 6.10: Número de métodos utilizados em mais de um tenant do SaaS.

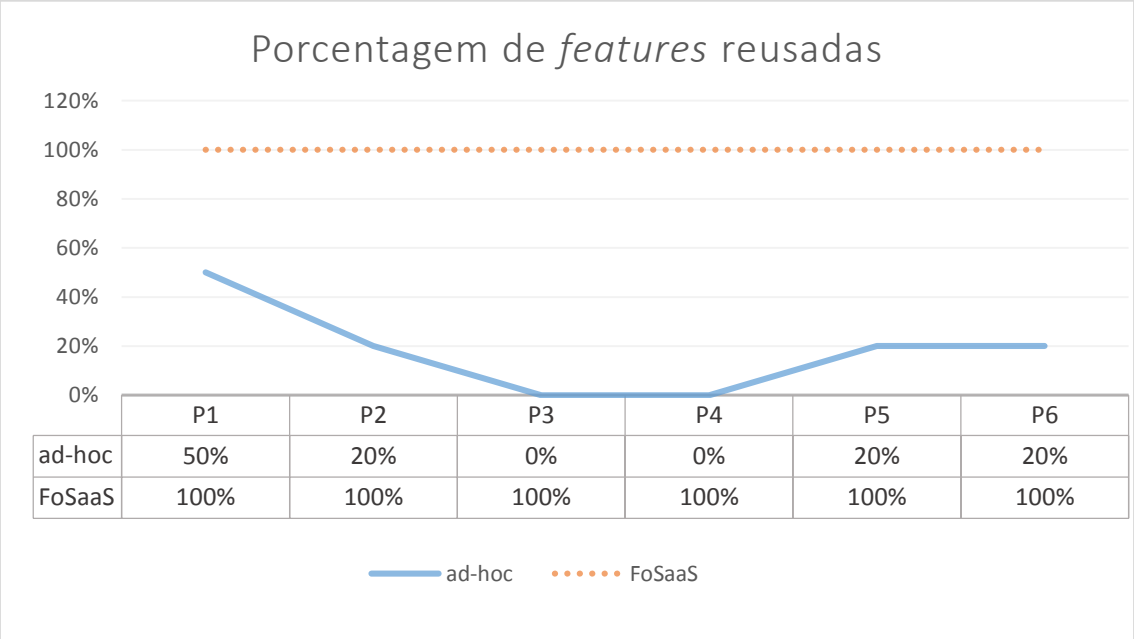
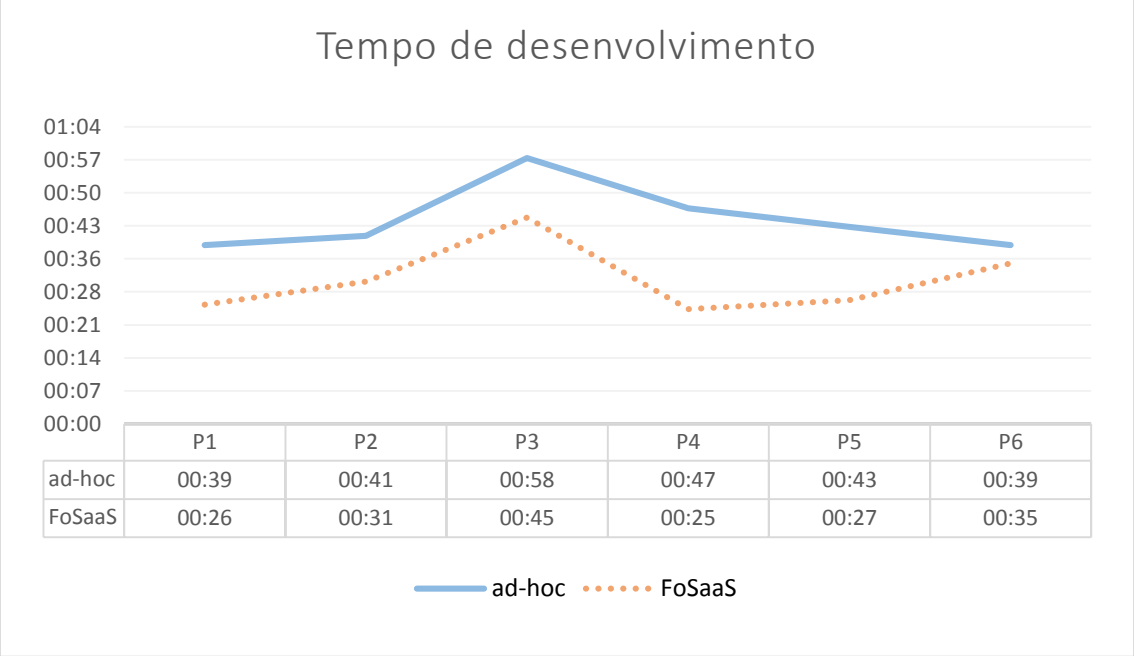


Figura 6.11: Número de métodos utilizados em mais de um *tenant* do SaaS.

Na Figura 6.11 as linhas pontilhadas e contínuas representam as métricas AQ2M1 e AQ2M2 respectivamente e são usadas para responder a questão Q2 do objetivo reúso: Qual é a porcentagem de reúso dos SaaS desenvolvidos com o FoSaaS? A interpretação das métricas diz:

Para todo AQ2M1 é maior que AQ2M2.

Com auxílio do FoSaaS os participantes obtiveram em média 78,3% a mais de *features* reusadas.



	P1	P2	P3	P4	P5	P6
ad-hoc	00:39	00:41	00:58	00:47	00:43	00:39
FoSaaS	00:26	00:31	00:45	00:25	00:27	00:35

Figura 6.12: Tempo gasto no desenvolvimento dos SaaS.

Na Figura 6.12 as linhas pontilhadas e contínuas representam as métricas BQ1M1 e BQ1M2, respectivamente, e são usadas para responder a questão Q1 do objetivo produtividade: O tempo gasto no desenvolvimento dos SaaS com auxílio do FoSaaS é menor que o desenvolvimento *ad-hoc*? A interpretação das métricas diz:

Para todo BQ1M1 é menor que BQ1M2.

Com auxílio do FoSaaS, os participantes reduziram em média 11,5 minutos do tempo de execução para o contexto deste experimento. A proximidade dos tempos provavelmente foi influenciada pela duração do exercício que tinha o limite de uma hora. Os tempos obtidos pelos participantes P3 e P6 podem ter sido influenciados pelo baixo conhecimento, uma vez que foram classificados como com pior conhecimento de acordo com o questionário apresentado na Seção 6.4.1.3.

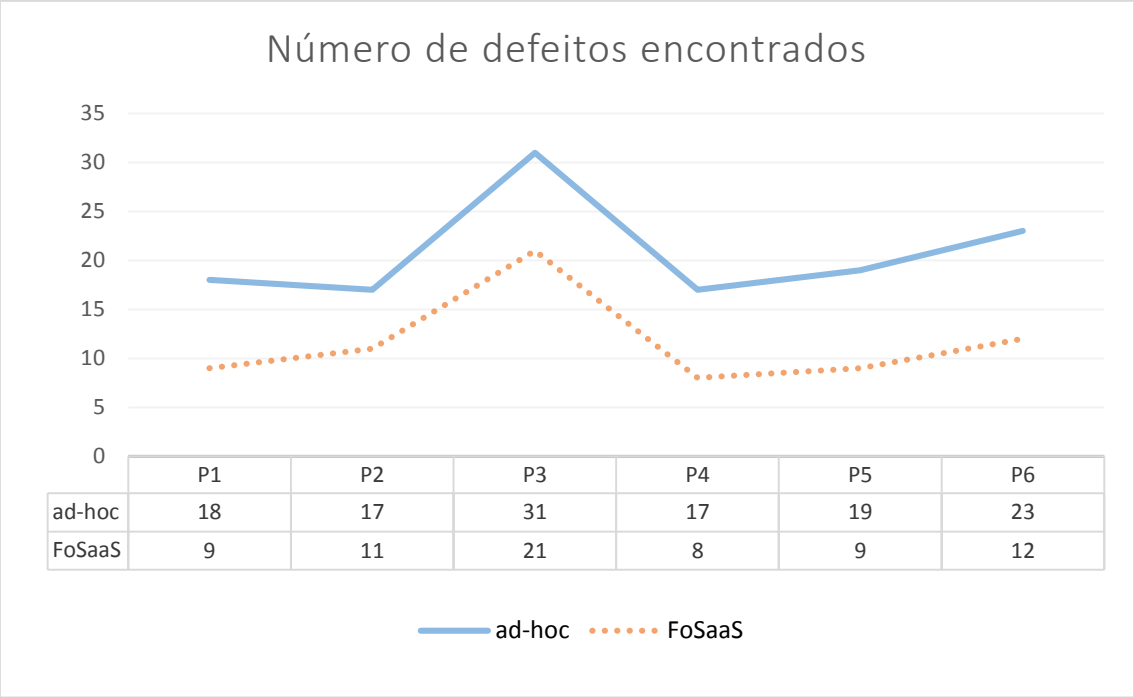


Figura 6.13: Número de defeitos encontrados nos SaaS.

Na Figura 6.13, as linhas pontilhadas e contínuas representam as métricas CQ1M1 e CQ1M2 respectivamente e são usadas para responder a questão Q1 do objetivo qualidade: Qual é a qualidade dos SaaS gerados com auxílio do FoSaaS? A interpretação das métricas diz:

Para todo CQ1M1 é menor que CQ1M2

Com auxílio do FoSaaS os participantes reduziram em média 9,17 defeitos de código para o contexto desse experimento.

Na Figura 6.14, as linhas pontilhadas e contínuas representam as métricas CQ1M3 e CQ1M4 respectivamente e essas são usadas para responder a questão Q1 do objetivo qualidade. A interpretação das métricas diz:

Para todo CQ1M3 é menor que CQ1M4

Com auxílio do FoSaaS os participantes reduziram em média 770 linhas de código para o contexto do experimento.

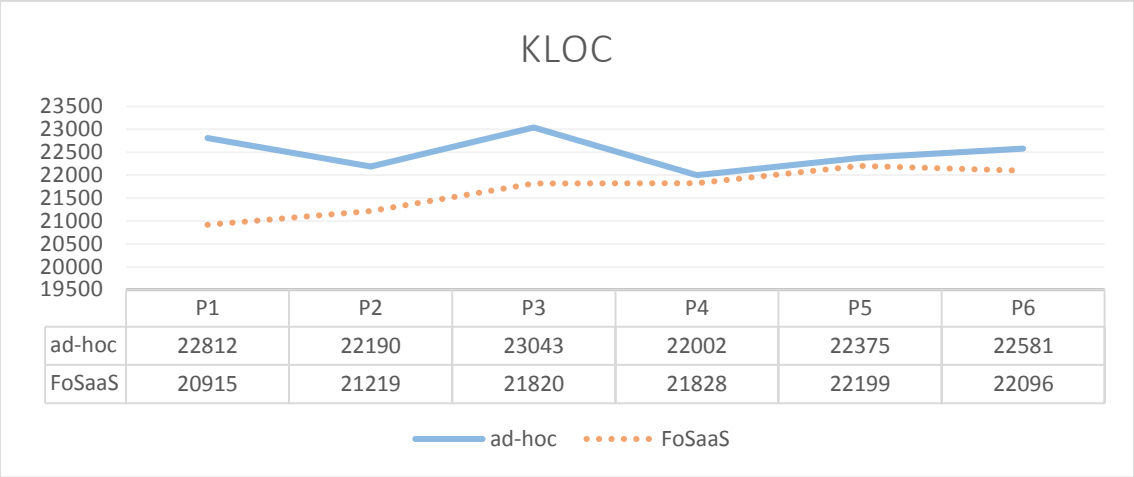


Figura 6.14: Número de linhas de código.

6.4.2.5 Ameaça a validade do Experimento 2

Algumas ameaças à validade podem ser consideradas:

- **Poucos participantes**: a amostra de participantes que realizaram o experimento foi pequena para que os resultados sejam considerados válidos.
- **Diferença de conhecimento entre os participantes:** a diferença de conhecimento entre os participantes do experimento podem ter gerado dados discrepantes. Esse ameaça foi contornada pela divisão dos grupos por níveis de conhecimento. Dessa forma, os grupos foram separados na tentativa de equilibrar o conhecimento.

6.5 Considerações Finais

Neste capítulo foram apresentados os experimentos realizados para avaliar os padrões propostos e o FoSaaS, desenvolvidos por este mestrado. Para complementar a avaliação foi realizado um estudo da aplicabilidade da proposta em projetos com requisitos reais, que evidenciou a efetividade dos padrões no desenvolvimento de SaaS MT customizáveis. Os experimentos realizados para avaliação foram previamente planejados com base nos objetivos pretendidos e o método GQM foi usado para orientar a execução.

Os resultados dos experimentos evidenciaram as hipóteses: aumento do reúso (Hr), qualidade (Hq) e produtividade (He) quando o desenvolvimento é apoiado pelo FoSaaS. Com o auxílio do *framework*, os participantes realizaram as atividades de forma eficiente, com maior reúso de métodos e menor inserção de falhas. Contudo, para que essas hipóteses sejam comprovadas, é necessário que os experimentos sejam replicados em populações maiores.

Capítulo 7

CONSIDERAÇÕES FINAIS

Software como um Serviço (SaaS) é uma forma de distribuição de software com enfoque no baixo custo de implantação, manutenção mais eficiente e distribuição em larga escala (Nitu, 2009). A arquitetura Multi-Tenant (MT) permite o compartilhamento da instância do SaaS por mais de um *tenant* simultaneamente. Contudo, existem problemas associados a arquitetura MT, causados pela variação dos requisitos funcionais e não-funcionais dos *tenants*, que dificultam a adoção dessa arquitetura.

O objetivo deste projeto de mestrado foi criar soluções para resolver o problema de customização dos *tenants* em SaaS MT. Dessa forma, foram propostos padrões para o desenvolvimento de SaaS, de qualidade e orientado ao reúso, visando a resolver o problema de customização dos *tenants* por meio do tratamento de variabilidades. O catálogo de padrões foi criado para manipular os requisitos específicos dos *tenants* em termos de interfaces com usuários, dados, processos e permissões. Com base nas técnicas de Linha de Produtos de Software (LPS) utilizadas na geração de aplicações, os padrões podem gerenciar as variabilidades dos *tenants* em tempo de execução, de acordo com o modelo *features* do SaaS. Além dos padrões, neste projeto também foi construído o *Framework of Software as Service* (FoSaaS), um artefato de software que automatiza a incorporação dos padrões ao SaaS e apoia o desenvolvimento por meio de geração automática de código. O objetivo do FoSaaS é reduzir a complexidade de SaaS MT e aumentar a qualidade do software e produtividade do desenvolvimento. Neste capítulo

são apresentadas as comparações dos padrões desenvolvidos para SaaS com os existentes na literatura, as contribuições deste projeto, as limitações e os trabalhos futuros.

O capítulo está organização em quatro seções. Na Seção 7.1 são descritas as contribuições geradas por este projeto. Na Seção 7.2 são expostas as limitações deste projeto e na Seção 7.3 são discutidos alguns dos trabalhos futuros que podem ser realizados para aumentar a abrangência e sanar algumas limitações.

7.1 Contribuições

Algumas contribuições alcançadas neste projeto são as listadas a seguir de acordo com uma ordem de relevância considerada por este autor:

- **Catálogo de padrões para desenvolvimento de SaaS MT customizáveis**: os padrões foram desenvolvidos para tornar a criação de SaaS MT mais eficiente, de forma sistematizada e com qualidade. Para facilitar o uso desses padrões, uma descrição formal foi apresentada, considerando todos os elementos do modelo de descrição de padrões apresentada por alguns autores (Zamani e Butler, 2009; Gamma *et al.*, 1995).
- **O *Framework of Software as a Services* (FoSaaS)**: foi desenvolvido para apoiar a incorporação dos padrões propostos no desenvolvimento de SaaS MT e auxiliar o desenvolvedor no reúso de software, com base em geração de código-fonte, que aumenta a produtividade e reduz a inserção de falhas. Visando a apoiar desenvolvedores e pesquisadores a replicarem o *framework* em seus trabalhos, este projeto descreveu o processo de criação do FoSaaS, apresentando as ferramentas utilizadas e como as técnicas de reúso foram aplicadas.
- **Popularização dos SaaS:** é uma solução para resolver o problema de customização, recorrente no desenvolvimento de SaaS MT (Kitano *et al.*, 2010; Li *et al.*, 2011). Deste modo os padrões podem ser uma forma de aumentar a abrangência dos SaaS e incentivar mais fornecedores de software a adota-los como um modelo de negócio.

7.2 Limitações

Os padrões desenvolvidos, o FoSaaS e a avaliação da proposta apresentam as seguintes limitações:

- Os padrões propostos não tratam a customização dos requisitos não-funcionais do *tenants*, tais como: eficiência, confiabilidade, portabilidade e interoperabilidade. Em outras palavras, significa que todos os *tenants* dos SaaS desenvolvidos com os padrões apresentaram o mesmo comportamento para esses requisitos;
- A manutenção do FoSaaS pode ser complexa para desenvolvedores inexperientes. O funcionamento do Grails e as estruturas que compõem a geração de código, tais como *templates*, apresentam sintaxe não familiar a desenvolvedores iniciantes; Outra complexidade do FoSaaS são suas estruturas desenvolvidas para reúso, com objetivo de permitir a aplicação do *framework* para domínios variados.
- O FoSaaS não manipula a variabilidade dos requisitos de segurança dos *tenants.* Apenas as funcionalidades de autenticação de usuários e controle de permissão são implementadas no FoSaaS.
- A avaliação da proposta foi limitada pelo número reduzido de participantes e pela pouca experiência em desenvolvimento de software. Outra limitação da avaliação foi a ausência de comparação da proposta com abordagens que buscam resolver o problema de customização em SaaS MT.

7.3 Trabalhos Futuros

A seguir são apresentadas sugestões para atenuar as limitações listadas na Seção 7.3, aumentar os recursos, a qualidade dos padrões e do FoSaaS:

- **Aplicar os padrões em outras linguagens de programação:** os padrões desenvolvidos nesta proposta foram implementados na linguagem de programação Groovy. Contudo, existem outras linguagens que são utilizadas no desenvolvimento de SaaS. A implementação desses padrões em outras linguagens pode fornecer aos desenvolvedores outros exemplos e disseminar o seu uso;
- **Aprimorar os padrões:** os SaaS apresentados no estudo da aplicabilidade foram considerados como estudos de caso para este projeto, outros desenvolvedores de software podem identificar melhorias a ser realizadas. Dessa forma, um estudo de aplicabilidade mais complexo e com várias regras de negócio deve ser realizado a fim de avaliar a cobertura dos padrões propostos e indicar quais outros podem ser desenvolvidos;
- **Transformar o FoSaaS em um *plug-in* Grails:** o modo mais simples para incorporar dependências às aplicações Grails é por meio de *plug-ins*, pois eles permitem a inclusão automática de nova funcionalidade. Dessa forma, a criação de um *plug-in* em Grails pode possibilitar o uso facilitado do FoSaaS aos desenvolveres e usuários desse *framework*;
- **Realizar mais avaliações, comparando a proposta com outras abordagens:** Os experimentos apresentados no capítulo de avaliação envolveram no total dezesseis participantes. Embora fossem obtidas evidências que comprovam os benefícios dos padrões, a avaliação pode ser melhorada por meio de experimentos com mais participantes e comparações com outras abordagens que igualmente buscam resolver o problema de customização em SaaS MT; Como os trabalhos apresentados por Ruehl e Andelfinger (2011) e Schroeter *et al.* (2012)
- **Desenvolver padrões para tratar a customização dos requisitos não-funcionais:** a personalização dos requisitos não-funcionais tais como eficiência e confiabilidade também são considerados desafios no desenvolvimento de SaaS MT. Em razão das evidências obtidas pela avaliação, pode-se estender esse estudo para criar padrões

capazes de resolver a variabilidade dos *tenants* em termos de requisitos não-funcionais;

- **Avaliar os padrões e o FoSaaS no desenvolvimento de software tradicional:** realizar uma avaliação para comparar o FoSaaS e outros *frameworks* de aplicação, verificando se os padrões propostos também podem ser aplicados em aplicações *single-tenant*.

REFERÊNCIAS

ARMBRUST, M.; FOX, A; GRIRIFFITH, R.; JOSEPH, A.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. A view of cloud computing. ACM, vol. 4, p. 50-58, 2010.

ATEYEH, K.; LOCKEMANN, P. C. Reuse and aspect-oriented courseware development. Educational Technology and Society, 2006.

ATLASSIAN. Dream Big, work smart, delivery fast, 2013. Disponível em: <https://www.atlassian.com>. Acesso em: 17 mai. 2014.

BABAR M. A.; ALI N.; CHEN L. Variability management in software product lines: a systematic review. ACM International Conference Proceeding Series. In: Proceedings of the 13th International Software Product Line Conference (SPLC '09), 13, Pittsburgh. vol. 446, 2009 p. 81–90.

BACHMANN, F; CLEMENTS, C. P. Variability in software product lines. Software Engineering Institute Pittsburgh USA, 2005.

BANERJEE, P.; FRIEDRICH, R.; BASH, C.; GOLDSACK, P.; HUBERMAN, B. A.; MANLEY, J.; PATEL, C.; RANGANATHAN, P.; VEITCH, A. Everything as a Service: Powering the New Information Economy. v. 44, n. 3, p. 36-43, 2011.

BASS, L.; CLEMENTS, P.; KAZMAN, R. Software architecture in practice. The SEI Series in Software Engeneering, Addison-Wesley, 2003.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The Goal Question Metric Approach. In: Encyclopedia of Software Engineering. Hoboken, NJ, USA: John Wiley & Sons, 1994, v. 2, p. 528-532.

AYER, J. O.; FLEGE, P.; KNAUBER, R.; LAQUA, D.; MUTHIQ, K.; SCHMID, T.; WIDEN e J. DEBAUD. PuLSE: A Methodology to Develop Software Product Lines. In: Proceedings of the Fifth Symposium on Software Reusability (SSR'99). Bridging the Gap between Research and Practice, 1999, Los Angeles. ACM Press, 1999 p. 122–131.

BOSCH, J. Design and use of software architectures: adopting and evolving a product-line approach. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

SOLINGEN, R. V.; BERGHOUT, E. The Goal/Question/Metric Method: a pratical guide for quality improvement of software development. Londres: McGraw Hill, 1999, p. 216.

BUHNE, S.; CHASTEK, G.; KAKOLA, T.; KNAUBER, P.; NORTHROP, L.; THIEL, S. Exploring the context of product line adoption. In: Proceedings 5th International Workshop, PFE 2003, 15, Siena. 2003 p. 4-6.

CHEROBIBO, V. SaaS: Quatro letras para conquistar as pequenas empresas. Disponível em: <http://computerworld.uol.com.br/gestao/2007/10/16/idgnoticia>. Acesso em: 29 mai. 2013.

CHEN, Y.; GANNOD, G. C.; COLLOFELLO J. S. A software product line process simulator. Software Process: Improvement and Practice, p. 385–409, 2006.

CHONG, F.; CARRARO, G. Architecture Strategies for Catching the Long Tail. Disponível em: http://msdn.microsoft.com/en-us/library/aa479069.aspx. Acesso em: 5 jun. 2013.

CHOQUET, C.; CORBIÈRE, A. Reengineering framework for systems in education. Educational Technology and Society, 2006.

CLEMENTS, P.; NORTHROP, L. Software product lines: Practices and patterns. SEI Series in Software Engineering. Addison-Wesley, 2002.

COHEN, S.; L. NORTHROP. Object-Oriented Technology and Domain Analysis. In: 5th International Conference on Software Reuse, 5, 1998, Los Alamitos. IEEE Computer Society, 1998. p. 86–93.

COHEN, S. Product line state of the practice report. Technical report, Software Engineering Institute, Carnegie Mellon University, 2002.

CZARNECKI, K; EISENECKER, U. Generative programming: methods, tools, and applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

DEBAUD, J. M.; SCHMID, K. A Systematic Approach to Derive the Scope of Software Product Lines. In: Proceedings of the IEEE International Conference on Software Engineering, 1999, Los Angeles, IEEE Computer Society Press, 1999 p. 34–43.

DEURSEN, A.; KLINT, P. Domain-specific language design requires feature descriptions. Journal of Computing and Information Technology, 2001.

DYBA, T.; DINGSOYR, T.; HANSSEN, G. K. Applying Systematic Reviews to Diverse Study Types: An Experience Report. In: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM '07). IEEE Computer Society, Washington, 2007 p. 225-234.

EIDSON, B.; MARON, J.; PAVLIK, G. SOA and the future of application development. In Proceedings of the First International Workshop on Design of Service-Oriented Application (WDSOA05), pages 1-8. IBM Research Division, IBM, November 2005.

ERRADI, A.; MAHESHWARI, P.; WLADIMIR, T. Policy-driven middleware for self-adaptation of web services compositions. In Maatern van Steen and Michi Henning, Middleware 2006, volume 4290 of Lecture Notes in Computer Science, pages 62-80. Springer, 2006.

EPCGLOBAL. The global language of business. Disponível em: <http://www.gs1.org/epcglobal>. Acesso em: 25 jun. 2014.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995. [REVER]

GIT. Distributed-even-if-your's-workflow-isnt. Disponível em: <http://git-scm.com/>. Acesso em: 19 jun. 2013.

GRAILS.Grails Framework Full Stack. Disponível em: <http://grails.org/>. Acesso em: 19 jun. 2013.

GOMMA, H. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison Wesley Longman Publishing Co., Inc., Redwood City, 2004.

GOMMA, H.; WEBBER, D. Modeling Adaptive and Evolvable Software Product Lines Using the Variation Point Model. In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04), 2004. 37., Washington. IEEE Computer Society, v. 9, 2004 p. 90268.

IDC. Cloud Computing's Role in Job Creation. Disponível em: <http://www.microsoft.com/enus/news/download/features/2012/IDC_Cloud_jobs_White_Paper.pdf>. Acesso em: 26 mai. 2012.

JACOBSON, I.; GRISS, M.; JONSSON, P. Software Reuse: Architecture, Process and Organization for Business Success. Reading, MA: Addison-Wesley, 1997.

JIANG, J.; RUOKONEN, A.; SYSTA, T. Pattern-based variability management inf web services development. In: Proceedings of the 3th European Conference on Web Services (ECOWS' 05), 2005, 3, Washington. IEEE Computer Society. 2005 p. 86.

KANG, K. Feature-oriented domain analysis feasibility study. Disponível em: <http://www.sei.cmu.edu/reports/90tr021.pdf>. Acesso em: 19 apr. 2013..

KITANO, T.; IGUCHI, K.; KOYAMA, K. Generating Robust XPaths for Service Customization. In: World Congress on Services, 6, Washington. 2010 p. 166-167.

KITCHENHAM, B. A. Procedures for performing systematic reviews**.** Technical Report, Keele University and NICTA, 2004.

KHAN, A.; KASTENER, C.; KOPPEN, V.; SAAKE, G. Service variability patterns**.** In Proceedings of the 30th international conference on Advances in conceptual modeling: recent developments and new directions (ER'11), Springer-Verlag, Berlin, Heidelberg, 2011 p. 130-140.

LI, D.; ZHANG, W.; ZHOU, S.; LIU, C.; JIN, W. Portal-based design for Software as a Service system presentation layer configurability. In: Computer Science Education (ICCSE), 2011 6th, International Conference, 2011 p. 1327-1330.

H2DATABASE. H2 Database Engine. Disponível em: <http://www.h2database.com/>. Acesso em: 25 jun. 2014.

HAN, J. Cloud Computing Review**,** Enterprise technology development, 2010.

HAYES-ROTH, B. A Domain-Specific Software Architecture for Adaptive Intelligent Systems. Software Engineering, IEEE Transactions on, v. 21, n. 4, p. 288–301, apr. 1995.

MIETZNER, R.; METZGER, A.; LEYMANN, F.; POHL, K. Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In: Proceedings of the Workshop on Principles of Engineering Service Oriented Systems (PESOS ’09), 2009, 09, Vancouver. IEEE Computer Society, 2009 p. 18–25.

NITU. Configurability in SaaS (software as a service) applications. In: Proceedings of the 2nd India software engineering conference, ISEC '09, New York, NY, USA., p. 19-26. ACM, 2009.

NORTHROP, L. M. Software product lines essentials. Disponível em <http://www.sei.edu/library/assets/spl-essentials.pdf>. Acesso em: 18 jan. 2013.

MA, D. The Business Model of "Software-As-A-Service". In: Conference on Services Computing (SCC), 1, IEEE International, 2007 p. 701- 702.

MYSQL. MYSQL: The World's Most Popular Open Source Database. 2011. Disponível em: <http://www.mysql.com/>. Acesso em: 16 mai. 2013.

RAHMAN S. S, ATEEQ K., GUNTER, S. Language Independent Rule-Based AOP Model for Adaptable Context-Sensitive Wev Service. In: Conference on Current Trends in Theory and Pratice of Computer Science, 2010, 36, Prague. Institute of Computer Science, v. 2, 2010 p. 87-99.

RUEHL, S. T., ANDELFINGER, U. Applying software product lines to create customizable software as a service application. In Proceedings of the 15th International Software Product Line Conference (SPLC '11), 15, 2011, New York. ACM, vol. 2, 2011 p. 1–4.

ROSCHELLE, J.; KAPUT, J.; STROUP, W; KAHN, T. M. Scaleable integration of educational software: Exploring the promise of component architectures. Journal of Interactive Media in Education, v. 98, p.6-98, nov. 1998.

SPRINGSOURCE. Spring Tool Suite. Disponível em: <http://www.springsource.org/sts>. Acesso em: 16 mai. 2014.

OASIS WS-BPEL. Technical Committee. Disponível em: <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel>. Acesso em: 16 mai. 2014.

OASIS WS-BPEL. **Technical Committee**. 2013. Disponível em: <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel>

PIVOTAL e GROOVY COMMUNITY. A dynamic language for the Java platform. Disponível em: <http://groovy.codehaus.org/>. Acesso em: 27 ago. 2012.

POHL K., Böckle G., Linden F. Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Berlin Heidelberg New York, 2005.

POSTGRESQL. PostgreSQL. Disponível em: <http://www.postgresql.org/>. Acesso em: 6 jan. 2014.

SEI PLP. Framework for Product Line Practice. Disponível em: <http://www.sei.cmu.edu/plp/>. Acesso em: 5 mar. 2013.

SCHROETER, J.; CECH, S.; GOTZ S.; WILKE, C.; AßMANN, U. Towards Modeling a Variable Architecture for Multi-Tenant SaaS-Applications. In: Proceedings of the sixth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS ’12), 2012 pages 111–120.

SOURCEFORGE. Solution center 2014. Disponível em: <http://sourceforge.net/>. Acesso em: 21 jun. 2012.

SPOCK. The enterprise ready specification framework. 2013. Disponível em: <https://code.google.com/p/spock/>. Acesso em: 8 jun. 2013.

SUN, W.; ZHANG, X.; GUO, C. J.; SUN, P.; SU, H. Software as a service: Configuration and customization perspectives. In: Proceedings of the 2008 IEEE Congress on Services (SERVICES-2 '08), 2008, 2, Washington. IEEE Computer Society, v. 2, 2008, p. 18-25.

SVAHNBERG, M.; VAN GURP, J.; BOSCH, J. A taxonomy of variability realization techniques: Review search articles. New York: Software—Practice & Experience, v. 35, n. 8, p. 705–754, jun. 2005.

TAO, C.; LIAO, H. An Anatomy to SaaS Business Mode Based on Internet. In: International Conference on Management of e- Commerce and e-Government (ICMECG), 2008, 08. Washington. ACM, 2008 p. 215-220.

TAURION, C. Computação em Nuvem: Transformando o Mundo da Tecnologia da Informação, v. 1, p. 228, 2009.

TICHY, W. F. Should computer scientists experiment more. Computer, v. 31, n. 5, p.32–40, nov. 1998.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. Introdução à engenharia de software experimental. 2002. Disponível em: <http://www2.ufpa.br/cdesouza/teaching/topes/4-ES-Experimental.pdf>. Acesso em: 9 maio 2013.

TURNER, M.; BUDGEN, D. Turning software into a service. Computer,v. 36, n. 10, p. 38-44, oct. 2003.

VAQUERO, L. M.; MERINO, L.; CACERES, J.; Lindner, M. A break in the clouds. ACM SIGCOMM Computer Communication Review, New York, v. 39, n. 1, p. 50-55, jan. 2009.

YAO, P.; WEN, H.; DAi, Q. Study on Presentation Layer Structure of Multi-tenant E-business Systems. In: Industrial Engineering and Engineering Management (IE&EM), 2011, Changchun. IEEE, 2011 p. 352-354.

YOUSEFF, L.; BUTRICO, M.; DA SILVA, D. Toward a Unified Ontology of Cloud Computing. In: Grid Computing Environments Workshop, 2008. 08., Austin. IEEE, 2008 p. 1–10.

WEI, Z.; NI, X.; JIA, D. Research on software reuse of user interface for mobile computing devices based on XML. In: Software Engineering and Service Sciences (ICSESS), 2010. 10., Beijing. IEEE, 2010 p. 146-149.

WEISS, D.; LAI, C. T. Software Product-Line Engineering: A Family-Based Software Development Process. Boston: Addison-Wesley, 1999. 426 p.

WEISSMAN, C. D.; BOBROWSKI, S. The design of the force.com multitenant internet application development platform. In: Proceedings of the 35th SIGMOD international conference on Management of data, 39, 2009, New York. ACM, 2009 p. 889-896.

ZAMANI, B.; BUTLER, G.; KAYHANI, Tool Support for Pattern Selection and Use: Electronic Notes in Theoretical Computer Science Electron, 233, 2009, Hungary. 2009 p. 127-142.

ZAINUNDDIN, E.; GONZALEZ, P. Configurability, Maturity, and Value Co-creation in SaaS: An Exploratory Case Study. In: ICIS 2011 Proceedings, 1, 2011, Shanghai. 2011 p. 9.

ZAINUNDDIN, E.; STAPLES, S. Enhancing SaaS with Insouciant Domain Knowledge. In: 16th UKAIS Conference Proceedings, 16, 2011, Oxford. 2007 p. 110-112.

ZHANG, K.; ZHANG, X.; SUN, W.; LIANG, H.; HUANG Y.; ZENG L.; LIU X. A Policy-Driven approach for Software-as-Services customization. In: E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 4, 2007, Tokyo. IEEE Computer Society, 2007 p.123-130.

APÊNDICE A

Questionário - Conhecimento do Participante do Experimento

Nome:
Linha de pesquisa:

1. Classifique seu conhecimento nas linguagens de programação entre: 1 – nenhum, 2 – pouco, 3 – muito, 4 – especialista:

( ) Java SE
( ) Java EE
( ) C
( ) C++
( ) C#
( ) Python
( ) Groovy
( ) Ruby
( ) PHP

2. Classifique seu conhecimento nos *frameworks* entre: 1 – nenhum, 2 – pouco, 3 – muito, 4 – especialista:

( ) Grails
( ) Rails
( ) Django
( ) Zend

3. Classifique seu entendimento entre: 1 – nada, 2 –pouco, 3 – bom e 4 – muito bom:

( ) Fluxo de execução Grails:
( ) Groovy Server Pages:
( ) Grails Controllers:
( ) Grails Domains (relacionamentos e restrições):
( ) Anotações:
( ) Funcionamento do *Framework* SaaS:

( ) Programação Web (cliente x servidor):

4. Classifique sua dificuldade entre: 1 – nenhuma, 2 – pouca, 3 – alta e 5 – muita alta:

( ) Criação e definição dos *tenants*:
( ) Escrita do modelo (Grails - *Domains*):
( ) Exercício proposto:

5. Explique quais foram suas dificuldades no exercício proposto:

6. Descreva de forma sucinta como você fez para diferenciar as funcionalidades acessíveis por *tenant* (trechos de código que podem ser usados na explicação). Caso você tenha alguma ideia de resolução que não conseguiu programar, descreva-a textualmente:

APÊNDICE B

Formulário de Execução do Treinamento

Obs: este formulário deverá ser entregue preenchido ao final do experimento,

1. Identificação do Participante

Nome:			Data: ___/____/_____
	COM auxílio da abordagem		SEM auxílio da abordagem

2. Execução do Experimento

Preenchido pelo Participante		Preenchido pelo Proponente do Experimento
Hora Início: ___ : ___	Hora Término: ___ : ___	**Tempo I (minutos):**

Interrupções

Preenchido pelo Participante		Preenchido pelo Proponente do Experimento
Hora Início: ___ : ___	Hora Término: ___ : ___	T1 (minutos):
Hora Início: ___ : ___	Hora Término: ___ : ___	T2 (minutos):
Hora Início: ___ : ___	Hora Término: ___ : ___	T3 (minutos):
Hora Início: ___ : ___	Hora Término: ___ : ___	T4 (minutos):
Hora Início: ___ : ___	Hora Término: ___ : ___	T5 (minutos):
TOTAL		**Tempo II (minutos):**

Tempo total do participante (Tempo I – Tempo II):	

3. Observações
