Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

Charles Wellington de Oliveira Fortes

PROPOSTA DE UMA ARQUITETURA PARA CUSTOMIZAÇÃO DE SISTEMAS USANDO MECANISMOS DE INJEÇÃO DE DEPENDENCIA

Belo Horizonte
2012

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de *Software*

PROPOSTA DE UMA ARQUITETURA PARA CUSTOMIZAÇÃO DE SISTEMAS USANDO MECANISMOS DE INJEÇÃO DE DEPENDENCIA

por

CHARLES WELLINGTON DE OLIVEIRA FORTES

Monografia de Final de Curso

Prof. Marco Túlio Valente
Orientador

Belo Horizonte
2012

CHARLES WELLINGTON DE OLIVEIRA FORTES

PROPOSTA DE UMA ARQUITETURA PARA CUSTOMIZAÇÃO DE SISTEMAS USANDO MECANISMOS DE INJEÇÃO DE DEPENDENCIA

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Especialista em Informática

Área de concentração: Especialização em Informática: Ênfase: Engenharia de *Software*

Orientador: Prof. Marco Túlio Valente

Belo Horizonte
2012

Folha de Aprovação

RESUMO

A customização de *software* de forma sustentável e escalável é um grande desafio do ponto de vista arquitetural para as empresas desenvolvedoras. As técnicas comumente adotadas pela comunidade de desenvolvimento de *software,* de forma geral, favorecem a incidência de erros que afetam o sistema não somente para o contratante da adaptação, mas também a outros clientes que utilizam as funcionalidades presentes no módulo alterado. Além disto, na maioria das vezes é percebido um aumento da complexidade do código fonte à medida que as customizações vão sendo incluídas, e uma maior dificuldade de cobrir o sistema com os testes necessários para garantir seu bom funcionamento.

Neste trabalho serão abordadas as técnicas mais utilizadas pela comunidade de desenvolvimento de *software*. Será apresentado e testado em um caso real, uma proposta de uma arquitetura que une padrões de projetos e técnicas de organização do código. Esta proposta visa proporcionar uma alternativa que remete a redução de incidência de erros, mantendo o código fonte do módulo original do sistema limpo destes detalhes de implementação.

Palavras Chaves: *Padrões de Projetos*, Arquitetura, Customização.

ABSTRACT

Software customization in a sustainable and scalable baseline is a great challenge in terms of architecture for the software houses. The techniques commonly adopted by the community of software development favor the incidence of errors affecting the system not only to the customer who contracted the customization, but also other customers who use the features present in the changed module. Moreover, in most cases it is perceived increases in the source code complexity as the customizations are being included, and it is more difficult to cover the system with the necessary tests to ensure its smooth operation.

This paper addresses techniques commonly used by software development community, presenting and testing in a real use case a proposal of an architecture that unites design patterns and techniques for organizing code that aim to provide an alternative that leads to reduction of errors, keeping the source code clean of implementation details.

Keywords: Design Patterns, Architecture, *Software* Customization.

LISTA DE ILUSTRAÇÃO

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Segundo Otávio Coelho, grande parte dos *software* atuais têm de lidar com a necessidade de customizações. O que significa não somente introduzir modificações conforme as necessidades particulares de uma empresa, mas também um grande esforço de implementação (COELHO, 2007).

A customização de *software* pode ser entendida como parte do seu processo de evolução, uma vez que promove a adição e modificação de recursos e funcionalidades, sejam elas específicas de um cliente, ou aberta a todos os seus usuários. Devido a isto, quando observamos a segunda lei da evolução de *software* apresentada no capitulo 3.1, observamos o aumento da complexidade do código fonte (LEHMAN, 2000) e o consequente aumento do custo de manutenção (BROOKS, 1995).

> "Quando realizada pela empresa fabricante do *software*, a customização pode ser, por vezes, uma fonte de renda, por outras, de prejuízo. Ela costuma ser uma fonte de renda quando a customização pode ser vendida e re-incorporada em novas versões do *software*, ou quando for possível agregar uma rede de serviços capaz de implementá-las sem modificar o produto. Por outro lado, a customização pode ser um ônus pesado quando ela implica em uma versão diferente do produto para cada cliente, causando problemas de suporte e manutenção, ou quando a exigência de customização pelo cliente não pode ser obedecida pelo fornecedor devido a limitações de recursos, aumentando o risco da perda do cliente para a concorrência." (COELHO, 2007)

Focando neste pensamento, este trabalho visa elicitar as técnicas de customização mais utilizadas pelas empresas de *software* através de uma pesquisa online que foca principalmente as empresas das regiões sul e sudeste do país. Ao final, nosso objetivo é propor um modelo de arquitetura que visa servir de alternativa para reduzir os problemas resultantes da implementação das customizações, sendo avaliada na prática em um projeto de médio porte real.

1.1 Problema

As empresas, independente de seu porte ou ramo de negócio, possuem necessidades particulares que necessitam ser supridas pelo *software* utilizados para apoio ao seu funcionamento. Mesmo em se tratando de empresas do mesmo ramo/setor, cada uma apresenta situações singulares decorrentes de sua cultura organizacional, fluxo de trabalho, especialização/diferencial dentre outros.

A adaptação de *software* para a adequação de suas funcionalidades em diversos contextos é um dos maiores desafios para a empresa que fabrica *software*. Encontrar o melhor caminho para a adaptação de um *software* sem que haja impacto no funcionamento de sua versão padrão e de forma que seja fácil de ser desenvolvida e mantida é uma tarefa que não tem sido fácil para os arquitetos e engenheiros de *software*.

Durante o trabalho, serão citadas as formas mais comuns utilizadas pelas empresas de *software* para adaptar seus produtos, sendo apresentados seus pontos positivos e negativos. Ao final do trabalho será proposta uma alternativa de arquitetura que visa que as adaptações do *software* sejam feitas de forma simples e com o mínimo de impacto no produto padrão.

1.2 Objetivo Geral

Este trabalho tem como objetivo geral analisar as formas de customização de *software* utilizada pelas empresas do setor, e propor uma alternativa de arquitetura que permita que um *software* seja adaptado com o mínimo impacto em sua versão padrão, oferecendo menos riscos de afetar outros clientes, sendo de simples implementação e de fácil manutenção.

1.3 Objetivos Específicos

Pesquisar e comparar as formas utilizadas por diversas empresas de *software* para a adaptação de seus produtos, elicitando seus pontos positivos e negativos.

Apresentar os resultados de qualidade e complexidade obtidos por empresas que utilizam as formas mais comuns de adaptação de *software* e compará-los com um projeto que utilizará a arquitetura proposta.

Abordar as técnicas de reflexão computacional e injeção de dependência para a elaboração de uma proposta de arquitetura.

Apresentar uma arquitetura que sirva de alternativa para ser usada e testada, de forma que possa com o passar do tempo e utilização pela comunidade de desenvolvimento de *software*, sofrer refinamentos a fim de superar os problemas detectados pela pesquisa.

1.4 Justificativa

Melhorar a qualidade das implementações de customizações específicas de clientes em *software*, buscando reduzir a incidência de erros e impactos a outros clientes que não aquele que contratou a modificação.

2 METODOLOGIA

Para o desenvolvimento deste trabalho são utilizadas pesquisas em materiais de auxílio para a elaboração de um estudo bibliográfico sobre as técnicas e padrões utilizados para a elaboração da arquitetura do sistema. As citações de trechos de obras em idioma estrangeiro, mais especificamente o inglês, serão traduzidas pelo autor do trabalho.

Além da pesquisa bibliográfica, serão utilizados formulários de pesquisas on-line que serão distribuídos entre profissionais de diversas empresas do país, com maior enfoque na região sudeste e sul, para o levantamento dos dados acerca das técnicas, metodologias e resultados obtidos para a customização de seus produtos.

É previsto que a entrevista atinja os mais diversos nichos de empresas de *software* e de vários tamanhos e cenários de atuação.

Para validação da arquitetura proposta no trabalho, a mesma é utilizada em um projeto de médio porte em uma empresa real instalada na capital de Minas Gerais, Belo Horizonte. Para avaliação foi feito o acompanhamento da equipe de desenvolvimento e arquitetura, recolhendo depoimentos sobre sua produtividade e qualidade utilizando a técnica de entrevista.

3 FUNDAMENTAÇÃO TEÓRICA

Em produtos de *software*, é muito comum que partes do *software* necessitem de adaptações para contextos específicos de determinadas empresas, domínios ou segmentos. Estas adaptações geram muitas vezes um vínculo com as funcionalidades do sistema que tornam estes fatores evolução e manutenção críticos.

Dentre as alternativas para se controlar estas modificações temos diversos recursos que vão desde a criação de *branches* das funcionalidades com recursos fornecidos pelo sistema de controle de versão (GOUSSET, KELLER, *et al.*, 2012) até controle de acesso via controle condicional “se-então-senão”.

No decorrer do trabalho serão analisados os prós e contras das técnicas utilizadas, focando em sua relação de custo, qualidade e complexidade do *software*, sendo apresentado uma arquitetura que une os benefícios da inversão de controle sugerida por Martin Fowler (FOWLER, 2004) e recursos de reflexão computacional fornecidos pelas linguagens de alto nível, focando no .NET Framework (NET FRAMEWORK DEVELOPER'S GUIDE, 2010) como exemplo.

3.1 Leis da Evolução de *Software*

As leis de evolução de *software*, que podem ser observadas na Tabela 1 - As oito leis de evolução de *software*, foram formuladas entre as décadas de 70 e 90 baseados nos resultados do projeto FEAST (sigla em inglês para *Feedback, Evolution And Software Technology* – Retroalimentação, Evolução e Tecnologias de *Software*), que “foi criado para explorar a hipótese de que para melhorar os processos de *software* do mundo real é preciso levar em conta as muitas interações de retroalimentação de tais processos” (LEHMAN, 2001).

Tabela 1 - As oito leis de evolução de *software*

Número	Ano	Nome	Lei
I	1974	Mudança contínua	Um *software* deve ser continuamente adaptado, caso contrário se torna progressivamente menos satisfatório.
II	1974	Complexidade crescente	À medida que um *software* é alterado, sua complexidade cresce, a menos que um trabalho seja feito para mantê-la ou diminuí-la.
III	1974	Auto-regulação	O processo de evolução de *software* é auto-regulado próximo à distribuição normal com relação às medidas dos atributos de produtos e processos.
IV	1978	Conservação da estabilidade organizacional	A não ser que mecanismos de retro-alimentação tenham sido ajustados de maneira apropriada, a taxa média de atividade global efetiva num *software* em evolução tende a ser manter constante durante o tempo de vida do produto.
V	1978	Conservação da Familiaridade	De maneira geral, a taxa de crescimento incremental e taxa de crescimento em longo prazo tendem a declinar.
VI	1991	Crescimento contínuo	O conteúdo funcional de um *software* deve ser continuamente aumentado durante seu tempo de vida para manter a satisfação do usuário.
VII	1996	Qualidade decrescente	A qualidade do *software* será entendida como declinante a menos que o *software* seja rigorosamente adaptado às mudanças no ambiente operacional.
VIII	1996	Sistema de Retro-alimentação	Processos de evolução de *software* são sistemas de retro-alimentação em múltiplos níveis, em múltiplos laços (loops) e envolvendo múltiplos agentes.

Fonte: Adaptado de (LEHMAN, 2000)

As três primeiras leis começaram a ser elaboradas no início da década de 70, as duas seguintes na década de 80 e as três restantes na década de 90 tendo como base a evolução de dois sistemas operacionais (IBM OS/360 e ICL VME Kernel), um sistema financeiro (Logica FW), um sistema de telecomunicações (Lucent) e um sistema de defesa (Matra BAE Dynamics).

Conforme observado na Tabela 1 - As oito leis de evolução de *software*, a necessidade de evolução do produto é um fator fundamental para sua sobrevida, sendo que estas evoluções podem ser feitas para atender a demandas tecnológicas do ambiente (LEHMAN, 2000) ou as necessidades específicas de um cliente (COELHO, 2007). Este processo resulta em um aumento da complexidade e queda da qualidade do código fonte, o que eleva os custos da manutenção do *software*, podendo ir de 40% do custo de desenvolvimento (BROOKS, 1995) e podendo inclusive excedê-lo (SOMMERVILE, 2007).

Além disto, durante o processo de alteração de um trecho do código fonte de um *software*, mesmo que para manutenções corretivas, esbarramos com a probabilidade de 20-50% de criar defeitos, além do aumento dos custos e necessidade de cobertura dos testes (BROOKS, 1995).

Devido a estes pontos de relevância, como já observado dentro das leis de *software*, é necessário que sejam investigados mecanismos e arquiteturas que promovam a facilidade de manutenção e evolução do sistema.

3.2 Design Patterns

Alexander (ALEXANDER, 1979) define que um padrão é uma solução para um problema em determinado contexto, sendo que padrões existem para quase todos os problemas de um projeto, podendo estes serem combinados para resolver um problema complexo.

Para (SHALLOWAY, 2002) *Design Patterns* constituem parte da vanguarda da tecnologia orientada a objeto, sendo que o propósito por trás dos *Design Patterns* é a qualidade do *software*. A aplicação destes padrões proporciona acima de tudo ganho no desempenho durante a produção do sistema e durante sua vida, quanto das manutenções e evoluções do sistema.

3.3 Princípios de Inversão de Controle

Martin Fowler explica que a inversão de controle é uma característica comum dos frameworks. Este padrão caracteriza-se por um conjunto de “*containers* que ajudam a montar componentes de projetos diferentes em uma aplicação coesa” (FOWLER, 2004).

Robert Martin a define como sendo a estratégia de trabalhar com interfaces ou funções e classes abstratas, ao invés de funções e classes concretas, que é o principio por trás dos projetos de componente, como exemplo o COM, CORBA, EJB, dentre outros (MARTIN, 2000).

Esta visão foi proposta ao pensarmos que os módulos de alto nível não deveriam depender diretamente dos módulos de implementação. Assim, passamos a apontar estas dependências para abstrações e transpassar o controle a estes (MARTIN, 2000).

Porém o termo inversão de controle é por demais generalista (FOWLER, 2004), sendo assim será focado em uma das técnicas de inversão de controle utilizadas nos *containers* mais comuns, chamada de Injeção de Dependência.

3.3.1 Injeção de Dependência

Segundo Martin Fowler, “A ideia básica da *Dependency Injection* é ter um objeto separado, o montador (*assembler*), que popula um campo em um objeto com uma implementação apropriada para a interface” (FOWLER, 2004) resultando em um comportamento como demonstrado na Figura 1 - As dependências para um Injetor de Dependências

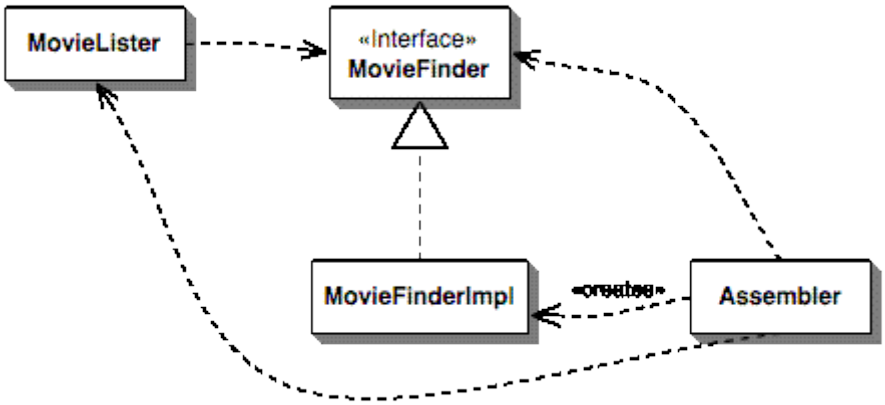

Figura 1 - As dependências para um Injetor de Dependências
Fonte: (MARTIN, 2000)

Desta forma, a injeção de dependência passa a poder construir o objeto concreto na memória conforme o contexto de necessidade para aquela abstração. Assim, no exemplo da Figura 1 - As dependências para um Injetor de Dependências pode-se perceber que a classe de alto nível *MovieLister* conhece apenas a interface que define o contrato do *MovieFinder*, sendo que o *Assembler* fica com a responsabilidade de conhecer e providenciar que a inversão de controle funcione desacoplando os componentes.

3.4 Reflexão para Instanciação de Classes

Conforme a documentação oficial da Microsoft (NET FRAMEWORK DEVELOPER'S GUIDE, 2010), reflexão computacional fornece objetos que encapsulam *assemblies*, módulos e tipos, e pode ser usada para criar instâncias de tipos dinamicamente, ligar o tipo a um objeto existente ou obter as definições de tipagem de um objeto existente. Além disto, os recursos de reflexão computacional permitem que sejam disparados métodos ou que seus campos e propriedades sejam acessados.

Os métodos de carga dinâmica em comparação aos métodos tradicionais possuem um custo maior, conforme pode ser observado na Figura 2 - Desempenho das chamadas de métodos pelos diferentes recursos. Segundo (COELHO, 2007), deve-se considerar, porém, o custo da carga do executável nos demais casos, pois este

“costuma ser maior do que o tempo de invocação – o que pode exigir o uso de cache” (COELHO, 2007).

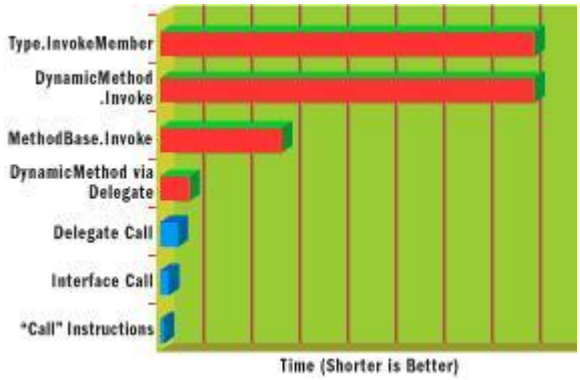

Figura 2 - Desempenho das chamadas de métodos pelos diferentes recursos
Fonte: (COELHO, 2007)

3.5 Conclusão do referencial teórico

A customização de *software* é um desafio para os arquitetos de *software* e as empresas de desenvolvimento. Ao longo dos anos diversos estudos vem comprovando as consequências das constantes modificações do código fonte, ao mesmo tempo em que se evidencia que estas são necessárias para que o *software* continue a atender às necessidades de seus clientes.

Com a utilização dos padrões de projetos e recursos das linguagens de auto nível, em especial o .NET framework, que foi escolhido para demonstrar o desenvolvimento deste trabalho, será proposta uma arquitetura de customização que pretende apresentar uma alternativa de fácil desenvolvimento e baixo impacto.

4 ARQUITETURA DE CUSTOMIZAÇÃO DE *SOFTWARE*

4.1 Análise da pesquisa de customização de *software*

Foi realizada no período de 03 de Outubro de 2011 a 14 de Janeiro de 2012 uma pesquisa online a qual foi respondida por 36 profissionais de nível sênior atuantes em arquitetura e desenvolvimento de sistemas, sendo que cada um destes representando uma única empresa de pequeno (16), médio (18) ou grande (2) porte, sendo estas localizadas nas cidades de Belo Horizonte (MG), São Paulo (SP), Rio de Janeiro (RJ) e Porto Alegre (RS), cujos nomes, por acordo, foram preservados.

O questionário aplicado está no anexo 1 desta monografia, e as respostas na íntegra no anexo 2.

4.1.1 Custo, Qualidade e Complexidade na Customização de *Software*

Alterações no código fonte de um *software*, mesmo quando focado em resolver problemas, possuem de 20% a 50% de chances de criar problemas (BROOKS, 1995), e ao observar o GRÁFICO 1 - Efeito dos erros causados pela implementação de customizações em clientes, podemos observar que em 36% das vezes, quando a implementação de uma nova customização gera um erro no *software*, este afeta a todos os clientes, e em apenas 17% das vezes estes erros não afetam os clientes.

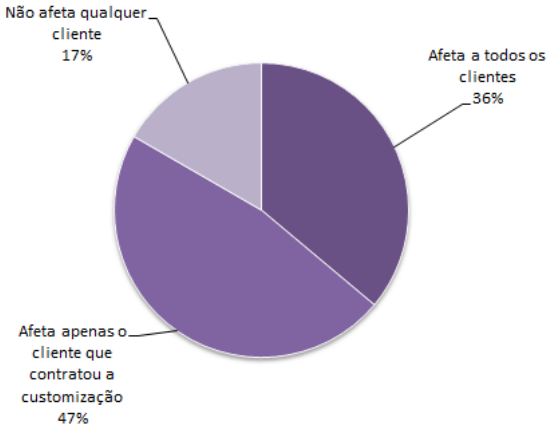

GRÁFICO 1 - Efeito dos erros causados pela implementação de customizações em clientes

Além disto, quanto mais o *software* sofre alterações, mais se torna complexo e tem sua qualidade diminuída, exigindo um trabalho contínuo (LEHMAN, 2000) evitando impactos para os clientes e prejuízos para a empresa desenvolvedora do *software*.

Customizar um *software* pode representar lucro ou prejuízo para a empresa desenvolvedora, dependendo da abordagem adotada. Por via de regra, uma customização traz lucro quando esta pode ser incorporada em novas versões do *software*, ou quando é possível implementá-la sem modificar o produto. E acaba por trazer ônus quando isto implica em uma versão diferente do produto para cada cliente, o que acarreta problemas de suporte e manutenção (COELHO, 2007).

O custo de se implementar uma customização deve ser pensado além do simples fator homem/hora. Sommervile (SOMMERVILE, 2007) aponta que o processo de evolução é impulsionado pelas solicitações de mudança, e que deve incluir a análise do impacto que esta mudança trará ao sistema e planejamento.

Segundo a pesquisa realizada para a elaboração deste trabalho observada no GRÁFICO 2 - Análise de complexidade agregada pela customização, 64% das respostas apontam um aumento de complexidade agregada ao *software* em decorrência das customizações, sendo que destas, 47% estão atreladas a complexidade de gestão das customizações e 17% atreladas a complexidade do código fonte. Devido a isto, é necessário que estes fatores sejam também levados em consideração durante a avaliação de impacto citada por Sommervile, pois podem influir diretamente nos custos de manutenção por todo o ciclo de vida do *software*, que normalmente já tendem a exceder os custos de criação (SOMMERVILE, 2007).

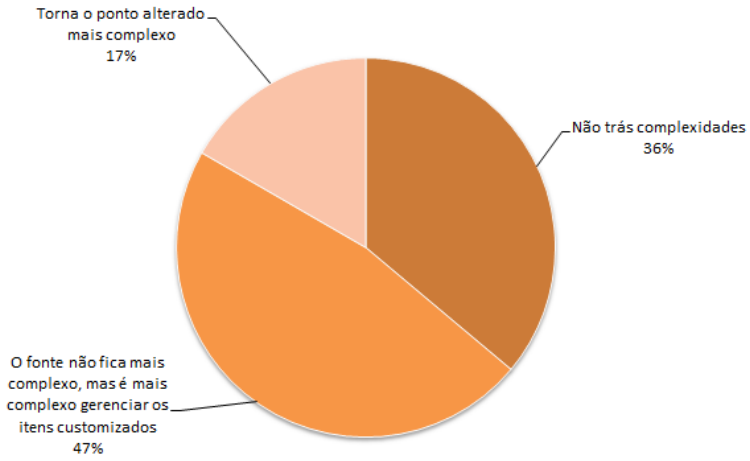

GRÁFICO 2 - Análise de complexidade agregada pela customização

A necessidade desta preocupação com as fases de manutenção dentro do ciclo de vida do *software* está diretamente relacionada ao custo que estas representam para a empresa que o desenvolveu, principalmente devido ao fato de que “a manutenção do *software* existente pode ser responsável por mais de 70% de todo o esforço despendido por uma organização de *software*” (PRESSMAN, 1995, p. 876), sendo que esta percentagem se eleva conforme novos *software* vão sendo produzidos.

4.1.2 Técnicas de Customização

Coelho (COELHO, 2007) classifica as fases de maturidade de customização de *software* em três fases, sendo elas: 1 código, 0/1 customizações; N códigos, N customizações; 1 código fonte, N customizações; conforme apresentado na Tabela 2 - Classificação das Customizações de *Software*

Tabela 2 - Classificação das Customizações de *Software*

Fase	Descrição
1 código fonte, 0/1 customizações	Ocorre em cenários onde a aplicação é muito simples, sem suporte à parametrização, ou na primeira venda e implantação
N códigos fontes, N customizações	Ocorre quando clientes importantes exigem customizações e não são implementadas práticas que garantam um único código (produto) para

	todos os clientes **[Branchs]**
1 código fonte, N customizações	Ocorre quando o produtor do *software* se organiza e produz um único código fonte através de uma ou mais das seguintes técnicas: • **Versionamento** com negociação com clientes. As customizações são demandadas, mas passam por um processo de generalização, a fim de que todos os clientes possam se beneficiar; • **Parametrização de dados**, onde técnicas de metadados permitem a configuração do comportamento da aplicação pelo cliente ou área de serviço; • Personalização de código **[Fluxo Condicional]** , onde técnicas de inserção de código permitem a alteração do comportamento da aplicação pelo cliente ou área de serviço; • **Uso de linguagens de programação**, onde o cliente ou a área de serviços pode estender a aplicação incluindo novas telas, relatórios e comportamentos.

Fonte: Adaptado de (COELHO, 2007)

Conforme observado no GRÁFICO 3 - Técnicas de customização adotadas, as empresas desenvolvedoras de *software* estão com seus sistemas na segunda e terceira fases apresentadas por Coelho. Devido a isto, este trabalho manterá seu foco apenas na segunda fase e nas técnicas listadas na terceira fase, as quais tiveram mais de 10% de adoção pelas empresas de *software*.

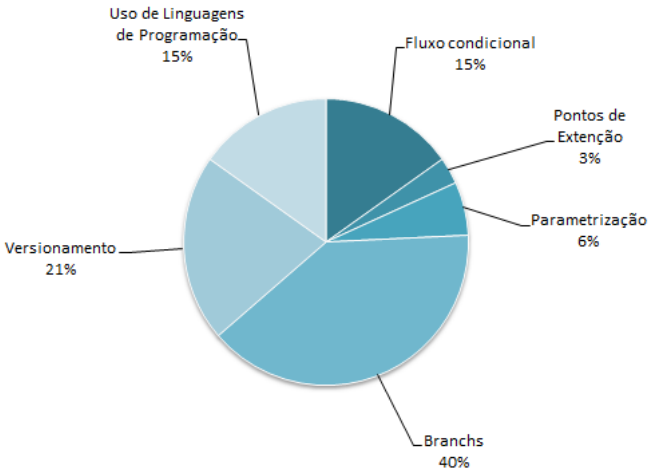

GRÁFICO 3 - Técnicas de customização adotadas

4.1.2.1 Branches

(GOUSSET, KELLER, *et al.*) Definem *branch* que “a maneira mais fácil de entender o que é *branch*, é que quando você faz um *branch* de seu código fonte,você está fazendo uma cópia dele” (GOUSSET, KELLER, *et al.*, 2012, p. 449).

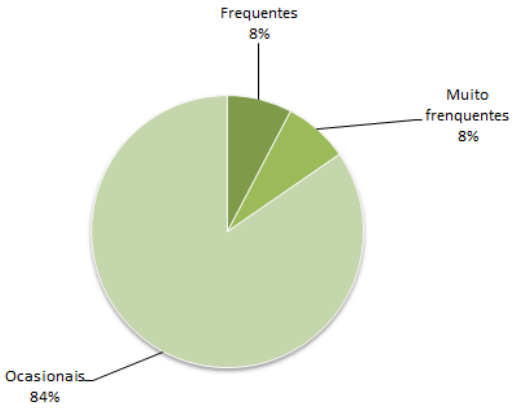

GRÁFICO 4 - Incidências de erro na técnica de branchs

GRÁFICO 5 - Impactos dos erros provocados pela utilização da técnica de branchs

Com relação a qualidade do código fonte para manutenção futura, é verificado no GRÁFICO 9 - Aumento de complexidade do código fonte provocado pela técnica de versionamento que em 100% dos casos não foi percebido um aumento da complexidade de manutenção do código fonte, sendo que em apenas 29% das vezes é percebido um aumento na complexidade de gerenciamento dos itens customizados.

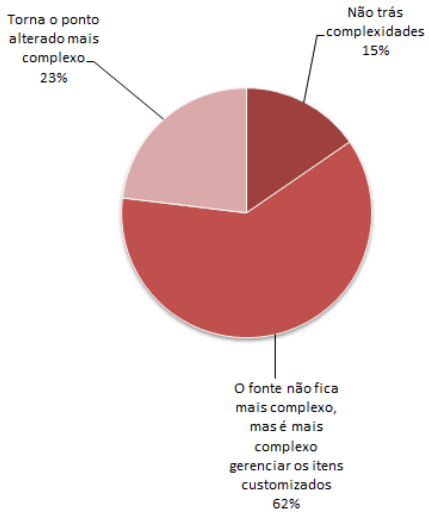

GRÁFICO 6 - Aumento de complexidade do código fonte provocado pela técnica de branchs

Na Tabela 4 - Prós e Contras da Técnica de Versionamento podemos acompanhar os prós e contras apresentados pelas empresas desenvolvedoras de *software* em relação a técnica de versionamento.

Tabela 3 - Prós e Contras da Técnica de Branchs

Prós	**Contras**
O produto se torna mais atraente para o cliente quando pode ser customizado à sua vontade.	A complexidade aumenta com regras de negócios específicas de cada customização.
Redução do custo total de desenvolvimento do *software* e qualidade final é alta após processo de QA.	Precisam ser feito mais testes e um longo processo de QA. Mas a qualidade final é alta.
Mantém um possível problema isolado a um cliente específico.	É preciso manter várias versões de um mesmo código.
Já que é um SaaS, separando o código desde a origem, consigo garantir que clientes não customizados sofram bugs relacionados a customização. Isso é segurança pra mim e para os clientes.	Fica mais difícil de testar e de fazer deploy. No final, acho que valerá a pena.
rápida entrega	difícil manutenção
Não vejo	Testes replicados Perca de tempo para rastreamento de alterações Alto custo de configuração (SCM)
Cada cliente recebe o que solicitou.	Um erro comum encontrado deverá ser corrigido nos fontes de todos os clientes.
Cada cliente é independente na customização do softaware.	Uma implementação feita em um cliente deve ser replicada para todos os clientes caso se queira essa funcionalidade.
Adaptação do *software* a necessidade do cliente	Retrabalho
Como o *software* é feito sob demanda, ele atende exatamente a necessidade do cliênte!	Se ouver migração de dados havera um trabalho maior para transformar dados para base nova na migração.
Facilidade de correção	Gerenciar muitas branchs
Agregar valor e revenda do produto customizado.	Tempo para implementação e elaboração

As customizações ficam isoladas e não ocorre o risco de afetar outros clientes.	Modificações na parte "geral" do sistema, que seriam benéficas a outros clientes, são mais difíceis de serem implantadas e gerenciadas.

4.1.2.2 Versionamento

A técnica de versionamento segundo Coelho (COELHO, 2007), consiste em generalizar as solicitações dos clientes de forma que possa ser implementada para o benefício de todos os clientes, e conforme observado no GRÁFICO 3 - Técnicas de customização adotadas, é a segunda técnica de customização mais utilizada pelas empresas desenvolvedoras de *software*.

A técnica de versionamento segundo pesquisa (GRÁFICO 7 - Incidências de erro na técnica de versionamento), tem apresentado bons resultados com relação a qualidade do produto, sendo que 14% das empresas apontam que esta técnica não contribui para o aumento dos erros de sistema, e 72% apontam que os erros ocorrem de forma ocasional. Porém conforme observamos no GRÁFICO 8 - Impactos dos erros provocados pela utilização da técnica de versionamento, quando estes erros ocorrem, em 57% das vezes afetam a todos os clientes da empresa, e em 29% afeta apenas o cliente que contratou a customização.

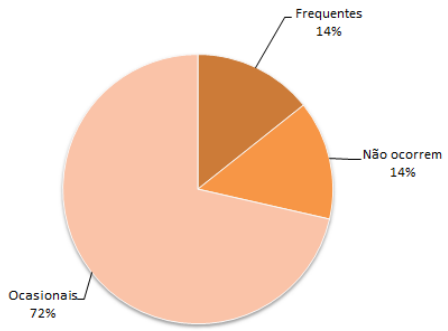

GRÁFICO 7 - Incidências de erro na técnica de versionamento

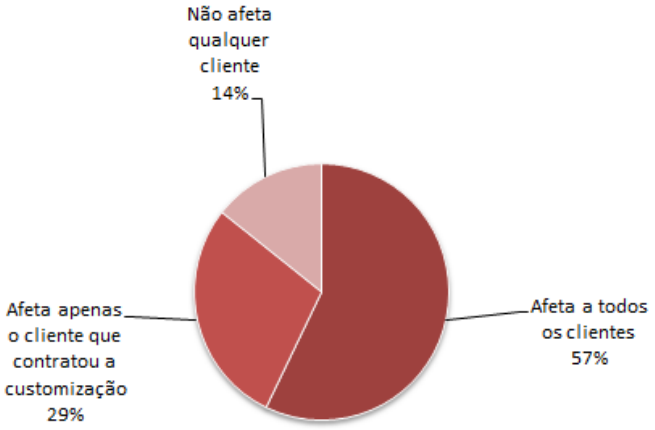

GRÁFICO 8 - Impactos dos erros provocados pela utilização da técnica de versionamento

Com relação a qualidade do código fonte para manutenção futura, é verificado no GRÁFICO 9 - Aumento de complexidade do código fonte provocado pela técnica de versionamento que em 100% dos casos não percebido um aumento da complexidade de manutenção do código fonte, sendo que em apenas 29% das vezes é percebido um aumento na complexidade de gerenciamento dos itens customizados.

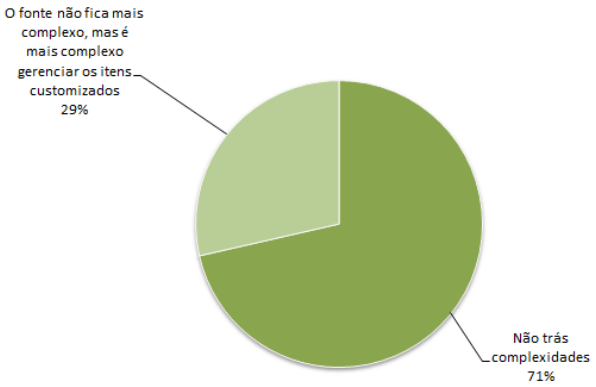

GRÁFICO 9 - Aumento de complexidade do código fonte provocado pela técnica de versionamento

Na Tabela 4 - Prós e Contras da Técnica de Versionamentopodemos acompanhar os prós e contras apresentados pelas empresas desenvolvedoras de *software* em relação a técnica de versionamento.

Tabela 4 - Prós e Contras da Técnica de Versionamento

Prós	Contras
Baixa complexidade do código fonte	Clientes que necessitam de situações muito específicas não são atendidos
As novas funcionalidades ajudam na evolução do produto e com isto a conquistar mais clientes	Funcionalidades que atendem a menos clientes são implementadas por último
Facilidade de manutenção	

4.1.2.3 Fluxo Condicional

A técnica de criação de fluxos condicionais consiste na inserção de código que proporciona caminhos diferenciados no sistema baseado em identificadores para uma funcionalidade ou para um cliente em específico (COELHO, 2007).

Quanto a qualidade do produto, podemos observar através do GRÁFICO 10- Incidências de erro na técnica de fluxos condicionais que a aparição de erros de customização devido a esta técnica é classificado por 20% das empresas desenvolvedoras de *software* como sendo muito frequentes, e 80% os apontaram como sendo ocasionais. Sendo que podemos observar no GRÁFICO 11- Impactos dos erros provocados pela utilização da técnica de fluxos condicionais quem em 60% das situações afeta a todos os clientes, e em 40% das vezes afetam o cliente que contratou a customização.

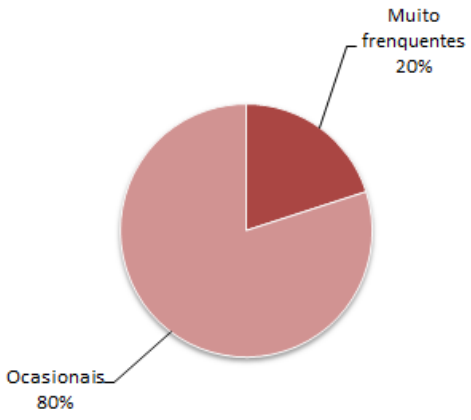

GRÁFICO 10- Incidências de erro na técnica de fluxos condicionais

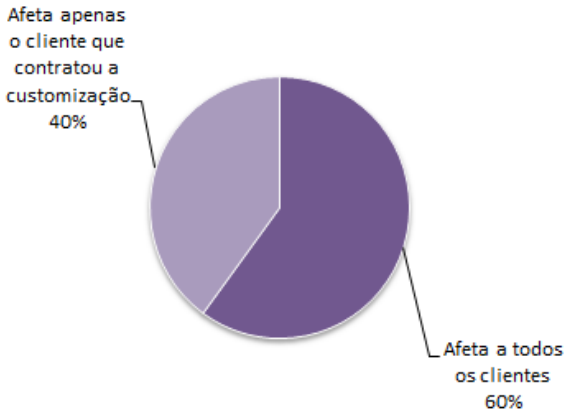

GRÁFICO 11- Impactos dos erros provocados pela utilização da técnica de fluxos condicionais

Conforme observado no GRÁFICO 12 - Aumento de complexidade do código fonte provocado pela técnica de fluxo condicional, para 20% das empresas desenvolvedoras de *software*, esta técnica torna o código fonte mais complexo, enquanto 60% das empresas dizem não perceber aumento da complexidade do código fonte, mas sim um aumento na complexidade em se gerenciar os itens customizados.

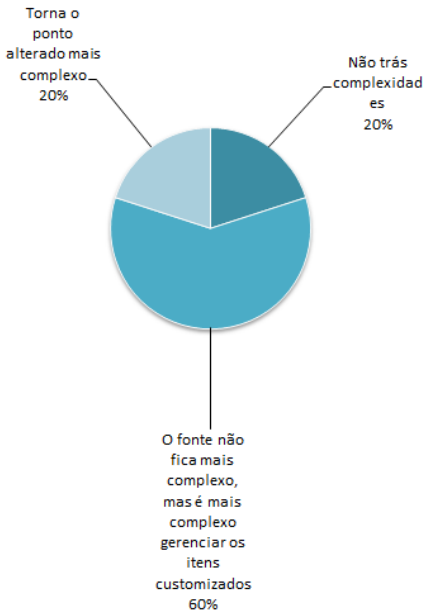

GRÁFICO 12 - Aumento de complexidade do código fonte provocado pela técnica de fluxo condicional

Abaixo, na Tabela 5 - Prós e Contras da Técnica de Fluxo Condicional, podemos acompanhar os prós e contras apresentados pelas empresas desenvolvedoras de *software* em relação a técnica de fluxo condicional.

Tabela 5 - Prós e Contras da Técnica de Fluxo Condicional

Prós	Contras
Facilidade de Implementar	Dificuldade de controlar os recursos específicos e gerais
Entrega rápida	Dificuldade de gerenciamento e rastreabilidade dos requisitos e testes
	Erros afetam um grande numero de clientes

4.1.2.4 Uso de Linguagem de customização

A utilização da técnica de customização do *software* com a utilização de uma linguagem de extensão específica como macros do Microsoft Office, a linguagem ABAP da SAP, ou o ADVPL da Microsiga, permitem ao cliente estender ou customizar as funcionalidades de uma aplicação sem a necessidade de se recompilar o *software* (COELHO, 2007), e com isto traz diversas vantagens nos quesitos de impacto, conforme podemos observar no GRÁFICO 13- Incidência de erro na técnica de utilização de linguagem de programação. Pode-se observar que 20% das empresas desenvolvedoras de *software* apontam que não são observados impactos devido aos erros, enquanto os demais 80% dizem ocorrer impactos ocasionais, das quais, conforme GRÁFICO 14 - Impactos dos erros provocados pela utilização da técnica utilização de linguagem de programação, em 40% das vezes não chega a afetar nenhum cliente, e nos demais casos, afetam apenas os clientes que contrataram a customização do *software*.

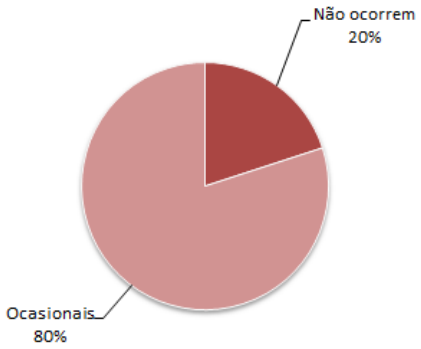

GRÁFICO 13- Incidência de erro na técnica de utilização de linguagem de programação

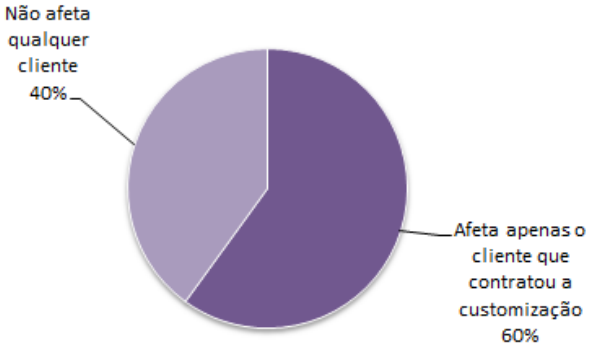

GRÁFICO 14 - Impactos dos erros provocados pela utilização da técnica utilização de linguagem de programação

O GRÁFICO 15 - Aumento de complexidade do código fonte provocado pela técnica de utilização de linguagem de programação nos mostra que em relação a complexidade do código fonte, 40% das empresas desenvolvedoras de *software* dizem perceber o aumento da complexidade do código fonte no ponto modificado, enquanto 60% dizem não perceber tais alterações, sendo que 40% das empresas dizem sofrer com o aumento da complexidade na gestão dos itens customizados.

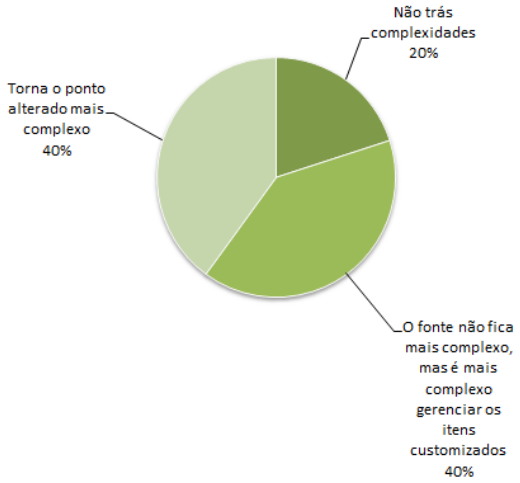

GRÁFICO 15 - Aumento de complexidade do código fonte provocado pela técnica de utilização de linguagem de programação

Abaixo, na Tabela 6 - Prós e Contras da Técnica de Utilização de Linguagem de Programação, podemos acompanhar os prós e contras apresentados pelas empresas desenvolvedoras de *software* em relação a técnica de fluxo condicional.

Tabela 6 - Prós e Contras da Técnica de Utilização de Linguagem de Programação

Prós	Contras
Adequação a regra de negócio do cliente.	Dificuldade de manutenção.
Maior facilidade de implementação	Pela facilidade de se customizar, esse modelo é utilizado de forma abusiva carregando o cliente de customizações desnecessárias.
Desacoplamento.	Alto custo.
	Como os aplicativos são voltados para clientes específicos o reaproveitamento deles é pequeno.
	Mudanças no processo de negócio do cliente podem afetar as funcionalidades implementadas no *software*, podendo gerar necessidade de alterações nas regras de negócio no código da aplicação.

5 PROPOSTA DE ARQUITETURA PARA CUSTOMIZAÇÃO

A arquitetura aqui proposta não visa a resolução de todos os problemas apontados no decorrer deste trabalho em relação a customização de *software*. Pretende apenas demonstrar uma opção focada em reduzir os impactos de erros derivados destas customizações nos clientes, e permitir que customizações não generalizáveis possam ser implementadas sem comprometer a complexidade e qualidade do código fonte do produto.

5.1 Identificação do contexto

Nosso sistema deve possuir um contexto de execução que identifique o que trataremos por *CustomizationType*, que é uma característica que identifique no sistema qual customização será utilizada naquele momento, o que pode ser um código de cliente, uma chave de acesso, ou qualquer outro identificador único para aquela customização.

Nosso gerenciador de contexto da aplicação será tratado no decorrer deste trabalho com o nome *AppContext*.

Vamos contemplar a caráter de exemplo o cenário de comportamento do sistema, que compreende a camada de negócios com serviços, validadores, Objetos de Domínio, dentre outros, sendo explanados com exemplos em C# .Net Framework. Contudo os conceitos aqui apresentados não se prendem a utilização de tecnologias Microsoft ou apenas a sistemas web, podendo ser adotadas em quaisquer linguagens e plataformas que usem linguagens de alto nível, como por exemplo Java.

5.2 Inserindo a Injeção de Comportamento

Para que a mudança de comportamento possa ser aplicada dinamicamente, é necessário que o sistema utilize algum container para inversão de controle, como o MS Unity ou o Spring.

A solução consiste em alterar o método responsável por resolver as dependências da Inversão de Controle, que iremos tratar com o nome *resolver,* para que antes de executar o método "*resolve*", verifique antes qual o tipo está sendo passado e tente localizar o tipo correspondente a ele, baseando-se no *CustomizationType* dentro do *AppContext*, sem que haja a necessidade de criar um mapeamento exclusivo para isto nas configurações do container.

5.3 Modificando o Resolver

Quando o método *resolve* for chamado, a primeira coisa que ele deve fazer é buscar o *CustomizationType* dentro do *AppContext*. De posse do tipo de customização corrente, o *resolver* irá tentar carregar um tipo, que é o resultado da concatenação do nome do tipo solicitado com o *CustomizationType* do contexto:

```
var newTypeName = typeof(T).FullName +
     AuthContext.CurrentContext.CurrentUser.Company.Parameters.AccessType;

Type newType = Type.GetType(newTypeName);
```

O tipo customizado deve estar no mesmo espaço de trabalho que o tipo originalmente passado ao método *resolve*. Após estes passos, caso o *newType* seja um objeto nulo, o *resolver* irá recuperar o objeto instanciado baseado no próprio tipo

enviado no parâmetro genérico (<T>). Caso contrário, ele usa *reflection* para carregar o objeto customizado.

Implementação do Método:

```
private static T Resolve<T>()
{
    object obj = null;

    var newTypeName = typeof(T).FullName +
    AppContext.CurrentCompany.CustomizationType;
    Type newType = Type.GetType(newTypeName);

    if (newType != null)
        obj = _dependencyResolver.GetType()
        .GetMethod("Resolve")
        .MakeGenericMethod(newType)
        .Invoke(_dependencyResolver, null);
    if (obj == null)
        obj = _dependencyResolver.Resolve<T>();

    return (T)obj;
}
```

Neste exemplo, caso o contexto ativo for o de uma empresa “Empresa A” (*CustomizationType = “CustomerA”*) que possui uma customização para o serviço de usuário (*UserService*), o método *resolve* irá procurar por uma classe *UserServiceCustomerA* no mesmo espaço de trabalho de *UserService*.

Neste mesmo exemplo, caso o contexto ativo seja de uma empresa “Empresa B” *(CustomizationType = “CustomerB”)* que não possui customização em *UserService* ou uma “Empresa C” que não possui um *customizationType*, o sistema irá retornar o “*UserService*” que é o comportamento padrão do sistema.

5.4 Criando customizações

As customizações são feitas criando uma nova classe que herde da classe a ser customizada, sendo que seu espaço de trabalho deve ser o mesmo espaço de

trabalho da classe a ser customizada, e seu nome seja a junção do nome da classe a ser customizada com o nome do *CustomizationType*:

Para customizar a classe de validação de empresa (CompanyValidator), adicionando novas validações específicas para um cliente, teremos por exemplo:

```
namespace Sample.Core.Validation
{
    public class CompanyValidatorCompanyA : CompanyValidator
    {
        public override bool ValidateRemove(Company toBeRemoved,
                    out IList<string> brokenRules)
        {
            base.ValidateRemove(toBeRemoved, out brokenRules);

            […]//condições de validação
            brokenRules.Add(CompanyAResources.RootCompanyCantBeDeleted);

            return brokenRules.Count == 0;
        }
    }
}
```

5.5 Organizando as customizações

Como os dois principais objetivos são facilitar a criação da customização e fazer com que ela não interfira no funcionamento padrão do sistema e em customizações de outros clientes, vamos criar uma biblioteca de classes cujo nome seja composto pelo nome da aplicação concatenada com "*.Custom*". Neste trabalho é tratada com o nome *Sample.Custom*.

Nesta biblioteca há apenas uma classe, chamada *SystemCustomTypes*, que é a classe que armazena os nomes das customizações disponíveis no sistema. Além disto, deve-se criar uma biblioteca cujo prefixo do nome é o nome da biblioteca que será customizada, seguida de "*.Custom*".

No exemplo abaixo (Figura 3 - Árvore de projetos da solução) mostramos uma customização em classes do pacote "*Sample.Core*" do sistema.

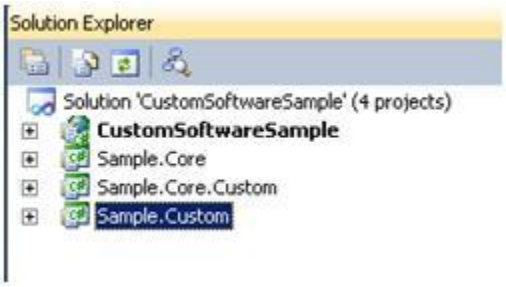

Figura 3 - Árvore de projetos da solução

Isto permite que a inclusão de uma nova customização não implique na geração de uma nova versão do sistema, bastando apenas encaminhar ao cliente que solicitou a nova customização, ou substituir apenas a biblioteca *Sample.Custom* (pois nela está o código de autorização dele) e *Sample.Core.Custom* (com a classe modificada)

Dentro da biblioteca *Sample.Core.Custom*, as classes estão organizadas de forma idêntica as classes de *Sample.Core*, incluindo os mesmos espaços de trabalho para que o Inversor de Controle consiga localizar a classe modificada.

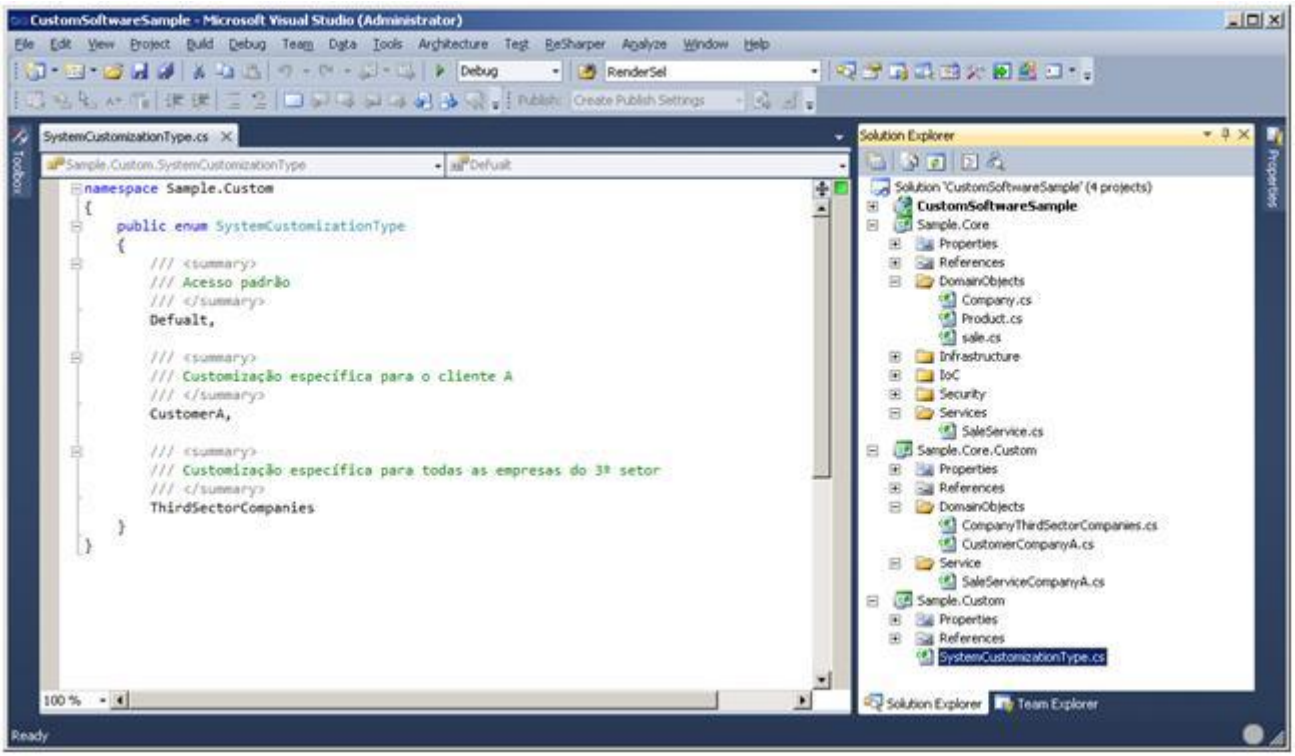

Figura 4 - Visão geral das classes de domínio da solução

6 RESULTADOS

O modelo proposto no capitulo 5 deste trabalho foi aplicado e testado por uma empresa que será identificada apenas por Empresa Desenvolvedora, empresa da cidade de Belo Horizonte em um projeto de médio onde atuaram 5 profissionais, sendo eles 3 desenvolvedores *seniores*, dois arquitetos de software *seniores*.

O projeto teve duração de 6 meses e por motivos de sigilo de negócio, o nome do projeto não poderá ser divulgado, sendo identificado neste trabalho apenas como **“Sistema”.**

A técnica foi usada no sistema para diferenciar o comportamento do software quando este é:

- Empresas de um cliente
- A empresa que administra o sistema
- Empresas de teste e demonstração

6.1 Sobre o sistema

O Sistema é um software desenvolvido para a plataforma web e vendido na forma de serviço pela Empresa Desenvolvedora, onde diversas empresas contratam estes serviços.

Ao se cadastrar no Sistema, as empresas ganham um identificador único que é utilizado sempre que um usuário da empresa tem que acessar o Sistema, informando sempre o código da instancia, o nome do usuário e a senha.

O sistema possui um cadastro de empresas, usuários, alguns ativos específicos, monitoramento de operações e diversas configurações, dentre as quais perfis de usuários, permissões de acesso, dentre outros.

6.2 Utilização do Modelo no sistema

O modelo foi aplicado para diferenciar, no momento em que o usuário acessa o sistema, qual o tipo de acesso sua empresa possui, baseado em uma característica que foi informada no momento de seu cadastro.

O desenvolvimento inicial, tido como fluxo convencional, foi o do acesso do tipo “Cliente”, sendo utilizadas o modelo descrito neste trabalho para diferenciar a “empresa raiz” (que representa a Prime Systems, administradora do sistema) e a “Empresa Falsa”, que é usada para testes e demonstrações.

6.2.1 Empresa Raiz

Quando utilizando a empresa Raiz, não só as telas de configuração e cadastro do sistema são diferenciadas como as opções do menu de acesso possuem itens diferenciados.

6.2.2 Empresa Falsa

Quando utilizando uma empresa sinalizada como falsa, as operações realizadas por esta empresa têm tratamentos diferenciados pelas classes responsáveis por prover os serviços de persistência e validação, além de exibições de informações diferenciadas pelo sistema.

6.3 Opinião da equipe sobre o modelo

De forma geral a equipe avaliou positivamente o modelo e a técnica abordadas, porém ressaltam a necessidade de mais testes e em ambientes mais dinâmicos como customizações específicas e massivas para clientes.

Abaixo, segue os relatos dos envolvidos no projeto conforme resposta aos formulários do Anexo 3. Por motivo de desligamento da empresa, nem todos os envolvidos puderam ser ouvidos.

- Arquiteto responsável

> “A técnica utilizada foi muito útil frente a demanda de alto grau de customização exigida pelo software. A solução atendeu a essa necessidade de maneira ágil e eficaz”.

- Arquiteto de usabilidade

> “Com a utilização da técnica descrita pelo Charles, tivemos grandes vantagens na implementação de customizações de sistemas, pois sempre que precisamos de uma determinada variação das regras ou mesmo que somente das *Views* não era necessário criar uma série de testes fixos no código. A técnica permitiu uma separação bem clara dos pontos onde o software foi customizado aumentando a organização dos arquivos e otimizando o tempo das manutenções”.

- Gerente de desenvolvimento

> “De maneira geral a técnica se apresentou positiva, principalmente no que se refere ás possibilidades de customização sem que maiores impacos sejam percebidas por clientes não optantes a customização. Importante ressaltar que ainda não temos em escala a solução, contudo, entendo que poderemos perceber bons retornos em escala maior inclusive”.

- Desenvolvedor Sênior

> “Com a utilização da técnica no projeto, ficou mais fácil dar manutenção nas *Views* do sistema e no entendimento da lógica vinculada a ela. Com isso foi eliminada as inúmeras condições dentro das *Views* de mostrar ou não certos campos de acordo com o perfil do usuário”.

7 CONCLUSÃO

A simples adaptação na Inversão de Controle nos permite criar uma customização de uma característica comportamental do sistema, seja ela qual for, de forma bem simples para o desenvolvedor, e sem a necessidade de alterações na classe com o comportamento padrão do sistema, reduzindo impactos durante o processo.

Este arranjo de práticas não soluciona todos os problemas acerca da customização de *software*, porém fornece uma alternativa robusta para se atingir qualidade e redução dos impactos nos clientes.

Diversas outras variações podem ser adotadas diante desta abordagem, de forma que ela pode ser refinada e adaptada aos diferentes cenários que possam vir a existir.

8 REFERÊNCIAS

ALEXANDER, C. I. S. . S. M. **The Timeless Way of Building**. New York: Oxford University Press, 1979.

BROOKS, F. P. **The mythical man-month:** essays on *software* engineering. 2ª. ed. [S.l.]: Addison-Wesley, 1995.

COELHO, O. P. Técnicas para Customização de *Software*. **Microsoft MSDN Brasil**, 2007. Disponivel em: <http://www.microsoft.com/brasil/msdn/tecnologias/arquitetura/Customizacao_*Software*.mspx>. Acesso em: 16 Janeiro 2012.

FOWLER, M. Inversion of Control Containers and the Dependency Injection, 2004. Disponivel em: <http://martinfowler.com/articles/injection.html>. Acesso em: 20 Janeiro 2012.

GOUSSET, M. et al. **Professional Application Lifecycle**. 1ª. ed. Indiana: Wiley Publishing, 2012.

LEHMAN, M. M. Rules and Tools for *Software* Evolution Planning and Management. **Imperial College - Department of Computing**, 2000. Disponivel em: <http://www.doc.ic.ac.uk/~mml/feast2/papers/pdf/611_2.pdf>. Acesso em: 15 Janeiro 2012.

LEHMAN, M. M. Feedback, Evolution And *Software* Technology - Brief Introduction. **FEAST**, 2001. Disponivel em: <http://www.doc.ic.ac.uk/~mml/feast/>. Acesso em: 15 Janeiro 2012.

MARTIN, R. C. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf. **Object Mentor**, 2000. Disponivel em:

<http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf>. Acesso em: 24 Janeiro 2012.

NET FRAMEWORK DEVELOPER'S GUIDE..NET Framework Developer's Guide: Reflection Overview. **MSDN**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/f7ykdhsy(VS.71).aspx>. Acesso em: 2010 julho 10.

PRESSMAN, R. S. **Engenharia de *Software***. 3ª. ed. São Paulo: MAKRON Books do Brasil, 1995.

SHALLOWAY, A. **Design patterns explained:** a new perspective on object-oriented design. New York: Addison-Wesley Publishing Co., 2002.

SOMMERVILE, I. **Engenharia de *Software***. 8ª. ed. São Paulo: Pearson, 2007.

Anexo 1 – Formulário de Pesquisa sobre a forma atual de customização de *software*

Pesquisa Sobre Customização de *Software*

Disponibilizado no formato digital através do endereço eletrônico:
https://docs.google.com/spreadsheet/viewform?hl=pt_BR&formkey=dGxVMFJsOUFBVVpwYkFUY2VPZElmOUE6MQ#gid=0

Esta pesquisa visa mapear as formas mais comuns utilizadas pelas empresas de *software* para customização de seus sistemas para clientes específicos. As informações postadas neste formulário são anônimas e serão divulgadas como conteúdo da monografia.

Este formulário foi criado como insumo para a monografia de pós-graduação do Instituto de Ciências Exatas - Departamento de Ciências da Computação da Universidade Federal de Minas Gerais: Curso de Especialização em Engenharia de *Software* - Turma 17 - Aluno: Charles Wellington de Oliveira Fortes

*Obrigatório

---

1. Sua empresa trabalha com *software* na forma de: *
   ( ) Produtos (o cliente compra o *software*)
   ( ) Serviços (o cliente paga mensalmente para utilizar o serviço)
   ( ) Outro (descreva):

2. O *software* produzido por sua empresa é: *
   ( ) Desktop
   ( ) Web
   ( ) Mobile
   ( ) Outro (descreva):

3. Descreva como são implementadas as customizações em sua empresa: *

(Exemplo: "é dado um código a cada cliente e nos pontos necessários são criados condicionais (SE (CLIENTE == ABC) ENTÃO { Faça(); })" -- "para cada cliente fazemos uma cópia do fonte e implementamos suas solicitações" -- "Usamos orientação a aspectos para adaptar pontos do sistema para aquele cliente" -- "Apenas implementamos o que é de comum interesse a todos os clientes, não havendo trechos específicos")

4. Você considera esta forma de customizar *software* *
   ( ) Muito Boa
   ( ) Boa
   ( ) Ruim
   ( ) Muito Ruim

5. Como você classifica a complexidade que esta forma de customização traz ao fonte do sistema? *
   ( ) Torna o ponto alterado mais complexo
   ( ) O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados
   ( ) Não trás complexidades
   ( ) Outro (descreva):

6. Quanto aos erros apresentados no *software* devido a estas implementações, você os classifica como: *
   ( ) Muito frenquentes
   ( ) Frequentes
   ( ) Ocasionais
   ( ) Não ocorrem

7. Quando aparecem erros causados pelas customizações, estes comumente afetam: *
   ( ) Afeta apenas o cliente que contratou a customização
   ( ) Não afeta qualquer cliente
   ( ) Afeta a todos os clientes

8. Como você classifica a dificuldade de cobrir as funcionalidades do sistema que possuem customizações específicas: *
   ( ) Indiferente
   ( ) Um pouco mais difícil de testar
   ( ) Muito mais difícil de testar

9. Informe pelo menos um ponto positivo desta forma de implementação *

10. Informe pelo menos um ponto negativo desta forma de implementação *

Anexo 2 – Resultado da Pesquisa sobre a forma atual de customização de *software*

Tipo Sistema	Ambiete	Técnica de Customização	Avaliação da Técnica	Classificação de Complexidade	Incidencia de erros	Impacto dos erros	Complexidade de Testes	Prós	Contras
Produtos (o cliente compra o *software*)	Desktop	Pontos de Extenção	Boa	Não trás complexidades	Muito frenquentes	Afeta a todos os clientes	Muito mais difícil de testar	Evita testes de condições excessivos.	É necessário que o cliente, pelo menos uma vez, tenha acesso à internet
Produtos (o cliente compra o *software*)	Desktop	Parametrização	Ruim	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta apenas o cliente que contratou a customização	Um pouco mais difícil de testar	O cliente tem o que ele precisa, sem ser afetado pelos requisitos específicos de outros clientes.	Várias versões do mesmo *software* têm que ser mantidas em paralelo, o que gera muito retrabalho e propensão a erros devido a implementações que deveriam ser comuns mas que podem não constar em uma ou outra versão.
O serviço é pré-pago. O cliente compra crédito	Web	Parametrização	Boa	O fonte não fica mais complexo, mas é mais comple	Não ocorrem	Não afeta qualquer cliente	Indiferente	As mudanças são extremamente pontuais.	Geralmente é feito com cópias de arquivos, o que torna o versioname

s para utilizar.				xo gerenciar os itens customizados					nto chato.
Produtos (o cliente compra o *software*)	Desktop	Branchs	Ruim	Torna o ponto alterado mais complexo	Frequentes	Afeta a todos os clientes	Indiferente	O produto se torna mais atraente para o cliente quando pode ser customizado à sua vontade.	A complexidade aumenta com regras de negócios específicas de cada customização.
Consultoria	Web	Branchs	Boa	Torna o ponto alterado mais complexo	Ocasionais	Não afeta qualquer cliente	Um pouco mais difícil de testar	Redução do custo total de desenvolvimento do *software* e qualidade final é alta após processo de QA.	Precisam ser feito mais testes e um longo processo de QA. Mas a qualidade final é alta.
Serviços (o cliente paga mensalmente para utilizar o serviço)	Desktop	Branchs	Ruim	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta apenas o cliente que contratou a customização	Um pouco mais difícil de testar	Mantém um possível problema isolado a um cliente específico.	É preciso manter várias versões de um mesmo código.
Produtos (o cliente compra o *software*)	Web	Branchs	Boa	O fonte não fica mais complexo, mas é mais complexo gerenci	Ocasionais	Afeta apenas o cliente que contratou a customização	Um pouco mais difícil de testar	Já que é um SaaS, separando o código desde a origem, consigo garantir	Fica mais difícil de testar e de fazer deploy. No final, acho que valerá a pena.

				ar os itens customizados				que clientes não customizados sofram bugs relacionados a customização. Isso é segurança pra mim e para os clientes.	
ambos	Web	Branchs	Ruim	Torna o ponto alterado mais complexo	Ocasionais	Afeta apenas o cliente que contratou a customização	Muito mais difícil de testar	rápida entrega	difícil manutenção
Produtos (o cliente compra o *software*)	Web	Branchs	Ruim	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta a todos os clientes	Muito mais difícil de testar	Não vejo	Testes replicados Perca de tempo para rastreamento de alterações Alto custo de configuração (SCM)
Serviços (o cliente paga mensalmente para utilizar o serviço)	Web	Versionamento	Boa	Não trás complexidades	Ocasionais	Afeta a todos os clientes	Indiferente	Como não há pontos específicos para cada cliente, não há milhares de condicionais no meio do fonte do sistema	Clientes que necessitam de situações muito específicas não são atendidos

Jogos sociais	Web	Versionamento	Boa	Não trás complexidades	Ocasionais	Afeta a todos os clientes	Indiferente	Não é preciso ficar fazendo condicionais através de todo o código	Não conseguimos atender a necessidades específicas de cada investidor/ marketing
De uso da própria empresa	Desktop	Versionamento	Boa	Não trás complexidades	Não ocorrem	Afeta a todos os clientes	Indiferente	Não se aplica.	Não se aplica.
Serviços (o cliente paga mensalmente para utilizar o serviço)	Web	Versionamento	Boa	Não trás complexidades	Ocasionais	Não afeta qualquer cliente	Indiferente	A maioria das funcionalidades do *software* foram sugeridas por clientes, o que aumentou o sucesso do produto.	Devido ao grande número de sugestões, foi necessário distinguir a prioridade de cada sugestão e o trabalho para implementalas.
Produtos (o cliente compra o *software*)	Desktop	Fluxo condicional	Ruim	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta a todos os clientes	Indiferente	Fácil de fazer	Dificuldade de controlar os recursos específicos ou nao.
Os dois	Web e Desktop	Fluxo condicional	Muito Ruim	Torna o ponto alterado mais complexo	Muito frenquentes	Afeta a todos os clientes	Muito mais difícil de testar	É mais rápido para entregar para o cliente a implementação	Dificuldade de gerenciamento e rastreabilidade dos requisitos e testes.

								solicitada.	
as duas acima	Web e Desktop	Fluxo condicional	Muito Ruim	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta apenas o cliente que contratou a customização	Indiferente	O dono ganha dinheiro e com meu trabalho dia e noite contribuo para que ele crie toda a familia com o luxo que lhe é devido.	o maior problema de customizaçao não é o cliente que pede e sugere alteraçoes a todo momento, e sim a empresa que só pensa em lucrar e nao tem a responsabilidade de domar o cliente causando transtornos e sofrimentos aos programadores. ...muita customizaçao descaracteriza o sistema e escraviza o programador.
Ambos	Web e Desktop	Uso de Linguagens de Programação	Muito Boa	Torna o ponto alterado mais complexo	Ocasionais	Não afeta qualquer cliente	Indiferente	Adequaçao a regra de negocio do cliente.	Dificuldade de manutençao.

Produtos (o cliente compra o *software*)	Desktop	Uso de Linguagens de Programação	Boa	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta apenas o cliente que contratou a customização	Indiferente	Levando em consideração a arquitetura do sistema a implantação das customizações é facilitada por nosso método.	Pela facilidade de se customizar, esse modelo é utilizado de forma abusiva carregando o cliente de customizações desnecessárias.
Produtos (o cliente compra o *software*)	Desktop	Uso de Linguagens de Programação	Muito Boa	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Não ocorrem	Não afeta qualquer cliente	Indiferente	Solução doméstica.	Alto custo.
Produtos (o cliente compra o *software*)	Desktop	Branchs	Muito Ruim	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta apenas o cliente que contratou a customização	Muito mais difícil de testar	Cada cliente recebe o que solicitou.	Um erro comum encontrado deverá ser corrigido nos fontes de todos os clientes.
Produtos (o cliente compra o *software*)	Desktop	Branchs	Boa	O fonte não fica mais complexo, mas é mais complexo gerenciar os	Muito frenquentes	Afeta apenas o cliente que contratou a customização	Um pouco mais difícil de testar	Cada cliente é independente na customização do softaware.	Uma implementação feita em um cliente deve ser replicada para todos os clientes caso se

				itens customizados					queira essa funcionalidade.
Produtos (o cliente compra o *software*)	Web	Branchs	Muito Boa	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta apenas o cliente que contratou a customização	Indiferente	Adaptação do *software* a necessidade do cliente	Retrabalho
Produtos (o cliente compra o *software*)	Desktop	Fluxo condicional	Boa	Não trás complexidades	Ocasionais	Afeta apenas o cliente que contratou a customização	Indiferente	rapidez	pessoal
Produtos (o cliente compra o *software*)	Web	Branchs	Boa	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta apenas o cliente que contratou a customização	Indiferente	Como o *software* é feito sob demanda, ele atende exatamente a necessidade do cliênte!	Se ouver migração de dados havera um trabalho maior para transformar dados para base nova na migração.
as duas alternativas.	Ambos	Uso de Linguagens de Programação	Muito Boa	Não trás complexidades	Ocasionais	Afeta apenas o cliente que contratou a customização	Indiferente	desacoplamento.	Como os aplicativos são voltados para clientes específicos o reaproveitamento deles é pequeno.

Desenvolve o produto personalizado para o cliente.	Ambos	Uso de Linguagens de Programação	Muito Boa	Torna o ponto alterado mais complexo	Ocasionais	Afeta apenas o cliente que contratou a customização	Indiferente	O *software* entregue é exatamente o que atende as necessidades e desejos do cliente.	Mudanças no processo de negócio do cliente podem afetar as funcionalidades implementadas no *software*, podendo gerar necessidade de alterações nas regras de negócio códigos da aplicação.
Software Próprio	Web	Versionamento	Muito Boa	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Frequentes	Afeta a todos os clientes	Indiferente	Este modelo de controle de versão permite criar ramificações do código fonte, e com isso é possível alterá-lo sem que a versão oficial seja afetada. Depois que o *software* passa por todas as fases (desenvolvimento,	Controle da versão é muito complexo, com isso o risco da versão estável sair com problemas é muito alto.

								teste, etc..) , as alterações vão para o código fonte estável para geração de versão.	
A própria empresa produz	Ambos	Versionamento	Muito Boa	Não trás complexidades	Ocasionais	Afeta apenas o cliente que contratou a customização	Muito mais difícil de testar	Cada departamento tem seu próprio sistema, sendo assim é mais fácil de dar manutenção.	As vezes os desenvolvedores ficam especialistas e viram "pai" do sistema.
Serviços (o cliente paga mensalmente para utilizar o serviço)	Web	Branchs	Boa	Não trás complexidades	Ocasionais	Afeta a todos os clientes	Indiferente	Facilidade de correção	Gerenciar muitas branchs
Serviços (o cliente paga mensalmente para utilizar o serviço)	Ambos	Versionamento	Boa	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta apenas o cliente que contratou a customização	Muito mais difícil de testar	Como há apenas "um cliente" não há como comparar.	Como há apenas "um cliente" não há como comparar.
Produtos (o cliente compra o	Web	Branchs	Muito Boa	Não trás complexidades	Ocasionais	Afeta apenas o cliente que	Um pouco mais difícil de	Agregar valor e revenda do produto	Tempo para implementação e elaboração

software)						contratou a customização	testar	customizado.	
Produtos (o cliente compra o *software*)	Web	Branchs	Ruim	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta apenas o cliente que contratou a customização	Indiferente	As customizações ficam isoladas e não ocorre o risco de afetar outros clientes.	Modificações na parte "geral" do sistema, que seriam benéficas a outros clientes, são mais difíceis de serem implantadas e gerenciadas.
Serviços (o cliente paga mensalmente para utilizar o serviço)	Desktop	Fluxo condicional	Muito Boa	O fonte não fica mais complexo, mas é mais complexo gerenciar os itens customizados	Ocasionais	Afeta a todos os clientes	Indiferente	Toda mudança no *software* é decidido por um comitê de responsabilidade do grupo do cliente, tendo assim mais controle quanto as customizações, que podem ser desenecessarias.	O caso de um erro detectado afetará um grande numero de cliente.
Produtos (o cliente compra o *software*)	Ambos	Versionamento	Ruim	Não trás complexidades	Ocasionais	Não afeta qualquer cliente	Indiferente	Menos complexo.	Abrange menos clientes.

Produtos (o cliente compra o *software*)	Web	Versionamento	Muito Ruim	Não trás complexidades	Ocasionais	Afeta a todos os clientes	Indiferente	É possível manter o foco no produto e não em customizações específicas.	Pode não atender algumas necessidades específicas de um cliente.
Produtos (o cliente compra o *software*)	Web	Versionamento	Boa	Não trás complexidades	Frequentes	Afeta a todos os clientes	Um pouco mais difícil de testar	O cliente ficará mais satisfeito, em ter um *software* que ele tenha melhor compreenção do seu ponto de vista. Facilitando o assim, a melhor adptação do *software* a empresa	Vai chegar um ponto que terá *software* muitos diferentes, dificultando o atualizaçõe s

Anexo 3 – Formulários de avaliação da técnica implementada no projeto da empresa desenvolvedora de softwares

Relatório de utilização da técnica apresentada como modelo na monografia de Pós-Graduação – Especialização em Engenharia de Software DCC – UFMG – Aluno Charles Wellington de Oliveira Fortes – Turma 17 – Aplicada em contexto de projeto real pela empresa F[redacted]s.

Nome Completo: [redacted]

Cargo: Analista Desenvolvedor .NET

Descreva em poucas palavras sua impressão geral sobre a técnica proposta:

Com a utilização da técnica descrita pelo Charles, tivemos grandes vantagens na implementação de customizações do sistema pois sempre que precisava-mos de uma determinada variação das regras ou mesmo que somente das "Vinis" não era necessário criar uma série de Testes fixos no código. A técnica permitiu uma separação bem clara dos pontos onde o sapetware foi customizado aumentando a organização dos arquivos e ~~manutenção~~ otimizando o tempo das manutenções.

Assinatura: [redacted]

Local e Data Belo Horizonte - 23/02/12

Relatório de utilização da técnica apresentada como modelo na monografia de Pós-Graduação – Especialização em Engenharia de Software DCC – UFMG – Aluno Charles Wellington de Oliveira Fortes – Turma 17 – Aplicada em contexto de projeto real pela empresa I[redacted]s.

Nome Completo: [redacted]

Cargo: ARQUITETO DE SOFTWARE

Descreva em poucas palavras sua impressão geral sobre a técnica proposta:

A TECNICA UTILIZADA FOI MUITO UTIL FRENTE A DEMANDA DE ALTO GRAU DE COSTOMIZAÇÃO EXIGIDA PELO SOFTWARE. ~~CONSEGUIMOS ATENDER A ESSA NECESSIDADE DE MANEIRA DE MAU~~

A SOLUÇÃO ATENDEU A ESSA NECESSIDADE DE MANEIRA AGIL E EFICAZ.

Assinatura: [redacted]

Local e Data BELO HORIZONTE - 23/02/12

Relatório de utilização da técnica apresentada como modelo na monografia de Pós-Graduação – Especialização em Engenharia de Software DCC – UFMG – Aluno Charles Wellington de Oliveira Fortes – Turma 17 – Aplicada em contexto de projeto real pela empresa [redacted].

Nome Completo: [redacted]

Cargo: Analista Desenvolvedor

Descreva em poucas palavras sua impressão geral sobre a técnica proposta:

Com a utilização da técnica no projeto, ficou mais fácil dar manutenção nas telas do sistema e no entendimento da lógica vinculadas a elas. Com isso foi eliminado as inúmeras condições dentro das telas de mostrar ou não certos campos de acordo com o perfil do usuário.

Assinatura: [redacted]

Local e Data Belo Horizonte - 24/02/12

Relatório de utilização da técnica apresentada como modelo na monografia de Pós-Graduação – Especialização em Engenharia de Software DCC – UFMG – Aluno Charles Wellington de Oliveira Fortes – Turma 17 – Aplicada em contexto de projeto real pela empresa F[redacted]s.

Nome Completo: [redacted]

Cargo: Gerente de Desenvolvimento.

Descreva em poucas palavras sua impressão geral sobre a técnica proposta:

De maneira geral a técnica se apresentou positiva, principalmente no que se refere às possibilidades de customizações sem que maiores impactos sejam percebidos por clientes não ligados à customização. Importante ressaltar que ainda não testamos em escala a solução, contudo, entendo que podemos perceber bons retornos em escala maior inclusive.

Assinatura: [redacted]

Local e Data Belo Horizonte - 23/02/2012