UMA LINHA DE PRODUTO DE SOFTWARE PARA SISTEMAS INTEGRADOS DE GESTÃO EMPRESARIAL: UM ESTUDO DE CASO NA INDÚSTRIA

Moisés Pôrto Rodrigues

Kleinner Silva Farias de Oliveira

Resumo: Desenvolver Sistemas Integrados de Gestão Empresarial (SIGE) ainda é considerado custo e propenso a erros. Isso pode ser explicado pela alta complexidade e dificuldade para entender os processos empresariais, pelas mudanças constantes dos requisitos, pela dificuldade de identificar e reusar de forma sistemática os módulos do sistema, bem como pela imprecisão na identificação das variabilidades do sistema. Este trabalho, portanto, tenta mitigar esta problemática ao projetar e implementar um SIGE seguindo os conceitos de linha de produto de software, os quais visam promover o reuso sistemático dos módulos do sistema, minimizar o esforço de desenvolvimento, e potencializar o gerenciamento das variabilidades. A linha de produto proposta foi extraída, bem como avaliada, a partir de um SIGE realístico da indústria, o que permitiu tirar conclusões mais prática sobre como potencializar o reuso aplicando conceitos de linha de produto de software em sistemas reais da indústria.

Palavras-Chave: Linha de Produtos de Software. Sistemas Integrados de Gestão Empresarial. Engenharia de Software.

1 INTRODUÇÃO

Conforme Clements e Northrop (2001), a Linha de Produtos de Software (LPS) trata-se de uma nova abordagem de reutilização em Engenharia de Software que vem obtendo muita atenção na indústria e na academia. Ela tem como objetivo central aumentar a produtividade ao viabilizar o reuso sistemático dos requisitos funcionais e não funcionais dos sistemas, também conhecidos como *features*. A identificação e gerenciamento de tais *features* do sistema viabiliza o desenvolvimento de famílias de sistemas, ao invés do desenvolvimento de aplicações individuais. Os produtos de uma linha possuem um núcleo comum e partes não comuns que são responsáveis pela diferenciação dos mesmos, as suas variabilidades. Estes produtos são construídos e baseados em um conjunto de componentes de software que são responsáveis por implementar as features distintas da linha de produto. Rezende e Abreu (2001) dizem que principais motivos para a indústria adotar LPS são a construção de produtos de software com cada vez

mais qualidade, custo reduzido, adaptabilidade a mudanças e rapidez de colocação no mercado.

O ERP, por sua vez, foi definido como um conjunto de sistemas de negócio, permitindo a automatização e integração da maioria dos processos de negócio, através de um conjunto de atividades executadas por software multimodular integrando as operações da empresa. Apesar de cada vez mais sistemas ERP tenham sido projetados e comercializados, o seu desenvolvimento ainda é custoso e frequentemente extrapola os prazos e orçamentos estabelecidos. Dito isso, fica justificada a necessidade de estudos que tentem resolver, ou mesmo minimizar, o desafio de desenvolver sistemas ERP de alta qualidade com prazos cada vez menores e com orçamento curtos. Portanto, a grande contribuição deste trabalho é usar LPS para desenvolver famílias de sistemas ERP. Isso permitirá uma melhor rastreabilidade entre os artefatos e features, permitirá o reuso sistemático de seus módulos, permitindo produzir novos produtos mais rapidamente através do gerenciamento efetivo de suas partes variantes. (COLÂNGELO FILHO, 2001).

Sendo assim, a Seção 1.1 apresenta a formulação do problema atacado neste trabalho. A Seção 1.2 especifica os objetivos gerais e específicos que se deseja atingir. Seção 1.3 descreve brevemente quais métodos de estudo serão utilizados para atingir os objetivos traçados, bem como define a maneira como tais estudos serão escutados. Visando posicionar o trabalho na comunidade científica, a Seção 1.4 caracteriza as contribuições científicas do trabalho e descreve o impacto da pesquisa no estado da prática de desenvolvimento de sistemas integrados de gestão empresarial. Por fim, a Seção 1.5 descreve como cada Capítulo do trabalho está organizado.

1.1 Formulação do problema

Para promover o desenvolvimento de ERP com alta qualidade em um menor espaço de tempo e com menor custo, é preciso atacar três problemas críticos, os quais são descritos a seguir.

1.1.1 Representação inadequada das features de ERPs

Dado que usualmente ERPs compartilham de um núcleo comum, os desenvolvedores precisam compreender de forma adequada quais são as features desse núcleo e o que o torna um produto único, isto é, suas particularidades. Porém, na prática, o problema é que os desenvolvedores acabam não compreendendo as *features* (seus requisitos funcionais e não funcionais do sistema), bem como a variabilidade entre os produtos. Isso pode explicado por alguns motivos: (1) a inadequada representação das features das aplicações e das dependências entre elas; (2) a não especificação das diferenças entre os produtos, bem como das suas variabilidades; e (3) a indefinição de quais features são necessárias para configurar um determinado produto. Portanto, tem-se, de fato, uma um problema na representação das features que compõem os produtos de ERPs, bem como na especificação das variabilidades entre os produtos.

1.1.2 Imprecisão na rastreabilidade entre features, artefatos e módulos de ERPs

O ganho de produtividade na manutenção e na evolução de ERP é alcançado quando os desenvolvedores compreendem claramente como as *features* do ERP são modularizadas em nível de projeto e implementação. Infelizmente, esse mapeamento raramente é realizado e mantido. Sendo assim, é particularmente desafiante para os desenvolvedores promover um reuso sistemático de tais features, e dos módulos que pertencentes a um ERP; dificultando, inevitavelmente, as atividades de evolução e manutenção das aplicações.

Esta falta de entendimento pode ser explicada por algumas razões: (1) as features que formam a família de produtos são raramente definidas e mapeadas para os artefatos que as implementam. Consequentemente, os desenvolvedores frequentemente não conseguem rastrear como requisitos, especificados em casos de uso, são contemplados em diagramas estruturais e comportamentais da UML. Isto é, os desenvolvedores são incapazes de “navegar” dos artefatos que especificam os requisitos para os artefatos que definem o projeto e a implementação das features, e vice-versa. Portanto, tem-se uma imprecisão na rastreabilidade entre as features e seus artefatos e módulos que as implementam.

1.1.3 Desconhecimento sobre os reais ganhos do uso de LPS para aplicações ERP

Como previamente mencionado, é aconselhável utilizar LPS para superar o desafio de desenvolver ERPs visando baixo custo e menor tempo de produção. Porém, a literatura atual falha ao não fornecer detalhes e evidências que comprovem se o uso de LPS pode trazer (ou não) reais benefícios ao se desenvolver ERP. Por exemplo, é possível que algumas regras de negócio em sistemas ERP não garantam os ganhos esperados com o uso de LPS. Portanto, existe um desconhecimento sobre os reais ganhos do uso de LPS para o desenvolvimento de sistemas ERP.

1.2 Objetivos

O presente trabalho tem como objetivo geral o desenvolvimento de uma linha de produto de software para ERP. Para isso, destaca-se os seguintes objetivos específicos:

1. compreender o estado da arte sobre linha de produto de software;
2. analisar o domínio de aplicação do software de gestão;
3. analisar e projetar uma linha de produto de software;
4. realizar a análise, projeto e implementação de uma linha de produto de software;
5. realizar um estudo de caso para avaliar os benefícios da linha de produto desenvolvida; e
6. gerar lições aprendidas sobre o tema abordado.

1.3 Metodologia do estudo

O trabalho foi realizado em seis etapas, as quais são apresentadas a seguir.

Etapa 1: esta etapa teve como objetivo central realizar a caracterização do problema estudado, entendimento dos aspectos teóricos do trabalho e do estado da arte. Em particular, nessa etapa foram executadas as seguintes atividades: (1) estudo dos aspectos teóricos do trabalho; (2) estudo do estado da arte; (3) formulação do problema estudado; e (4) definição dos objetivos de pesquisa.

Etapa 2: de acordo com o entendimento adquirido na Etapa anterior, a segunda etapa se concentrou na elaboração da linha de produto de software de ERP. Em particular, nessa etapa foram executadas as seguintes atividades: (5)

análise e projeto da LPS; e (6) estudos dos métodos de implementação de variabilidade de LPS.

Etapa 3: esta etapa se concentrou na escrita da monografia, sendo executada em paralelo as etapas 1 e 2. Em particular, as seguintes atividades foram executadas: (7) redação da monografia; e (8) revisão e entrega do texto.

Etapa 4: o foco desta etapa foi no desenvolvimento da linha de produto idealizada nas Etapas 1 e 2. A atividade realizada nesta etapa foi a (9) implementação da linha de produto.

Etapa 5: esta etapa avaliou a linha de produto desenvolvida. Para isto, duas atividades foram executadas: (10) a realização de um estudo de caso exploratório com o objetivo de extrair a linha de produto a partir de um ERP proprietário e amplamente utilizado na indústria; e (11) avaliação dos produtos derivados da LPS implementada em termos de modularidade e custo-benefício.

Etapa 6: esta etapa se concentrou na elaboração do artigo científico. À medida que as atividades 9, 10 e 11 foram finalizadas, este artigo foi elaborado com o conteúdo produzido. Em particular, uma macroatividade executada: (12) escrita e entrega do artigo científico.

Por fim, é necessário destacar que todos os métodos de pesquisa utilizados neste trabalho seguirão as recomendações descritas no estado da arte na área de Engenharia de Software Experimental (WOHLIN et al, 2000).

1.4 Contribuições

As contribuições deste trabalho são geradas a partir da execução das etapas desta pesquisa descrita anteriormente. Desse modo, é possível elencar as seguintes contribuições: (1) uma análise do estado da arte na área de LPS de ERP; (2) a análise e o projeto de uma LPS para ERP; (3) uma análise de domínio para a elaboração da LPS para ERP; (4) conhecimento prático sobre os reais benefícios do uso de LPS para minimizar os problemas associados à inadequada representação das features de uma aplicação ERP; (5) redução da imprecisão na rastreabilidade entre as features, os artefatos e os módulos de um ERP; e (6) lições aprendidas identificadas durante a análise, projeto e desenvolvimento da LPS para ERP.

1.5 Organização do trabalho

Este trabalho é organizado da seguinte forma. A Seção 2 apresenta o referencial teórico do trabalho descrevendo os principais conceitos necessário para o bom entendimento dos próximos capítulo. A Seção 3 descreve como a ferramenta do desenvolvida. A Seção 4 descreve os resultados obtidos. A Seção 5 contrasta o trabalho desenvolvido com a literatura atual em LPS para ERP. Por fim, a Seção 6 descreve as considerações finais e os trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este Seção tem como objetivo central descrever os principais conceitos para o projeto de uma LPS de ERP. Para isso, o conteúdo apresentado nas próximas Seções abordará informações sobre linhas de produto de *software* (Seção 2.1), sistema integrados de gestão empresarial (Seção 2.2) e projeto de linhas de *software* com UML (Seção 2.3).

2.1 Linhas de Produto de Software

O conceito de Linhas de Produto de Software (LPS) surge como uma nova metodologia de desenvolvimento de software que visa aumentar a produtividade da organização através do reuso sistemático dos ativos de software. Essa metodologia surge para fazer frente ao reuso *ad-hoc* praticado até então. Clements e Northrop (2001) definem LPS como um conjunto de sistemas de software que compartilham um conjunto comum e gerenciado de características[1], as quais satisfazem necessidades específicas de um segmento de mercado, e que são desenvolvidos de uma forma pré-definida a partir de um conjunto comum de ativos.

Czarnecki e Eisenecker (2000) definem característica como sendo “uma propriedade de um sistema que é relevante para um cliente e que é usada para capturar os aspectos comuns e diferentes entre produtos de uma linha”. Tais características podem ser organizadas em um modelo de características (*feature model*) e são classificadas como, obrigatórias, opcionais e alternativas. O reuso sistemático é atingido quando é possível compreender como uma família de

[1] Do inglês *feature*

sistemas de software compartilham características; e entender como as diferenças entre eles se manifestam.

Kang et al (1990), demonstra o método Feature-Oriented Domain Analysis (FODA), que consiste em focar em representar o domínio do problema através de features e seus relacionamentos. O autor ainda segue, dizendo, que um domínio não precisa, ocorrer em um nível específico de granularidade de software. Um domínio é um conceito mais geral, o qual pode ser alongado para ser aplicar a maior parte de qualquer potencial classe de sistemas.

Conforme Kang et al. (1998), o reuso de artefatos de software é uma das soluções mais promissoras para a crise de software. Descoberta sistemática de pontos comuns entre os produtos de software relacionados e representação destes de uma forma explorável são requisitos técnicos fundamentais para alcançar uma bem sucedida reutilização de software. O autor desenvolveu o método Feature-Oriented Reuse Method (FORM), que FORM é um método sistemático que procura e captura as semelhanças e as diferenças de aplicação de um domínio em termos de *features*, e utiliza os resultados desta análise para desenvolver arquiteturas de domínio e componentes. O modelo que capta as similaridades e as diferenças se chama *feature model*, que: (1) é utilizado para dar suporte tanto para engenharia de artefatos de domínio reutilizáveis quando para desenvolvimento de aplicações utilizando artefatos de domínio; (2) é capaz de representar as características comuns encontradas entre vários sistemas de software, bem como as *features* que diferenciam tais sistemas.

Tomando por base os métodos FODA e FORM, Gomaa (2011) propõe um método de desenvolvimento de LPS baseado em artefatos UML, chamado *Product Line UML-Based Software Engineering* (PLUS). É baseado em um processo de desenvolvimento iterativo e orientado a objetos, chamado *Evolutionary Software Product Line Engineering* (ESPLEP). Esta abordagem ocasiona o fim da distinção entre o desenvolvimento e manutenção de software, permitindo que o sistema possa evoluir por meio de iterações, para isso o sistema deve ser pensado e projetado, com a visão de que estas possíveis mudanças ocorram durante as iterações. O ESPLEP consiste em duas atividades principais: (1) **Engenharia de Domínio** – desenvolvimento da arquitetura que servirá de base para a LPS, definição de características comuns e variáveis da LPS –; e (2) **Engenharia da Aplicação** –

baseado no resultado da Engenharia da Linha de Produtos, são gerados os modelos de linha de produto e arquitetura de acordo com as *features* configuradas. A arquitetura determina qual dos componentes reusáveis são necessários para derivação e configuração do software.

Gomaa (2011), diz que a Engenharia da LPS pode ser dividida em três atividades principais: (a) *requisitos*, que é a definição dos requisitos funcionais, funcionalidades comuns e variáveis dentre os produtos – é feita uma análise de escopo, além do desenvolvimento do modelo de requisitos, que consiste do modelo de casos de uso e modelo de *features* –, caso os requisitos não sejam entendidos claramente, um protótipo descartável pode ser desenvolvido para esclarecimento ; (b) *análise*, que é a realização da decomposição do problema para que se possa entende-lo de uma melhor maneira – é realizada a modelagem estática, representada pelo diagrama de classes e análise de dependência entre features e classes, e a modelagem dinâmica, representados pelos diagramas de sequência e comunicação; e (c) *projeto e desenvolvimento*, que é a sintetização da solução, implementação incremental de componentes. Testes funcionais devem ser realizados a cada iteração de desenvolvimento destes componentes.

2.2 Sistemas Integrados de Gestão Empresarial

Sistemas Integrados de Gestão Empresarial[2] (SIGE), mais conhecidos como *Enterprise Resource Planning* (ERP), podem ser definidos como um conjunto de sistemas de *software* desenvolvido para suportar os diferentes processos de uma empresa e integrar as informações reunidas nestes processos. Davenport (1998) afirma que o ERP é uma solução genérica que reflete uma série de afirmações a respeito do modo de como as organizações operam, ao contrário dos sistemas proprietários, construídos sobre requisitos específicos de uma empresa. Essa contradição gera a imposição dos já padronizados processos estratégicos, culturais e estruturais.

Cavalcanti (2001 apud OLIVEIRA et al., 2005, p. 4650) diz que um sistema ERP pode ser definido como uma solução de software que atende as necessidades

[2] Do inglês *Enterprise Resource Planning* (ERP)

do negócio, levando em consideração a visão do processo de uma organização com a finalidade de encontrar as metas dessa organização, integrando de forma estreita todas as áreas e funções do negócio. Segundo Deloitte (1998), ERP também pode ser definido como um pacote de software de negócios que permita a automatização e integração da maioria dos processos de negócio de uma empresa, compartilhamento de dados e de práticas comuns por toda a empresa, e à produção e acesso às informações em tempo real.

Podemos considerar que a evolução dos sistemas MRP (*Materials Requirements Planning*) e MRP II (*Manufacturing Resources Planning*) – aplicados em empresas de manufatura para controle de estoque e planejamento de recursos para compra e produção –, trouxe um novo conceito de sistemas, o ERP. A falta de suporte ao planejamento de custos e de capacidade, além da inexistência de integração com outras aplicações, eram deficiências dos sistemas MRP (CORRÊA; GIANESI; CAON, 1997 e PADILHA; MARINS, 2005).

Os sistemas ERP se enquadram na definição de pacotes de software, porém possuem uma série de características que os distinguem, além de torná-los mais abrangentes. Souza e Zwicker (2000) dizem que ERP são pacotes de software comerciais que incorporam modelos padronizados de processos de negócio e integram diversas áreas da empresa via fluxo de dados, possuem uma grande abrangência funcional porém requerem procedimento de ajuste.

2.3 Projeto de Linhas de Produto de Software com UML

Para o projeto proposto, serão utilizados quatro artefatos UML para que seja possível descrever a estrutura idealizada. Os artefatos que serão utilizados serão os casos de uso, diagrama de *feature*, diagrama de classe e diagrama de sequência. A descrição destes artefatos será realizada, conforme segue abaixo.

2.3.1 Casos de uso

De acordo com Use (2013), o caso de uso em sua forma mais simples é uma representação da interação do usuário com o sistema e descrevendo as especificações de um caso de uso. Um diagrama de caso de uso pode retratar os diferentes tipos de usuários de um sistema e as várias maneiras que eles interagem

com o sistema. Em UML2 (2013) é descrito que os diagramas de Caso de Uso trazem uma visão global dos requisitos de utilização de um sistema. São úteis em apresentações para gerenciar e/ou projeta partes interessadas, mas para desenvolvimento de software eles proveem um valor muito mais significante porque eles representam o cerne dos requisitos atuais.

2.3.2 Modelo de Feature

Kang et al. (1990), diz que no desenvolvimento de software, um modelo de *features* é uma representação compacta de todos os produtos da SPL em termos de *features*. O objetivo do modelo de *feature* é descrever os requisitos de um domínio específico. O modelo deve abranger o maior número de requisitos quanto possível, para incluir a gama mais completa de recursos e valores de recurso. A implementação específica de um domínio, pode ser pensada como uma instanciação do modelo de recurso, ou um conjunto de valores característicos que descrevem as suas capacidades particulares. Ser capaz de descrever um sistema proposto a um cliente, em termos de características possíveis, que podem ser fornecidas, simplifica o processo de levantamento de requisitos, e podem esclarecer os diversos *trade-off* de decisões implícitos que devem ser efetuadas.

2.3.3 Diagrama de classes

Segundo Ambler (2005), os diagramas de classe mostram as classes de um sistema, as suas relações, operações e atributos. Eles são utilizados para explorar conceitos de um domínio na forma de um modelo, analisar requisitos na forma de um modelo conceitual ou analítico, e retratar o projeto detalhado de sistemas orientados a objeto. Sparks (2001) diz que o diagrama de classe é o principal bloco de modelagem orientado a objeto, e pode ser utilizado para modelagem conceitual genérica da sistemática da aplicação, e, ainda, para modelagem mais detalhada traduzindo os outros artefatos, e ainda podem ser utilizados para modelagem de dados.

2.3.4 Diagrama de sequência

Conforme Barros (1998), o diagrama de sequência mostra a colaboração dinâmica entre os vários objetos de um sistema. O mais importante aspecto deste diagrama é que a partir dele percebe-se a sequência de mensagens enviadas entre os objetos. Gomaa (2011) relata que o diagrama de sequência retrata as interações dos objetos organizados em uma linha de tempo, e que, também, é um diagrama bidimensional, onde os objetos participantes na interação são retratados horizontalmente e a dimensão vertical representa o tempo.

3 PROJETO

O projeto e implementação da LPS para ERP contempla duas etapas. A primeira é a *engenharia de Domínio* que foca na elaboração da descrição dos requisitos, dos casos de uso, do modelo de feature e da configuração dos produtos. A segunda é a *Engenharia da Aplicação* que se concentra na definição da arquitetura, diagrama de classes, diagrama de sequência e mapeamento dos produtos, bem como da implementação da LPS.

3.1 Engenharia de Domínio

As features da LPS foram extraídas analisando um ERP da indústria. Quadro 1 apresenta uma descrição sucinta das features identificadas. As features apresentadas parcialmente no Quadro 1 foram mapeadas 1:1 para os requisitos funcionais e não funcionais do LPS. Isso implicada dizer que para cada feature identificada foi criado um caso de uso e elaborada a sua especificação.

Quadro 1 – Tabela parcial de Casos de uso

Nome	Descrição	Tipo
Cadastrar produtos	Permitir ao usuário registrar produtos no sistema, com todos os dados necessários ao seu escopo.	Opcional
Cadastrar serviços	Permitir ao usuário registrar serviços no sistema, com todos os dados necessários ao seu escopo.	Opcional
Listar produtos	Permitir ao usuário listar todos os produtos registrados no sistema.	Opcional
Listar serviços	Permitir ao usuário listar todos os serviços registrados no sistema.	Opcional
Realizar Entrada de Nota Fiscal	Permitir ao usuário registrar a Entrada de Nota Fiscal de produtos ou serviços adquiridos.	Obrigatório
Realizar Ordem de Compra	Permitir ao usuário registrar uma Ordem de Compra de produtos ou serviços a adquirir.	Opcional
Realizar Venda	Permitir ao usuário registrar uma Venda de produtos ou serviços vendidos.	Obrigatório
Realizar	Permitir ao usuário registrar um Orçamento de produtos ou	Opcional

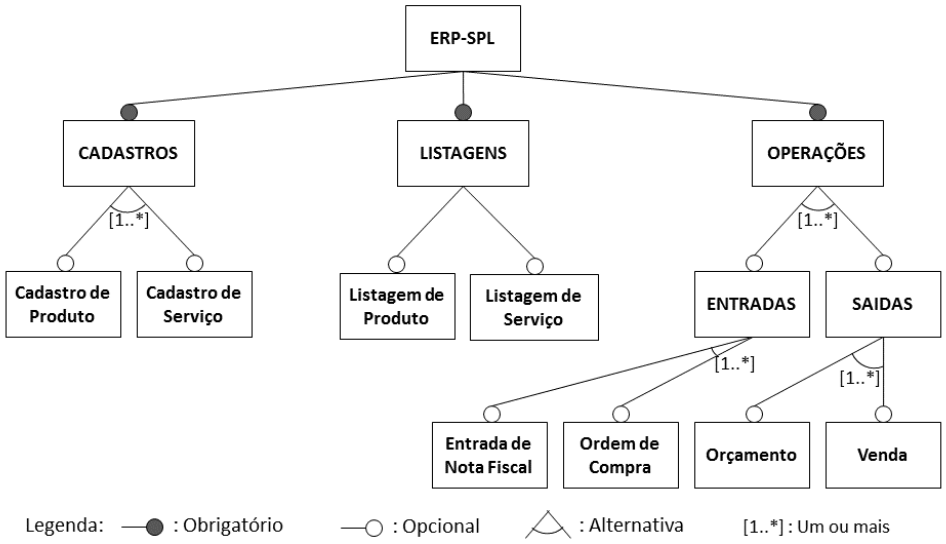
Orçamento	serviços a vender.	

Fonte: Elaborada pelo autor.

Baseando-se nas Descrições dos Requisitos, demonstramos as necessidades básicas em Casos de Uso e seus diagramas. Com a etapa dos Casos de Uso concluída, foi possível visualizar, de uma forma agradável, as necessidades da empresa.

As features identificadas são representadas através de um diagrama de features com o objetivo de representá-las de forma abstrata, bem como identificar explicitamente as suas dependências e suas variabilidades. O Diagrama de Features, resultado desta etapa, é mostrado parcialmente na Figura 1, e por completo no Apêndice A.

Figura 1 - Diagrama Parcial de Features



Fonte: Elaborada pelo autor.

Considerando as features descritas anteriormente, foi realizado um mapeamento de tais features para os possíveis produtos que podem ser extraídos da LPS proposta. Este mapeamento visa identificar de forma explícita quais features são necessárias para gerar um produto em particular. Desse modo, nesta etapa de mapeamento de features em relação os possíveis produtos, o modelo de feature, apresentado anteriormente, desempenhou um papel fundamental, visto que ele potencializou identificar quais versões do sistema podem ser produzidas considerando um conjunto de features comum, e considerando algumas partes variantes. Ao realizar este mapeamento, configura-se os possíveis produtos da LPS.

Com isto em mente, quatro configurações fixas foram idealizadas considerando algumas *features* previamente selecionadas. Uma configuração básica

(C1), sem nenhuma das *features* opcionais selecionadas, uma considerada como intermediária (C2), com as *features* relacionadas ao controle de estoque selecionadas, uma considerada como intermediária (C3), com as *features* relacionadas ao controle financeiro selecionadas, e uma completa (C4), com as principais *features* selecionadas.

As configurações de produto demonstradas colocarão em evidência somente as diferenças em termos de *feature*, previamente citadas, no Quadro 1, e nele serão utilizados alguns marcadores para demonstrar o estado de configuração das *features*: selecionada (S), não selecionada (N) e indisponível (I). No Quadro 2, temos uma visão parcial das configurações – não foram utilizadas todas as features disponíveis –, apenas para elucidação das configurações elaboradas. O quadro completo se encontra no Apêndice B.

Quadro 2 – Mapeamento parcial de Features

Feature	C1	C2	C3	C4
Cadastro de produto	**N**	**N**	**S**	**S**
Cadastro de serviço	**N**	**S**	**N**	**S**
Listagens	**N**	**S**	**S**	**S**
Listagem de produtos	**I**	**N**	**S**	**S**
Listagem de serviços	**I**	**S**	**N**	**S**
Entradas	**N**	**S**	**S**	**S**
Ordens de compra	**I**	**N**	**S**	**S**
Entradas de nota fiscal	**I**	**S**	**S**	**S**
Saídas	**S**	**S**	**S**	**S**
Orçamento	**N**	**N**	**S**	**S**
Venda	**S**	**S**	**S**	**S**

Fonte: Elaborado pelo autor.

3.2 Engenharia da Aplicação

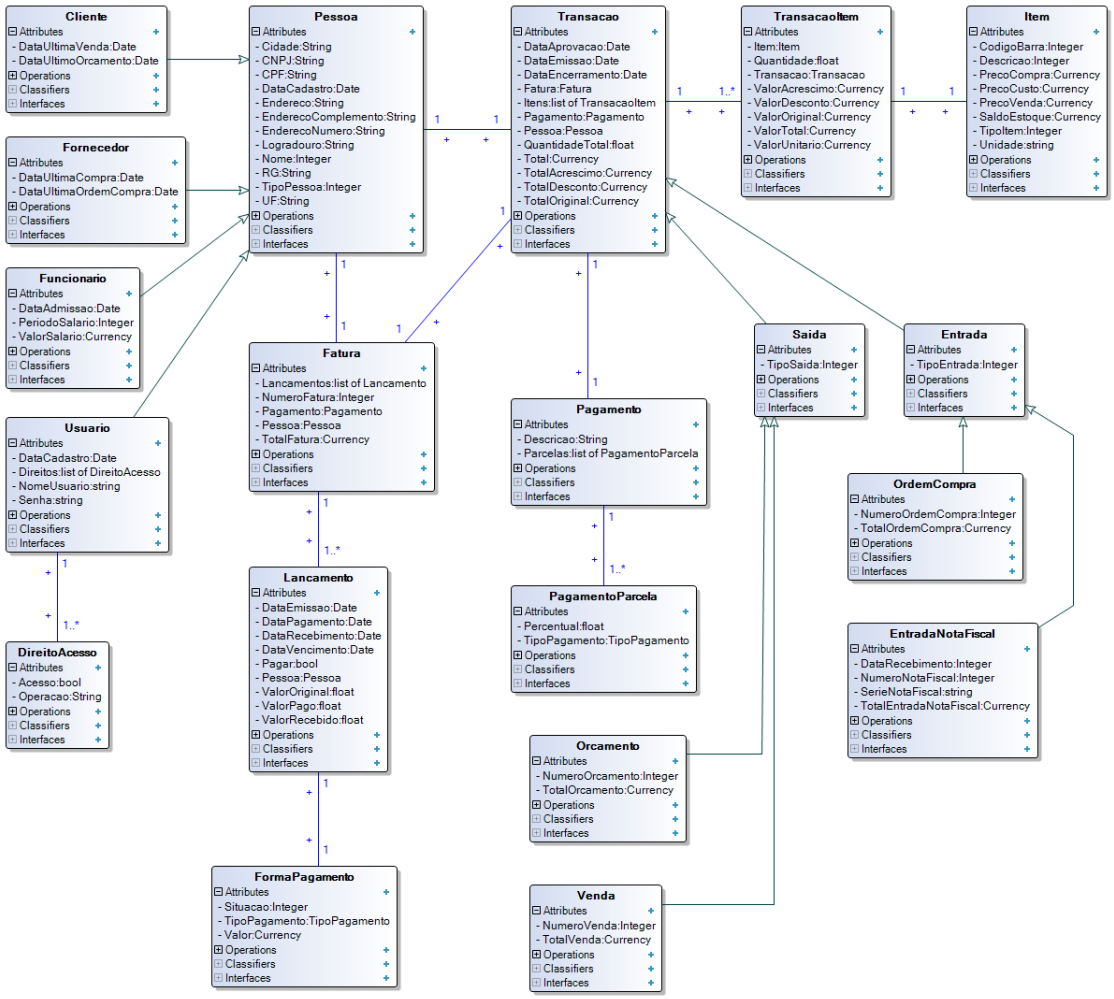
A arquitetura planejada tem como objetivo central suportar a implementação de todos os produtos da LPS e ser flexível o bastante para acomodar futuras adaptações e evoluções. Para representá-la, foi utilizado o diagrama de componentes da UML, visto que é possível fazer um alinhamento entre as features da LPS e os componentes do produto, bem como identificar quais são as interfaces requeridas e providas de cada componente.

A tentativa de criar uma aplicação que possa ser estendida de uma forma sistemática, foi possível devido ao mapeamento das *features* em componentes. As *features* obrigatórias, que são de uso comum do sistema, foram agrupadas em

pacotes separados por funcionalidade. Para as outras *features*, criou-se pacotes de acordo com a sua necessidade.

Concluída a etapa de Definição de Arquitetura, é hora de seguir para a etapa de Diagrama de Classes. Como os dados que serão informados ao sistema serão manipulados por um sistema de gerenciamento de banco de dados, as classes foram idealizadas visando o mapeamento de classes para as tabelas em um banco de dados relacional. O diagrama de classes entidade pensado é apresentado a seguir, na Figura 2.

Figura 2 – Diagrama de classes entidade



Fonte: Elaborada pelo autor.

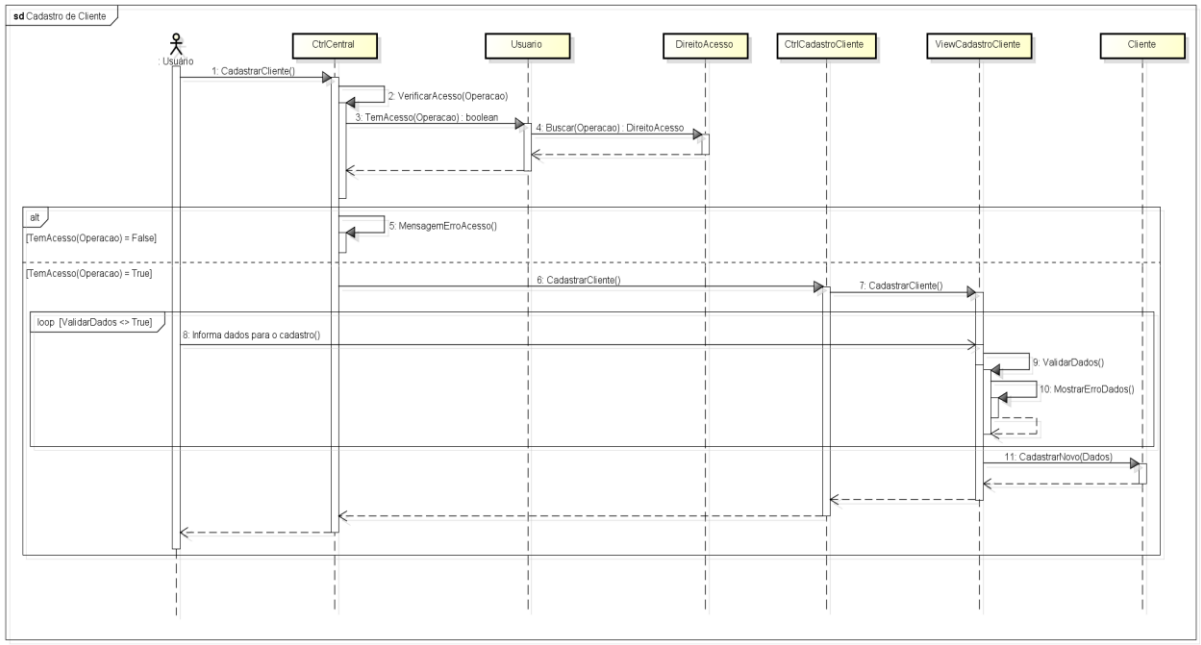
Além das classes entidade apresentadas no diagrama, as classes referentes aos controladores[3] e classes de visualização[4] foram projetadas, porém não foram

[3] Do inglês *Control*

incluídas no diagrama mostrado anteriormente. A estrutura das classes tem por base o modelo MVC, onde: (1) as classes *Model* são representadas pelas classes entidade, um exemplo é a classe Cliente; (2) as classes *View* são representadas pelas classes de visualização, um exemplo é a classe ViewCliente; (3) as classes *Control* são representadas pelas classes controle, um exemplo é a classe CtrlCliente.

O propósito da etapa a seguir é mostrar através do Diagrama de Sequência como os objetos do sistema trocam mensagens entre si para implementar uma funcionalidade. Os diagramas de sequência são baseados no diagrama de classes previamente apresentado. É importante destacar que nos diagramas de sequência foram utilizadas as classes controladoras e de visualização, as quais não foram representadas anteriormente no diagrama de classes. A Figura 3 mostra a troca de mensagens entre os objetos das classes que são responsáveis pela implementação do caso de uso “cadastro de cliente”, e também o “controle de acesso de usuário”.

Figura 3 – Diagrama de Sequência de cadastro de cliente



Fonte: Elaborada pelo autor.

Prosseguindo para a próxima e última etapa, de Mapeamento dos Produtos, começamos a identificar quais classes são necessárias para implementar as

[4] Do inglês *View*

features de formam um produto. Apenas uma parte das *features* foi apresentada no Quadro 2, apenas para uma visualização elucidativa. O mapeamento definido pela rastreabilidade, definido no Quadro 3, segue o mesmo padrão e será mostrado apenas uma parte do Quadro originalmente projetado, mostrado no Apêndice C, para ser breve e conseguir demonstrar que a rastreabilidade das classes foi possível.

Quadro 3 – Mapeamento parcial das classes

Classes	C1	C2	C3	C4
Transacao	X	X	X	X
TransacaoItem	X	X	X	X
ViewTransacaoItem	X	X	X	X
Saida	X	X	X	X
Orcamento		X		X
CtrlOrcamento		X		X
ViewOrcamento		X		X
Venda	X	X	X	X
CtrlVenda	X	X	X	X
ViewVenda	X	X	X	X
Entrada		X	X	X
OrdemCompra			X	X
CtrlOrdemCompra			X	X
ViewOrdemCompra			X	X
EntradaNotaFiscal		X	X	X
CtrlEntradaNotaFiscal		X	X	X
ViewEntradaNotaFiscal		X	X	X

Fonte: Elaborada pelo autor.

3.3 Execução

A tecnologia utilizada para a implementação da LPS proposta foi o Embarcadero Delphi XE5 (Delphi, 2013) em conjunto com o Sistema Gerenciador de Banco de Dados (SGBD) Firebird SQL 2.5 (Firebird, 2014). Esta escolha pode ser justificada considerando alguns motivos: (1) é um ambiente de desenvolvimento integrado (IDE) para a construção de aplicações — o Delphi faz parte do Embarcadero RAD Studio; (2) oferece um conjunto abrangente de ferramentas para racionalizar e simplificar o ciclo de vida do desenvolvimento; (3) usando a interface de *design visual* do RAD Studio, é possível interfaces gráficas de usuário, arrastando e soltando componentes da paleta de ferramentas a um formulário; (4) usando os *designers*, é possível criar aplicativos *Windows Forms* que usam a extensa biblioteca de componentes visuais (VCL); (5) é possível personalizar as aplicações para diferentes versões do Windows; (6) o autor do trabalho tem uma larga experiência

como o desenvolvimento de aplicações empresarias usando o RAD Studio; e (7) é possível utilizar arquivos de configuração para implementar a LPS.

A escolha do Firebird foi motivada considerando também alguns fatores: (1) é um banco de dados relacional de código aberto que oferece muitas características no padrão ANSI SQL; (2) tem suporte para as plataformas Linux, Windows, e em uma variedade de plataformas Unix; (3) oferece excelente concorrência de dados, alta performance e um poderoso suporte à linguagem para *Stored Procedures* e *Triggers*.

Por fim, a possibilidade de criar um software com modularização através de pacotes com o Delphi é uma característica muito importante, haja vista que o projeto previamente desenvolvido e arquitetado, prevê que cada feature a ser desenvolvida deve ser implementada como um pacote de código, encapsulando o escopo da feature a ser implementada. Com isso teremos um projeto com acoplamento dinâmico, conforme o que foi definido na configuração de features do produto.

4 AVALIAÇÃO

Este capítulo descreve como a avaliação do trabalho foi realizada. A Seção 4.1 apresenta os desafios encontrados, bem como os procedimentos adotados para contorná-los durante as fases de análise, projeto e implementação da LPS. A Seção 4.2 apresenta as adaptações realizadas no projeto inicialmente proposto, bem como as justificativas de tais adaptações. Por fim, a Seção 4.3 descreve detalhes sobre como foi possível implementar a LPS e aponta as limitações encontradas para implementar o projeto da LPS.

4.1 Desafios encontrados

No decorrer do desenvolvimento do projeto, foi encontrado um problema crítico: a utilização de classes entidade na codificação do projeto, representadas na Figura 2. Como o projeto foi pensado de uma forma modular, dependendo da configuração do produto algumas classes entidades se tornariam inúteis, pois nunca seriam utilizadas. O acoplamento e coesão do projeto também foram levados em consideração. Com a utilização das classes entidades, o acoplamento seria maior do que o desejado, e a estrutura estaria menos coesa.

Devido a facilidade de realizar as operações CRUD (*create, retrieve, update, delete*) diretamente no banco de dados, através dos componentes FireDac – uma biblioteca de Acesso Universal a Dados, para o desenvolvimento de aplicações para múltiplos dispositivos conectados a bancos de dados – de conexão e manipulação de informação no banco de dados, começou-se a questionar a real necessidade de utilizar classes entidade no desenvolvimento projeto. Apesar de ter que tomar mais cuidado na codificação de cada uma das features, o sistema adquire uma arquitetura com bem menos acoplamento e com uma maior coesão.

4.2 Modificações do projeto

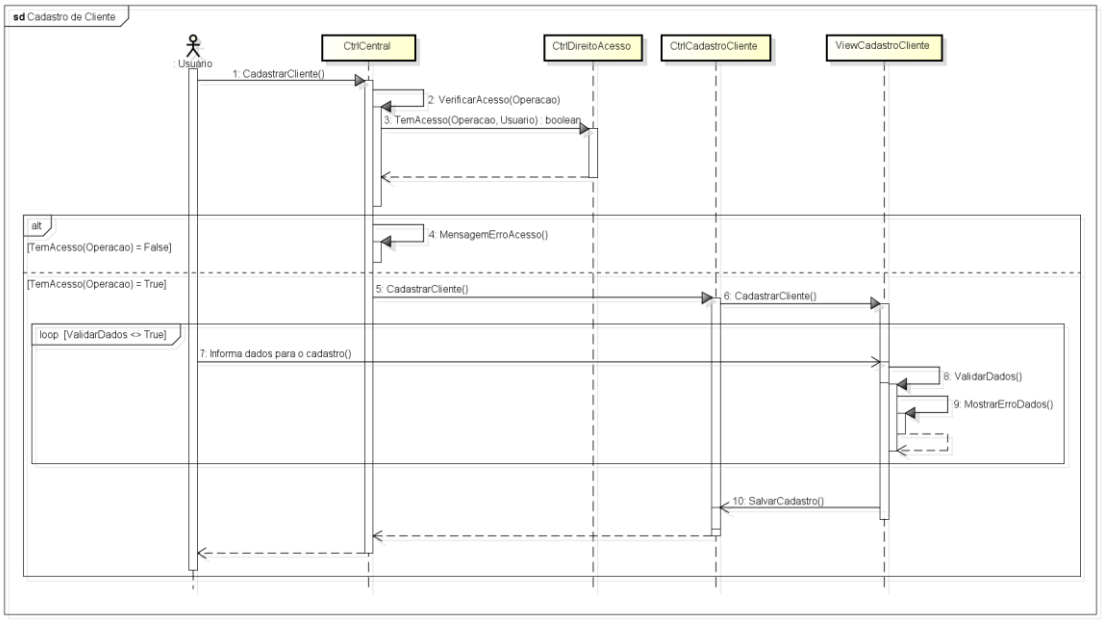
Nesta seção serão mostradas as modificações e execução do projeto em duas partes: (1) modificações e execução no escopo de ERP; e (2) modificações e execução no escopo de LPS.

4.2.1 ERP

Logo ao início da execução do projeto, foi estabelecido que as classes entidades não seriam implementadas, devido aos fatores previamente citados. Todas as operações CRUD que foram projetadas para que fossem executadas pelas classes entidades, foram migradas para as classes controle de cada uma das features.

Exemplificando a estrutura de classes estabelecida podemos usar a feature de Cadastro de produto, onde o seu pacote encapsularia a classe controle TCtrlCadastroProduto, classe entidade TProduto e classe de visualização TViewCadastroProduto. A alteração previamente citada fez com que as interações da classe controle com a classe entidade fossem cessadas e as operações CRUD – que seriam executadas na classe entidade – são agora executadas na classe de controle. Essa alteração foi extensiva para todas as features que dependiam de classes entidades para realizar as operações CRUD. Podemos visualizar essa alteração no diagrama apresentado na Figura 4.

Figura 4 - Novo diagrama de sequência para cadastro de cliente



Fonte: Elaborada pelo autor.

As classes entidades não foram utilizadas para o desenvolvimento diretamente no Delphi, mas foram base para o Diagrama Entidade Relacionamento (ER), responsável por estruturar toda a base de informação a ser utilizada para guardar os dados informados ao sistema ERP-SPL.

4.2.2 Implementação da variabilidade

Como anteriormente mencionado, o mecanismo que implementa a variabilidade da LPS é algo central para promover os reais benefícios esperados com o uso de LPS. Dentre os possíveis mecanismos para implementar a variabilidade de LPS, o uso de arquivo de configuração foi escolhido. É importante destacar que as outras formas de implementar variabilidade, incluindo compilação condicional e programação orientada a aspectos, não foram utilizadas porque o Delphi não suporta tais técnicas. Sendo assim, instaladores foram utilizados, os quais incorporam os arquivos de configuração dos produtos. Foi definido um instalador para cada tipo de produto que a LPS pode produzir. Cada um dos instaladores, é constituído de uma coleção de pacotes. Cada pacote é responsável por encapsular as funcionalidades de uma feature em particular.

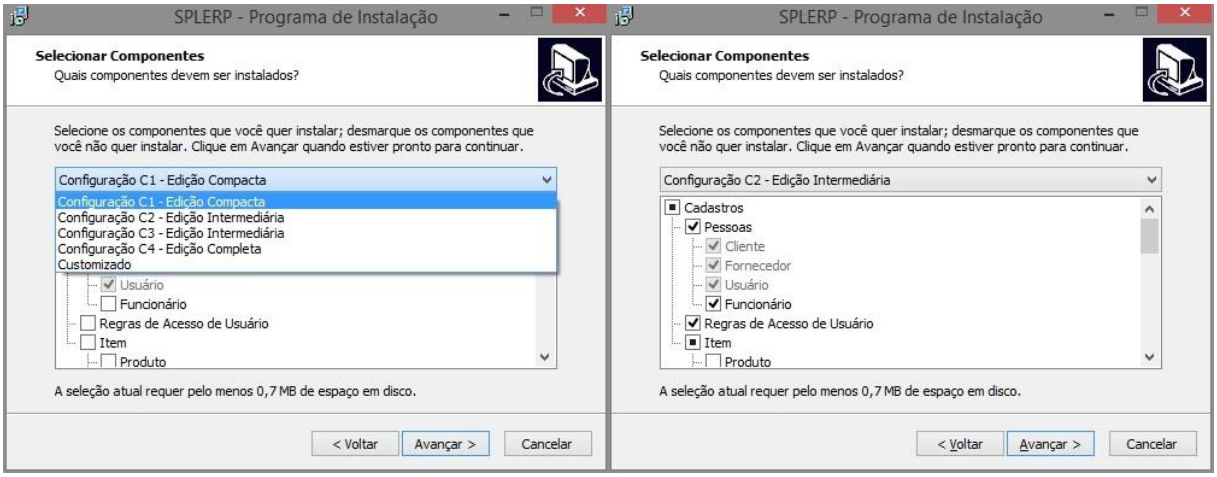
Existe um instalador para cada tipo de produto que pode ser derivado da LPS. Tem-se verificado um conjunto de vantagens ao implementar variabilidade usando

esta abordagem. Primeiro, *a manutenibilidade do processo de derivação de produtos da LPS se tornou menos propensa a erro*. Se novas features são inseridas na LPS, ou mesmo se as features pré-existente são alteradas, é necessário apenas modificar o instalador corresponde daquele produto, não sendo necessário modificar qualquer trecho de código da aplicação. Mais especificamente, as configurações dos produtos nos instaladores serão alteradas, e não qualquer trecho de código da aplicação. Pelo fato do instalador concentrar todo o conhecimento sobre como um produto em particular deve ser derivado, isso potencializa alguns benefícios em termos de manutenibilidade, visto que as modificações passam a ser localizadas em um único módulo, ao invés de ficarem espalhadas por vários módulos. Esta modularização na implementação da variabilidade dos produtos se alinha com os benefícios preconizados, por exemplo, com o uso de programação orientada a aspectos.

Segundo, *o esforço para derivar novos produtos é reduzido*. É sabido que, tendo um conjunto considerável de features opcionais e alternativas implementadas, é possível derivar um número expressivo de produtos da LPS. Porém, para isso se tornar possível, é necessário configurar como cada produto será derivado, ao mesmo tempo que assegure uma quantidade de esforço a ser investido; caso contrário, todos os benefícios adquiridos com o reuso podem ser comprometidos. Diante deste cenário é que o arquivo de configuração ajuda a reduzir o esforço e a propensão a erros, dado que as modificações ficam restritas a um único local.

É importante destacar que o programa instalador também é responsável por levar todos os arquivos necessários para que o sistema ERP-SPL esteja em plenas condições de uso. Os arquivos necessários incluem todos as bibliotecas utilizadas para a implementação, desde componentes visuais até cliente de banco de dados. O SGBD Firebird 2.5 também é faz parte dos pré-requisitos básicos, e será instalado caso ainda não esteja. Além disso, as features opcionais devem ser selecionadas durante a instalação – conforme a Figura 5 – para que as mesmas estejam disponíveis para o uso no sistema ERP-SPL. Cada uma das features opcionais selecionada terá o seu pacote correspondente instalado, para que seja respeitada a seleção feita pelo usuário.

Figura 5 – Instalador e configurações de produtos



Fonte: Elaborada pelo autor.

4.3 Discussão

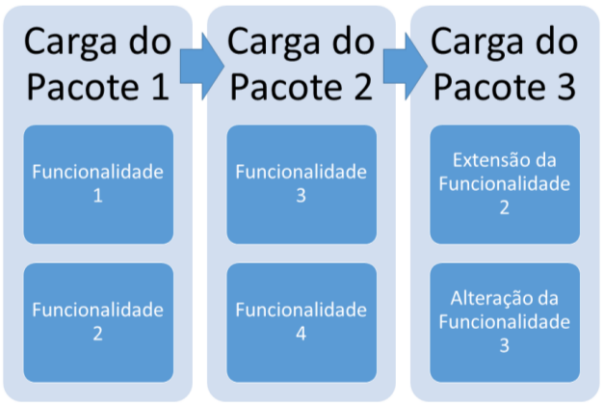
Uma parte muito importante a ser discutida é de como foi implementada a parte da variabilidade dos pacotes utilizando o Delphi, que não traz ferramentas nativas para agilizar este processo. Um dos aspectos mais aproximados à variabilidade foi utilizado para este procedimento, a carga dinâmica de pacotes em conjunto com um repositório central de funcionalidades.

A estrutura de repositório central foi criada para que fosse possível registrar e disponibilizar a execução de toda e qualquer funcionalidade declarada nos pacotes. Em uma visão de implementação, o repositório é uma variável global, sendo possível utilizá-lo em qualquer outro objeto do sistema. Ele é criado durante a inicialização primária do sistema, e tem como principais funcionalidades: (1) registro de funcionalidade e (2) execução de funcionalidade.

A carga dinâmica consiste em fazer com que um pacote, e todo o seu conteúdo, seja carregado em memória. Nessa etapa, a carga do pacote dispara um procedimento de inicialização do mesmo, que registra no repositório central, todas as funcionalidades declaradas no pacote.

A implementação da variabilidade foi possível graças ao uso desta estrutura. Ao declararmos uma funcionalidade em um pacote, durante a carga do pacote, esta funcionalidade estará disponível para o uso. A variabilidade consiste em estender ou modificar uma funcionalidade em outro pacote, exemplificado na Figura 6.

Figura 6 - Exemplo de variabilidade



Fonte: Elaborada pelo autor.

Um aspecto muito importante para que seja possível a implementação correta da variabilidade é que a hierarquia dos pacotes deve ser obedecida. Tem-se a Figura 6 como exemplo: caso a carga do pacote 3 tenha sido realizada antes da carga do pacote 1, a extensão da funcionalidade 2 e a alteração da funcionalidade 3 serão carregadas e disponibilizadas, porém a carga do pacote 1 irá sobrescrever as alterações e irá registrar a funcionalidade 2 como estava sem a sua extensão realizada no pacote 3.

Os resultados obtidos com a execução do que foi estipulado no projeto e nas suas alterações realizadas ao decorrer da implementação, foram satisfatórios. Foi possível criar um software voltado a Gestão Empresarial, suprindo as necessidades básicas de uma empresa. Também foi possível criar uma LPS, aliando toda a base de código gerada e separada em pacotes, com a criação das configurações estipuladas com a ferramenta de criação de instaladores *Inno Setup*.

Toda a estrutura criada, voltada para a modularização das funcionalidades, foi feita visando a variabilidade e a futura adição de novas features. Com isso a modularização realizada por implementação em pacotes foi um ponto crucial para que fosse possível gerar a LPS. A utilização do *Inno Setup* para converter as configurações dos produtos em produtos, foi simples devido à todo o suporte que o projeto ofereceu para isso.

5 TRABALHOS RELACIONADOS

O objetivo deste capítulo é realizar uma análise comparativa do trabalho proposto com alguns trabalhos encontrados na literatura que abordam direta ou

indiretamente o escopo do trabalho. Essa análise comparativa será apresentada nas próximas três Seções.

5.1 Sistema de Gestão Integrada para Micro e Pequenas Empresas de Transporte Rodoviário de Cargas

O trabalho proposto, por Bill (2013), apresenta um sistema de ERP para micro e pequenas empresas de transporte rodoviário de cargas secas. O trabalho foca na resolução de problemas relacionados ao transporte rodoviário de cargas secas. A metodologia utilizada para o desenvolvimento do trabalho é descrita em um ciclo de vida de desenvolvimento de sistemas, e a metodologia escolhida foi a de Processo Unificado (PU). De acordo com Krutchen (2003), o PU propõe um desenvolvimento interativo e incremental, centrado na arquitetura e orientado a casos de uso.

O projeto foi descrito e detalhado usando os artefatos preconizados pelo PU: (a) modelo de processos de negócio; (b) visão de casos de uso; (c) diagrama de pacotes como visão lógica do sistema; (d) diagrama de desenvolvimento[5] como visão de implantação; (e) diagrama de componentes como visão de implementação; (f) diagrama de classes como modelo conceitual de dados; (g) diagrama entidade-relacionamento como modelo lógico de dados; e (h) diagramas de transição de estados como modelos comportamentais.

Embora o trabalho suporte um número considerável de funcionalidades, o projeto peca no que se refere a sua capacidade de suportar novas *features.* Consequentemente, alterar, ou mesmo adicionar novas *features*, pode ser tornar algo custoso e propenso a erros. Tem-se observado que o trabalho anterior é rígido, não sendo complacente com boas práticas e princípios de projeto como, por exemplo, o preconizado pelo princípio *Open-Closed* – um software deve ser aberto para extensão e fechado para modificação. Adaptar tal projeto para outro contexto exigiria um esforço de desenvolvimento grande, dada a quantidade de modificações a serem realizadas e a imprevisibilidade da propagação das mudanças.

Por outro lado, o trabalho proposto nesta monografia visa contornar esta problemática ao projetar uma LPS modularizada, a qual é suportada por uma

---

[5] Do inglês *deployment*

arquitetura flexível. A vantagem ao se utilizar desta abordagem de LPS é que novas *features* poderão ser adicionadas, alteradas ou removidas de uma forma mais sistemática —. Isso é possível devido ao mapeamento das *features* e rastreabilidade de tais *features* o que potencializa o reuso e garante a produtividade da equipe de desenvolvimento.

5.2 Implementação e Análise de uma Linha de Produtos de Software

O trabalho realizado em Ferreira (2009), apresenta a implementação de uma LPS de uma ferramenta de geração automática de casos de teste. Fazendo uso de estratégias de desenvolvimento de LPS, o objetivo central é apresentar e analisar o LPS da TaRGet (*Test and Requirements Generation Tool*), contribuir com a identificação de pontos de variação da LPS TaRGeT, implementar as variações das *features* da LPS TaRGeT, e analisar soluções alternativas para implementação da LPS. A TaRGeT é uma ferramenta que tem como funcionalidade a geração de casos de teste, objetivando a automação da abordagem sistemática ao lidar com artefatos de requisitos e de teste, de uma maneira integrada.

Foram mostrados os pontos de variação identificados e as variações implementadas a partir de cada um deles. Em sequência, foi apresentada uma análise de implementações alternativas que envolveu não apenas o uso de técnicas isoladas, mas também de combinações de técnicas a fim de propor soluções mais abrangentes e adequadas para a implementação da linha.

O trabalho realizado apresentou o estudo e alteração de todos os artefatos utilizados no projeto do sistema TaRGeT, que contrário ao sistema ERP-SPL, foi construído após realizar estudos de como seria a melhor maneira de realizar o sistema especificado. Ambos os trabalhos realizam análise e desenvolvimento de projeto baseado em LPS, a alteração no sistema TaRGeT cobre um escopo de trabalho muito pequeno, atendendo apenas a uma pequena área de atuação, se comparado ao ERP-SPL.

5.3 Ligo: Uma linha de produtos de software para gerenciamento de igrejas cristãs

O objetivo principal do trabalho de Teixeira (2007), é o de desenvolver uma LPS capaz de gerenciar igrejas cristãs, englobando todos os aspectos relacionados ao gerenciamento de igrejas, facilitando a integração e compartilhamento de dados entre os setores. Para isso, foram utilizadas as técnicas de LPS e PLUS.

Para a modelagem de requisitos, as atividades realizadas foram: (a) *modelagem de requisitos*: análise de escopo, modelagem de *features*, modelagem de casos de uso, relação entre *features* e casos de uso; (b) *análise*: modelagem estática, representado por diagramas de classes entidade, e modelagem dinâmica, representado por diagramas de sequência; e (c) *projeto e desenvolvimento*: uso de herança para implementar a variabilidade entre produtos, arquivos de configuração para gerenciar a variabilidade implementada anteriormente, programação orientada a aspectos foi adotada pois, de acordo com a análise do autor, mostrou-se a maneira mais adequada para implementar as variações de *features*.

Para a engenharia de aplicação, o autor utilizou um diagrama de atividades para demonstrar as iterações necessárias para configurar os produtos, segundo os aspectos levantados no domínio de negócio.

O trabalho Ligo mostrou semelhanças com o sistema ERP-SPL, pois tratam-se de dois trabalhos que desenvolvem uma LPS, utilizando o método PLUS, para desenvolver um ERP, cada um voltado a sua área de domínio. Uma diferença notada foi que, na engenharia de aplicação do sistema Ligo, mostrou-se apenas a configuração personalizada pelo usuário do sistema, enquanto que no ERP-SPL foi realizada toda a engenharia como projeto de software.

6 CONSIDERAÇÕES FINAIS

Este sistema ERP, ao contrário de muitos encontrados no mercado, foi projetado e executado utilizando técnicas e conceitos da metodologia de engenharia de software de Linha de Produtos de Software (LPS), que permite a criação de mais de um produto de software fazendo reuso dos artefatos UML criados e de códigos gerado na implementação. Esta metodologia também facilita e potencializa a realização de reengenharia do projeto de uma maneira menos custosa devido aos conceitos empregados.

Para realizar o desenvolvimento do sistema ERP com qualidade em um menor espaço de tempo e com pouco custo, foi preciso ter muito cuidado com três

problemas críticos: (1) Representação adequada das features, para isso foi preciso compreender de forma apropriada quais são as features do núcleo comum – compartilhado pelos produtos ERP – e suas particularidades; (2) Rastreabilidade entre Features, artefatos e módulos de ERPs, foi realizado para que houvesse um ganho de produtividade na manutenção e na evolução do ERP – que foi alcançado porque o desenvolvedor compreende claramente como as Features do ERP foram modularizadas em nível de projeto e implementação; e (3) Conhecimento sobre o real ganho do uso de LPS para aplicações ERP, a literatura atual é falha ao não fornecer evidências que comprovem se o uso de LPS pode trazer benefícios ao desenvolver um ERP.

Durante a execução do trabalho foi possível demonstrar algumas contribuições: (1) o estado da arte de LPS e ERP; (2) a análise e o projeto de uma LPS para ERP; (3) uma análise de domínio para a elaboração da LPS para ERP; (4) conhecimentos práticos sobre os reais ganhos e benefícios ao utilizar LPS para adequar a representação das features de um sistema ERP; (5) aumento da precisão da rastreabilidade entre features, artefatos e os módulos de um ERP; (6) aprendizado identificado durante a análise, projeto e desenvolvimento da LPS para ERP; (7) possibilidade de implementar a variabilidade de LPS, com o uso do Delphi aliado ao gerador de instaladores; e (8) facilidade de manutenibilidade e expansão do arquivo de configuração do gerador de instaladores, concentrador de todo o conhecimento das configurações dos produtos da LPS.

Portanto, podemos ver este artigo como um primeiro passo para uma agenda mais ambiciosa de propor um quadro mais estabelecido e empiricamente sólido de referência para a avaliação do modelo de esforço composição em diferentes contextos do mundo real. Embora o modelo tenha sido derivado de estudos empíricos, precisamos entender melhor se o modelo proposto prevê algum ganho quando: (1) aplicado para orientar estudos empíricos considerando novas noções de qualidade, e (2) com relação ao tempo gasto para entender e fazer um uso eficaz do quadro qualitativo. Por fim, esperamos que as questões apresentadas ao longo do artigo encorajem outros pesquisadores a realizar seus estudos, seguindo o modelo proposto e também avaliá-lo, no futuro, em circunstâncias diferentes. Por estes motivos podemos descrever este trabalho como um guia, ou *roadmap*, sobre como projetar e implementar a variabilidade de uma LPS com Delphi. Para o futuro, serão

observadas, projetadas e implementadas novas necessidades – utilizando a generalização da variabilidade obtida – vindouras de empresas que optem pelo uso deste sistema ERP.

A SOFTWARE PRODUCT LINE TO BUSINESS MANAGEMENT INTEGRATED SYSTEMS: A CASE STUDY IN THE INDUSTRY

The management of business processes is very complex and requires a lot of organization. In many cases, the organization of these processes is affected by this high complexity. This ends up creating confusion among everyone involved, and can result in severe problems. This work presents the design of a software that assists in the organization and conduct of processes of a company. The software in question is the Sistemas Integrados de Gestão Empresarial (SIGE), also known as Enterprise Resource Planning (ERP). Unlike many ERP systems found on the market, this one was designed using techniques and concepts from engineering methodology Software Product Line (SPL), which allows the creation of more than one product making reuse of created Unified Modeling Language (UML) artifacts and implementation codes. This methodology also allows the realization of reengineering project, if necessary, in a manner facilitated by the concepts applied. The project developed allows the implementation of the software, creating a tool to manage, organize and view business processes specified in its scope.

Keywords: Software Product Line, Enterprise Resource Planning, Software Engineering.

REFERÊNCIAS

AMBLER, S. W. Enterprise Unified Process: Extending the Rational Unified Process. 2005.

BARROS, Pablo. Linguagem de Modelagem Unificada em Português. 1998. Disponível em: <http://cc.usu.edu/~slqz9/uml/>. Acesso em: 17 nov. 2013.

BILL, Felipe Luiz. Sistema de Gestão Integrada para Micro e Pequenas Empresas de Transporte Rodoviário de Cargas. 2013. 82 f. Trabalho de Conclusão de Curso (Bacharel em Sistemas de Informação). Curso de Sistemas de Informação. Universidade Tecnológica Federal do Paraná, Curitiba, PR, 2013.

CAVALCANTI, M. Gestão estratégica de negócio: evolução, cenários, diagnóstico e ação. 2001.

CLEMENTS, Paul; NORTHROP, Linda. Software product lines: practices and patterns. 2001.

COLÂNGELO FILHO, Lucio. Implantação de Sistemas ERP. 2001.

CORRÊA, H. L.; GIANESI, I. G. N.; CAON, M. Planejamento, programação, e controle da produção: MRP II/ERP: Conceitos, uso e implantação. 1997.

CZARNECKI, K.; EISENECKER, U. W. Synthesizing objects. Concurrency: Practice and Experience, v. 12, n. 14, p. 1347-1377, 2000. Disponível em: < http://onlinelibrary.wiley.com/doi/10.1002/1096-9128(20001210)12:14%3C1347::AID-CPE513%3E3.0.CO;2-N/abstract>. Acesso em 26 nov. 2013.

DAVENPORT; T. H. Putting de enterprise into the enterprise system. Harvard Business Review. p. 1221-1231. 1998.

DELOITTE. ERP's Second Wave: Maximizing the Value of ERP-Enabled Processes. Relatório de pesquisa publicado pela Deloitte Consulting. 1998. Disponível em: <http://www.ctiforum.com/technology/CRM/wp01/download/erp2w.pdf>. Acesso em 21 nov. 2013.

Delphi XE5. 2013. Disponível em: <http://www.embarcadero.com/>. Acesso em 21 jan. 2014.

FERREIRA, Felype Santiago. Implementação e Análise de uma Linha de Produtos de Software. 2009. 79 f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação). Curso de Ciência da Computação. Universidade Federal de Pernambuco, Recife, PE, 2009.

Firebird SQL 2.5. 2014. Disponível em <http://www.firebirdsql.org/>. Acesso em 22 jan. 2014.

GOMAA; Hassan. Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures. 2011.

KANG, Kyo C.; COHEN, Sholom G.; HESS, James A.; NOVAK, William E.; PETERSON, A. Spencer. Feature-Oriented Domain Analysis (FODA): Feasibility Study. 1990.

KANG, Kyo C.; KIM, Sajoong; LEE, Jaejoon; KIM, Kijoo; SHIN, Euiseob; HUH, Moonhang. 1998. FORM: A feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. 5, p. 143-168. 1998.

PADILHA, T. C. C.; MARINS, F. A. S. Sistemas ERP: características, custos e tendências. Prod., São Paulo, v. 15, n. 1, p. 102-113. 2005 . Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-65132005000100009&lng=pt&nrm=iso>. Acesso em 20 nov. 2013.

REZENDE, Denis Alcides; ABREU, Aline França. Tecnologia da Informação Aplicada a Sistemas de Informação Empresariais. 2001.

SOUZA, C. A.; ZWICKER, R. Ciclo de vida de sistemas ERP. Caderno de Pesquisas em Administração, São Paulo. v. 1, n. 11. 2000. Disponível em: <http://profjayrfigueiredo.com.br/STI_AC_08.pdf>. Acesso em 21 nov. 2013.

SPARKS, Geoffrey. Database Modelling in UML. 2001.

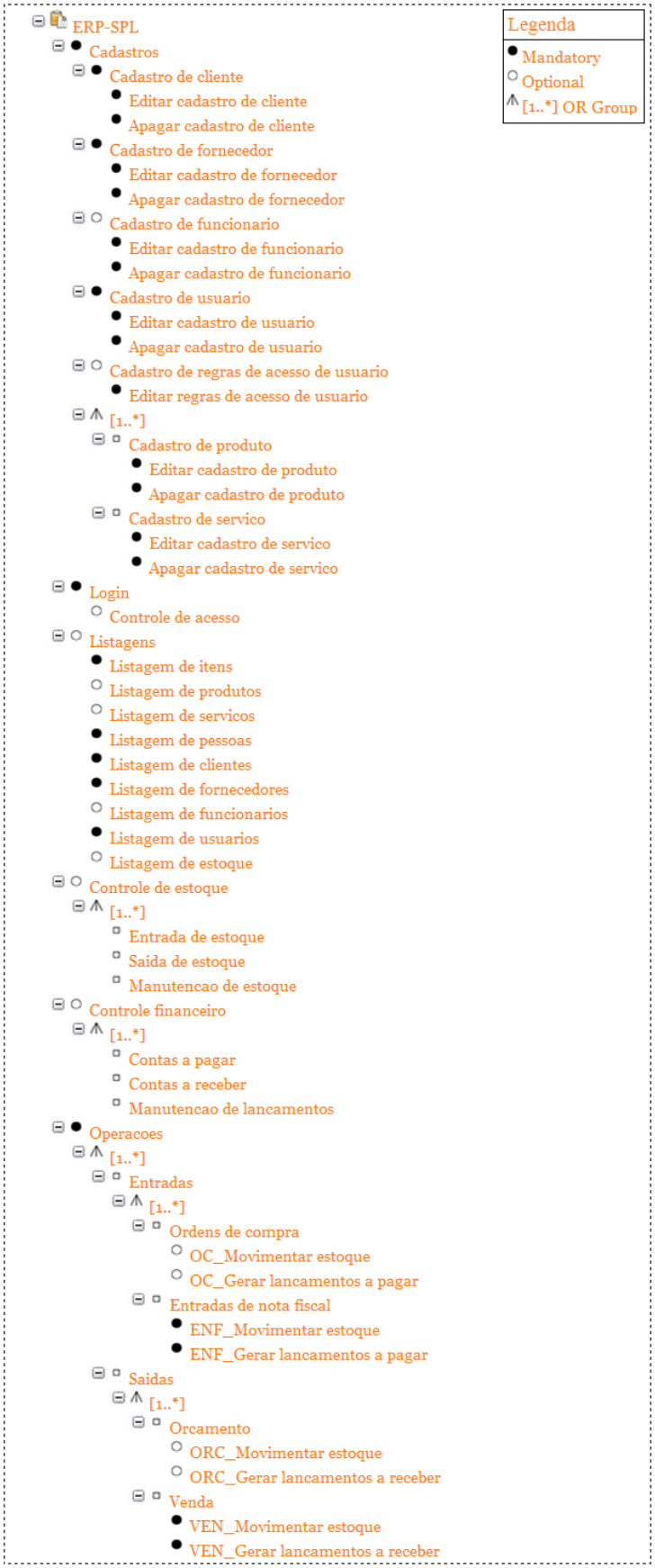
TEIXEIRA, Leopoldo Motta. Ligo: Uma linha de produtos de software para gerenciamento de igrejas cristãs. 2007. 56 f. Trabalho de Conclusão de Curso (Bacharel em Engenharia da Computação). Curso de Engenharia da Computação. Escola Politécnica de Pernambuco – Universidade de Pernambuco, Recife, PE, 2007.

UML2 Use Case Diagrams, 2013. Disponível em: <http://www.agilemodeling.com/artifacts/useCaseDiagram.htm>. Acesso em: 17/11/2013.

USE Case Diagram, 2013. Disponível em: <http://en.wikipedia.org/wiki/Use_Case_Diagram>. Acesso em: 17/11/2013.

WOHLIN, Claes; RUNESON, Per; HÖST, Martin; OHLSSON, Magnus C.; REGNELL, Bjöorn; WESSLÉN, Anders. Experimentation in Software Engineering: An Introduction. 2000.

APÊNDICE A – DIAGRAMA DE FEATURES



APÊNDICE B – QUADRO DE DESCRIÇÃO DE FEATURES

Feature	Descrição	Configuração
Cadastros	Sistema de cadastros	Obrigatória
Cadastro de cliente	Funcionalidade para cadastrar cliente	Obrigatória
Editar cadastro de cliente	Funcionalidade que permita edição de um cadastro de cliente	Obrigatória
Apagar cadastro de cliente	Funcionalidade que permita exclusão de um cadastro de cliente	Obrigatória
Cadastro de fornecedor	Funcionalidade para cadastrar fornecedor	Obrigatória
Editar cadastro de fornecedor	Funcionalidade que permita edição de um cadastro de fornecedor	Obrigatória
Apagar cadastro de fornecedor	Funcionalidade que permita exclusão de um cadastro de fornecedor	Obrigatória
Cadastro de funcionário	Funcionalidade para cadastrar funcionário	Opcional
Editar cadastro de funcionário	Funcionalidade que permita edição de um cadastro de funcionário	Obrigatória, caso a feature Cadastro de funcionário seja selecionada
Apagar cadastro de funcionário	Funcionalidade que permita exclusão de um cadastro de funcionário	Obrigatória, caso a feature Cadastro de funcionário seja selecionada
Cadastro de usuário	Funcionalidade para cadastrar usuário	Obrigatória
Editar cadastro de usuário	Ferramenta que permita edição de um cadastro de usuário	Obrigatória
Apagar cadastro de usuário	Funcionalidade que permita exclusão de um cadastro de usuário	Obrigatória
Grupo OU (Cadastro de item)	Grupo OU, que permita a realização de cadastro de todos os tipos de item	Alternativa (OU)
Cadastro de produto	Funcionalidade para cadastrar item do tipo produto	Alternativa opcional do Grupo OU (Cadastro de item)
Editar cadastro de produto	Funcionalidade que permita edição de um cadastro de item do tipo produto	Obrigatória, caso a *feature* Cadastro de produto seja selecionada
Apagar cadastro de produto	Funcionalidade que permita exclusão de um cadastro de um item do tipo produto	Obrigatória, caso a *feature* Cadastro de produto seja selecionada
Cadastro de serviço	Funcionalidade para cadastrar item do tipo serviço	Alternativa opcional do Grupo OU (Cadastro de item)

Editar cadastro de serviço	Funcionalidade que permita edição de um cadastro de item do tipo serviço	Obrigatória, caso a *feature* Cadastro de serviço seja selecionada
Apagar cadastro de serviço	Funcionalidade que permita exclusão de um cadastro de um item do tipo serviço	Obrigatória, caso a *feature* Cadastro de serviço seja selecionada
Cadastro de regras de acesso de usuário	Funcionalidade que permita o cadastro de regras de acesso do usuário a ferramentas do sistema	Opcional
Editar regras de acesso de usuário	Funcionalidade que permita a edição de regras de acesso do usuário a ferramentas do sistema	Obrigatória, caso a *feature* Cadastro de regras de acesso de usuário seja selecionada
Login	Controle de acesso ao sistema via usuário e senha	Obrigatória
Controle de acesso	Controle de acesso a ferramentas do sistema, via regras e acesso cadastradas	Opcional, porém obrigatória quando a *feature* Cadastro de regras de acesso de usuário for selecionada
Listagens	Grupo de ferramentas que possibilitem uma série de listagens	Opcional
Listagem de itens	Funcionalidade que liste os itens de todos os tipos cadastrados no sistema	Obrigatória, caso a *feature* Listagens seja selecionada
Listagem de produtos	Funcionalidade que liste os itens de todos os itens do tipo produto cadastrados no sistema	Opcional, caso a *feature* Listagens seja selecionada
Listagem de serviços	Funcionalidade que liste os itens de todos os itens do tipo serviço cadastrados no sistema	Opcional, caso a *feature* Listagens seja selecionada
Listagem de pessoas	Funcionalidade que liste as pessoas de todos os tipos cadastrados no sistema	Obrigatória, caso a *feature* Listagens seja selecionada
Listagem de clientes	Funcionalidade que liste as pessoas do tipo cliente cadastrados no sistema	Obrigatória, caso a *feature* Listagens seja selecionada
Listagem de fornecedores	Funcionalidade que liste as pessoas do tipo fornecedor cadastrados no sistema	Obrigatória, caso a *feature* Listagens seja selecionada
Listagem de funcionários	Funcionalidade que liste as pessoas do tipo funcionário cadastrados no sistema	Opcional, caso a *feature* Listagens seja selecionada
Listagem de usuários	Funcionalidade que liste as pessoas do tipo usuário cadastrados no sistema	Obrigatória, caso a *feature* Listagens seja selecionada
Listagem de estoque	Funcionalidade que liste os itens	Opcional, caso a *feature*

	cadastrados no sistema em um formato de inventário de estoque	Listagens seja selecionada
Controle de estoque	Grupo de ferramentas que possibilitem o controle de estoque dos itens	Opcional
Entrada de estoque	Funcionalidade que realize o movimento de entrada de estoque	Alternativa opcional, do grupo de ferramentas de Controle de estoque
Saída de estoque	Funcionalidade que realize o movimento de saída de estoque	Alternativa opcional, do grupo de ferramentas de Controle de estoque
Manutenção de estoque	Funcionalidade que possibilite a manutenção do estoque	Alternativa opcional, do grupo de ferramentas de Controle de estoque
Controle financeiro	Grupo de ferramentas que realize o controle financeiro	Opcional
Contas a pagar	Funcionalidade para gerenciar as contas a pagar	Alternativa opcional, do grupo de ferramentas de Controle financeiro
Contas a receber	Funcionalidade para gerenciar as contas a receber	Alternativa opcional, do grupo de ferramentas de Controle financeiro
Manutenção de lançamentos	Funcionalidade para dar manutenção aos lançamentos cadastrados no sistema	Alternativa opcional, do grupo de ferramentas de Controle financeiro
Operações	Grupo de ferramentas para realizar operações de entrada e saída de itens	Obrigatória
Entradas	Funcionalidade para realizar, e gerenciar operações de entrada de itens	Alternativa opcional, do grupo de ferramentas de Operações
Ordens de compra	Funcionalidade para realizar operações de entrada do tipo Ordem de compra de itens	Alternativa opcional, do grupo de ferramentas de Entradas
OC_Movimentar estoque	Funcionalidade para movimentar estoque automaticamente ao encerrar uma Ordem de Compra	Opcional, caso a feature Ordens de compra seja selecionada
OC_Gerar Lançamentos a pagar	Funcionalidade para gerar lançamentos a pagar ao encerrar uma Ordem de Compra	Opcional, caso a feature Ordens de compra seja selecionada
Entradas de nota fiscal	Funcionalidade para realizar operações de entrada do tipo Entrada de Nota Fiscal de itens	Alternativa opcional, do grupo de ferramentas de Entradas

ENF_Movimentar estoque	Funcionalidade para movimentar o estoque automaticamente ao encerrar uma Entrada de Nota Fiscal	Obrigatória, caso a feature Entradas de nota fiscal seja selecionada
ENF_Gerar Lançamentos a pagar	Funcionalidade para gerar lançamentos a pagar ao encerar uma Entrada de Nota Fiscal	Obrigatória, caso a feature Entradas de nota fiscal seja selecionada
Saídas	Ferramentas para realizar operações de saída de itens	Alternativa opcional, do grupo de ferramentas de Operações
Orçamento	Ferramenta para realizar a operações de saída do tipo Orçamento	Alternativa opcional, do grupo de ferramentas de Saídas
ORC_Movimentar estoque	Funcionalidade para movimentar o estoque automaticamente ao encerrar um Orçamento	Opcional, caso a feature Orçamento seja selecionada
ORC_Gerar lançamentos a receber	Funcionalidade para gerar lançamentos a receber ao encerar um Orçamento	Opcional, caso a feature Orçamento seja selecionada
Venda	Funcionalidade para realizar operações de saída do tipo Venda	Alternativa opcional, do grupo de ferramentas de Saídas
VEN_Movimentar estoque	Funcionalidade para movimentar o estoque automaticamente ao encerrar uma Venda	Obrigatória, caso a feature Venda seja selecionada
VEN_Gerar lançamentos a receber	Funcionalidade para gerar lançamentos a receber ao encerar uma Venda	Obrigatória, caso a feature Venda seja selecionada

APÊNDICE C – QUADRO DE MAPEAMENTO DE CLASSES

Classes	C1	C2	C3	C4
CtrlCentral	X	X	X	X
ViewCentral	X	X	X	X
CtrlLogin	X	X	X	X
ViewLogin	X	X	X	X
Pessoa	X	X	X	X
Cliente	X	X	X	X
CtrlCliente	X	X	X	X
ViewCliente	X	X	X	X
Fornecedor	X	X	X	X
CtrlFornecedor	X	X	X	X
ViewFornecedor	X	X	X	X
Funcionario		X	X	X
CtrlFuncionario		X	X	X
ViewFuncionario		X	X	X
Usuario		X	X	X
CtrlUsuario		X	X	X
ViewUsuario		X	X	X
DireitoAcesso		X	X	X
CtrlDireitoAcesso		X	X	X
ViewDireitoAcesso		X	X	X
Transacao	X	X	X	X
TransacaoItem	X	X	X	X
ViewTransacaoItem	X	X	X	X
Item	X	X	X	X
CtrlItem	X	X	X	X
ViewItem	X	X	X	X
Saida	X	X	X	X
Orcamento		X		X
CtrlOrcamento				X
ViewOrcamento				X
Venda	X	X	X	X
CtrlVenda	X	X	X	X
ViewVenda	X	X	X	X
Entrada		X	X	X
OrdemCompra			X	X
CtrlOrdemCompra				X
ViewOrdemCompra				X
EntradaNotaFiscal		X	X	X
CtrlEntradaNotaFiscal		X	X	X
ViewEntradaNotaFiscal		X	X	X
Pagamento	X	X	X	X

CtrlPagamento	X	X	X	X
ViewPagamento	X	X	X	X
PagamentoParcela	X	X	X	X
ViewPagamentoParcela	X	X	X	X
Fatura	X	X	X	X
CtrlFatura			X	X
ViewFatura			X	X
Lancamento			X	X
CtrlLancamento			X	X
ViewLancamento			X	X
FormaPagamento			X	X
CtrlFormaPagamento			X	X
ViewFormaPagamento			X	X
ViewListagem		X	X	X
CtrlListagemItem		X	X	X
ViewListagemItem		X	X	X
CtrlListagemProduto			X	X
ViewListagemProduto			X	X
CtrlListagemServico		X		X
ViewListagemServico		X		X
CtrlListagemPessoa		X	X	X
ViewListagemPessoa		X	X	X
CtrlListagemCliente		X	X	X
ViewListagemCliente		X	X	X
CtrlListagemFornecedor		X	X	X
ViewListagemFornecedor		X	X	X
CtrlListagemFuncionario		X	X	X
ViewListagemFuncionario		X	X	X
CtrlListagemUsuario		X	X	X
ViewListagemUsuario		X	X	X
CtrlListagemEstoque		X		X
ViewListagemEstoque		X		X