UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

COORDENAÇÃO DO CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CURSO SUPERIOR DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

BRUNA ROSSETTO DELAZERI

ELLEN CRISTINA WOLF

MODELAGEM DE UM SISTEMA ORGANIZADOR BASEADO EM LINHAS DE PRODUTO

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2012

BRUNA ROSSETTO DELAZERI

ELLEN CRISTINA WOLF

MODELAGEM DE UM SISTEMA ORGANIZADOR BASEADO EM LINHAS DE PRODUTO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, da Coordenação de Análise e Desenvolvimento de Sistemas (COADS), da Universidade Tecnológica Federal do Paraná.

Orientador: Profª. Drª. Simone Nasser Matos.

PONTA GROSSA

2012

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

MODELAGEM DE UM SISTEMA ORGANIZADOR BASEADO EM LINHAS DE PRODUTO

por

BRUNA ROSSETTO DELAZERI
ELLEN CRISTINA WOLF

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 25 de maio de 2012 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Profª Drª Simone Nasser Matos
Orientadora

_____
Prof. Alison Roger Hajo Weber
Membro titular

_____
Profª. Msc. Helyane B. Borges
Responsável pelos Trabalhos de Conclusão de Curso

_____
Profª. Msc. Eliana Claudia Mayumi Ishikawa
Membro titular

_____
Profª. Msc. Simone de Almeida
Coordenadora do Curso
UTFPR - Câmpus Ponta Grossa

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

RESUMO

DELAZERI, Bruna Rossetto; WOLF, Ellen Cristina. **Modelagem de um Sistema Organizador Baseado em Linhas de Produto.** 2012. 84 f. Trabalho de Conclusão de Curso Tecnologia em Analise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2012.

O desenvolvimento baseado em linha de produto permite a identificação das diferenças e semelhanças entre aplicações de forma que se possa construir uma arquitetura genérica que as atenda em suas especificidades e variabilidades. Este trabalho identificou que alguns métodos que podem ser usados para a criação de linha de produto que não possuem definidos os artefatos de entrada e saída que devem ser produzidos em cada fase e cada um deles é composto de fases diferentes. Desta forma, os métodos da literatura foram usados como base para a adaptação de um método em que se procurou definir os artefatos, aproveitando seus pontos positivos. Aplicou-se o método proposto na modelagem de um sistema organizador de compromissos de forma a produzir uma arquitetura de componentes reutilizáveis para este domínio, além de se construir os diagramas de características e casos de uso para o domínio e a aplicação. Discute-se as facilidades e dificuldades que foram identificadas durante a criação da modelagem usando os conceitos de linha de produto.

Palavras-chave: Linhas de Produto. Modelagem. Sistema Organizador. Arquitetura.

ABSTRACT

DELAZERI, Bruna Rosseto; WOLF, Ellen Cristina. **Modeling of a Organization System based on Product in Line**. 2012. 84f. Trabalho de Conclusão de Curso Tecnologia em Analise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2012.

The development based on Product in Lines allows the identification of the differences and similarities between applications so that be possible to build a generic architecture that meets on yours specificities and variability. This works identified some methods that can be used for the creation of the product in line but don't have defined the input and output artifacts that must be produced on which phase and which one is formed of different phases. However, the methods from the literature were used has a base for the adaption of a method where was looked for defined the artifacts, taking advantage of their positive points. The proposed method was applied on an organization system modeling on the way to produce a architecture of reusable components for that domain, also build the features and use cases diagrams for the application domain. Analyzes also the facilities and the difficulties that were indentified during the creation of the modeling using the concepts of the product in line.

Keywords: Product inLine. Modeling. Organization System. Architecture.

LISTA DE ILUSTRAÇÕES

Figura 1 - Atividades essenciais no processo de linhas de produtos 17
Figura 2 - Atividades do método FODA.....25
Figura 3 - Processo do desenvolvimento ESPLEP 28
Figura 4 - Atividade de engenharia de linha de produtos do ESPLEP 28
Figura 5 - Fases do método FAST 30
Figura 6 - Interface Odyssey 34
Figura 7 - Interface fmp 35
Figura 8 - Interface XFeature 37
Figura 9 - Interface pure::variants 38
Figura 10 - Modelo de fases do Método Adaptado.....41
Figura 11 - Modelo de Contexto 49
Figura 12 – OpenProj – Cadastro de Projeto 50
Figura 13 - OpenProj – Tarefas.....51
Figura 14 - OpenProj – Predecessora.....51
Figura 15 - OpenProj - Calculo do Tempo.....52
Figura 16 - OpenProj - Dias de Folga.....52
Figura 17 - Tarefas Atrasadas.....53
Figura 18 - Google Agenda - Cadastro de Evento 53
Figura 19 - Google Agenda - Repetir Evento 54
Figura 20 - Google Agenda - Adicionar Lembrete 54
Figura 21 - Google Agenda - Compromissos por Data 55
Figura 22 - Google Agenda - Tempo Ocupado 55
Figura 23 - Google Agenda - Próximos Compromissos 56
Figura 24 - Diagrama de Características das Similaridades 58
Figura 25 - Diagrama de Caso de Uso das Similaridades Agenda/ Cronograma 59
Figura 26 - Interfaces do Software Organização dos casos de uso das similaridades Agenda/Cronograma 63
Figura 27 - Interfaces com Métodos do Software Organização dos casos de uso das similaridades Agenda/Cronograma 64
Figura 28 - Modelo de Tipo de Negócio Agenda/Cronograma 65
Figura 29 Diagrama de responsabilidades das interfaces do modelo de tipo de negócio.....66
Figura 30 - Configuração arquitetural na camada de sistema.....67
Figura 31 - Componentes das interfaces de negócios 67
Figura 32 Arquitetura do Software de Organização 68
Figura 33 - Variabilidades do Cronograma.....69
Figura 34 - Variabilidades da Agenda 70
Figura 35 - Caso de Uso das Variabilidades do Cronograma 70
Figura 36 - Caso de Uso das Variabilidades da Agenda.....72

Figura 37 - Modelos de classe para instanciação do Cronograma.....76
Figura 38 - Modelos de classe para instanciação da Agenda 76
Figura 39 – Modelo de classes para instanciação do Post-it 80

LISTA DE QUADROS

Quadro 1 - Comparação entre os métodos de desenvolvimento de Linhas de Produtos 32
Quadro 2 - Diagramas que cada Método Utiliza 32
Quadro 3 - Quadro comparativo das ferramentas LPS 39
Quadro 4 - Quadro demonstrando os modelos que cada software possui 40
Quadro 5 - Artefatos de entrada e saída produzidos no Método Adaptado 46
Quadro 6 - Lista de Requisitos 56
Quadro 7 - Cenário para o Caso de Uso “Acionar Lembrete” 59
Quadro 8 – Cenário para o Caso de Uso “Manter Tarefa/Compromisso” 60
Quadro 9 - Cenário para o Caso de Uso “Validar Dados” 60
Quadro 10 - Cenário para o Caso de Uso “Visualizar Tempo Ocupado” 61
Quadro 11 - Cenário para o Caso de Uso “Verificar Tempo Restante” 71
Quadro 12 - Cenário para o Caso de Uso “Manter Projeto” 71
Quadro 13 - Cenário para o Caso de Uso “Adicionar Dias de Folga” 72
Quadro 14 - Cenário para o Caso de Uso “Visualizar Compromissos por Data” 73
Quadro 15 - Cenário para o Caso de Uso “Pesquisar Data” 73
Quadro 16 - Cenário para o Caso de Uso “Visualizar Próximos Compromissos” 74
Quadro 17 - Cenário para o Caso de Uso “Definir Data/Hora do Lembrete” 74
Quadro 18 - Cenário para o Caso de Uso “Repetir Evento” 75

LISTA DE SIGLAS

LPS	Linhas de Produtos de Software
FODA	*Feature Oriented Domain Analysis*
DARTS	*Design Approach for Real-Time Systems*
PLUS	*Product Line UML-Based Software Engineering*
ESPLEP	*Evolutionary Software Product Line Engineering*
RUP	*Rational Unified Process*
FAST	*Family-Oriented Abstraction, Specification and Translaction*
FMP	*Feature Modeling Plug-In*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*
API	*Application Programming Interface*
XSLT	*eXtensible Stylesheet Language for Transformation*
GEF	*Graphical Editing Framework*
XSL	*EXtensible Stylesheet Language*

SUMÁRIO

TERMO DE APROVAÇÃO 3
1 INTRODUÇÃO **13**
1.1 OBJETIVOS 14
1.1.1 Objetivo Geral 14
1.1.2 Objetivos Específicos 14
1.2 ORGANIZAÇÃO DO TRABALHO 15
2 LINHAS DE PRODUTOS (*PRODUCT IN LINE*) **16**
2.1 DESENVOLVIMENTO BASEADO EM LINHAS DE PRODUTOS 16
2.1.1 Engenharia de Domínio 20
2.1.2 Engenharia de Aplicação 22
2.1.3 Gerenciamento 24
2.2 MÉTODOS PARA A CONSTRUÇÃO DE LPS 24
2.2.1 FODA (Feature Oriented Domain Analysis) 24
2.2.2 PLUS (Product Line UML-Based Software Engineering) 26
2.2.3 FAST (Family-Oriented Abstraction, Specification and Translaction) 29
2.2.4 Análise Comparativa dos Métodos de Desevolvimento LPs 31
2.3 FERRAMENTAS DE APOIO PARA A CONSTRUÇÃO DE SOFTWARE BASEADO EM LINHAS DE PRODUTO 33
2.3.1 Sistema ODYSSEY 33
2.3.2 fmp 34
2.3.3 XFeature 36
2.3.4 pure::variants 37
2.3.5 Comparação entre as ferramentas 38
3 MÉTODO PROPOSTO PARA DESENVOLVIMENTO DE LPS **41**
3.1 PROCESSO GERAL 41
3.2 ENGENHARIA DE DOMÍNIO 42
3.2.1 Análise de Domínio 43
3.2.2 Requisitos do Domínio 43
3.2.3 Modelagem do Domínio 43
3.2.4 Projeto do Domínio 44
3.2.5 Implementação do Domínio 44
3.2.6 Testes do Domínio 44
3.3 Engenharia de Aplicação 44
3.3.1 Requisistos da Aplicação 44
3.3.2 Implementação da Aplicação 45
3.3.3 Testes da Aplicação 45
3.3.4 Entrega e Suporte da Aplicação 45
3.4 ARTEFATOS PRODUZIDOS EM CADA FASE DO MÉTODO PROPOSTO 45

4 APLICAÇÃO DO MÉTODO PROPOSTO: MODELAGEM DE UM SISTEMA ORGANIZADOR 48
4.1 ANÁLISE DE DOMÍNIO 48
4.2 IDENTIFICAÇÃO DE CARACTERISTICAS 49
4.2.1 Requisitos de Domínio 49
4.2.2 Modelagem do Domínio 58
4.2.3 Projeto do Domínio 61
4.2.3.1 Modelagem da Arquitetura 63
4.2.3.1.1 Identificação dos Componentes 63
4.3 REQUISITOS DA APLICAÇÃO 68
5 RESULTADOS E DISCUSSÕES SOBRE A MODELAGEM BASEADA EM LINHA DE PRODUTO 78
5.1 DIFICULDADES E LIMITAÇÕES NA CONSTRUÇÃO DE UMA LINHA DE PRODUTO 78
5.2 VANTAGEM DE UTILIZAR LINHAS DE PRODUTO NA AGENDA E NO CRONOGRAMA 79
5.3 REUSO DA LPS PROPOSTA PARA O SISTEMA ORGANIZADOR 79
5.4 MODELAGEM DE SOFTWARE BASEADO EM LP X OUTROS TIPOS DE MODELAGEM 80
6 CONCLUSÃO 81
6.1 TRABALHOS FUTUROS 82

1 INTRODUÇÃO

O aumento da produção nas indústrias de software vem crescendo significativamente (POHL; BOCKLE; LINDEN, 2008), por consequência aumenta também a necessidade de produção rápida sem deixar de lado a qualidade. Uma das formas das empresas se adequarem à esta mudança sem perder o mercado é a reusabilidade (NEIVA, 2008).

Linhas de produto de software (LPS) é uma solução útil e viável para a questão do reuso de software. Este é um paradigma de desenvolvimento que tem como objetivo não apenas o reuso de código, mas também da arquitetura e dos requisitos. Além disso, a LPS proporciona diminuição nos custos de desenvolvimento, aumento da qualidade e reduz o tempo de entrega do produto (NEIVA, 2008).

A LPS abrange um conjunto de softwares similares (família) dentro de um domínio e enfatiza as similaridades e variabilidades entre eles de modo a ficar explicito as partes que poderão ser utilizadas ou adaptadas à nova aplicação (GIMENES; TRAVASSOS, 2002).

O método FAST (*Family-Oriented Abstraction, Specification and Translaction*) se mostra como uma alternativa ao processo clássico de desenvolvimento de software, tornando-o completo e baseado em Linhas de Produto (HARSU, 2002). O método PLUS (*Product Line UML-Based Software Engineering*) é baseado em UML e é compatível com os métodos convencionais de construção de software RUP e Espiral, além disso, é baseado em *features* (TEIXEIRA, 2007). O método FODA (*Feature Oriented Domain Analysis*) também é baseado em *features* e possui como objetivo a identificação das características comuns e variáveis que possam ser reutilizadas em softwares de mesmo domínio (LOBO;RUBIRA, 2009).

Neste trabalho é apresentado um método adaptado para construção de software em LPS baseando-se nos métodos já existentes na literatura: FAST , PLUS e FODA. O método proposto foi adaptado de outros em que se utilizou suas melhores práticas e incluiu-se novos artefatos, pois os métodos já existentes não deixam explicito os artefatos que dão total suporte a modelagem baseada em linhas de produto.

Aplicou-se o método em um Sistema Organizador de forma a produzir os artefatos de entrada e saída para cada fase e subfase e com isto gerar a modelagem para o sistema.

Como resultado criou-se uma arquitetura de domínio que contempla os pontos comuns entre os sistemas analisados pertencentes ao Organizador, a saber, *Agenda* e *Cronograma*. Além disto, no modelo instanciação dos produtos, usando diagrama de classes, fica visível os pontos de variabilidades entre eles. Comenta também as facilidades e dificuldades em se usar a modelagem baseada em linha de produto.

1.1 OBJETIVOS

Os objetivos do trabalho são descritos a seguir. A subseção 1.1.1 descreve o objetivo geral do trabalho. A subseção 1.1.2 relata os objetivos específicos do mesmo.

1.1.1 Objetivo Geral

Criar e aplicar um método baseado em Linhas de Produtos de Software na modelagem de um sistema de Organização de Compromissos, identificando-se as variabilidades e similaridades.

1.1.2 Objetivos Específicos

- Analisar os métodos e ferramentas para construção de linhas de produtos de software.
- Criar um método baseado em linhas de produtos de software.
- Aplicar o método adaptado em aplicação real criando sua modelagem.

1.2 ORGANIZAÇÃO DO TRABALHO

Este trabalho é constituído de quatro capítulos. O Capítulo 2 define o desenvolvimento baseado Linhas de Produtos de Software, suas fases, seus métodos e ferramentas de apoio.

O Capítulo 3 apresenta um método adaptado para criação de sistemas baseados em LPs. O Capítulo 4 descreve as similaridades e variabilidades de uma Agenda e um Cronograma que fazem parte do sistema de organização. O Capítulo 5 discute as evidências resultantes do uso de um método para modelagem de uma aplicação baseada em LPs.

Por fim, o último capítulo apresenta as considerações finais sobre esta pesquisa e expõe os prováveis trabalhos futuros que podem ser desenvolvidos a partir deste estudo.

2 LINHAS DE PRODUTOS (*PRODUCT IN LINE*)

Este capítulo apresenta a abordagem do desenvolvimento de software baseado em linhas de produto, seus métodos e ferramentas, que proveem uma aplicação de reuso em linhas de código para sistemas similares. Isto permite um ganho de produtividade em larga escala, mas garante a qualidade do produto. A Seção 2.1 descreve o conceito sobre linhas de produtos de software, bem como suas vantagens. A Seção 2.2 aborda os métodos utilizados para o desenvolvimento de software baseado em linhas de produto. A seção 2.3 apresenta as ferramentas mais utilizadas para a construção de software baseado em linhas de produtos.

2.1 DESENVOLVIMENTO BASEADO EM LINHAS DE PRODUTOS

Com o mercado de software aquecido, as empresas precisam acompanhar as tecnologias que estão surgindo para construir aplicativos com qualidade e de baixo custo.

Com o desenvolvimento destas tecnologias, a reusabilidade de componentes nas indústrias em geral é importante. Para utilizar este tipo de abordagem de construção é necessário que as empresas trabalhem com componentes genéricos que obedeçam a determinadas regras de padronização.

Um método que torna possível a reusabilidade de componentes chama-se Engenharia de Linhas de Produto (LPs), está é definida por Pohl, Bocke e Linden (2005, p. 37) como "... um paradigma para desenvolver aplicativos de software (sistemas de software intensivos e produtos de software) usando plataformas e customização de massa."

A abordagem de linhas de produto de software tem como principio procurar diferenças e semelhanças entre produtos e artefatos de software similares que serão construídos a partir de uma arquitetura genérica, que permite adaptar estes artefatos de acordo com suas necessidades específicas. As diferenças entre estes deverão aparecer na definição dos requisitos e serão representadas ao longo do processo (OLIVEIRA, 2009).

As linhas de produto estão ligadas diretamente com custo do projeto e seu tempo de entrega e tem como princípio o reuso de código, o que gera economia nas duas situações. Porém, quando se tem a intenção de aplicar a LPS no desenvolvimento dos projetos da empresa é necessário que esta realize um investimento inicial para que seja construído o *core asset* (núcleo) (GIMENES; TRAVASSOS, 2002).

As linhas de produto software também podem ser chamadas família de sistemas e possui três atividades em seu desenvolvimento (BRITO; COLANZI, 2010; NEIVA, 2008) ilustradas na Figura 1.

Figura 1 - Atividades essenciais no processo de linhas de produtos
Fonte: Clements; Northrop (2002)

As atividades são as seguintes:

- Engenharia do Domínio: Desenvolvimento do *core assets*, infraestutura para reuso e a base do desenvolvimento de produto. Exemplos: componentes, modelo de domínio, especificação de requisitos, entre outros.

- Engenharia da Aplicação: a partir dos *core assets* são desenvolvidos os produtos.
- Gerenciamento técnico e organizacional da linha de produto.

As organizações que utilizam linhas de produto possuem vantagem competitiva comparada com as que não a utilizam, isto porque existe um aumento da produtividade em larga escala, da qualidade, da redução de custos, do tempo de entrega e dos riscos dos produtos (CLEMENTS; NORTHROP, 2002).

Para iniciar o processo de implementação em linha de produto, a organização deve estabelecer objetivos sólidos e específicos, pois este tipo de desenvolvimento de software necessita um investimento inicial para a criação e manutenção dos *core assets*. Os custos iniciais para aplicar as linhas de produto são maiores que os lucros, porém com o passar do tempo, o aumento de produtividade supera os custos iniciais.

Os benefícios encontrados nas linhas de produto podem ser classificados da seguinte maneira (COHEN, 2003):

- Tangíveis: medidos de maneira direta, podem ser citados a redução de *time-to-market* ou redução de defeitos. São classificados como:
    - Lucratividade: permite a organização produzir artefatos para um determinado segmento, observado nos aumentos de lucratividade e da participação de mercado.
    - Qualidade: pode ser medida pela redução de tempo de correção e do efeito *ripple* (novos defeitos a partir de correções executadas).
    - Desempenho dos produtos de software: aumento de desempenho no desenvolvimento, com ativos mais otimizados.
    - Tempo de integração: tempo de desenvolvimento incremental facilitado.
    - Produtividade: equipe de desenvolvimento reduzida, custo menor, cronograma reduzido, flexibilidade documentada, facilidade no atendimento de solicitações.
- Intangíveis: são relatados pelos desenvolvedores, podendo ser:
    - Desgaste de profissionais: menor desgaste, ou seja, reduz o *turnover*.

- Aceitabilidade dos desenvolvedores: satisfação em trabalhar com tal abordagem.
- Satisfação profissional: Os desenvolvedores se concentram em atividades mais interessantes tais como: aperfeiçoamento e inovação.
- Satisfação do Cliente: redução de riscos, defeitos e melhor previsão de entrega.

Apesar dos benefícios, a implementação de uma linha de produto necessita cuidado e planejamento. Além disso, são enfrentadas dificuldades e riscos tais como (COHEN, 2003):

- Falta de um líder comprometido: quando o líder de projeto não acredita no modelo ou não possuí autoridade pode haver risco de desistência.
- Falta de compromisso da gerência: quando a gerência não esta convicta da viabilidade, a implantação de linhas de produto se torna um projeto secundário ou pode até ser esquecido.
- Abordagem inadequada: quando os produtos não possuem similaridades o modelo pode não trazer o retorno planejado.
- Falta de compromisso da equipe: assim como a gerência e o líder, a equipe deve acreditar no processo, ao contrário, o desenvolvimento é impossibilitado.
- Interação insuficiente entre as equipes: a falta de colaboração pode dificultar o alcance dos objetivos planejados.
- Padronizações desapropriadas: quando os padrões não são estudados, a tecnologia da linha pode ser prejudicada.
- Adaptação insuficiente: as práticas organizacionais que não possuem adaptação podem gerar desvios não previstos e diminuem a otimização e o desempenho das equipes.
- Evolução da abordagem: se o processo não for revisado e atualizado de maneira periódica, pode tornar as práticas do mesmo obsoletas e inapropriadas, gerando práticas não previstas para sanar necessidades.

- Falha de disseminação: o líder deve construir e distribuir documentações necessárias, treinar os envolvidos e apoiar o processo. Se isto não for cumprido, os cronogramas e os objetivos das linhas podem ser prejudicados.

Outros riscos encontrados são (COHEN, 2003; DURSCKI et. al,2004, p. 7):

> "Clientes desinformados ou céticos com relação às mudanças.
> Tendências da organização de retornar ao modelo antigo de desenvolvimento.
> Momento inoportuno para implantação."

A seguir serão descritas as fases da criação de software baseados em linhas de produtos: Engenharia de Domínio, Engenharia de Aplicação e Gerenciamento.

2.1.1 Engenharia de Domínio

A Engenharia de Domínio é a responsável pela criação do *core asset*, formado de vários *assets*, em que são armazenadas as funcionalidades necessárias dos produtos (OLIVEIRA, 2009). Esta etapa do desenvolvimento das Linhas de produtos pode ser dividida nas seguintes fases (BORBA, 2009):

Gerenciamento de Produto

Têm como objetivo integrar o desenvolvimento, a produção e o *marketing*. Isto para maximizar o investimento e satisfazer os clientes, utilizando técnicas para definir o que faz ou não parte do escopo da linha de produto.

Engenharia de Requisitos de Domínio

São identificados requisitos e analisados os pontos comuns e variáveis dos produtos de LPS. Nesta etapa os requisitos são descritos, especificados, analisados e verificados. Nesta fase, são identificados os *stakeholders* que representam qualquer pessoa envolvida no projeto (LARMAN, 2008), e que interagem com as funcionalidades dos sistemas e tornam claras suas funcionalidades (OLIVEIRA, 2009).

Os requisitos são modelados de forma gráfica e textual, e a análise destas modelagens assegura que o documento de requisitos represente as funcionalidades

reais da Linha de Produto de Software. Para modelagem são usados diagramas de caso de uso que identificam *features*, características do sistema visíveis ao usuário, que representam um ou mais casos de uso (OLIVEIRA, 2009; BORBA, 2009).

Nesta etapa também existe o desenvolvimento do modelo de *features* que as relaciona com requisitos funcionais do domínio do problema e os requisitos não funcionais. Pode-se também visualizar as funcionalidades comuns e opcionais do sistema que esta sendo modelado. A partir disso, são identificadas *features* que podem ser reutilizadas em determinado domínio especifico.

O modelo também demonstra as relações das *features* com os produtos LPS, para identificação de funcionalidades comuns e variáveis na LPS (OLIVEIRA, 2009; BORBA, 2009).

Projeto de Domínio

É escolhida uma arquitetura de alto nível do sistema que é comum em todos os produtos. Nesta será definida os pontos de variação, plataforma de suporte e a produção em larga escala, os quais serão adequados às necessidades do cliente.

Define-se também a plataforma de hardware e software onde serão construídos e implantados os produtos e especifica os componentes baseados nos modelos de classes. São utilizadas o reuso de modelos e classes de *frameworks*.

Os modelos de classes são refinados e é realizada a especificação detalhada com a criação de protótipos dos métodos e atributos dos objetos. Estruturar as classes em componentes ou padrões facilita a reutilização dos *core asset* (OLIVEIRA, 2009; BORBA, 2009).

Realização de Domínio

É feito o tratamento dos detalhes do projeto e da implementação dos componentes reutilizáveis do produto.

Testes de Domínio

Validação e verificação de componentes reutilizáveis. Os componentes são construídos e testados. O Engenheiro de Domínio gera parte do código se baseando nas classes dos componentes. Estas são testadas até que alcancem seu objetivo.

Nas linhas de produto uma característica pode estar presente em mais de um requisito e um requisito pode ser aplicado em mais de uma característica.

As variabilidades encontradas dos artefatos que farão parte da “família” dos produtos criados devem ser sempre representadas. As variabilidades são diferenças entre os produtos, descritas em termos de pontos de variação e variantes. Um ponto de variação é um lugar onde uma decisão do projeto é conectada e cada variação é associada a um conjunto de variantes correspondentes as alternativas do projeto (BRITO; COLANZI, 2010).

A engenharia de domínio também se utiliza de incrementos horizontais para o desenvolvimento das características de um produto e incrementos verticais, onde são implementadas todas as variabilidades do produto (BRITO; COLANZI, 2008).

Na disciplina de requisitos são identificados e gerenciados os requisitos para o desenvolvimento do produto. Estes requisitos podem ser comuns, isto é, todos os produtos vão possuir, ou opcionais, que são os requisitos que apenas alguns produtos vão possuir (OLIVEIRA, 2009).

Ao final da engenharia de domínio estão criados os core *assets*, modelos e códigos, organizados, estruturados e disponíveis para o reuso (OLIVEIRA, 2009).

2.1.2 Engenharia de Aplicação

A engenharia de aplicação também é chamada de desenvolvimento do produto e em sua criação possui os artefatos: modelo de domínio, que identifica os requisitos do cliente; *framework* de arquitetura de linha de produtos; e um conjunto de componentes reutilizáveis (GIMENES;TRAVASSOS, 2002).

Na Engenharia de aplicação são reutilizados os *core assets* criados na Engenharia de Domínio, para isto são consideradas as *features* obrigatórias e opcionais, pois apenas as opcionais são implementadas e as obrigatórias reutilizadas (OLIVEIRA,2009).

O processo de desenvolvimento do produto é iniciado com uma análise do modelo de domínio para especificar o que deve ser desenvolvido. Este é associado a um modelo de decisão, que considera aspectos do desenvolvimento e as necessidades que são encontradas após o início da engenharia de aplicação, os quais são chamados de novos requisitos (GIMENES;TRAVASSOS, 2002).

Após a análise do modelo de domínio, é estudado o *framework* de arquitetura, até que seja transformado em uma arquitetura específica do produto. O desenvolvimento desta arquitetura ocorre de forma iterativa.

Por fim, a arquitetura é completada com componentes adequados e os requisitos especificados. Realizam-se também testes tais como: unidade, integração, entre outros (GIMENES;TRAVASSOS, 2002).

As fases da engenharia de aplicação podem ser divididas da seguinte forma (BORBA, 2009):

Engenharia de Requisitos de Aplicação

São identificados os requisitos dos *stakeholders* para aplicação e o engenheiro modela os requisitos do produto por meio de diagramas de caso de uso, demonstrando as funcionalidades escolhidas. Após isso, são criadas *features* com o mapeamento dos casos de uso.

Nesta etapa é elaborado o documento com a lista das *features* do produto. São identificadas diferenças entre os requisitos da aplicação e os resultantes da Engenharia de Domínio (OLIVEIRA, 2009; BORBA, 2009).

Projeto de Aplicação

É a produção da arquitetura da aplicação que determina a estrutura de uma aplicação específica. Adota-se a mesma arquitetura do *core assets* e estes são reutilizados.

Um modelo de componentes é elaborado, o qual deve considerar o reuso dos componentes e do *core assets* (OLIVEIRA,2009; BORBA,2009).

Realização de Aplicação

Nesta fase o produto é desenvolvido de acordo com o desejado, isto é, realiza-se sua implementação de acordo com os componentes reutilizáveis e as características especificas da aplicação. (BORBA, 2009).

Teste de Aplicação

Validação do produto com objetivo de verificar se os requisitos dos *stakeholders* e os erros encontrados foram corrigidos. É útil possuir uma documentação dos testes (OLIVEIRA, 2009; BORBA, 2009).

2.1.3 Gerenciamento

É feito o gerenciamento do produto concluído e isto é fundamental para o sucesso do mesmo.

O gerenciamento pode ser técnico, engloba o desenvolvimento do produto e do *core assets*, ou organizacional, que trata a organização.

2.2 MÉTODOS PARA A CONSTRUÇÃO DE LPS

A seguir serão detalhados os métodos FODA, PLUS e FAST (seção 2.2.1, 2.2.2 e 2.2.3, respectivamente), os quais são utilizados para o desenvolvimento de linhas de produtos de software. Estes três métodos foram analisados porque são atualmente os mais utilizados pela sua simplicidade e também por obter resultados satisfatórios na produção de software baseado em linhas de produtos (LOBO; RUBIRA, 2009). Existem também outros métodos para construção de LPS, tais como: ODM, DSSA, PuLSE e KobrA, porém não descritos nesta pesquisa.

2.2.1 FODA (Feature Oriented Domain Analysis)

O método *Feature Oriented Domain Analysis* (FODA) foi desenvolvido pelo SEI (*Software Engineering Institute*) em 1990 e é baseado em *features*. Este consiste na análise do contexto e modelagem de domínio das aplicações, que tem como seu principal objetivo obter e representar informações de sistemas que possuem características comuns e que possam ser amplamente reutilizáveis para a concepção de outros softwares de mesmo domínio (LOBO; RUBIRA, 2009). A Figura 2 ilustra as atividades contidas no método FODA.

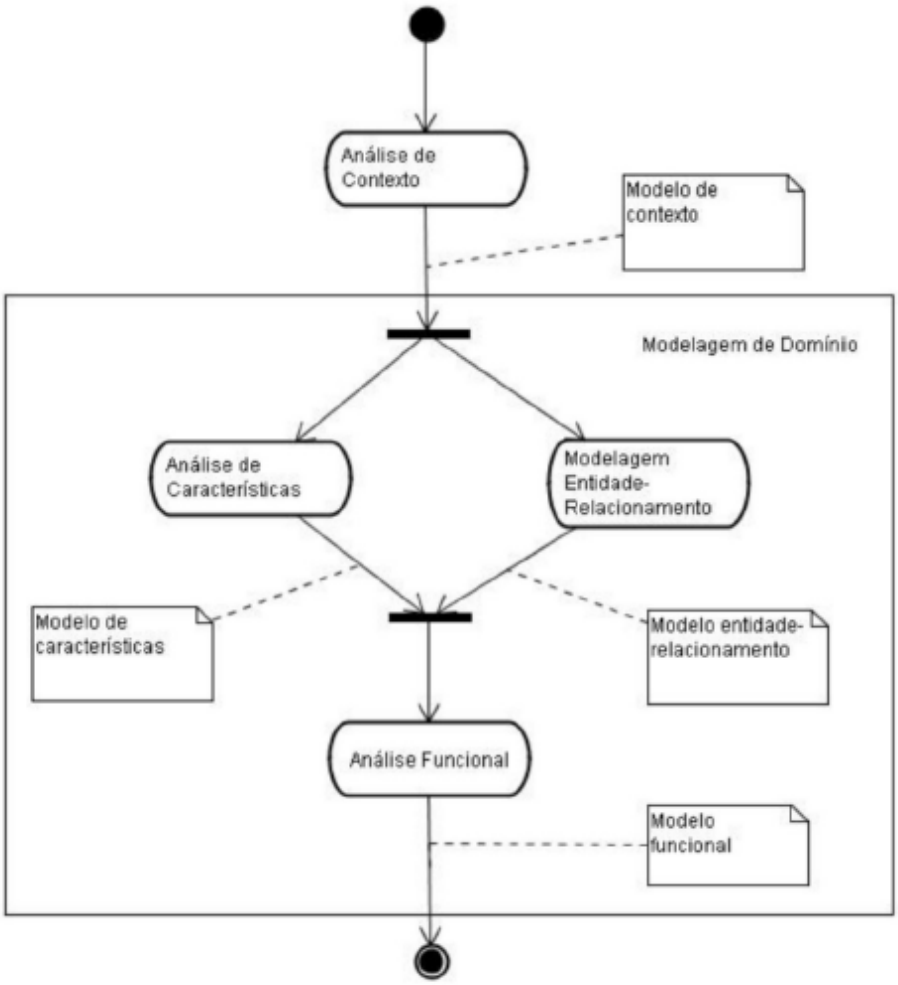

Figura 2 - Atividades do método FODA
Fonte: Lobo; Rubira (2009)

A fase de *Análise de contexto* tem como objetivo definir o escopo de um domínio que poderá ser explorado para obter seus artefatos. No final desta fase é criado um documento de modelo de contexto, o qual define o escopo da modelagem de domínio (LOBO; RUBIRA, 2009; KANG et al., 1990).

Na fase de *Modelagem de domínio* são analisadas as características comuns e variáveis entre as aplicações. Esta fase possui três subatividades descritas a seguir:

- Análise de Características: É considerada a principal fase do método FODA. Sua finalidade é representar em um modelo as características e relacionamentos que são comuns e variáveis entre as aplicações. Este modelo consiste na criação do diagrama de características, que contém informações tais como: descrições, restrições, dependências e

lógicas de uso. O processo de análise de características é dividido em três subfases:

- Identificação das características.
- Abstração e classificação das características como um modelo e definição das características (modelagem das características).
- Validação de modelo.

- Modelagem Entidade-Relacionamento: captura e define o conhecimento de domínio, representando explicitamente suas entidades e relacionamentos.
- Análise Funcional: identifica as funcionalidades comuns e variáveis do domínio entre as aplicações. As funcionalidades comuns são alienadas em um modelo funcional. A identificação do modelo funcional pode ser classificada em duas partes:
  - Especificações de funções: representada por diagrama de atividades.
  - Especificações de comportamento: representada por diagrama de estados.

O resultado final da fase de *Modelagem de domínio* é validado com um modelo funcional, que pode ser representado por um diagrama de caso de uso (LOBO; RUBIRA, 2009). Concebe-se também a modelagem da arquitetura, a qual tem o objetivo de fornecer uma “solução” para os problemas da aplicação em questão. A arquitetura é feita em camadas para que a reutilização do código possa ser adequada e para que a alteração possa ser facilmente localizada dentro da aplicação.

No modelo de arquitetura do método FODA o foco está na identificação dos processos comuns nas aplicações, na alocação dos recursos e nas funções. Utiliza também a metodologia DARTS (*Design Approach for Real-Time Systems*) para desenvolver a modelagem da arquitetura (KANG et al., 1990).

2.2.2 PLUS (Product Line UML-Based Software Engineering)

O método de desenvolvimento de linhas de produtos de software PLUS (*Product Line UML-Based Software Engineering*) foi desenvolvido baseado em UML

(*Unified Modeling Language*), e está fundamentado no processo *Evolutionary Software Product Line Engineering* (ESPLEP) – processo de desenvolvimento iterativo e orientado à objetos (TEIXEIRA, 2007).

Este método é compatível com os de construção de software RUP (*Rational Unified Process*) e o Espiral, tendo como principal objetivo explicitar as características comuns e variáveis do software, além disso, é baseado em *features* (TEIXEIRA, 2007).

O ESPLEP consiste em duas atividades principais, ilustradas na Figura 3, descritas a seguir:

- Engenharia de Linhas de Produtos (*Software Product Line Engineering*): O objetivo desta atividade é produzir artefatos reutilizáveis que modelam o domínio da linha de produto. Para isto, desenvolve-se um: modelo de caso de uso, modelo de análise e da arquitetura de linhas de produto e também componentes reusáveis. Após o término dos modelos, estes são guardados em um repositório da linha de produto de software.
- Engenharia de Aplicação (*Software Application Engineering*): Esta é a última fase do método PLUS, na qual um membro da LPS desenvolve uma aplicação individual. Para o desenvolvimento dos modelos desta aplicação são utilizados os modelos contidos no repositório, sendo eles adaptados para construção dos modelos de aplicação.

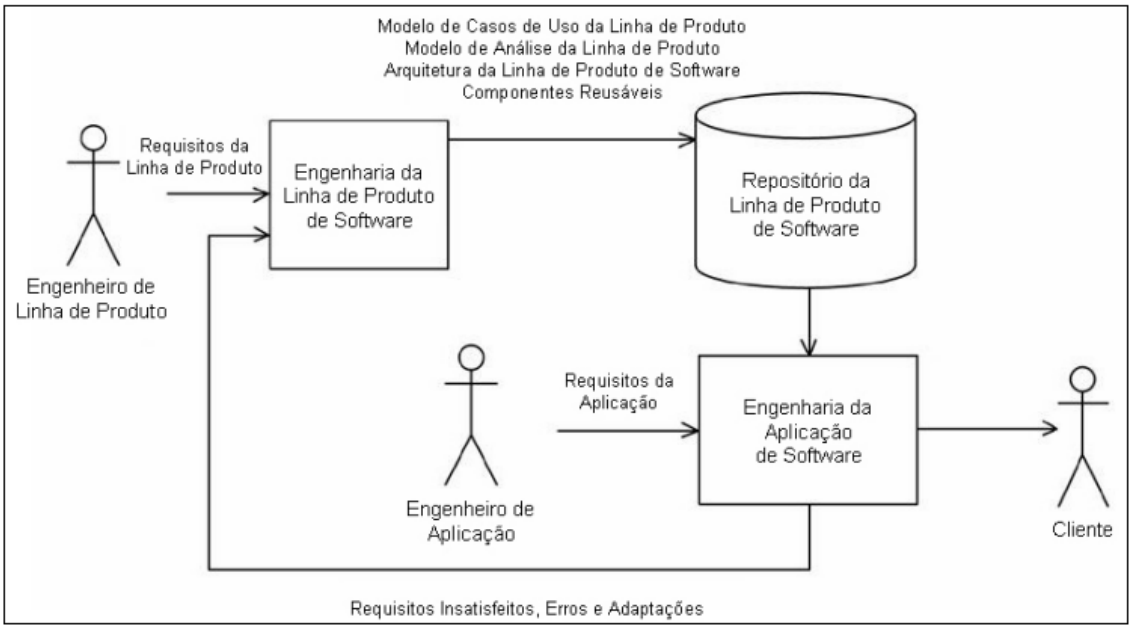

Figura 3 - Processo do desenvolvimento ESPLEP
Fonte: Teixeira (2007)

A figura 4 ilustra detalhadamente a atividade de engenharia de linhas de produtos baseada na metodologia ESPLEP.

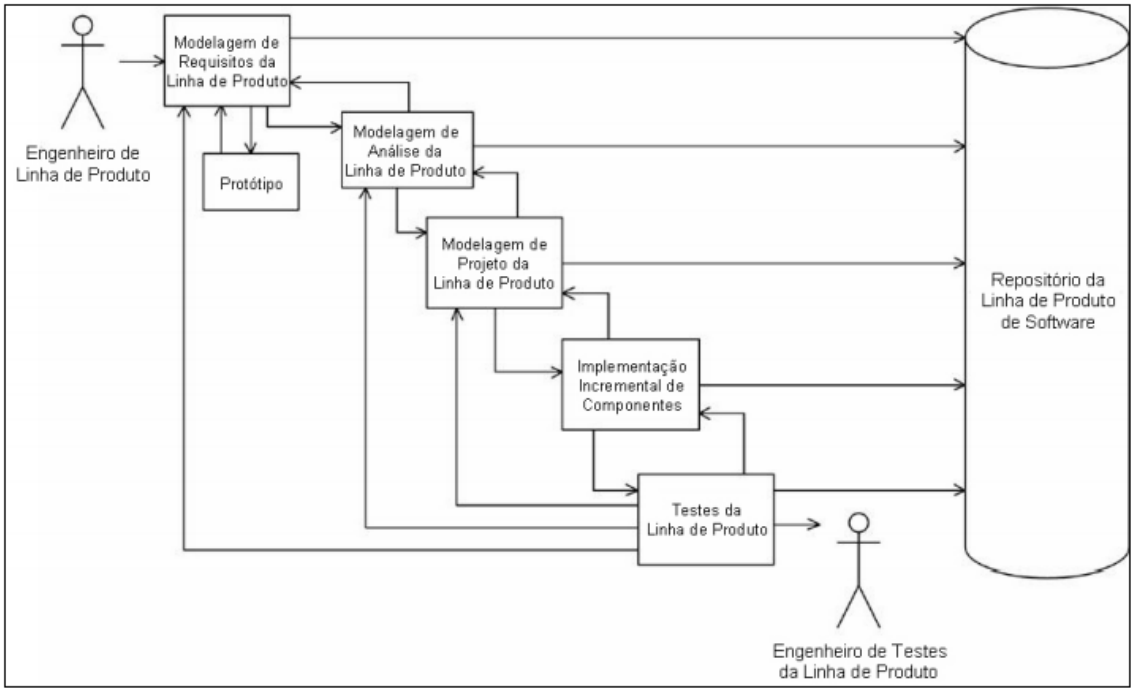

Figura 4 - Atividade de engenharia de linha de produtos do ESPLEP
Fonte: Araújo (2010)

A seguir descreve-se resumidamente cada uma das atividades apresentadas na Figura 4 (TEIXEIRA, 2007):

- Modelagem de Requisitos: Nesta fase se definem os requisitos comuns e variáveis entre os produtos. No final, é desenvolvida a modelagem do caso de uso e as *features*, que são requisitos e características reutilizáveis em uma linha de produto de software.
- Modelagem de Análise: Nesta fase é feita a decomposição do problema para que se possa melhor entendê-lo. Ao final, desenvolvem-se a modelagem estática – define o relacionamento estrutural entre as classes do domínio – e a modelagem dinâmica – casos de uso são descritos por meio de diagramas de sequência e comunicação.
- Modelagem de Projeto: Os artefatos descritos nas fases anteriores são sintetizados. Ao final, cria-se o modelo do projeto de software baseado em componentes.
- Implementação Incremental de Componentes: Um subconjunto da linha de produto é separado para que seja implementado em cada iteração. Esta implementação é feita nos casos de uso com o detalhamento da arquitetura, codificação e teste dos componentes.
- Teste: É realizado os testes dos componentes de LPS com ênfase em sua integridade e funcionalidade.

O ESPLEP é desenvolvido baseado em casos de uso, portanto, no modelo de requisitos as funcionalidades são definidas como atores e casos de uso, no modelo de análise os casos de uso são descritos em forma de objetos que participam das iterações. Após a criação destes modelos, a arquitetura baseada em componentes é desenvolvida.

2.2.3 FAST (Family-Oriented Abstraction, Specification and Translaction)

A *Family-Oriented Abstraction, Specification, and Translation* (FAST) foi criada por David Weiss no início de 1990. Este é um método que define um processo completo de engenharia de software em linhas de produto, ou seja, é uma alternativa ao processo clássico de desenvolvimento de software e tem como objetivo tornar o software mais eficiente com a redução das múltiplas tarefas,

redução do custo de produção e o refinamento do tempo de comercialização (MATINLASSI, 2004; ARAGÓN, 2004).

O método FAST é dividido em três fases: Qualificação de Domínio, Engenharia de Domínio e Engenharia de Aplicação, como mostra a figura 5.

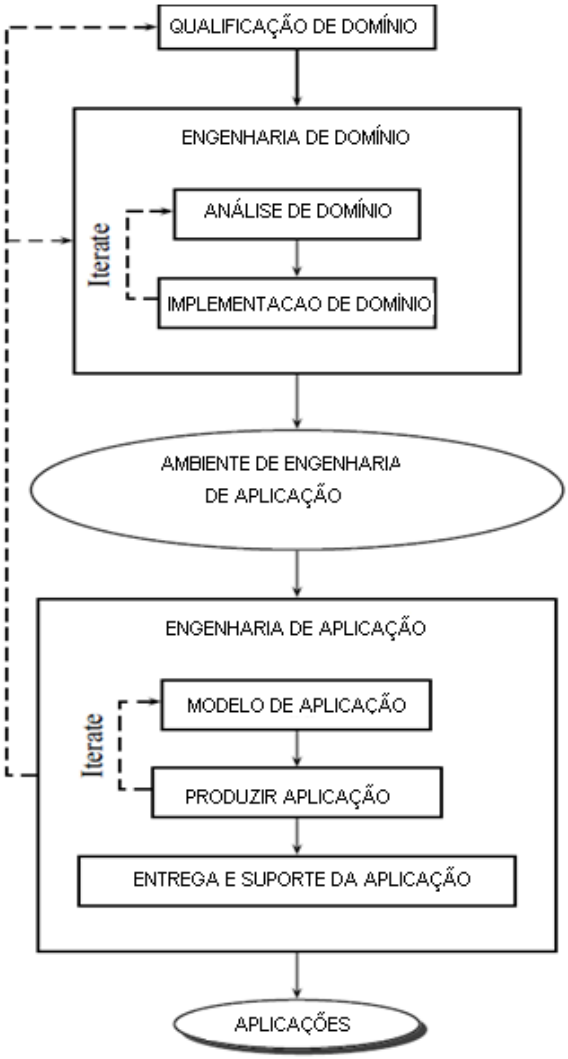

Figura 5 - Fases do método FAST
Fonte: Harsu (2002).

A fase de *Qualificação de Domínio* é caracterizada pela análise da família dos produtos. A primeira família é denominada de Potencial, a qual se qualifica por um conjunto de domínios que possuem *features* para serem analisadas. Em seguida têm-se a Semi-família, a qual tem por objetivo identificar as variabilidades e similaridades dentro do domínio. Logo após, a família Definida, que tem por objetivo fazer a análise econômica realizando uma comparação de custo-benefício do

domínio em questão. Por último, a família Projetada, que define a aplicação do processo de LPS (ARAGÓN, 2004).

Na fase de *Engenharia de Domínio* são identificados os requisitos comuns e específicos dentro da família do software (HARSU, 2002).

A fase de *Engenharia de Aplicação* pode ser feita em paralelo à fase de *Engenharia de Domínio* e utiliza os requisitos comuns para que rapidamente possa ser feita outra aplicação da família do software.

A *Engenharia de Aplicação* é um processo iterativo, onde o cliente ajuda na identificação e refinamento dos requisitos. Caso o cliente não fique satisfeito com a nova aplicação os requisitos são novamente refinados e o processo volta ao início (HARSU, 2002).

2.2.4 ANÁLISE COMPARATIVA DOS MÉTODOS DE DESEVOLVIMENTO LPS

Para fazer a comparação entre os métodos de desenvolvimento de software baseado em linhas de produto foram usadas as seguintes características identificadas no trabalho de Aragón (2004):

- Tipo de Abordagem: Baseado em *features* ou baseado em famílias.
- Modificação: Esta característica é dividida em três outras subcaracterísicas, são elas:
  - Caixa – Branca: Componentes são reutilizados por modificação e adaptação.
  - Caixa – Preta: Componentes são reutilizados sem modificação.
  - Adaptativo: Utiliza grandes estruturas de softwares como invariantes e restringe a variabilidade a um conjunto de argumentos ou parâmetros.
- Alcance do Domínio: Esta característica é dividida em outras duas subcaracterísticas, são elas:
  - Vertical: Reutilização dentro do mesmo domínio de aplicação.
  - Horizontal: Reutilização dentro de diferentes domínios de aplicação.
- Alcance do Desenvolvimento: Esta característica é dividida em outras duas subcaracterísticas, são elas:

- Interno: Mede o nível de reutilização de componentes de um repositório do mesmo projeto.
- Externo: Mede o nível de reutilização de componentes que provêm de repositórios externos ou a proporção de produtos que foram adquiridos.

O Quadro 1 apresenta uma comparação entre os métodos FODA, FAST e PLUS considerando as características relatadas anteriormente.

Quadro 1 - Comparação entre os métodos de desenvolvimento de Linhas de Produtos

CARACTERÍSTICAS	METODOS		
	FODA	**FAST**	**PLUS**
Tipo de Abordagem	Baseado em *features*	Baseado em família	Baseado em *features*
Modificação	Caixa- Branca	Adaptativo	Caixa- Branca Adaptativo
Alcance do Domínio	Horizontal	Vertical	Horizontal
Alcance do Desenvolvimento	Externo	Interno	Externo

Fonte: Adaptado de Aragón (2004)

Observa-se neste quadro que os métodos FODA e PLUS possuem características semelhantes, já em relação ao FAST são diferentes. Isto mostra a diversificação entre os métodos que contém uma mesma finalidade, isto é, aplicação de Linhas de Produto de Software.

O Quadro 2 mostra os diagramas que são utilizados em cada método de linhas de produtos de software. Estes diagramas foram identificados por meio do estudo realizado sobre os métodos de desenvolvimento baseado em linhas de produto.

Quadro 2 - Diagramas que cada Método Utiliza

DIAGRAMAS	MÉTODO		
	FODA	**FAST**	**PLUS**
Caso de Uso	X	X	X
Classe	X	X	-
Sequência	-	-	X
Estado	-	X	X
Modelo de Características	X	X	-

Fonte: Autoria Própria

Observa-se que o diagrama de Caso de Uso é usado por todos os métodos, tornando-se assim necessário durante a modelagem baseada em linhas de produto.

Os diagramas: Classe, Estado e Modelo de Características; são também importantes para a criação do modelo da aplicação, apesar de serem usados por 67% dos métodos. Nota-se que os outros diagramas, Modelo de Análise e de Sequencia, podem ser considerados opcionais, visto que são adotados somente por 33% dos métodos.

2.3 FERRAMENTAS DE APOIO PARA A CONSTRUÇÃO DE SOFTWARE BASEADO EM LINHAS DE PRODUTO

Nesta seção serão apresentadas as ferramentas de apoio a construção de softwares usando linhas de produto, sendo estas as mais citadas nas referências bibiográficas: *Odyssey*, *fmp*, *XFeature*, *pure:variants* e *Gears*.

2.3.1 Sistema ODYSSEY

Ambiente desenvolvido pela COPPE/UFRJ com a finalidade de criar uma base para reutilização baseada em modelagem de domínio, linhas de produto e desenvolvimento baseado em componentes.

Uma das versões do *Odyssey*, representado na Figura 6 chama-se *OdysseyLight*, que possui funcionalidades de apoio a atividades de Engenharia de Domínio e Engenharia de Aplicação.

Suas ferramentas de núcleo são: editor de diagramas UML, editor de diagramas de características, navegador inteligente, gerador de aplicações, máquina de processos, suporte a desenvolvimento baseado em componentes, persistência objeto-relacional e documentação de artefatos de software (LOBO, RUBIRA, 2009).

Esta ferramenta é utilizada por engenheiros e especialistas de domínio e aplicação, pois permite modelar os diagramas e as características de cada nó separadamente. Os diagramas que o *Odyssey* possui são: diagrama de contexto, diagrama de *features*, diagrama de negócios, diagrama de casos de uso e diagrama estrutural (PROJETO YANA, 2010).

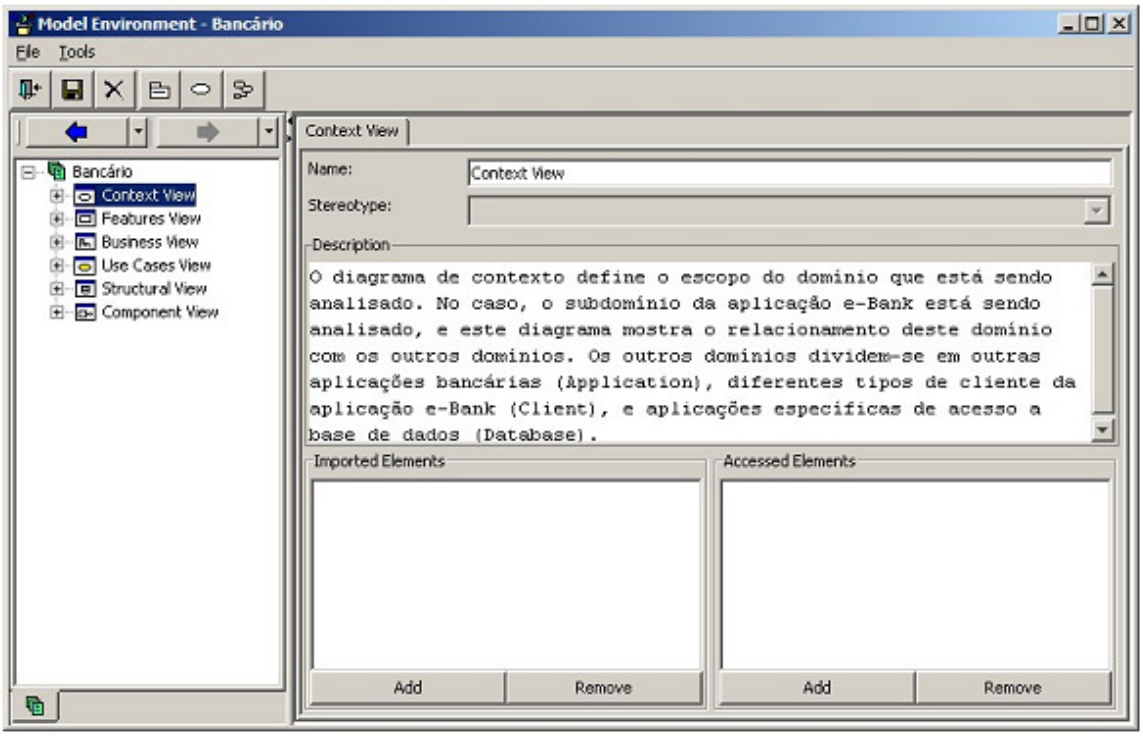

Figura 6 - Interface Odyssey
Fonte: Lobo; Rubira (2009)

A notação nativa para modelar *features* é chamada *Odyssey-FEX* e é uma extensão proposta no método FODA. Esta notação representa as características das *features* no próprio diagrama e relacionamentos baseados em UML (PROJETO YANA, 2010).

2.3.2 fmp

O *fmp*, representado na Figura 7 é um *plug-in* do Eclipse desenvolvido pela *Generative Software Development Lab*, da *University of Waterloo*. É uma ferramenta gratuita e *open source* que utiliza como base o *Eclipse Modeling Framework* (SILVA et al., 2011; LIMA, 2008; GENERATIVE SOFTWARE DEVELOPMENT LAB, 2012).

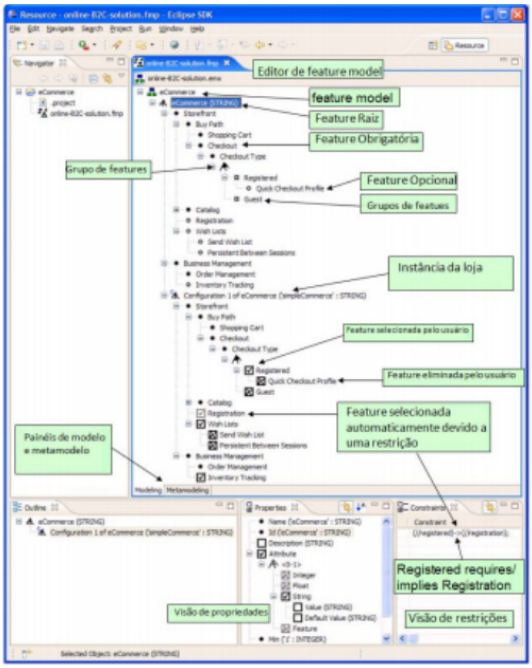

Figura 7 - Interface fmp
Fonte: Silva et al. (2011)

No Eclipse o *fmp* é utilizado sozinho, porém quando na *Rational Software Modeler* ou no *Rational Software Architect* ele pode ser utilizado em conjunto com o *fmp2rsm*, um *plug-in* que permite ao fmp modelar em UML (GENERATIVE SOFTWARE DEVELOPMENT LAB, 2012).

A ferramenta *fmp* possui suporte a modelagem de domínio, porém não contém o *configuration knowlegde* e nenhuma outra funcionalidade que permita o mapeamento entre *features* e artefatos da linha de produtos (LIMA, 2008).

O *fmp* possui um *background* acadêmico e grava seus modelos de *feature model* em XML, o que permite que geradores que utilizam XSLT processem o modelo. Possui API bem definida e a possibilidade de integração com outras ferramentas. Sua modelagem é baseada em (SILVA et al., 2011; LIMA, 2008):

- Cardinalidade de *feature* e de grupo.
- Atributos de *feature*.
- Referencias e anotações definidas pelo usuário.

A cardinalidade possibilita que sejam especificados os números máximo e mínimo de filhos de uma *feature* que podem ser selecionados e os atributos permitem que as *features* possuam valores. As anotações adicionam novas

propriedades às *features*, a união destas aumenta a expressividade e customização do modelo (SILVA et al., 2011; LIMA, 2008).

2.3.3 XFeature

Desenvolvido pela P&P Software, é uma ferramenta que automatiza o processo de modelagem e configuração de artefatos reusáveis de software. É gratuito e seu código esta disponível para correções e inclusão de novas *features* que estejam de acordo com o padrão do *XFeature* (SILVA et al., 2011; LIMA, 2008; P&P SOFTWARE and ETH ZÜRICH, 2012).

Este *plug-in* possui a possibilidade de customização do metamodelo da família de produtos e o suporte para a modelagem de domínio, porém não possui a funcionalidade de mapeamento entre *features* e artefatos. Seu editor é baseado no GEF (*Graphical Editing Framework*) do Eclipse e seu *background* é acadêmico e corporativo (LIMA, 2008).

Depende do XML e de transformações XSL para sua utilização. Possui um processo de criação complexo, porém permite ao usuário maior controle sobre como são criados os *feature models*, podendo escolher a modelagem de *features* a ser utilizada.(LIMA, 2008)

Os modelos e instâncias podem ser validados contra seu metamodelo e após a sua validação podem ser usados como entrada para outras ferramentas. A edição e visualização dos modelos no *XFeature* é em forma de árvore, conforme ilustrado na Figura 8 (SILVA et al., 2011; LIMA, 2008).

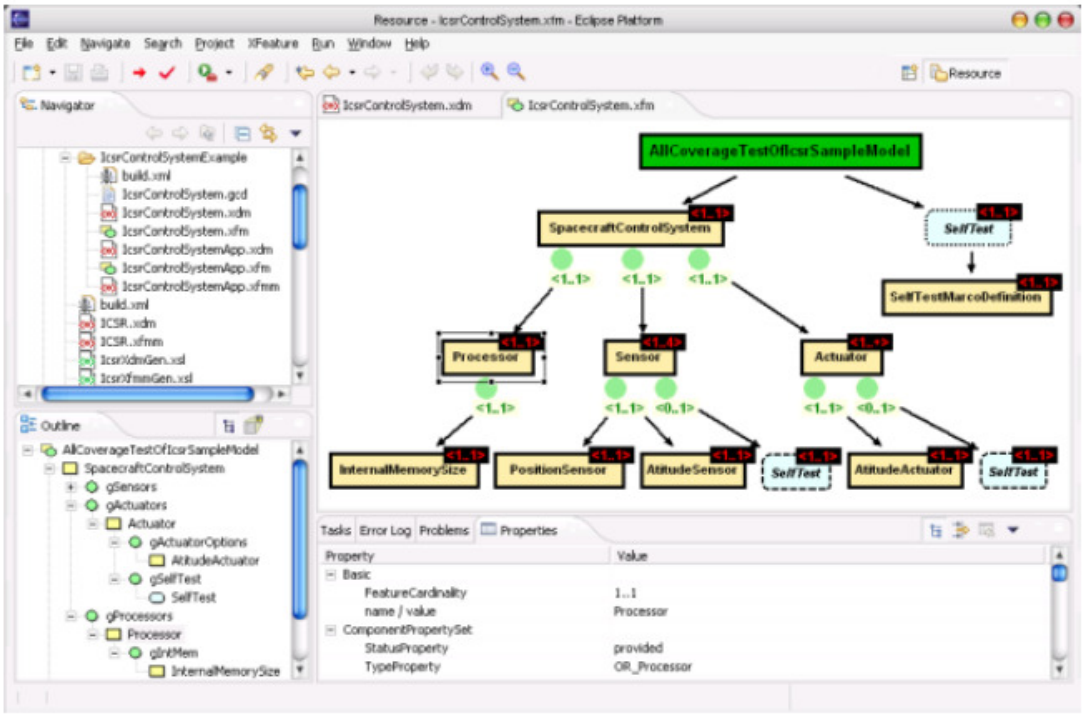

Figura 8 - Interface XFeature
Fonte: Silva et al. (2011)

2.3.4 pure::variants

Desenvolvido pela *GmbH*, o software *pure::variantes*, representado na Figura 9 tem como objetivo de suportar o desenvolvimento e a implantação de linhas de produto e famílias de software, provendo suporte nas atividades de análise, modelagem, implementação e implantação. Ou seja, pode ser usada em todas as partes do desenvolvimento de produtos de software (LIMA, 2008; PURE-SYSTEMS GMBH, 2012).

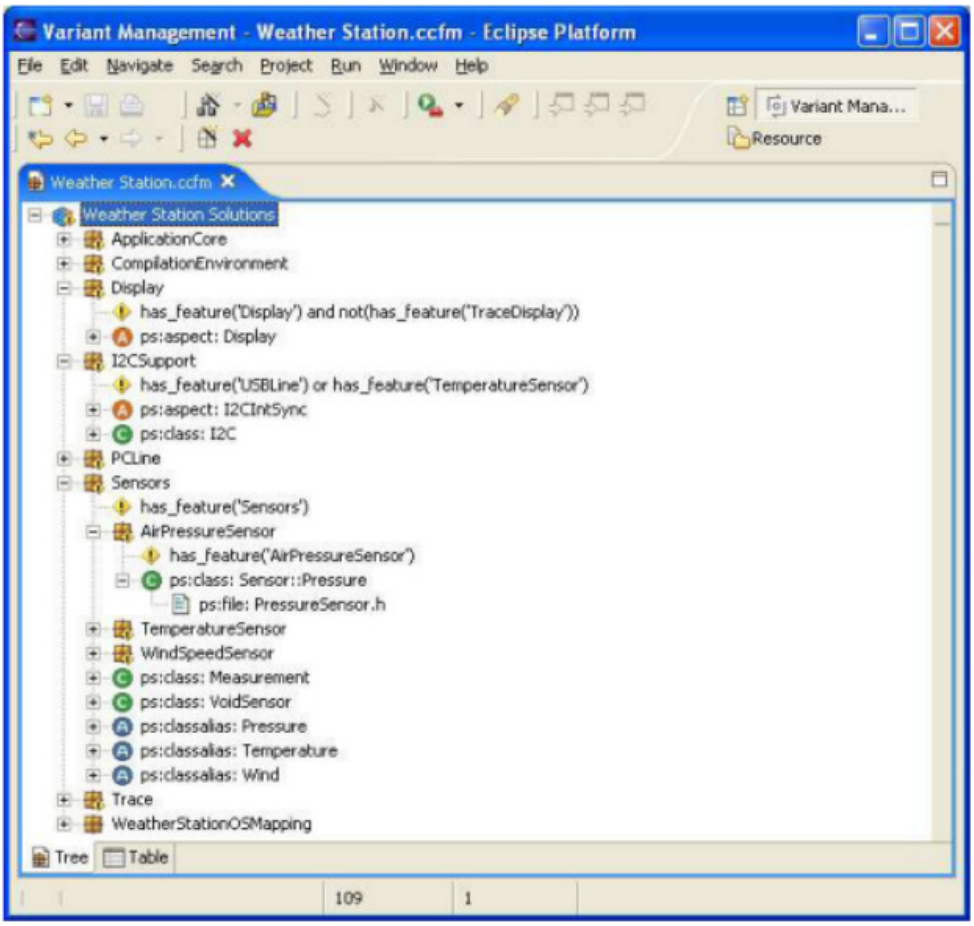

Figura 9 - Interface pure::variants
Fonte: Silva et al. (2011)

Suporta modelagem do domínio e possui *Configuration Knowledge*, funcionalidade que permite o mapeamento entre *features* e artefatos.

A *pure::variants* é uma ferramenta paga e seu preço depende da quantidade de licenças e versão. É baseada no método FODA e possui três tipos de modelos: *feature model*, onde são definidas as características comuns e variabilidades; *family model*, parte do *configuration knowlegde*; *variant models*, onde é definida uma instância da linha de produtos. Os *variant models* podem ser validados e utilizados com o *family model* e o *feature model* (SILVA et al., 2011; LIMA, 2008).

2.3.5 Comparação entre as ferramentas

Os Quadro 3 e 4 apresentam as diferenças entre as ferramentas, de acordo com os estudos realizados na literatura. De acordo com a teoria, as características

gerais identificadas com maior relevância, representadas no Quadro 3, são (LIMA, 2008):

- Gratuita: tipo de licença da ferramenta.
- Facilidade de uso: se a ferramenta é considerada de fácil utilização pelos pesquisadores da literatura.
- *Configuration Knowledge*: se a ferramenta possui esta funcionalidade, ou alternativa que possibilite o mapeamento entre *features* e artefatos.
- *Background Acadêmico*: qual o tipo de *background* que a ferramenta disponibiliza para o usuário.

Quadro 3 - Quadro comparativo das ferramentas LPS

Ferramentas	**Gratuito**	**Facilidade de uso**	***Configuration Knowledge***	***Background* acadêmico**
Odyssey	x	x		
Fmp	x	x		x
XFeature	x			x
pure::variants		x	x	

Fonte: Autoria própria

De acordo com o Quadro 3 no quesito Gratuito, a ferramenta *pure::variants* é paga, enquanto o *Odyssey*, *fmp* e *xfeature* são livres de taxa.

Quanto a facilidade de uso apenas a *xfeature* é considerada de nível difícil. A única ferramenta que possui *Configuration Knowledge* é a *pure::variants*. Por fim, 50% das ferramentas são de *background* acadêmico.

O Quadro 4 apresenta os tipos de modelo usados durante o desenvolvimento baseado em Linhas de Produto, sendo:

- *Feature model:* conjunto de diagramas de *Features*, isto é, diagramas para demonstrar o que há de comum entre os produtos de uma mesma linha ou para discriminá-los (LIMA, 2008).
- *Family model: meta-modelo do domínio* (LIMA, 2008;p.25).
- *Variant model: modelo que define uma instância da linha de produtos* (LIMA, 2008; p.29).

Quadro 4 - Quadro demonstrando os modelos que cada software possui

Ferramentas	***Feature model***	***Family model***	***Variant model***
Odyssey	X		
Fmp	X		
XFeature	X		
pure::variants	X	x	x

Fonte: Autoria própria

De acordo com o Quadro 4 apenas 25% das ferramentas possui todos os modelos citados na teoria deste trabalho, os outros 75% contemplam apenas o modelo *Feature Model*.

Os métodos pesquisados neste trabalho não relacionam todos os artefatos que devem ser gerados em cada fase, por isto se adaptou as fases presentes nos métodos e propôs um método adaptado para o desenvolvimento de sistemas baseado em linhas de produto.

Apesar da ferramenta *Odyssey* apresentar características inferiores à outras, foi adotada porque os artefatos produzidos pelo método proposto podem ser modelados nesta ferramenta.

3 MÉTODO PROPOSTO PARA DESENVOLVIMENTO DE LPS

Este capítulo descreve um método adaptado para desenvolvimento de linhas de produtos de software. A Seção 3.1 apresenta o processo geral e uma breve descrição sobre as subfases do método. A Seção 3.2 descreve as subfases da *Engenharia de Domínio*. A Seção 3.3 relata as subfases da *Engenharia de Aplicação*. A Seção 3.4 descreve detalhadamente os artefatos de entrada e saída de cada fase e subfase.

3.1 PROCESSO GERAL

A Figura 10 ilustra as fases do método adaptado para linhas de produtos. Este propõe um conjunto de fases adaptadas dos métodos PLUS, FAST e FODA. Esta adaptação foi feita para que o novo método proporcione fases, documentações e diagramas mais específicos.

Figura 10 - Modelo de fases do Método Adaptado
Fonte: Autoria Própria

O método é composto de duas fases principais: *Engenharia de Domínio* e *Engenharia de Aplicação*. A fase de *Engenharia de Domínio* têm como objetivo a produção de artefatos reutilizáveis que modelam o domínio da linha de produto e por isto é realizada de forma iterativa. É composta de algumas subfases:

- Análise de domínio: Esta subfase tem como objetivo analisar e definir o escopo do software a ser desenvolvido.
- Identificação de Características: Têm como finalidade identificar as características comuns e variáveis do domínio em questão. Esta subfase possui outras etapas listadas a seguir:
    - Requisitos do domínio.
    - Modelagem do domínio.
    - Projeto do domínio.
    - Implementação do domínio.
    - Testes do domínio.

A segunda fase, *Engenharia de Aplicação*, têm como finalidade o desenvolvimento de uma aplicação individual baseando-se no que já foi definido na *Engenharia de Domínio*. Suas subfases são as seguintes:

- Requisitos da aplicação.
- Implementação da Aplicação.
- Testes da Aplicação.
- Entrega e Suporte da Aplicação.

No método proposto, a fase de Gerenciamento, que faz parte da Linha de Produto de Software, faz-se por meio da implementação, testes e suporte da aplicação.

A seguir são descritas minuciosamente todas as subfases e etapas contidas no método adaptado.

3.2 ENGENHARIA DE DOMÍNIO

Detalham-se as subfases da fase de *Engenharia de Domínio*.

3.2.1 Análise de Domínio

Esta subfase deriva-se do método FAST, embora os outros métodos também a contenham com outro nome. Escolheu-se utilizar a ideia dada pela FAST por ser a mais completa.

Na análise de domínio o objetivo é analisar e definir o escopo do software que será explorado para obtenção de seus artefatos. Ao final, cria-se o modelo de contexto o qual define o escopo da modelagem do domínio.

3.2.2 Requisitos do Domínio

Esta subfase provém do método FODA (Análise de Características) e também do método PLUS. Apenas nestes dois métodos existe esta fase, e por ser muito importante para extração de requisitos optou-se por utilizá-la no método adaptado.

Ambos os métodos utilizam esta fase que tem por objetivo identificar os requisitos contidos no sistema e observar os pontos comuns e variáveis de LPS.O artefato criado é o modelo de características onde estão explicitadas as características comuns contidas no domínio.

3.2.3 Modelagem do Domínio

Esta provém da teoria contida sobre as Linhas de Produtos de Software e também do método PLUS. Tem como objetivo fazer a decomposição do problema em questão e a modelagem estática do software que está sendo modelado.

A modelagem estática define o relacionamento estrutural entre as classes do domínio ou o comportamento do sistema. Desta forma, pode-se utilizar como artefatos diagramas de classe ou os diagramas de caso de uso. Ambos diagramas são usados para identificar as características contidas no sistema.

3.2.4 Projeto do Domínio

Assim como a fase de modelagem, esta subfase também provém da teoria de LPS e do método PLUS e foi incluído no método adaptado por ser uma fase que permite a escolha da arquitetura em todos os produtos. Ao final, cria-se um modelo do projeto do software baseado em componentes.

3.2.5 Implementação do Domínio

Esta subfase foi adaptada do método PLUS e da teoria de LPS. Escolheu-se permanecer com esta subfase, pois é nela que se faz a implementação dos componentes reutilizáveis e também se define a linguagem de programação que será utilizada no software. Neste trabalho esta subfase não foi desenvolvida porque o foco é a modelagem do sistema.

3.2.6 Testes do Domínio

Nesta subfase são realizados os testes nos componentes de LPS, validando e verificando os componentes reutilizados, dando ênfase na integridade e funcionalidade.

Esta subfase foi adaptada do método PLUS e da teoria sobre Linhas de Produto de software, pois permite realizar a validação de componentes. Esta subfase também não foi executada neste trabalho.

3.3 ENGENHARIA DE APLICAÇÃO

Detalham-se as subfases da fase de *Engenharia de Aplicação*.

3.3.1 REQUISISTOS DA APLICAÇÃO

Esta subfase foi adaptada do método PLUS e FAST e também da teoria LPS. Escolheu-se permanecer com esta subfase porque seu objetivo é identificar e

modelar os requisitos da aplicação em diagramas de caso de uso. Também deve ser criado o diagrama de características das variabilidades contidas no domínio.

3.3.2 IMPLEMENTAÇÃO DA APLICAÇÃO

Esta subfase provém do método FAST e também da teoria de LPS, mais especificadamente da engenharia de aplicação, pois permite que seja realizada a implementação do produto conforme a extração de requisitos obtida em subfases anteriores. Esta subfase também não foi desenvolvida durante a aplicação deste trabalho, pois o foco concentrou-se na modelagem.

3.3.3 TESTES DA APLICAÇÃO

Esta subfase foi adaptada somente da teoria aplicada em linhas de produtos de software, retirada da engenharia da aplicação, e tem como objetivo a validação do produto. Para isto são realizados testes funcionais e de integração e a análise da integridade dos requisitos. Está subfase também não foi usada no estudo de caso descrito no capítulo posterior.

3.3.4 ENTREGA E SUPORTE DA APLICAÇÃO

Esta subfase originou-se do método FAST e é considerada de grande importância para a satisfação do cliente. Nesta é feita a entrega da aplicação ao cliente ou posteriormente para o suporte, caso seja necessário. Está subfase também não foi desenvolvida no estudo de caso.

3.4 ARTEFATOS PRODUZIDOS EM CADA FASE DO MÉTODO PROPOSTO

O Quadro 5 ilustra os artefatos produzidos em cada subfase, bem com suas respectivas entradas e saídas.

Quadro 5 - Artefatos de entrada e saída produzidos no Método Adaptado

Fases	Subfases		Artefatos de Entrada	Artefatos de Saída
Engenharia de Domínio	Análise de domínio		Definição do Domínio a ser modelado	Modelo de contexto definindo o escopo do domínio
	Identificação de Características	Requisitos do domínio	Dois exemplos de aplicações no domínio, no mínimo.	-Descrição narrativa de cada exemplo. Caso não se tenha a descrição narrativa, devem-se utilizar os aplicativos disponíveis. Neste caso, a análise será realizar por meio da execução do software. Identificação dos pontos comuns entre os estudos de caso. - Requisitos identificados - Diagrama de características (contendo os pontos de comuns)
		Modelagem do domínio	Descrição narrativa de cada exemplo. Pontos de comuns	- Diagrama de Caso de Uso - Cenários dos casos de uso - Diagrama de classe
		Projeto do domínio	Diagrama de Caso de Uso	- Arquitetura da parte Similar (baseada em componentes) - Diagramas de classe para a concepção da arquitetura - Especificação das Interfaces do Sistema. - Identificação das Interfaces de Negócio. Identificação dos Componentes.
		Implementação do domínio	Arquitetura da parte Similar	- Codificação dos componentes da arquitetura similar
		Testes do domínio	Codificação	Validação dos componentes
Engenharia de Aplicação	Requisitos da aplicação		Requisitos da aplicação (pontos de variabilidade) oriundos da fase Requisitos do domínio	- Lista de requisitos das variabilidades -Diagrama de caso de uso da aplicação - Diagrama de classe
	Implementação da Aplicação.		Diagrama de classe	- Codificação dos pontos variáveis
	Testes da Aplicação		Codificação da aplicação	- Plano de testes funcionais e de integração - Produto validado
	Entrega e Suporte da Aplicação		Aplicação validada	- Aplicação

Fonte: Autoria Própria

A seguir se descreve a aplicação do método em um Sistema Organizador. Salienta-se que foi realizado somente as subfases referentes a geração de modelos e as de implementação e testes não foram usadas, pois não é foco deste trabalho.

4 APLICAÇÃO DO MÉTODO PROPOSTO: MODELAGEM DE UM SISTEMA ORGANIZADOR

Este capítulo descreve o sistema utilizado para o estudo de caso, bem como a aplicação do método proposto para gerar os artefatos de saída. Os diagramas foram criados na ferramenta *Odyssey*. A Seção 4.1 descreve a *Análise de Domínio*. A Seção 4.2 apresenta a *Identificação das Características* composta pelas seguintes etapas: Requisitos do domínio (Seção 4.2.1), Modelagem do Domínio (seção 4.2.2), Projeto do Domínio (Seção 4.2.3). A seção 4.3 relata a modelagem da aplicação, onde são descritos seus requisitos.

4.1 ANÁLISE DE DOMÍNIO

O sistema utilizado como domínio foi um Software de Organização, utilizando como base o programa *Daisho* (DAISHO BLACKSMITH, 2012), o qual possui diversas funcionalidades tais como: Agenda de Compromissos, Sistema de Cronograma, Gerenciador de Contatos e um Gerenciador de Objetivos. Este programa foi escolhido, pois supre as necessidades para a realização da modelagem de domínio.

Para estudo de caso desenvolvido neste trabalho foram escolhidos os Sistemas de *Cronograma* e a *Agenda de Compromissos*, os quais possuem rotinas similares, o que possibilita a criação de *core assets*. A Figura 11 ilustra o diagrama de contexto para o sistema de Organização usando como referência o *Daisho*.

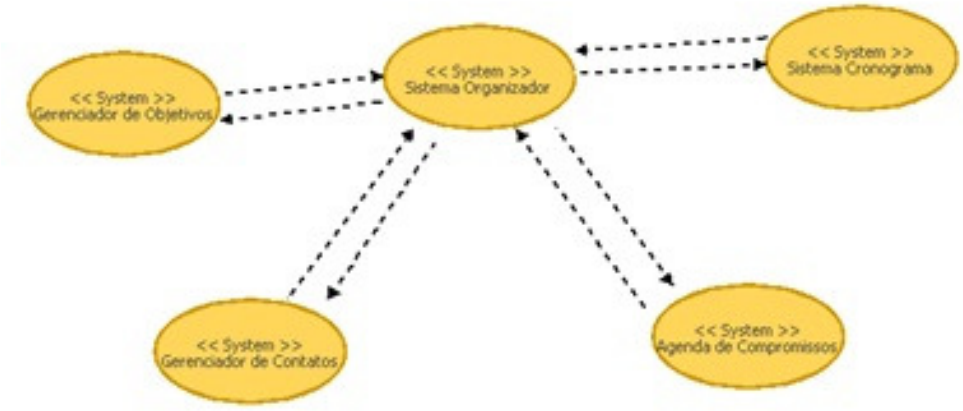

Figura 11 - Modelo de Contexto
Fonte: Autoria própria.

Esta figura ilustra os sistemas que compõem um Organizador bem como seus relacionamentos.

4.2 IDENTIFICAÇÃO DE CARACTERISTICAS

Apresenta-se a aplicação e os artefatos gerados por esta subfase em suas respectivas etapas.

4.2.1 Requisitos de Domínio

Descrevem-se a seguir os dois sistemas presentes em um Organizador e que foi base para o desenvolvimento do estudo de caso, a saber, *Cronograma* e *Agenda*.

Sistema de Cronograma

O cronograma utilizado como exemplo de domínio neste trabalho foi baseado no programa *OpenProj* (OPENPROJ - PROJECT MANAGEMENT, 2012). Este software foi escolhido, pois possui as características necessárias para um cronograma e é gratuito.

Um cronograma é um programa que ajuda ao usuário a administrar um determinado projeto com diversas tarefas. A partir da análise do funcionamento do

software *OpenProj*, foi possível constatar que um cronograma possui as seguintes *feature*s:

- Realiza o cadastro de um projeto, onde são cadastradas as tarefas, que possui: nome, gerente, data de inicio, planejamento adiantado e notas, como mostra a Figura 12.

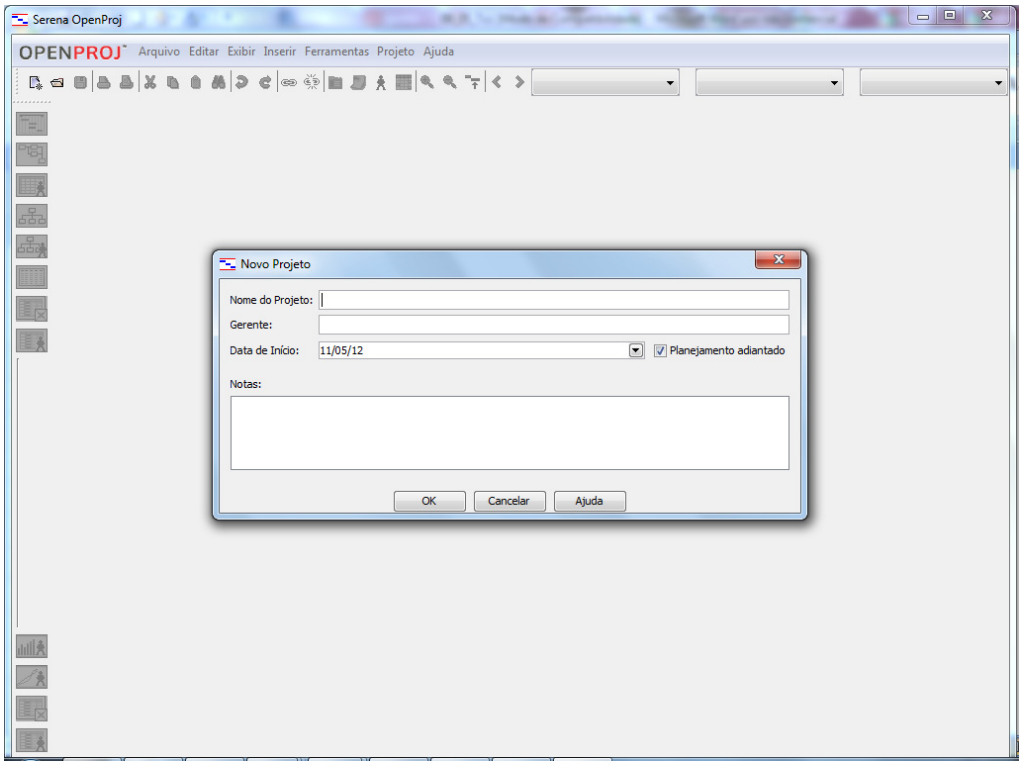

Figura 12 – OpenProj – Cadastro de Projeto
Fonte: OpenProj (2012)

- Faz o cadastro de uma tarefa, que possui: nome, duração, início, término, predecessores e recurso, representado na Figura 13.

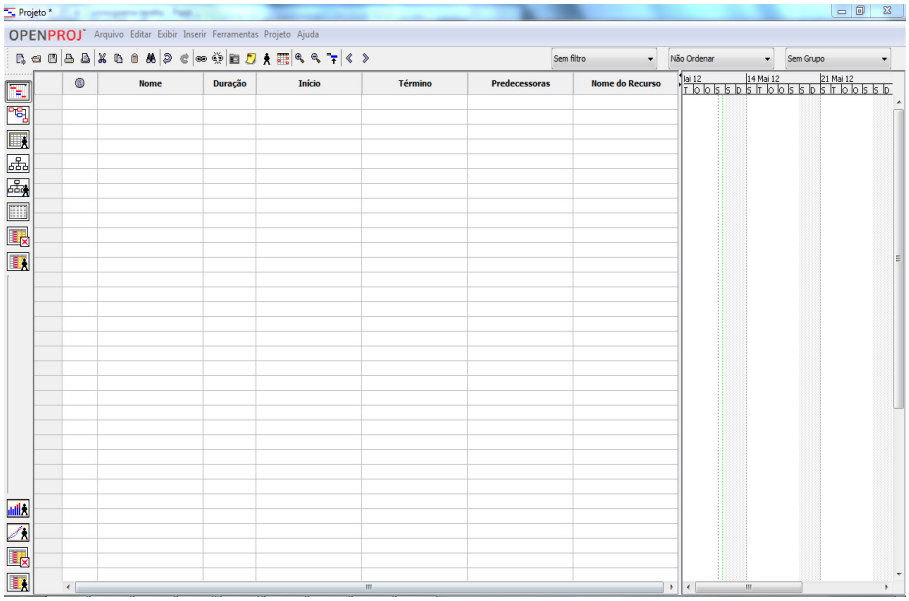

Figura 13 - OpenProj – Tarefas
Fonte: OpenProj (2012)

- Quando uma tarefa possui dependência de outra, não pode iniciar antes que sua predecessora seja finalizada, sendo que a data de início apenas aceita após a sua predecessora estar concluída, representado na Figura 14.

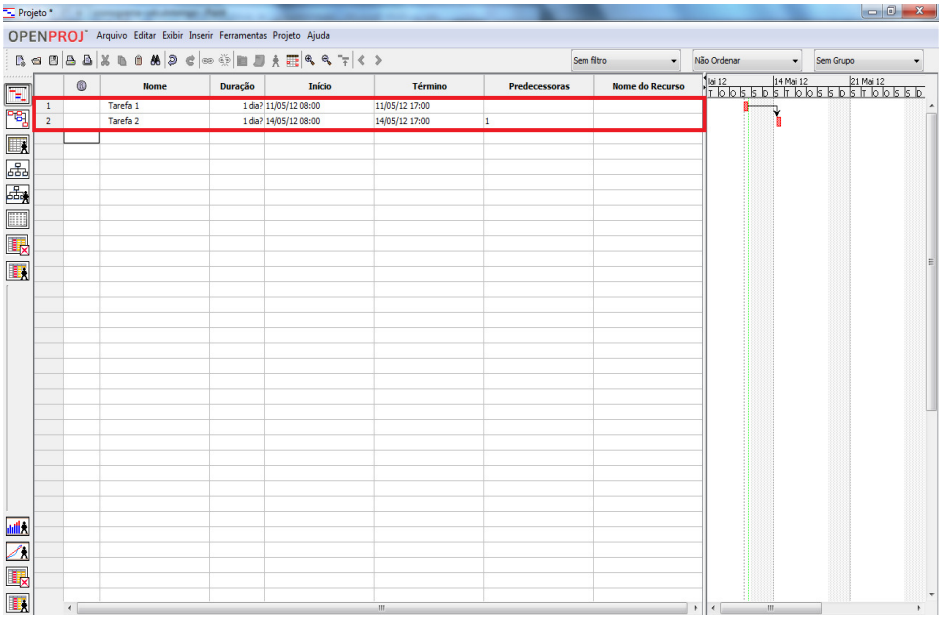

Figura 14 - OpenProj – Predecessora
Fonte: OpenProj (2012)

- Calcula em quanto tempo o projeto será finalizado, de acordo com o tempo dedicado a cada tarefa, a partir de seu término, ilustrado na Figura 15.

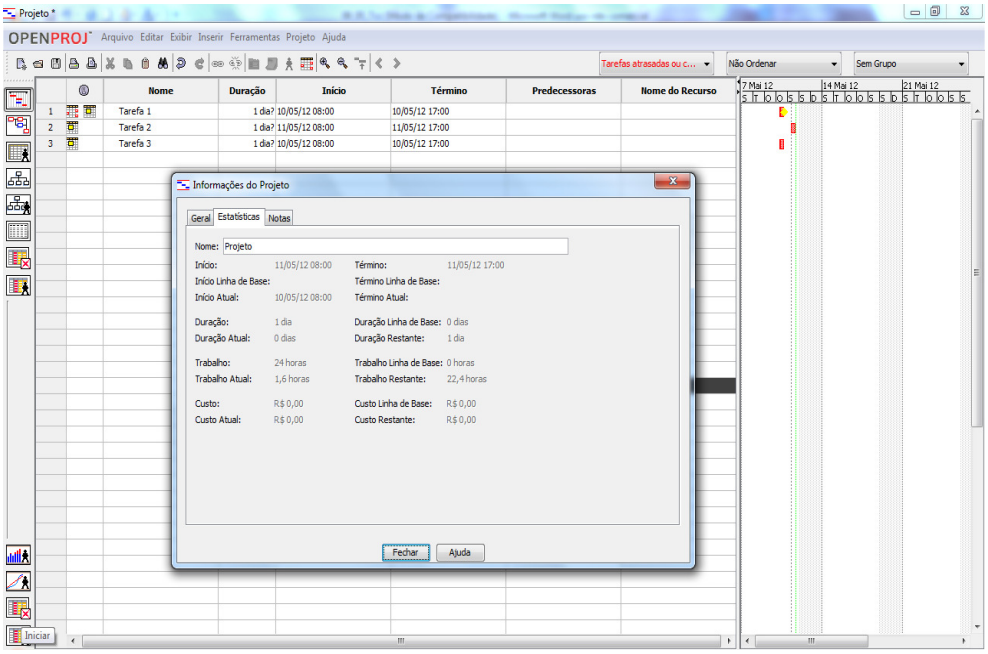

Figura 15 - OpenProj - Calculo do Tempo
Fonte: OpenProj (2012)

- É possível colocar dias em que não será realizado nenhum trabalho, como finais de semana e feriados, representado na Figura 16.

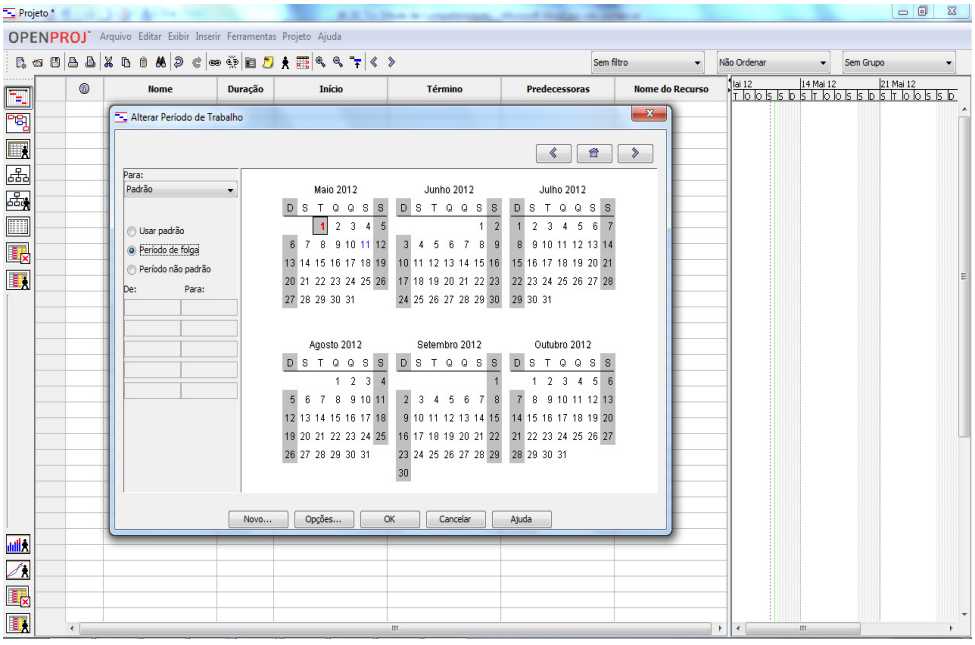

Figura 16 - OpenProj - Dias de Folga
Fonte: OpenProj (2012)

- Quando uma tarefa estiver atrasada, é possível controlar seu atraso, ilustrado na Figura 17.

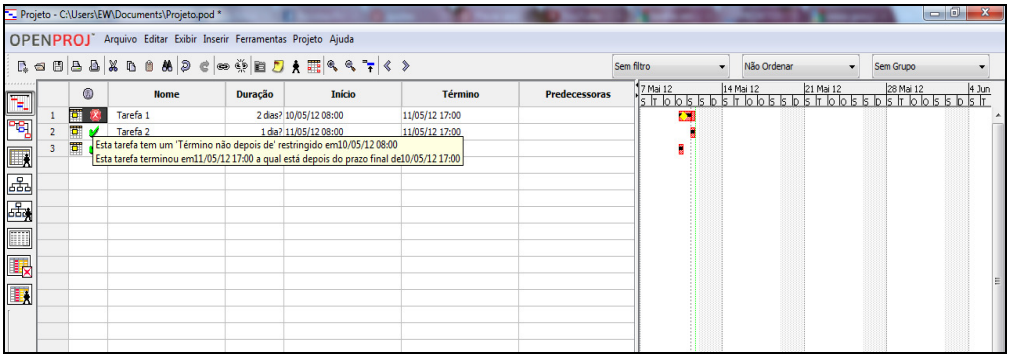

Figura 17 - Tarefas Atrasadas
Fonte: OpenProj (2012)

Sistema de Agenda

A agenda usada no estudo de caso foi baseada na *Agenda Virtual do Google,* ilustrado na Figura 18, (GOOGLE AGENDA, 2012). Uma agenda é utilizada para programar eventos, e organizar suas atividades. Nela é possível realizar as seguintes funcionalidades:

- Criar eventos contendo: representado na Figura 18, possui título, hora, local, usuário, descrição, cor, *status* e privacidade.

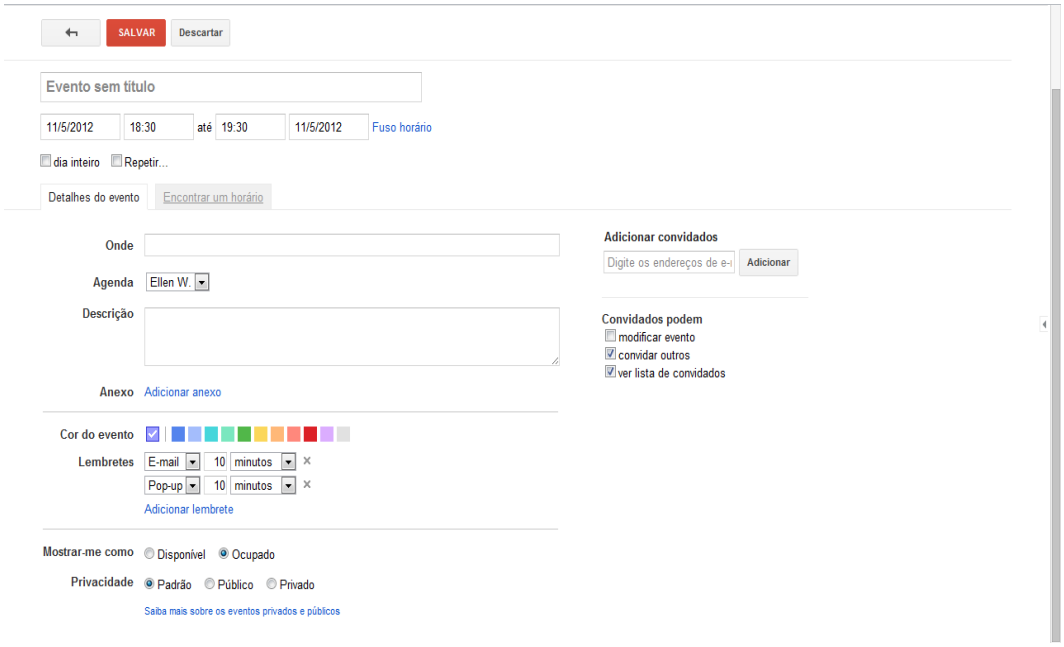

Figura 18 - Google Agenda - Cadastro de Evento
Fonte: Google Agenda (2012)

- Quando um evento ocorre mais de uma vez, o usuário tem a opção de repeti-lo acionando um botão onde deverá informar em quanto tempo este será repetido, representado na Figura 19.

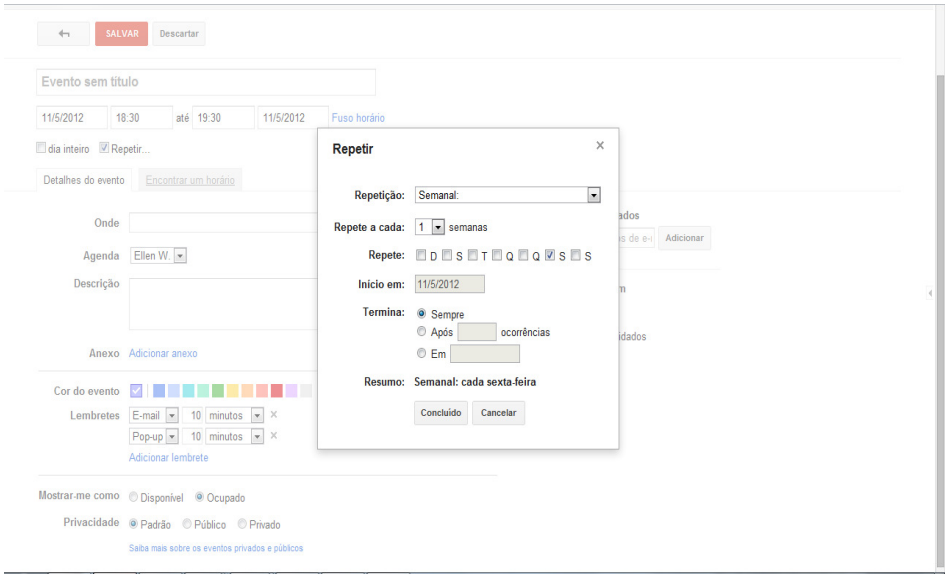

Figura 19 - Google Agenda - Repetir Evento
Fonte: Google Agenda (2012)

- Permitir ao usuário receber lembretes sobre seus compromissos. O usuário poderá escolher quanto tempo antes do compromisso ele quer receber o aviso, ilustrado na Figura 20.

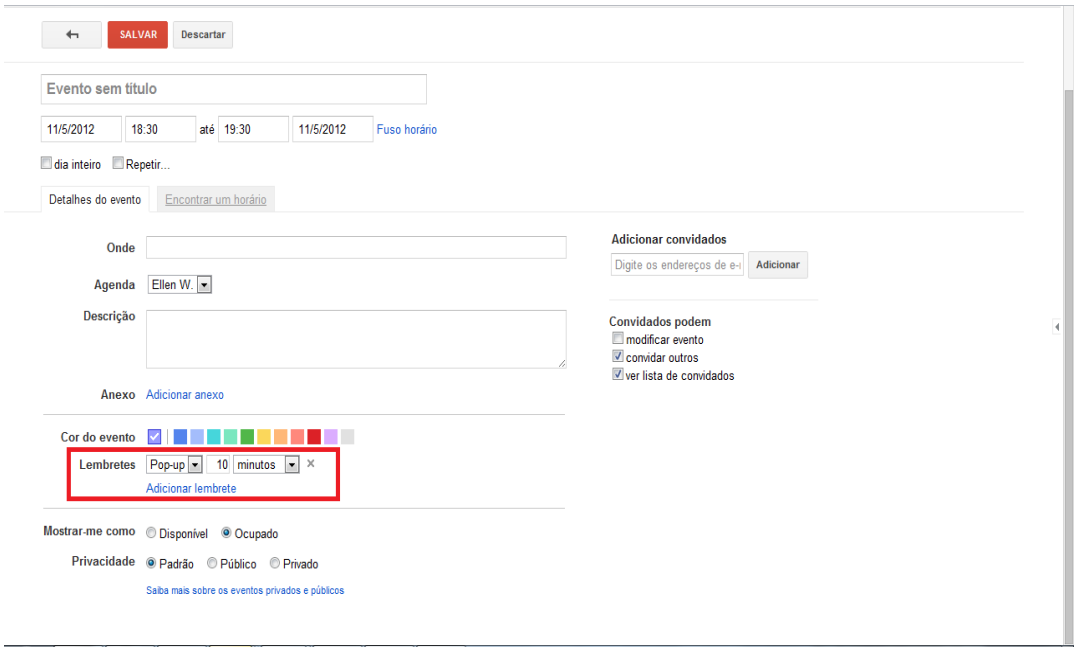

Figura 20 - Google Agenda - Adicionar Lembrete
Fonte: Google Agenda (2012)

- O usuário tem a opção de visualizar os compromissos a partir de uma data que deseja, representado na Figura 21.

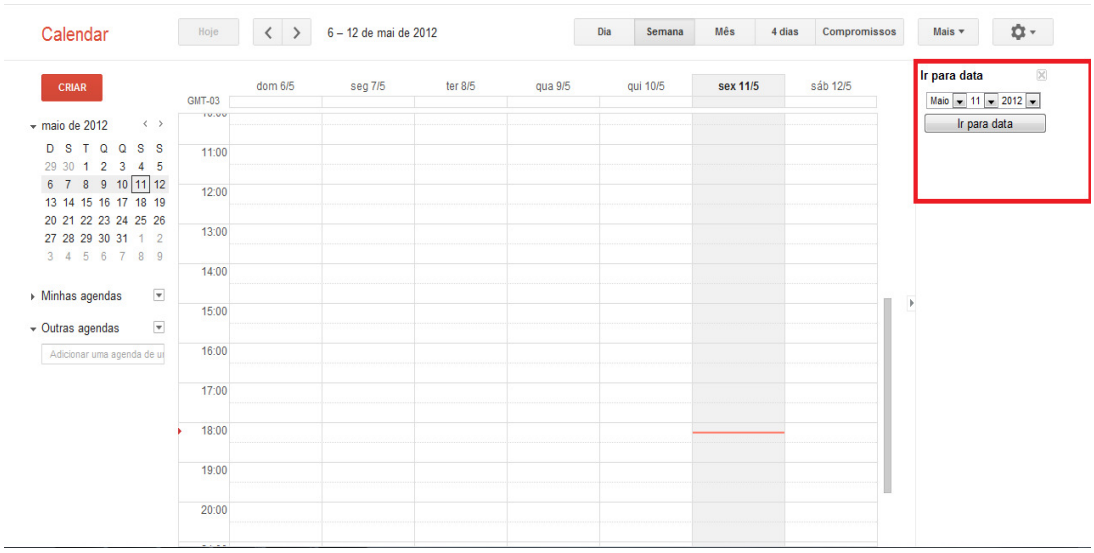

Figura 21 - Google Agenda - Compromissos por Data
Fonte: Google Agenda (2012)

- A agenda possibilita ao usuário ver os horários que possuem compromissos, representado na Figura 22.

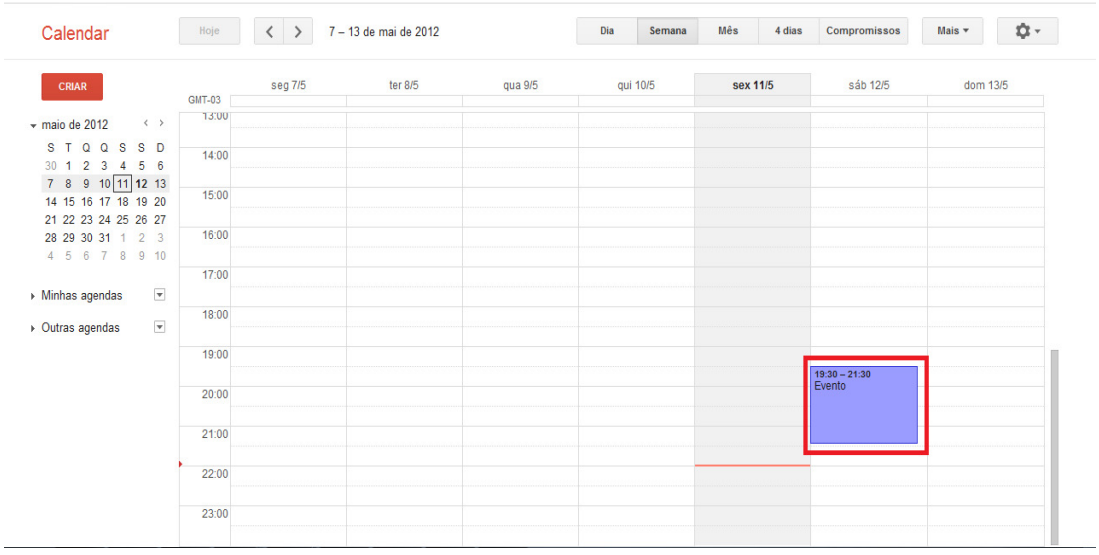

Figura 22 - Google Agenda - Tempo Ocupado
Fonte: Google Agenda (2012)

- O usuário pode visualizar de maneira resumida seus próximos, representado na Figura 23.

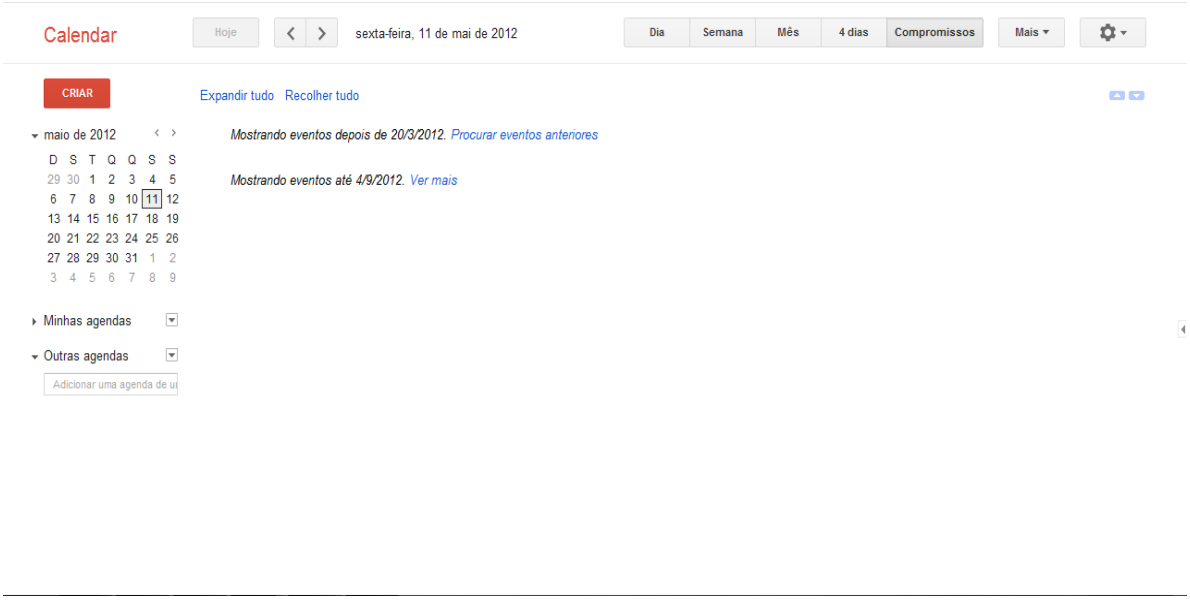

Figura 23 - Google Agenda - Próximos Compromissos
Fonte: Google Agenda

Após realizar a descrição narrativa dos sistemas, identifica-se para cada um deles suas funcionalidades e se estas são pontos similares (*S*) ou de variabilidade (*V*). O Quadro 6 apresenta a lista de requisitos para os dois sistemas.

Quadro 6 - Lista de Requisitos

Sistema	Funcionalidades	Classificação
Agenda	Manter Compromisso	*S*
	Controlar Tempo Ocupado	*S*
	Acionar Lembrete	*S*
	Visualizar Compromisso por Data	*V*
	Visualizar Próximos Compromissos	*V*
	Repetir Evento	*V*
Cronograma	Manter Tarefa	*S*
	Controlar Dependência de Tarefas(Tempo Ocupado)	*S*
	Calcular de Tempo Restante da Tarefa	*V*
	Adicionar Dias de Folga	*V*
	Avisar Tarefas Atrasadas(Lembrete)	*S*

Fonte: Autoria Própria

Os Sistemas Cronograma (*OpenProj*) e Agenda (*Google Agenda*) possuem pontos comuns (ou seja, similares) identificados por meio da análise dos aplicativos. Essas similaridades podem ser usadas para a aplicação em linhas de produto e estão listadas a seguir:

- Cadastro – Ambos aplicativos possuem o cadastro em que se identificou alguns atributos similares tais como: nome/titulo, duração, início/hora, entre outros, os quais podem ser adaptados e desta forma construir um núcleo para ambos os aplicativos.
- Tempo Ocupado - Na Agenda é possível ver quando um horário está ocupado. Por sua vez no Cronograma quando uma tarefa possui uma predecessora o sistema cadastra a tarefa automaticamente depois da finalização da predecessora. Assim sendo, pode-se implementar um núcleo que será responsável por comparar se existe algum tempo ocupado de cadastro de tarefa ou compromisso.
- Lembrete - Na Agenda é possível acionar um alarme para ser feita a lembrança de um compromisso, no Cronograma, por sua vez, quando uma tarefa está atrasada isto é avisado ao usuário. Portanto, será desenvolvido um núcleo que verifica a necessidade da utilização da mensagem de informação.

Após a análise dos softwares, nesta subfase se cria o Diagrama de *Feature*s que ilustra as similaridades entre os aplicativos analisados. A Figura 24 ilustra as similaridades obtidas entre os aplicativos. As similaridades são obtidas por meio da análise detalhada de cada requisito.

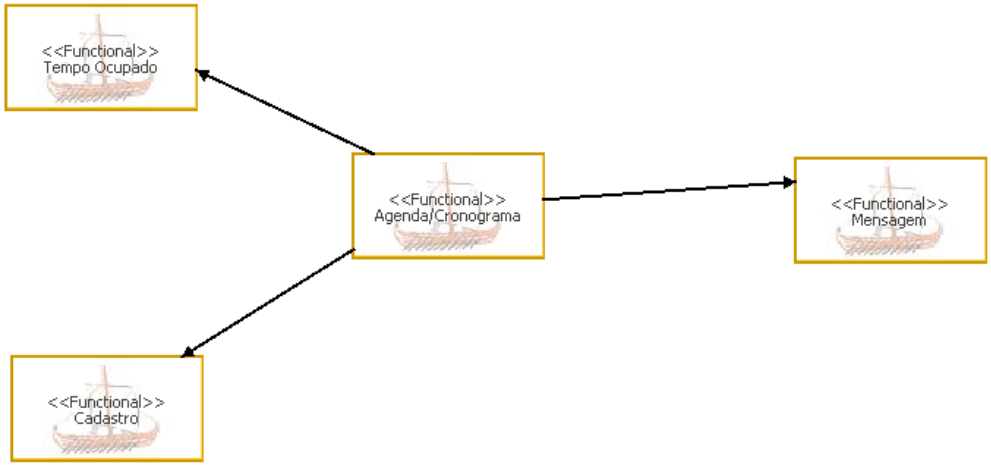

Figura 24 - Diagrama de *feature*s das Similaridades
Fonte: Autoria própria

4.2.2 Modelagem do Domínio

Nesta fase é feito a decomposição do problema através de uma modelagem estática ou de comportamento do sistema, sendo esta representada por diagrama de classes ou de caso de uso, respectivamente. Neste caso, optou-se pela utilização do diagrama de casos de uso, pois este deixa o problema em questão mais claro e de fácil entendimento.

A Figura 25 ilustra o diagrama de caso de uso para as similaridades entre Agenda e Cronograma que foram identificadas. As similaridades foram obtidas da etapa de *Requisitos do Domínio*.

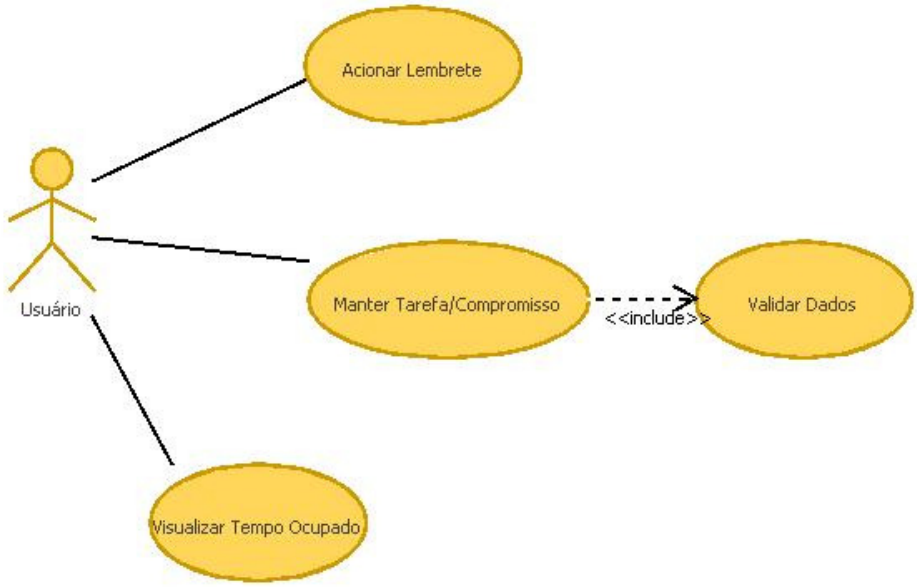

Figura 25 - Diagrama de Caso de Uso das Similaridades Agenda/ Cronograma
Fonte: Autoria Própria

Os cenários para os casos de uso ilustrados na Figura 25 estão descritos nos Quadros 7, 8, 9 e 10.

O Quadro 7 descreve o cenário do Acionar Lembrete que é responsável pelos lembretes que o sistema mostra ao usuário.

Quadro 7 - Cenário para o Caso de Uso "Acionar Lembrete"

Use Case	Acionar Lembrete
Descrição	Permite acionar um Lembrete a Tarefa/Compromisso
Pré-Condições	Tarefa/Compromisso em aberto.
Fluxo Basico	
Ação do Ator	**Resposta do Sistema**
1. Inicia quando a opção acionar Lembrete é escolhida. 3. Informa dados.	2. Sistema permite que os dados sejam informados 4. Valida os dados. **[A1]** 5. Registra dados. **[A2]**
Fluxos Alternativos	
[A1] – Valida os dados informados 4.1 Retorna uma mensagem de erro caso os dados não estejam de acordo com seus valores aceitos e retorna ao Passo 3 **[A2] -** Acesso ao Registro. 5.2 Se ocorrer erro de acesso ao registro retornar uma mensagem de erro e finaliza o caso de uso.	
Pós-Condições	Lembrete acionado.

Fonte: Autoria Própria

O Quadro 8 descreve o cenário do Manter Tarefa/Compromisso que é responsável por incluir, alterar e excluir tarefas e compromissos.

Quadro 8 – Cenário para o Caso de Uso "Manter Tarefa/Compromisso"

Use Case	Manter Tarefa/Compromisso
Descrição	Permite incluir, alterar e excluir uma tarefa ou compromisso
Pré-Condições	Dados da Tarefa/Compromisso
Fluxo Básico	
Ação do Ator	**Resposta do Sistema**
1. Inicia quando a opção cadastrar Tarefa/Compromisso é escolhida 3. Informa os dados 5. Escolhe uma opção.	2. Sistema permite que os dados sejam informados 4. Requisita caso de uso *Validar Dados*.**[A1]** 6. Executa operação escolhida **[A2], [A3]** e **[A4]** 7. Registra dados**[A5]**
Fluxo Alternativo	
[A1] Verificar resultado do validar 4.1 Os dados não foram validados com sucesso, o sistema não executará os passos posteriores. **[A2] -** Incluir 6.1 Os dados serão registrados e uma mensagem de sucesso será exibida ao usuário. **[A3] -** Alterar 6.1 Os dados serão alterados e uma mensagem de sucesso será exibida ao usuário. **[A4] -** Excluir 6.1 Os dados serão excluídos e uma mensagem de sucesso será exibida ao usuário. **[A5] -** Acesso ao Registro. 5.2 Se ocorrer erro de acesso ao registro retornar uma mensagem de erro e finaliza o caso de uso.	
Pós-Condições	Tarefa/Compromisso cadastrado

Fonte: Autoria Própria

O Quadro 9 descreve o cenário do Validar Dados que é responsável pela validação dos dados do sistema.

Quadro 9 - Cenário para o Caso de Uso "Validar Dados"

Use Case	Validar Dados
Descrição	O sistema valida os dados.
Pré-Condições	Dados.
Curso de Eventos Básicos	
Ação do Ator	**Resposta do Sistema**
	1. O sistema valida os dados. 2. O sistema retorna mensagem sobre validação.**[A1]**
Cursos Alternativos	
[A1] Retorno de mensagem 2.1 Os dados foram preenchidos corretamente, o retorno será uma mensagem de sucesso.	
Pós-Condições	Dados validados.

Fonte: Autoria Própria

O Quadro 10 descreve o cenário do Visualizar Tempo Ocupado que é responsável por mostrar ao usuário o horário está ocupado por outra tarefa/compromisso.

Quadro 10 - Cenário para o Caso de Uso "Visualizar Tempo Ocupado"

Use Case	Visualizar Tempo Ocupado
Descrição	Permite a visualização do tempo que está ocupado.
Pré-Condições	Tarefa/Compromisso Cadastrados
Fluxo Básico	
Ação do Ator	**Resposta do Sistema**
1. O usuário entra no sistema.	2. O sistema busca por data tarefa/compromisso.**[A1]** 3. Compara a data da tarefa/compromisso cadastrado com a data atual. **[A2]** 4. Visualiza o compromisso.
Fluxo Alternativo	
[A1] Tarefa/compromisso não registrados. 2.1 O sistema informa que nenhuma tarefa ou compromisso foram registrados e finaliza o caso de uso. **[A2]** Compara data. 3.1 Caso nenhuma data de tarefa/compromisso for igual a data atual o sistema não executará o passo 4.	
Pós-Condições	O sistema mostra o Tempo que está ocupado

Fonte: Autoria Própria

4.2.3 Projeto do Domínio

Nesta fase será definida a arquitetura baseada em componentes do domínio. Para a definição da arquitetura foi escolhido o processo de desenvolvimento de software *UML Components* (SILVA, 2011).

O *UML Components* possui uma arquitetura pré-definida dividida em quatro camadas, porém, duas delas se destacam no processo de desenvolvimento, são elas:

- Camada de Sistema: Nesta camada, os componentes que implementam as funcionalidades específicas do sistema são agrupados.
- Camada de Negócio: Definem os componentes que implementam as funcionalidades estáveis do tipo de negócio. Nesta camada se faz a concepção da arquitetura e pode ser dividida em seis fases: *Especificação de*

requisitos, *Especificação de componentes*, *Provisionamento dos componentes*, *Montagem do sistema*, *Testes* e *Implantação*.

O escopo desta etapa se limita apenas ao nível arquitetural, por esta questão, serão apenas detalhadas as fases de: *Especificação de requisitos* e *Especificação de componentes* para a definição da arquitetura do sistema usado no estudo de caso.

Especificação de Requisitos

Nesta subfase os artefatos gerados são os casos e uso, utilizados para representar os requisitos funcionais do sistema e o modelo conceitual do negócio, representando o domínio do problema. Porém neste trabalho foi produzido somente o caso de uso, pois está definido na etapa de *Modelagem do domínio* (seção 4.2.2). Este diagrama será utilizado na fase de especificação de componentes como artefato de entrada.

Especificação de Componentes

Esta é considerada a fase mais detalhada do processo *UML Components*. Os artefatos de entrada são os produzidos na *Especificação de requisitos*. Como artefato de saída, esta fase produz a especificação das interfaces, especificação dos componentes e a arquitetura do sistema. Esta fase é dividida em três subfases, são elas:

- Identificação dos componentes.
- Interação entre os componentes.
- Especificação final.

Neste trabalho será aplicada apenas a fase de Identificação dos Componentes, pois é nela que é concebida a arquitetura do sistema, sendo esta o foco principal do trabalho.

4.2.3.1Modelagem da Arquitetura

Para a concepção da arquitetura do sistema é necessário a aplicação das fases de *Especificação de requisitos* e *Especificação de componentes*. Esta subseção irá aplicar os conceitos de ambas as fases.

4.2.3.1.1 Identificação dos Componentes

Foram produzidos os seguintes artefatos: Especificação *das interfaces do sistema*, *Identificação das interfaces de negócio* e *Identificação dos componentes*.

Especificação das Interfaces do Sistema

De acordo com o modelo *UML Components*, cada caso de uso se transforma em uma interface da camada do sistema, os métodos das interfaces são identificados a partir da análise do caso de uso em questão. Para essa transformação, será utilizado o caso de uso das Similaridades Agenda/Cronograma (ver Figura 25) .

Inicialmente, cada caso de uso virá uma interface do sistema, como mostra a figura 26.

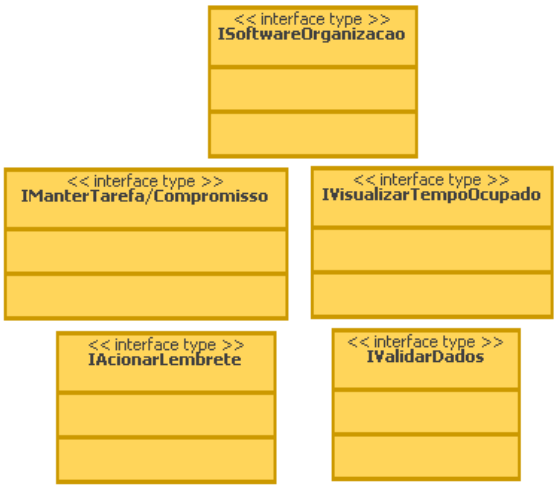

Figura 26 - Interfaces do Software Organização dos casos de uso das similaridades Agenda/Cronograma
Fonte: Autoria Própria

A seguir é feito outro diagrama com as interfaces, já eleitas, do sistema com os respectivos métodos que compõe cada uma, ilustrados na figura 27.

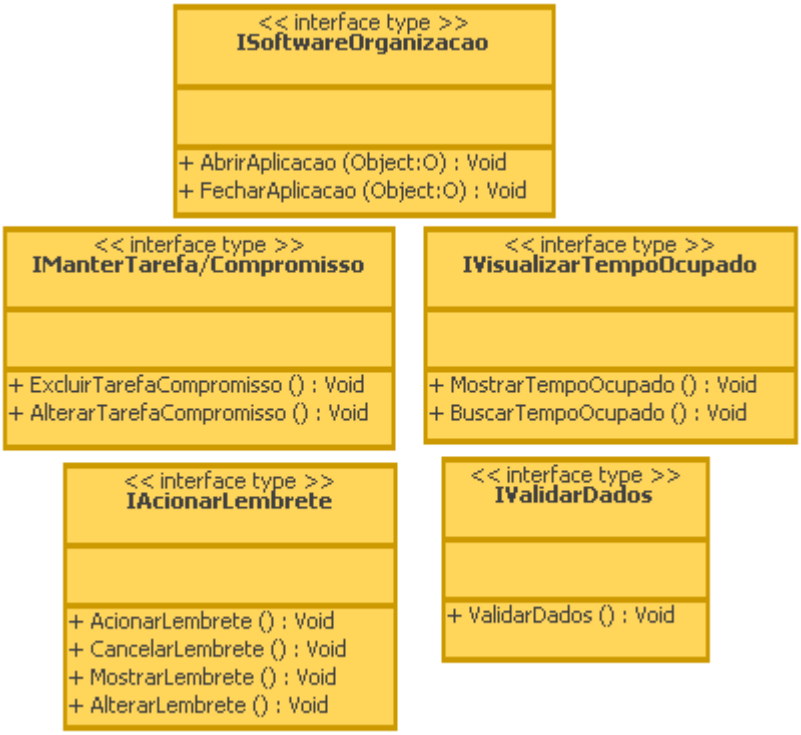

Figura 27 - Interfaces com Métodos do Software Organização dos casos de uso das similaridades Agenda/Cronograma
Fonte: Autoria Própria

Identificação das Interfaces de Negócio

As abstrações das informações que o sistema deve gerenciar são ditas interfaces de negócio. Para identificar as interfaces de negócio devem-se seguir os seguintes passos:

1. Criar o modelo do tipo de negócio: este modelo deve conter informações específicas do negócio e que são guardadas pelo sistema especificado, incluindo detalhes dos atributos que constitui o modelo.
2. Identificar *Core Business Type*: neste são descritas as dependências das informações contidas no sistema.
3. Criar uma interface para os *core types* e incluí-los no modelo.
4. Fazer o refinamento do modelo de tipo de negócio e indicar as responsabilidades das interfaces.

Os dois primeiros passos foram aplicados e criou-se o diagrama representado na Figura 27. Foi desenvolvido o modelo de tipo de negócio com as informações específicas juntamente com os atributos que compõe o domínio,

definindo-se também os *Core Business Type*, conforme indicado no primeiro e segundo passos, respectivamente.

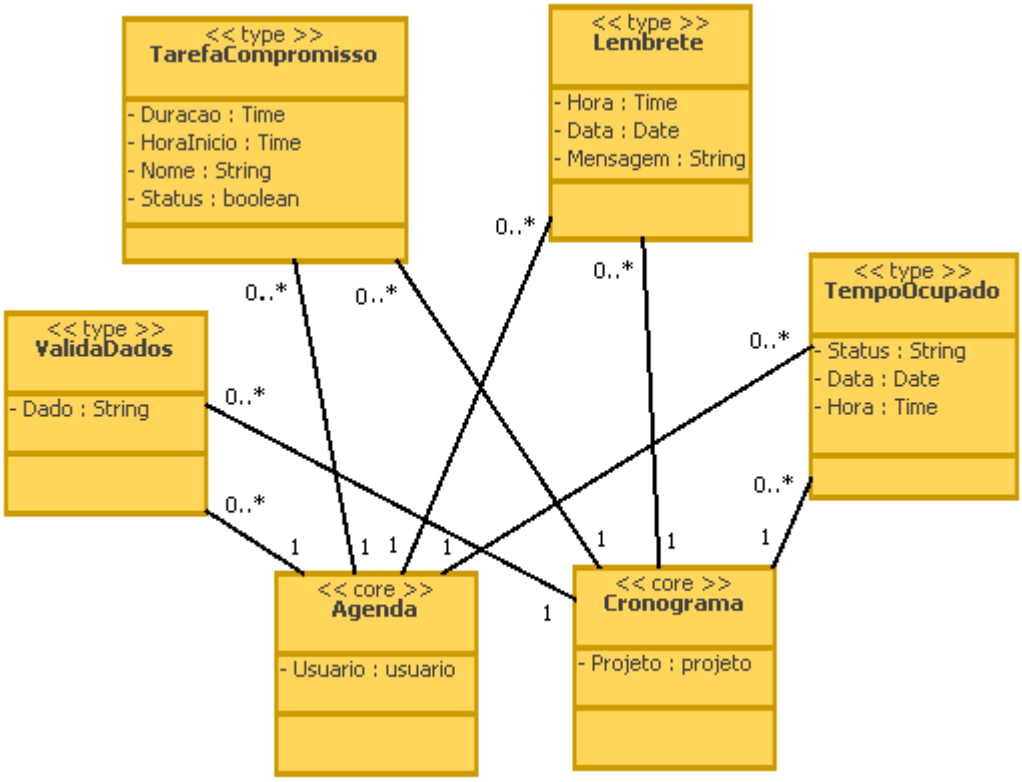

Figura 28 - Modelo de Tipo de Negócio Agenda/Cronograma
Fonte: Autoria Própria

No modelo de negócio, na Figura 28, é definido, os *core types*, neste caso, a *Agenda* e o *Cronograma*. Eles são do tipo *core* (negócio que possui independência de outro negócio), e os outros negócios, que possuem dependência, são do tipo *type*.

Seguindo os passos, deve-se criar uma interface para cada *core type*, adicionar ao modelo e indicar suas respectivas responsabilidades, abaixo estão descritas as interfaces e sua responsabilidades:

- IAGENDA: Na agenda estão todos os compromissos cadastrados, alterados e excluídos, bem como seu tempo ocupado, e o lembrete, caso o usuário selecione esta opção.
- ICRONOGRAMA: No cronograma está contido todos os cadastros, alterações e exclusões de tarefas, com o tempo que a mesma ocupa, e o lembrete, caso o usuário escolha esta opção.

Conclui-se a fase de especificação das interfaces de negócio com as devidas identificações das interfaces, aplicação das responsabilidades e adicionando-as ao modelo de negócio, ilustrado na Figura 29.

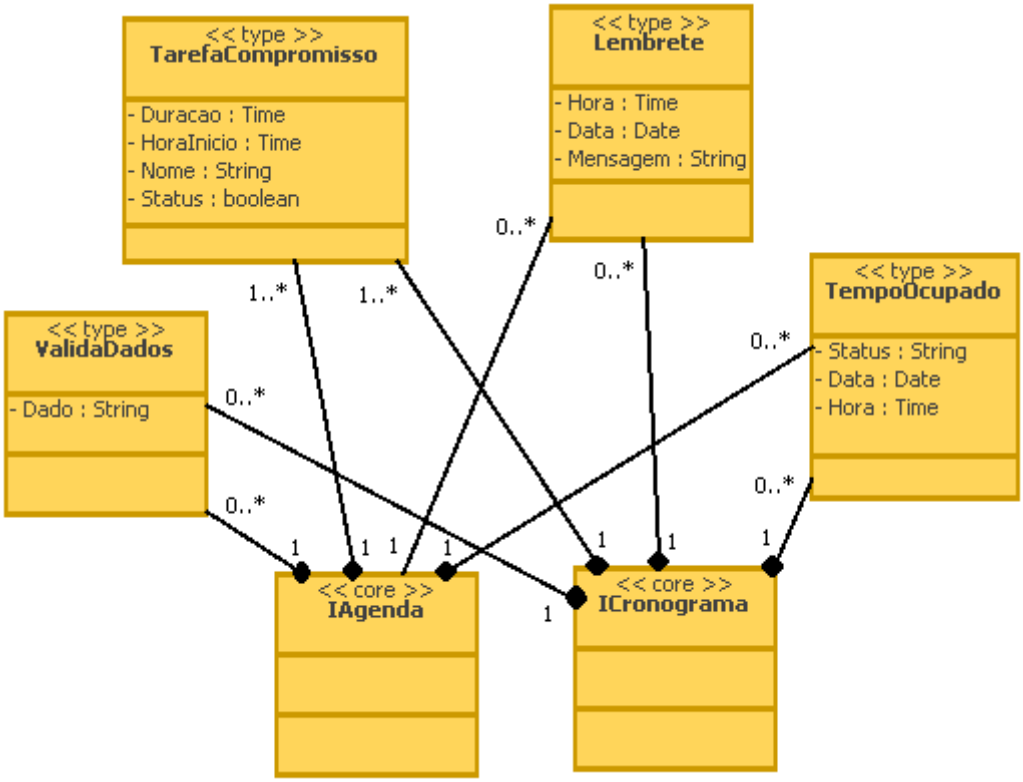

Figura 29 Diagrama de responsabilidades das interfaces do modelo de tipo de negócio
Fonte: Autoria Própria

Identificação dos Componentes

Foram identificados os componentes que implementam as interfaces especificadas nas subfases anteriores. Seguindo o processo *UML Components*, deve-se associar um novo componente à cada interface, tanto de sistema quanto de negócio.

Para demonstrar a criação da camada Sistema, usa-se o caso de uso das Similaridades Agenda/Cronograma (Figura 25). Neste foram identificadas as seguintes interfaces: *ISoftwareOrganização*, *IAcionarLembrete*, *IManterTarefa/Copromisso*, *IVisualizarTempoOcupado*, *IValidarDados.*

Cada interface identificada deve ter um componente associado. No contexto de linha de produto, sabe-se que *ISoftwareOrganização* controla as operações

contidas no sistema, portanto, esta interface deve estar associada a todas as outras interfaces que fazem operações dentro do sistema.

Os mesmos estereótipos relacionados com os casos de uso devem estar incluídos nos componentes que possuem tais funcionalidades, onde todas as interfaces do sistema devem ser identificadas como componentes, assim, uma primeira visão arquitetural pode ser concebida para o Software de Organização, como ilustra a Figura 30.

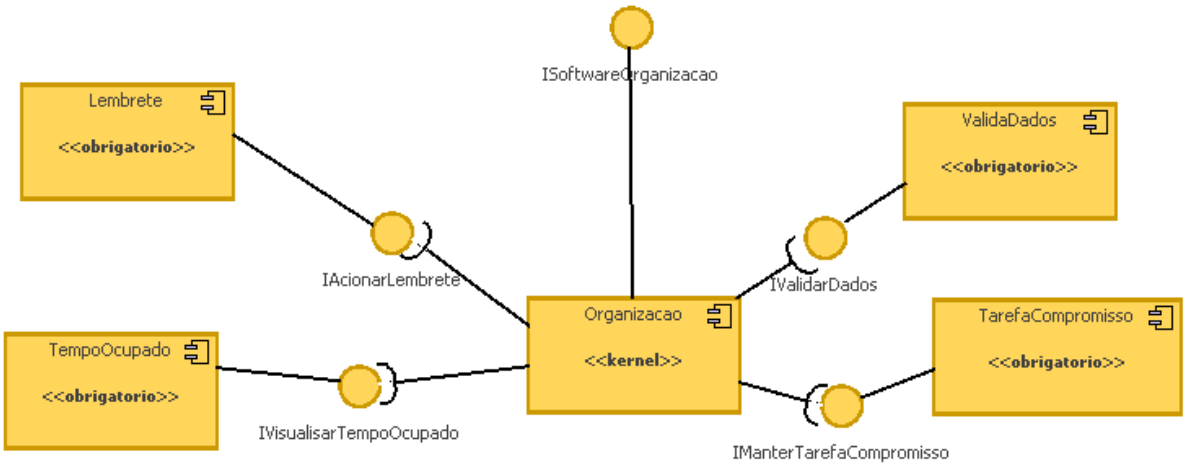

Figura 30 - Configuração arquitetural na camada de sistema
Fonte: Autoria Própria

Depois de criada a camada do sistema, os componentes que implementam as interfaces de negócio (*IAgenda* e *ICronograma*) devem ser associados um componente para cada interface, como ilustra a Figura 31.

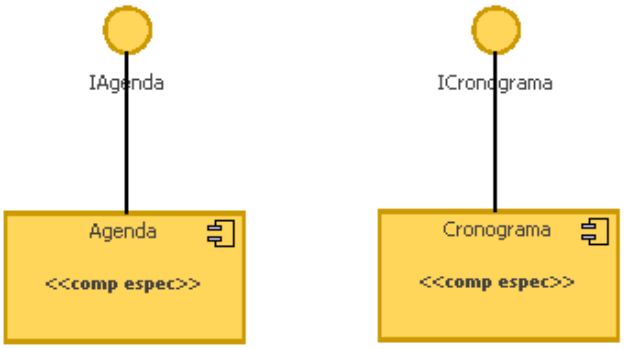

Figura 31 - Componentes das interfaces de negócios
Fonte: Autoria Própria

No processo *UML Components*, a camada de Sistema utiliza operações da camada de Negócio, então, deve haver uma associação entre essas interfaces. Portanto, a Figura 32 representa a arquitetura final dos componentes reutilizáveis do Software de Organização, que é uma junção das camadas de *Sistema* e *Negócio* ilustradas anteriormente pelas Figuras 30 e 31, respectivamente.

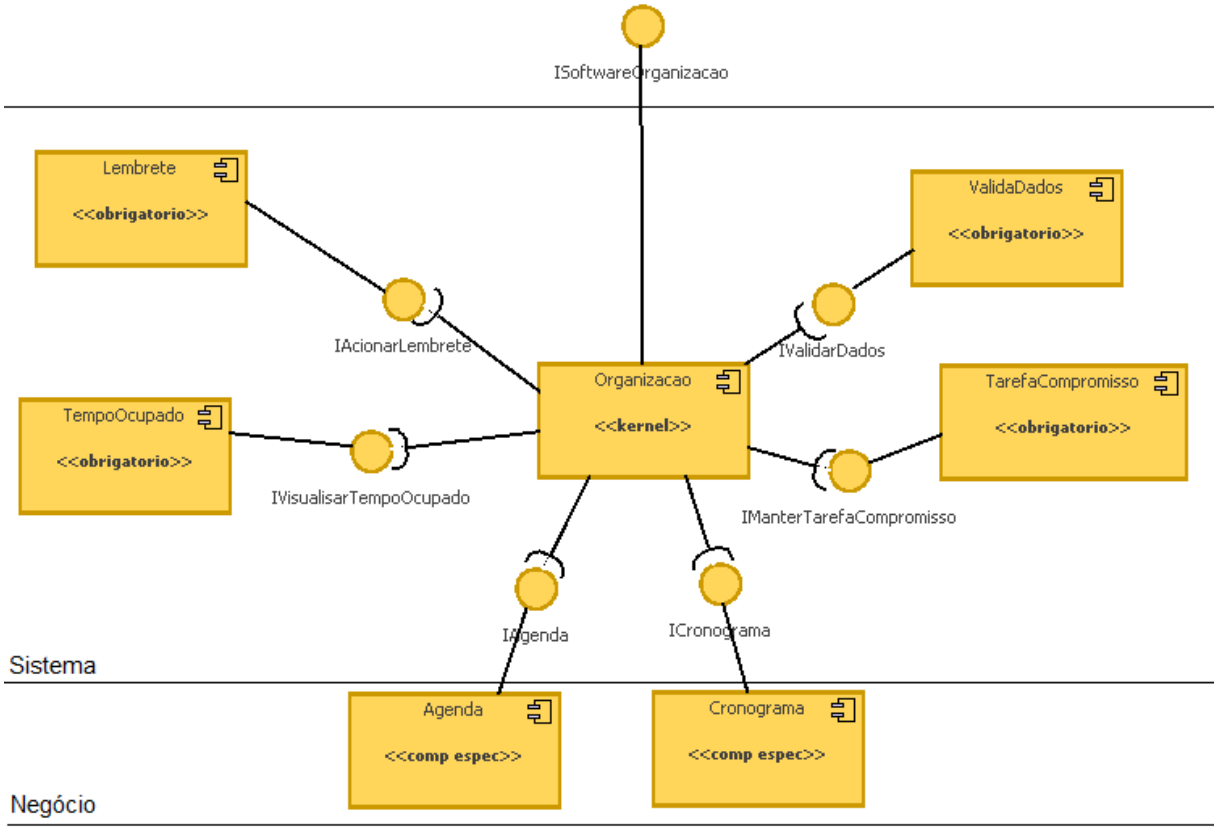

Figura 32 Arquitetura do Software de Organização
Fonte: Autoria Própria

A implementação e testes da *Agenda* e do *Cronograma* se dá por meio da instanciação das interfaces definidas na visão sistema e negócio, nas quais são definidas todas as interfaces comuns entre eles. Neste trabalho a implementação não foi realizada, pois o objetivo é a modelagem do sistema.

4.3 REQUISITOS DA APLICAÇÃO

Os requisitos da aplicação são oriundos do levantamento realizado na etapa de Requisitos de Domínio, os quais foram descritos no Quadro 6.

As variabilidades encontradas no *Cronograma* são:

- Calculo de tempo restante: funcionalidade onde o sistema mostra ao usuário quanto tempo falta para o término da tarefa que esta sendo realizada.
- Dias de folga: onde o usuário pode adicionar dias onde não será realizado trabalho.

Ainda de acordo com as análises realizadas sobre a descrição na seção 4.2 pode-se perceber que as variabilidades da *Agenda* são:

- Visualizar Compromisso: funcionalidade onde o usuário escolhe determinada data e o sistema mostra quais os compromissos que ele possui.
- Compromissos Resumidos: onde o usuário pode visualizar quais são os próximos compromissos que ele possui.
- Repetir Evento: a funcionalidade que permite ao usuário repetir o mesmo evento em diferentes dias.

Após a análise detalhada de cada requisito, podem-se construir os Diagramas de *Feature*s que ilustram as variabilidades dos aplicativos analisados. A Figura 33 ilustra as variabilidades do *Cronograma* e a Figura 34 as variabilidades da *Agenda*.

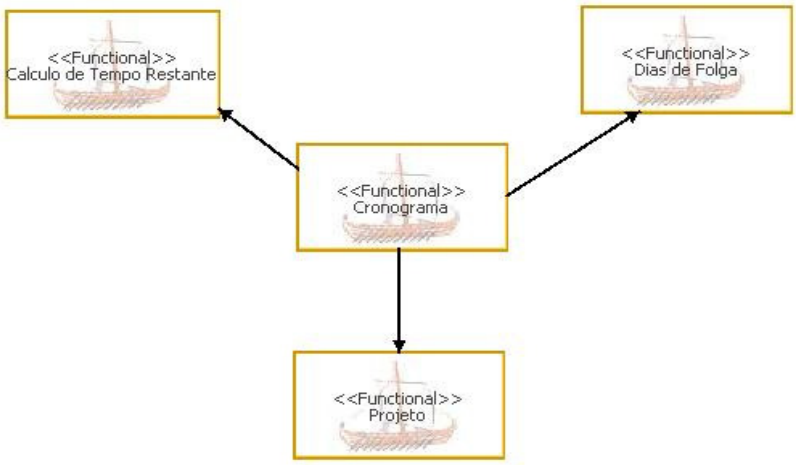

Figura 33 - Variabilidades do Cronograma
Fonte: Autoria própria

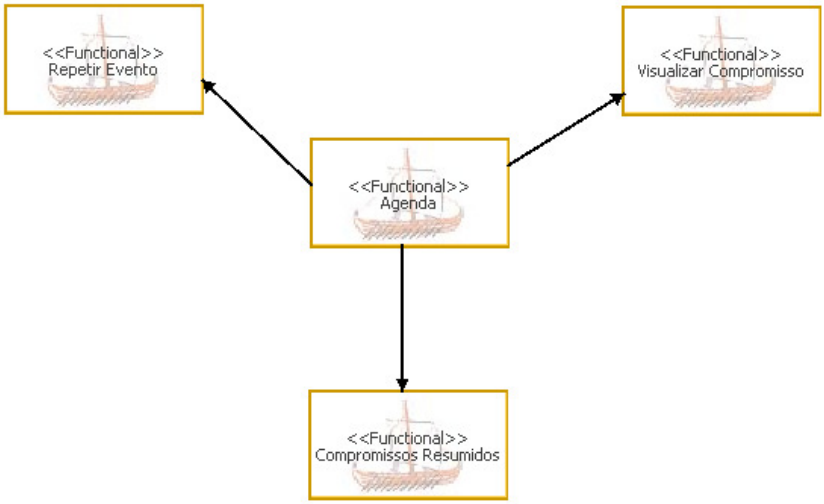

Figura 34 - Variabilidades da Agenda
Fonte: Autoria Própria

Também utilizando-se dos resultados obtidos a partir das duas análises que referenciam as variabilidades das aplicações *Agenda* e *Cronograma*, pode-se então ilustrá-las a partir de casos de uso, sendo representadas as variabilidades do *Cronograma* na Figura 35.

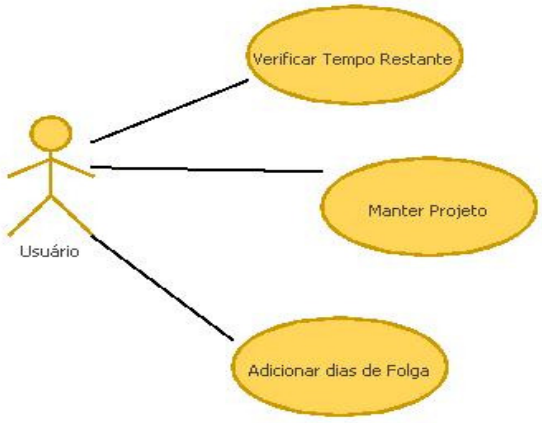

Figura 35 - Caso de Uso das Variabilidades do Cronograma
Fonte: Autoria Própria

Os cenários para os casos de uso ilustrados na Figura 35 estão descritos nos Quadros 11, 12 e 13.

O Quadro 11 descreve o cenário do Verificar Tempo Restante que é responsável por calcular quanto tempo falta para a finalizar a tarefa.

Quadro 11 - Cenário para o Caso de Uso "Verificar Tempo Restante"

Use Case	Verificar Tempo Restante
Descrição	Permite visualizar o tempo restante do Projeto
Pré-Condições	Estar com o projeto em aberto
Fluxo Básico	
Ação do Ator	**Resposta do Sistema**
1 .O use case inicia quando selecionada a opção para Visualizar Tempo Restante	2. Calcular o Tempo Restante de cada Projeto.**[A1]** 3. O sistema mostra o Tempo Restante do Projeto.
Fluxos Alternativos	
[A1] Calcular Tempo. 2.1 Uma mensagem será exibida se o cálculo do tempo for menor ou igual a zero, pois neste caso nenhuma tarefa foi adicionada ao projeto e finaliza-se o caso de uso.	
Pós-Condições	Visualização do Tempo Restante.

Fonte: Autoria Própria

O Quadro 12 descreve o cenário do Manter Projeto que é responsável por incluir, alterar e excluir um projeto.

Quadro 12 - Cenário para o Caso de Uso "Manter Projeto"

Use Case	Manter Projeto
Descrição	Permite incluir, alterar e excluir um projeto
Pré-Condições	Dados do Projeto
Fluxo Básico	
Ação do Ator	**Resposta do Sistema**
1. Inicia quando a opção cadastrar Projeto é escolhida 3. Informa os dados 5. Escolhe uma opção.	2. Sistema permite que os dados sejam informados 4. Requisita caso de uso *Validar Dados*.**[A1]** 6. Executa operação escolhida **[A2], [A3]** e **[A4]** 7. Registra dados**[A5]**
Fluxo Alternativo	
[A1] Verificar resultado do validar 4.1 Os dados não foram validados com sucesso, o sistema não executará os passos posteriores. **[A2] -** Incluir 6.1 Os dados serão registrados e uma mensagem de sucesso será exibida ao usuário. **[A3] -** Alterar 6.1 Os dados serão alterados e uma mensagem de sucesso será exibida ao usuário. **[A4] -** Excluir 6.1 Os dados serão excluídos e uma mensagem de sucesso será exibida ao usuário. **[A5] -** Acesso ao Registro. 5.2 Se ocorrer erro de acesso ao registro retornar uma mensagem de erro e finaliza o caso de uso.	
Pós-Condições	Projeto cadastrado

Fonte: Autoria Própria

O Quadro 13 descreve o cenário do Adicionar Dias de Folga que é responsável por incluir dias em que nenhuma tarefa do projeto será executada.

Quadro 13 - Cenário para o Caso de Uso "Adicionar Dias de Folga"

Use Case	Adicionar Dias de Folga
Descrição	Permite ao usuário adicionar dias de folga
Pré-Condições	Projeto em aberto
Curso de Eventos Básicos	
Ação do Ator	**Resposta do Sistema**
1. O caso de uso inicia quando o usuário seleciona a opção Adicionar Dias de Folga. 3. Informa dados.	2. Sistema permite que os dados sejam informados 4. Valida os dados **[A1]** 5. Registra dados**[A2]**
Cursos Alternativos	
[[A1] – Valida os dados informados 4.1 Retorna uma mensagem de erro caso os dados não estejam de acordo com seus valores aceitos e retorna ao Passo 3 **[A2] -** Acesso ao Registro. 5.1 Se ocorrer erro de acesso ao registro retornar uma mensagem de erro e finaliza o caso de uso.	
Pós-Condições	Dias de folga adicionados.

Fonte: Autoria Própria

A variabilidades da *Agenda* estão ilustradas no diagrama de caso de uso da Figura 36.

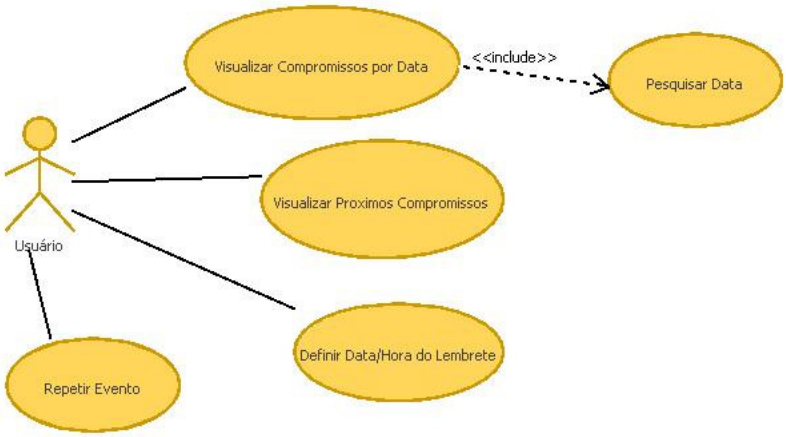

Figura 36 - Caso de Uso das Variabilidades da Agenda
Fonte: Autoria Própria

A descrição dos casos de uso da Figura 36, encontram-se nos Quadros 14, 15,16, 17 e 18.

O Quadro 14 descreve o cenário do Visualizar Compromissos por Data , onde são exibidos os compromissos de uma determinada data.

Quadro 14 - Cenário para o Caso de Uso "Visualizar Compromissos por Data"

Use Case	Visualizar Compromissos por Data
Descrição	Permite visualizar os compromissos de uma determinada data
Pré-Condições	Compromissos cadastrados.
Fluxo Básico	
Ação do Ator	**Resposta do Sistema**
1 .O use case inicia quando se escolhe a opção de Visualizar os Compromissos por Data, informando uma data.	2. Requisita caso de uso *Pesquisar Data* 3. Verifica o resultado retornado pela pesquisa de data. **[A1]** 4. Exibe os Compromissos Cadastrados na Data escolhida.
Fluxos Alternativos	
[A1] – Dados não encontrados 2.1 Retorna uma mensagem de erro se não compromissos cadastrados e nenhum passo posterior será executado.	
Pós-Condições	Visualização dos compromissos por data.

Fonte: Autoria Própria

O Quadro 15 descreve o cenário do Pesquisar Data, onde são buscados os compromissos de uma determinada data.

Quadro 15 - Cenário para o Caso de Uso "Pesquisar Data"

Use Case	Pesquisar Data
Descrição	Permite ao usuário escolher a data a ser pesquisada.
Pré-Condições	Compromisso cadastrado
Curso de Eventos Básicos	
Ação do Ator	**Resposta do Sistema**
	1. O sistema procura os compromissos que estão cadastrados a partir da data informada. 2. Retorna os compromissos daquela data.
Cursos Alternativos	
Pós-Condições	Pesquisa por data concluída.

Fonte: Autoria Própria

O Quadro 16 descreve o cenário do Visualizar Próximos Compromissos, onde são mostrados os próximos compromissos.

Quadro 16 - Cenário para o Caso de Uso "Visualizar Próximos Compromissos"

Use Case	Visualizar Próximos Compromissos
Descrição	Permite visualizar os próximos compromissos cadastrados no sistema.
Pré-Condições	Compromissos cadastrados.
Fluxo Básico	
Ação do Ator	**Resposta do Sistema**
1 .O use case inicia quando selecionada a opção para Visualizar os Proximos Compromissos.	2. O sistema busca os próximos compromissos a partir da data e hora atual. 3. Exibe os próximos Compromissos Cadastrados.**[A1]**
Fluxos Alternativos	
[A1] – Dados não encontrados 3.1 Retorna uma mensagem de erro se não compromissos cadastrados e finaliza o caso de uso.	
Pós-Condições	Visualização dos próximos compromissos.

Fonte: Autoria Própria

O Quadro 17 descreve o cenário do Definir Data/ Hora do Lembrete, onde é adicionada uma hora específica para o Lembrete.

Quadro 17 - Cenário para o Caso de Uso "Definir Data/Hora do Lembrete"

Use Case	Definir Data/Hora do Lembrete
Descrição	Permite ao usuário adicionar uma data e hora ao lembrete.
Pré-Condições	Compromisso em aberto
Curso de Eventos Básicos	
Ação do Ator	**Resposta do Sistema**
2. O usuário informa a data e a hora do Lembrete.	1. O sistema mostra a opção para adicionar data e hora. 3. Requisita o caso de uso *Valida Dados*.**[A1]** 4. Registra os dados informados.**[A2]**
Cursos Alternativos	
[A1] Verificar resultado do validar 3.1 Os dados não foram validados com sucesso, o sistema não executará os passos posteriores. **[A2] -** Acesso ao Registro. 4.1 Se ocorrer erro de acesso ao registro retornar uma mensagem de erro e finaliza o caso de uso.	
Pós-Condições	Data inclusa no Lembrete.

Fonte: Autoria Própria

O Quadro 18 descreve o cenário do Repetir Evento, que permite ao usuário a opção de repetir um determinado compromisso.

Quadro 18 - Cenário para o Caso de Uso "Repetir Evento"

Use Case	Repetir Evento
Descrição	Permite ao usuário cadastrar uma repetição de um evento.
Pré-Condições	Compromisso em aberto
Curso de Eventos Básicos	
Ação do Ator	**Resposta do Sistema**
1. O caso de uso inicia quando o usuário seleciona a opção Repetir Evento. 3. O usuário escolhe os intervalos de tempo.	2. O sistema mostra a tela de Repetir compromisso 4. Requisita o caso de uso *Validar Dados*.**[A1]** 5. Registra os dados do intervalo de repetição.**[A2]**
Cursos Alternativos	
[A1] Verificar resultado do validar 4.1 Os dados não foram validados com sucesso, o sistema não executará os passos posteriores. **[A2] -** Acesso ao Registro. 5.1 Se ocorrer erro de acesso ao registro retornar uma mensagem de erro e finaliza o caso de uso.	
Pós-Condições	Repetir Evento Adicionado.

Fonte: Autoria Própria

Por fim, de acordo com a análise realizada, pode-se modelar o produto como instância do metamodelo, identificando as classes de domínio já modeladas e as de aplicação adicionadas com a realização desta fase. A Figura 37 ilustra o diagrama de classe para a instanciação da aplicação de *Cronograma,* desenvolvido com base nos diagramas da fase de Projeto de Domínio. As classes com os estereótipos *<<Application>>* indicam que elas pertencem a fase de aplicação, isto é, são variabilidades, <<type>> refere-se as classes definidas na fase de domínio, isto é as similaridades e o *<<core>>* é o *Kernel* do sistema. Ressaltando que para cada classe de domínio deve-se utilizar as interfaces definidas na fase de Projeto de Domínio e que foram ilustradas na Figura 32. As classes *Tarefa/Compromisso* e *Lembrete*, além de conterem os atributos de domínio mostrados na Figura 28, nesta fase, foram acrescentados os atributos de variabilidade.

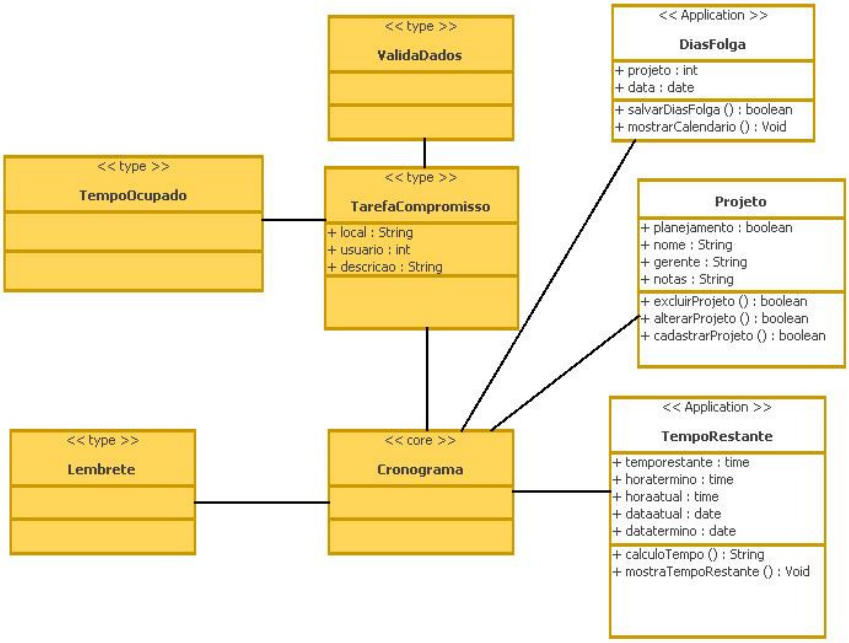

Figura 37 - Modelos de classe para instanciação do Cronograma
Fonte: Autoria Própria

A Figura 38 ilustra o diagrama de classe necessário para a criação da aplicação da *Agenda*. As classes de variabilidades são: IntervaloTempo, Evento, Data, DataHoraLembrete e Compromisso.

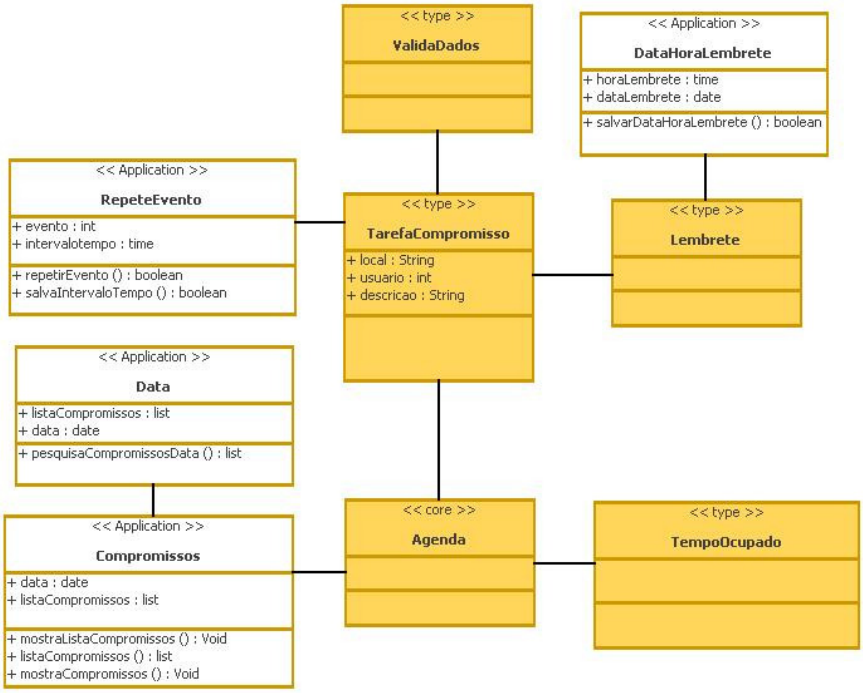

Figura 38 - Modelos de classe para instanciação da Agenda
Fonte: Autoria Própria

A implementação, teste e entrega da aplicação tanto da *Agenda* e do *Cronograma* se dá por meio da instanciação das interfaces definidas na visão sistema e negócio onde são definidas todas as interfaces comuns entre eles, incluídas de suas classes específicas. Isto não foi realizado por este trabalho, pois o objetivo é a modelagem de um sistema usando os conceitos de linha de produto.

5 RESULTADOS E DISCUSSÕES SOBRE A MODELAGEM BASEADA EM LINHA DE PRODUTO

Este capítulo tem como finalidade discutir os pontos relevantes e difíceis que foram identificados durante o estudo de caso. A Seção 5.1 apresenta as dificuldades e limitações na modelagem das aplicações usando LPs. A Seção 5.2 descreve a vantagem em se usar LPs. A Seção 5.3 relata quais os componentes reusáveis que foram alcançados com o estudo de caso. Por fim, a Seção 5.4 apresenta as diferenças entre a criação usando ou não LPS.

5.1 DIFICULDADES E LIMITAÇÕES NA CONSTRUÇÃO DE UMA LINHA DE PRODUTO

Para construir aplicações baseadas em Linhas de Produto, se faz necessário a realização de pesquisas sobre métodos e suas aplicações.

Um das dificuldades encontradas durante o estudo foi localizar materiais completos sobre a modelagem de sistema baseado em LPs. Por isto, se fez necessário utilizar diversos autores para chegar a uma conclusão sobre qual seria um método ideal para modelagem. Além disto, cada método apresentava seus respectivos artefatos de entrada e saída e muitas vezes não os deixa claramente identificados.

Outra dificuldade foi referente a utilização de ferramentas para modelagem baseadas em LPS, as quais não oferecem todo o suporte para realizar o ciclo de desenvolvimento do produto e as mais completas são pagas.

A modelagem baseada em LPS também exige do analista um conhecimento sobre outros assuntos, como por exemplo, durante a criação da arquitetura do estudo de caso foi necessário um estudo mais detalhado sobre *UML Components*. Além deste, o desenvolvedor deve ter conhecimento sobre: padrões de projeto, *framework*, entre outros.

O profissional que está construindo uma LPs deve possuir profundo conhecimento sobre o sistema a ser desenvolvido, pois se faz necessário a criação de núcleos que sejam flexíveis o suficiente para se encaixar em todas as aplicações que o utilizarão.

Além disso, o início da implementação de LPs é mais dispendiosa que a implementação de softwares que não utilizam este método, pois a criação dos núcleos que serão utilizados para uma família de softwares são implementados a partir de métodos diferenciados. Estes métodos tem como objetivo garantir o funcionamento correto da funcionalidade e sanar as necessidades da funcionalidade que se encontra no núcleo em cada um dos softwares da família.

5.2 VANTAGEM DE UTILIZAR LINHAS DE PRODUTO NA AGENDA E NO CRONOGRAMA

Apesar das dificuldades encontradas para criação do modelo, deve-se salientar que LPS é uma tecnologia que pode trazer vantagens aos desenvolvedores de software.

Nos sistemas analisados neste trabalho, sendo estes Cronograma e Agenda, pode-se concluir que com a aplicação de LPS e a criação de núcleos, o trabalho que será despendido para a implementação dos sistemas é menor, o que acarretara a diminuição do tempo de implementação e o corte de custos.

Além disto, as mudanças realizadas no núcleo, que é utilizado por diversos produtos, são repassados à aplicação sem a necessidade de uma nova codificação.

Além de possibilitar um menor trabalho aos desenvolvedores, a documentação resultante das fases realizadas para a aplicação de uma LP é importante, pois demonstra de maneira clara como o projeto foi desenvolvido e a partir deste se pode instanciar quantas aplicações forem necessárias. Lembrando que para este processo de instanciação é necessário que as mesmas apresentem similaridades com o *core assets*.

5.3 REUSO DA LPS PROPOSTA PARA O SISTEMA ORGANIZADOR

Se os softwares analisados por este trabalho forem implementados, as fases que seriam melhores utilizadas são as de descrição e de modelagem, as quais explicam o passo a passo de como os produtos devem ser desenvolvidos. Isto porque durante a modelagem os atributos e métodos já estão definidos.

A utilização dos estereótipos tanto no modelo de *feature*s quanto no modelo de negócio e de instanciação facilitam a identificação de quais classes devem ser efetivamente produzidas e as que pertencem ao núcleo durante a instanciação do produto.

Além do reuso dos modelos que foram desenvolvidos, o método proposto também pode ser utilizado para o desenvolvimento de uma nova aplicação, pois os artefatos de entrada e saída estão claramente definidas no quadro 5.

Um exemplo de aplicação que pode utilizar um dos núcleos citados neste trabalho é o software *Post-it® Digital Notes*, que permite ao usuário que ele adicione pequenos lembretes em seu navegador (POST-IT DIGITAL NOTES, 2012). Neste caso, este produto pode utilizar o núcleo de *Adicionar Lembrete*, onde é adicionado um lembrete ao abrir um navegador. O Diagrama de classes da instanciação deste exemplo está ilustrado na Figura 39.

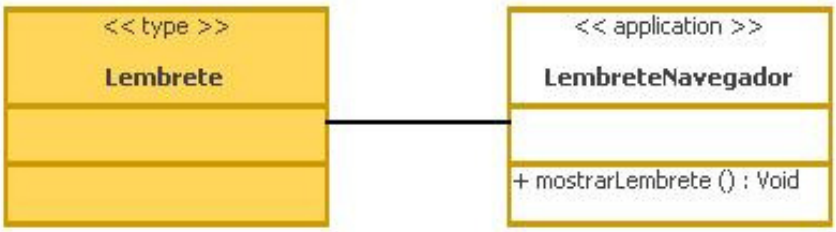

Figura 39 – Modelo de classes para instanciação do Post-it
Fonte: Autoria Própria

5.4 MODELAGEM DE SOFTWARE BASEADO EM LP X OUTROS TIPOS DE MODELAGEM

Com o estudo realizado se pode notar algumas diferenças entre os softwares baseados em LPS e os softwares que não são. Uma das diferenças encontradas é que o início da modelagem de uma LPS é mais complexa visto que se deve analisar no mínimo duas aplicações e delas identificar os pontos comuns e variáveis.

Usando a LPs consegue-se um aumento no reuso porque o núcleo será utilizado por *n* aplicações do domínio.

Por meio de softwares baseados em LPs é possível diminuir a quantidade de código, pois parte deste já está presente no núcleo. Facilita a manutenção, pois as regras de variabilidade ficam em uma camada e o núcleo em outra.

6 CONCLUSÃO

Para realização deste trabalho foi criado um método para a modelagem do sistema e foram identificadas as similaridades e variabilidades na construção da modelagem do Sistema Organizador, especificamente os módulos Agenda e Cronograma. Para se chegar à criação do método foram analisados os métodos FODA, FAST e PLUS, no qual se utilizou suas melhores práticas. Estudou-se as ferramentas de modelagem *Odyssey*, *fmp*, *XFeature* e *pure::variants*, sendo escolhida a *Odyssey*, pois está possuí os diagramas necessários para a criação dos artefatos de entrada e saída proposto pelo método.

Durante a fase de *Engenharia de Domínio*, a qual inicia com a subfase *Análise de Domínio*, construiu-se o modelo de contexto que demonstra o sistema no qual os produtos estarão inseridos. Após esta subfase, é realizada a *Identificação de Características*, que começa com a análise de requisitos do domínio na qual se procura identificar a lista de requisitos contendo as similaridades e variabilidades entre as aplicações, elaborando o diagrama de características com os pontos similares dos sistemas. Em seguida, é feita a *Modelagem de Domínio*, onde são produzidos os diagramas de caso de uso de domínio e seus cenários. Por fim, desenvolve-se a arquitetura do domínio dos produtos estudados, conforme ilustrou este trabalho para o Sistema Organizador.

Na *Engenharia de aplicação* foram analisados os requisitos considerados de variabilidades e a partir deles criou-se os respectivos diagramas de características e casos de usos (com os cenários) e a partir deles foi construído um diagrama de classes como forma de instanciação de cada produto analisado.

Conclui-se a partir dos estudos realizados neste trabalho que a utilização de linhas de produto no desenvolvimento de software traz vantagem para os desenvolvedores, pois durante a produção se criam núcleos reutilizáveis, ou seja, possibilita que estes sejam usados para a criação de novos produtos, que apresentam a mesma funcionalidade.

Além do reuso do núcleo, os artefatos gerados em cada fase, subfrase ou etapa demonstram como a modelagem linha do projeto foi construída. Isto facilita o entendimento por parte do desenvolvedor quando o mesmo precisar instanciar uma nova aplicação a partir da que já foi desenvolvida.

6.1 TRABALHOS FUTUROS

A seguir estão citados trabalhos futuros que podem ser desenvolvidos tendo como base o presente trabalho:

- Refinar os requisitos da *Agenda* e *Cronograma*, o que pode gerar um núcleo mais completo para estes sistemas.
- Implementar e testar o modelo proposto em que se identificou os pontos comuns e de variabilidade dos sistemas *Agenda* e *Cronograma*.
- Modelar os módulos *Gerenciador de objetivos* e *Gerenciador de contatos*.
- Incorporar a modelagem dos sistemas de objetivos e contatos na arquitetura proposta.
- Implementar e testar os módulos Gerenciador de Objetivos e Gerenciador de Contatos.
- Aplicar método adaptado em outros estudos de caso.

REFERÊNCIAS

ARAGÓN, C.R. **Processo de Desenvolvimento de Uma Linha de Produtos Para Sistemas de Gestão de Bibliotecas**. 2004. 93f. Tese (Mestrado em Ciência da Computação) - Curso de Pós- Graduação em Ciência da Computação, Universidade Federal do Mato Grosso do Sul, Campo Grande, 2004.

ARAÚJO, D. O. **Elaboração de Especificações de Casos de Uso para Linhas de Produto de Software Baseada em Fragmentos**. 2010. 132f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2010.

BORBA, C. C. **Uma Abordagem Orientada a Objetivos para as Fases de Requisitos de Linhas de Produto de Software.** 2009. 176 f. Dissertação (Mestrado em Ciencia da Computação) - Centro de Informatica da Universidade Federal de Pernambuco, Universidade Federal de Pernambuco, Recife, 2009.

BRITO, R. de C., COLANZI, T. E. **Avaliação da adequação do uso de aspectos na implementação de variabilidades de linha de produto de software.** 2010. Trabalho de Conclusão de Curso (Especialização em Desenvolvimento de Sistemas para Web). Universidade Estadual de Maringá.

CLEMENTS, P.; NORTHROP, L.. **Software Product Lines: Practices and Patterns.** 3. ed. Boston: Addison-wesley, 2002. 563 p.

COHEN, S.. **Predicting When Product Line Investment Pays**. 2003. The Software Engineering Institute, Carnegie Mellon University, 2003.

DAISHO BLACKSMITH. Disponível em: < http://www.daisho-blacksmith.com/en/software.html/>. Acesso em: 30 jan. 2012.

DURSCKI, R. C. et al. Linhas de Produto de Software: riscos e vantagens de sua implantação. **In:** Simpósio Internacional de Melhoria de Processos de Software, 6.,2004,São Paulo. SIMPROS, 2004. p.155-166.

GENERATIVE SOFTWARE DEVELOPMENT LAB. Disponível em: <http://gp.uwaterloo.ca/fmp>. Acesso em: 20 jan. 2012.

GIMENES, I. M.,TRAVASSOS, G. H. O Enfoque de Linha de Produto para Desenvolvimento de Software **In:** XXI JAI – Livro Texto Ed.Florianópolis: Sociedade Brasileira de Computação, 2002.

GOOGLE AGENDA. Disponível em: < https://www.google.com/calendar/>. Acesso em: 30 jan. 2012.

HARSU, M.; **FAST product-line architecture process**; Tempere University of Technology; Tempere; 1 – 43.2002.

KANG, K. C.; COHEN, S. G.; HESS J. A.; NOVAK W. E.; PETERSON A.S. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Pittsburg, Carnegie Mellon University, 1990, 161 p., **Technical Report.**

LARMAN, C.. **Utilizando UML e Padrões**: Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. Porto Alegre: Bookman, 2008. 696p.

LIMA JUNIOR, R. A. de. **Comparação Entre Ferramentas para Linha de Produtos de Software.** 2008. 46 f. Trabalho de Conclusão de Curso (Graduação) - Bacharelado em Computação, Universidade Federal de São Carlos, São Carlos, 2008.

LOBO, A. E. de C.; RUBIRA, C. M. F. Um Estudo para Implantação de Linha de Produto de Software Baseada em Componentes. Campinas, Universidade Estadual de Campinas, 2009, 30 p., **Relatório Técnico.**

MATINLASSI, M.; Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA; **In:** 26th International Conference on Software Engineering; Finland; 2004, 1-10.

NEIVA, D. F. S. **Uma revisão de engenharia de requisitos para linha de produto de software**. 2008. 166 f. Dissertação (Mestrado em Ciência da Computação) – Faculdade de Ciência da Computação, Universidade Federal de Pernambuco, Recife, 2008.

OLIVEIRA, R. P. de. **UbiComSPL: Desenvolvimento Baseado em MDA, de Linha de Produto de Software no Domínio de Aplicações Ubiquias.** 2009. 78 f. Dissertação (Mestrado em Ciencia da Computação) – Departamento de Engenharia da Computação, Escola Politécnica de Pernambuco, Universidade de Pernambuco, Recife, 2009.

OPENPROJ. Disponível em: <http://sourceforge.net/projects/openproj/>. Acesso em: 20 jan. 2012.

P&P SOFTWARE and ETH ZÜRICH. Disponível em: < http://www.pnp-software.com/XFeature/>. Acesso em: 20 jan. 2012.

POHL, K.; BOCKLE, G.;LINDEN, F. **Software Produtc Line Engineering: Foundations, Principles, and Techniques.** Berlin: Springer – Verlag, 2005.

POST-IT DIGITAL NOTES. Disponível em:<<http://www.post-it.com/wps/portal/3M/en_US/Post_It/Global/Products/Catalog/?PC_7_RJH9U523000P60II85TCFL1863000000_nid=GS5C682K2WgsQBFGNHBXGFgl10GG5C0NK3bl&WT.mc_id=www.3m.com/us/office/postit/products/prod_digital.html>>. Acesso em: 18 mai 2012.

PROJETO YANA. Disponível em: <https://sites.google.com/site/projetoyanaufpb/home>. Acesso em: 20 jan. 2012.

PURE SYSTEMS GMLB. Disponível em: < http://www.pure-systems.com/>. Acesso em: 20 jan. 2012.

SILVA et al. Linhas de Produtos de Software: Uma tendência da indústria. In: ERCEMAPI, 5., 2011, Teresina. **Livro Texto dos Minicursos.**Teresina: Sociedade Brasileira de Computação, 2011. p. 8 - 31.

SILVA, A. P. da. **Uma Linha de Produto de Software baseada na Web Semântica para Sistemas Tutores Inteligentes**. 2011. 185f. Tese (Doutorado em Ciência da Computação) – Curso de Pós- Graduação em Ciência da Computação, Universidade Federal de Campina Grande, Campina Grande, 2011.

SOBERIT. Disponível em: <http://www.soberit.hut.fi/kumbangtools/>. Acesso em: 20 jan. 2012.

TEIXEIRA, L. M. **Ligo: Uma Linhas de Produtos de Software Para Gerenciamento de Igrejas Cristãs**. 2007. 56f. Trabalho de Conclusão de Curso (Graduação) – Escola Politécnica de Pernambuco, Universidade de Pernambuco, Recife, 2007.