PODER EXECUTIVO

MINISTÉRIO DA EDUCAÇÃO

UNIVERSIDADE FEDERAL DO AMAZONAS

INSTITUTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Rafael Normando Cunha

SPLIT: Um Conjunto de Técnicas de Inspeção de Modelos de Linha de Produto de Software

Manaus

2013

PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Rafael Normando Cunha

SPLIT: Um Conjunto de Técnicas de Inspeção em Modelos de Linha de Produto de Software

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Amazonas como requisito parcial para obtenção do grau de Mestre em Informática.

Área de concentração: Engenharia de Software

Profa. D. Sc. Tayana Uchôa Conte
Instituto de Computação – UFAM
Orientadora

Prof. D. Sc. Raimundo da Silva Barreto
Instituto de Computação – UFAM

Prof. D. Sc. Edson A. Oliveira Junior
Departamento de Informática – UEM

Manaus
2013

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Cunha, Rafael Normando
C972s SPLIT: Um Conjunto de Técnicas de Inspeção de Modelos de Linha de Produto de Software / Rafael Normando Cunha. 2013
109 f.: il.; 31 cm.

Orientadora: Tayana Uchôa Conte
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Linha de Produto de Software. 2. Técnica de Inspecão. 3. SPLIT. 4. LPS. I. Conte, Tayana Uchôa II. Universidade Federal do Amazonas III. Título

DEDICATÓRIA

À Minha família e amigos pelo incentivo
durante os anos de pós-graduação

AGRADECIMENTOS

À minha família pelo constante apoio.

Ao Daniel Tadeu, pelo companheirismo desde a época da Graduação, que auxiliou de forma constante durante toda a caminhada acadêmica.

Aos amigos e integrantes do USES (Grupo de Pesquisa em Usabilidade e Engenharia de Software) da UFAM, que auxiliaram e forneceram o suporte necessário para a execução dessa pesquisa.

Ao professor Leandro Galvão, por ter participado na banca de defesa do exame de qualificação e pelas sugestões para a pesquisa.

Ao INCT-SEC (Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos), no qual este trabalho está incluso, no suporte no decorrer da pesquisa para cooperação entre universidades e apoio para execução de experimento e participação em conferências.

Aos professores Valter Vieira de Camargo e Rosângela Delloso Penteado por fornecer a linha de produto que foi utilizada nesta pesquisa; sem a qual não seria possível a realização do estudo de caso.

Aos professores José Carlos Maldonado e Itana Maria de Souza Gimenes pelas contribuições ao trabalho.

À secretaria do programa de pós-graduação, em especial à Elienai Nogueira no apoio nas atividades necessárias junto à Universidade.

Aos professores Raimundo da Silva Barreto e Edson A. Oliveira Jr. aceitarem participar da Banca de defesa da Dissertação.

À Prof.ª Tayana Conte, minha orientadora, pelo apoio, empenho e dedicação na realização desse projeto.

RESUMO

Linha de produto de software permite que organizações desenvolvem um número similar de produtos específicos em um mesmo domínio de aplicação, reduzindo o tempo de desenvolvimento e o custo de manutenção, e aumentando a produtividade. Especificações de linha de produto de software necessitam ser avaliadas para aumentar a qualidade do software. Neste cenário, inspeções de software visam garantir que os artefatos estejam completos, consistentes e corretos ao encontrar defeitos em estágios iniciais do ciclo de vida de desenvolvimento. Neste trabalho, é proposta a SPLIT (*Software Product Line Inspection Technique*), um conjunto de técnicas de inspeção baseada em modelos para avaliar especificações de linha de produto de software. Um estudo de viabilidade foi conduzido para comparar a SPLIT com uma abordagem de inspeção baseada em tipos de defeitos. O objeto do estudo de viabilidade foi um cliente de Twitter desenvolvido para a execução do experimento. Os resultados indicaram que o conjunto de técnicas SPLIT encontrou um número maior de defeitos que uma abordagem de inspeção baseada em tipos de defeitos. Um segundo experimento foi conduzido para comparar a SPLIT com uma abordagem baseada em tipos de defeitos usando uma linha de produto de software real para veículos robóticos móveis. Este estudo visou verificar se o conjunto de técnicas se adaptam ao ciclo de vida de desenvolvimento real. O segundo experimento corroborou o resultado do estudo de viabilidade ao encontrar um número de defeitos maior que uma abordagem de inspeção baseada em tipos de defeitos. Com isso, a garantia da qualidade em linhas de produto de software pode ser melhorada ao utilizar a SPLIT para encontrar defeitos em estágios iniciais do desenvolvimento.

Palavras-chaves: Linha de Produto de Software, Técnica de Inspeção, SPLIT, LPS

ABSTRACT

Software Product Lines enable organizations to develop a number of similar products in the same application domain, which reduces development time and maintenance cost, and increases productivity. Software product line specifications need to be evaluated for improving software. In this context, software inspections aims do guarantee complete, consistent and correct artifacts finding defects in early stages in software lifecycle. In this work, we propose SPLIT (Software Product Line Inspection Technique), which is a set of model-based inspection techniques for evaluating software product line specifications. A feasibility study was conducted for comparing SPLIT against a defect type-based inspection approach. The object of the feasibility study was a Twitter client software product line specification created for the experiment. The results indicated that the set of techniques found a greater number of defects than a defect type-based inspection approach. A second empirical study was conducted for comparing SPLIT against a defect type-based inspection approach using a real software product line for mobile robot vehicles. This study aimed to verify whether the set of techniques fits real world life-cycle development. The second empirical results supported the feasibility study in which SPLIT found a greater number of defects than a defect type-based inspection approach. Thus, software product line quality assurance can be improved using SPLIT for detecting defects in early stage of development.

Keywords: Software Product Line, Inspection Technique, SPLIT, SPL

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1. INTRODUÇÃO

Linha de produto de software (LPS) é uma abordagem que tem como principal objetivo promover a geração de produtos similares, porém específicos, com base na reutilização de uma infraestrutura central, reduzindo o tempo de desenvolvimento e manutenção, e aumentando a produtividade [Denger e Kolb, 2006]. Essa abordagem é adequada aos domínios que exigem uma demanda por produtos que possuem características comuns, mas que contém pontos de variação bem definidos.

A abordagem de LPS auxilia a reusabilidade por meio do desenvolvimento de um conjunto de produtos que compartilham de uma parte central e diferem em variabilidades [Weiss e Lai, 1999]. Por conseguinte, diferentemente do desenvolvimento de software tradicional, onde acontece a modelagem de um produto por vez, ao utilizarmos a abordagem de linha de produto de software, tem-se a modelagem de um conjunto de produtos de uma única vez.

Uma razão essencial para a introdução de engenharia de linha de produto é a redução de custos [Pohl *et al.*, 2005]. Isso ocorre pelo fato de cada produto usar as mesmas funcionalidades centrais e algumas opcionais, gerando uma redução de custo para cada produto gerado pela LPS.

Os artefatos de uma linha de produto de software podem ser reutilizados, porém é necessário investimento para criá-los, planejando o mecanismo de reuso, para que a reusabilidade das funcionalidades possa ser gerenciada. Em uma linha de produto de software, em geral, o *time to market* é inicialmente maior, pois os artefatos comuns devem ser criados primeiro. No entanto, o tempo para o mercado é diminuído consideravelmente assim que os artefatos podem ser reutilizados em cada novo produto [Pohl *et al.*, 2005].

1.1. Contexto

Este trabalho está inserido em uma pesquisa conduzida pelo Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos (INCT-SEC) [INCT-SEC, 2012]. Em uma de suas linhas de pesquisa, o INCT-SEC visa definir um ambiente de desenvolvimento de sistemas embarcados críticos buscando prover suporte automatizado à construção de sistemas desse domínio, dando ênfase principalmente às atividades que são críticas à qualidade desses sistemas.

Com a utilização de técnicas de linha de produto em sistemas embarcados críticos, o Centro de Linha de Produto de *Software* para Sistemas Embarcados Críticos do INCT-SEC tem como objetivo auxiliar a cooperação e comunicação entre os grupos de pesquisa de engenharia de *software* e sistemas embarcados do INCT-SEC e grupos de pesquisa e empresas no Brasil e no exterior nas áreas de veículos autônomos móveis aéreos e terrestres.

1.2. Definição do Problema

A abordagem de linha de produto de software necessita resolver os mesmos problemas que o desenvolvimento de software tradicional, como por exemplo, inconsistências entre requisitos e especificações de software. Uma abordagem aplicada ao desenvolvimento de software tradicional que melhora a qualidade e diminui os custos são técnicas de inspeção, utilizadas para identificar defeitos em estágios iniciais do desenvolvimento [Travassos *et al.*, 1999].

Inspeções de software visam garantir que um determinado artefato de software é completo, consistente, não ambíguo e contém o menor número de defeitos possíveis para efetivamente apoiar o desenvolvimento de software. As inspeções de software dependem da compreensão humana para a identificação dos defeitos, por isso apresentam o benefício de serem conduzidas assim que o artefato de software é concluído, e podem ser utilizadas em diferentes artefatos e notações. No contexto do desenvolvimento de software tradicional, é possível identificar aumentos substanciais de qualidade e redução de defeitos devido à adoção de inspeções e outras técnicas de revisão (Travassos et al. 1999; Leite et al. 2005).

Inspeções de projetos de software são essenciais, pois defeitos de projeto podem afetar diretamente a qualidade e o esforço necessário para implementação [Travassos et al., 1999].

Técnicas para assegurar a qualidade como testes e inspeções são eficientes e efetivos para detectar e remover defeitos [Geppert *et al.*, 2005]. Apesar da comunidade de linha de produto de software estar ciente da necessidade de técnicas para assegurar a qualidade, aspectos de LPS ainda não são suficientemente considerados nas abordagens atuais [Denger e Kolb, 2006], principalmente, no que tange a questão da variabilidade em linhas de produto de software. Dessa forma, o uso em linhas de produto de software, pode resultar em um software sem qualidade, e assim, reduzir os benefícios da inspeção de software. Segundo Denger e Kolb (2006), pelo fato de modelos de LPS serem diferentes de modelos de desenvolvimento de um sistema único, técnicas padrões, utilizadas no desenvolvimento de software tradicional, são insuficientes para verificar as características específicas de sistemas de reusabilidade. Dessa maneira, novas técnicas de inspeções específicas para linhas de produto de software são necessárias.

Esse cenário motivou o objetivo desta pesquisa: definir um conjunto de técnicas de inspeções para auxiliar a garantia de qualidade para modelos de linhas de produto de software.

1.3. Objetivos

A meta deste trabalho consiste em propor tecnologias de software que apoiem a avaliação da qualidade, para atender os requisitos específicos de linha de produto de software de acordo com a definição do problema apresentado na Seção 1.2.

Esta meta pode ser decomposta em três objetivos (1) identificar características dos modelos de linha de produto de software; e (2) definir um conjunto de técnicas de inspeção em especificações de linha de produto de

software; e (3) avaliar experimentalmente o conjunto de técnicas de inspeção em linha de produto de software proposto.

1.4. Metodologia

O método de pesquisa utilizado no desenvolvimento deste trabalho é baseado em evidências para a definição de novas tecnologias de software, proposto em [Mafra *et al.*, 2006]. Essa metodologia, uma extensão daquela proposta por [Shull *et. al*, 2001], estabelece, além das fases a serem seguidas para a introdução de tecnologias de software na indústria, uma etapa de definição inicial de uma tecnologia.

A metodologia utilizada neste trabalho está especificada na **Figura 1.1**. Nesta, foi proposta a realização de um estudo secundário, por meio de um mapeamento sistemático, para a criação de um corpo de conhecimento acerca do tema proposto. E, baseada nesse estudo secundário, é definida a proposta inicial da tecnologia, que evolui no decorrer da pesquisa com a realização de experimentos para a sua validação e futuras melhorias.

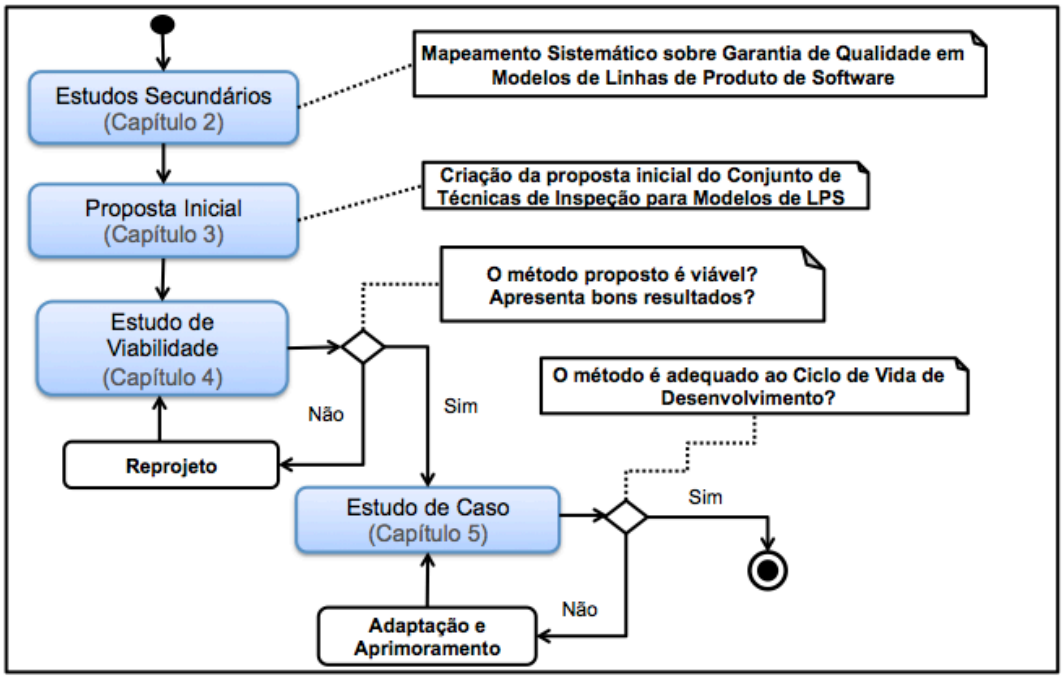


Figura 1.1. Método de pesquisa baseado em Mafra *et al.* (2006) e Shull *et al.* (2001).

1.5. Organização do texto

O restante deste trabalho está estruturado da seguinte forma: o Capítulo 2 descreve os conceitos de linha de produto de software, os principais modelos de especificação utilizados nessa abordagem, e o planejamento e a execução do mapeamento sistemático acerca de garantia de qualidade em modelos de linha de produto de software; o Capítulo 3 apresenta a proposta do conjunto de técnicas de inspeção em modelos de linha de produto, chamada SPLIT (*Software Product Line Inspection Technique*); o Capítulo 4 mostra o estudo de viabilidade realizado, descrevendo os seus objetivos, planejamento, execução e resultados; o Capítulo 5, assim como o capítulo anterior, apresenta um estudo experimental, porém trata-se um estudo de caso, utilizando um exemplo real de uma linha de produto de software de veículos robóticos móveis; e o Capítulo 6 contém as considerações sobre os resultados obtidos no decorrer da pesquisa e trabalhos futuros.

CAPÍTULO 2. REFERENCIAL TEÓRICO

2.1. Linha de Produto de Software

Uma linha de produto de software é um sistema intensivo de software, que compartilha um conjunto comum e gerenciado de funcionalidades que satisfazem à necessidade específica de um segmento particular do mercado ou de uma missão, sendo desenvolvido a partir de um conjunto comum de artefatos de forma planejada [Clements e Northrop, 2001].

A abordagem de LPS possibilita às organizações explorar semelhanças entre seus produtos, aumentando, assim, a reutilização de artefatos. Como consequência, tem-se uma diminuição dos custos e tempo no desenvolvimento [Heymans e Trigaux, 2003].

Durante o processo de desenvolvimento de uma LPS, são identificados e descritos os produtos criados pela linha de produto por meio das diferenças pelas funcionalidades que proveem, os requisitos que atendem e, até mesmo, da sua infraestrutura de arquitetura [Clements e Northrop, 2001]. Esta flexibilidade é chamada de variabilidade e constitui a base de engenharia da linha de produto de software, pois esta representa a diferença em cada produto específico.

Segundo Pohl *et al.* (2005), a engenharia de linha de produto é um paradigma de desenvolvimento de software (sistemas de software intensivos e produtos de software) que utiliza uma infraestrutura central e customização em massa. Para isso, a infraestrutura central é o plano estratégico para o desenvolvimento de artefatos reutilizáveis, juntamente com a sua utilização; e a customização em massa significa a utilização de variabilidade gerenciável, no qual as diferenças e similaridades da aplicação devem ser modeladas de maneira única.

2.1.1. Motivações para a utilização de LPS

Segundo Pohl *et al.* (2005), a principal razão para a engenharia de linha de produto é fornecer produtos customizados a um preço competitivo. As

principais motivações, para o desenvolvimento de software utilizando a abordagem de linha de produto de software são descritas nas subseções a seguir.

2.1.1.1. Redução do custo de desenvolvimento

Uma razão essencial para a introdução da engenharia de linha de produto é a redução de custos. A partir da utilização de uma infraestrutura central em diferentes sistemas, obtém-se a redução de custo em cada sistema. Porém, é necessário um investimento inicial da empresa, para que sejam desenvolvidos artefatos que proveem um reuso gerenciado, possibilitando, posteriormente, ser alcançada uma redução do custo por produto.

A **Figura 2.1** apresenta o custo acumulado de desenvolvimento de acordo com o número de produtos, comparando a abordagem de linha de produto de software com a abordagem de desenvolvimento tradicional. A linha sólida apresenta o custo de desenvolvimento de sistemas independentes, enquanto a linha tracejada apresenta o custo para uma engenharia de linha de produto. No caso de um pequeno número de sistemas, os custos de uma linha de produto de software são relativamente altos, porém são significantemente menores para uma grande quantidade de sistemas.

Neste caso, é possível notar que inicialmente temos um custo maior para o desenvolvimento de uma LPS, porém, em um determinado momento, estes custos se igualam. No caso, avaliações experimentais sugerem que este ponto é de aproximadamente três sistemas [Pohl *et al.*, 2005]. Esta métrica, porém, depende das características da organização, a experiência da equipe, base de usuário, mercado e variedade e tipo de produtos.

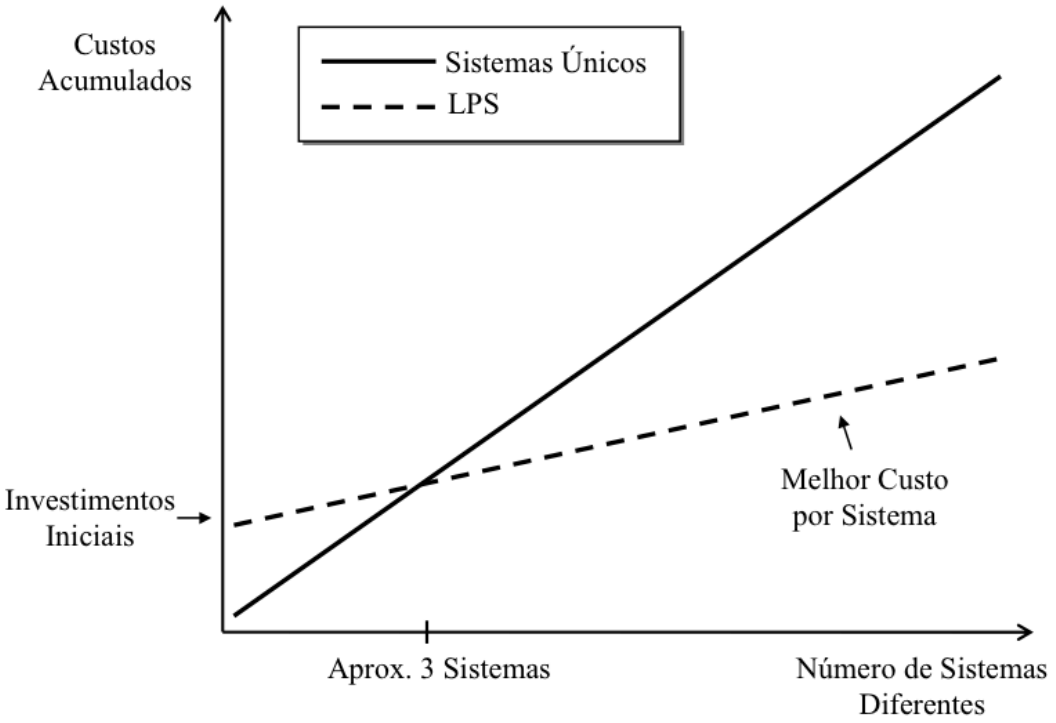


Figura 2.1. Gráfico de comparação entre o custo de desenvolvimento para Sistemas Únicos e para LPS segundo Pohl *et al.* (2005)

2.1.1.2. Melhoria da qualidade

Os artefatos da infraestrutura central de uma LPS são revisados e testados em uma série de produtos. Eles tem que apresentar um funcionamento correto em mais de um tipo de produto. Isto implica em uma maior chance de detectar erros e corrigi-los, e dessa maneira, melhorar a qualidade de todos os produtos.

Técnicas usadas em linhas de produto de software contribuem para a diminuição dos erros, principalmente pela propriedade de reuso de código, em que a correção de um único defeito pode ter impacto em todos os produtos que utilizam esse componente de código, não sendo então necessário corrigir vários erros. Isso implica em uma menor necessidade de suporte aos clientes, ciclos de testes menores e menor atividade de correção de erros.

2.1.1.3. Redução de *Time To Market*

Para o desenvolvimento tradicional, no qual apenas um produto é desenvolvido isoladamente, existe um tempo para o desenvolvimento de um produto. Já no caso de produtos específicos de uma LPS[1], após o tempo e investimento inicial para o desenvolvimento da infraestrutura central, temos o tempo reduzido para o mercado devido à reutilização do núcleo de artefato, como podemos perceber na **Figura 2.2**.

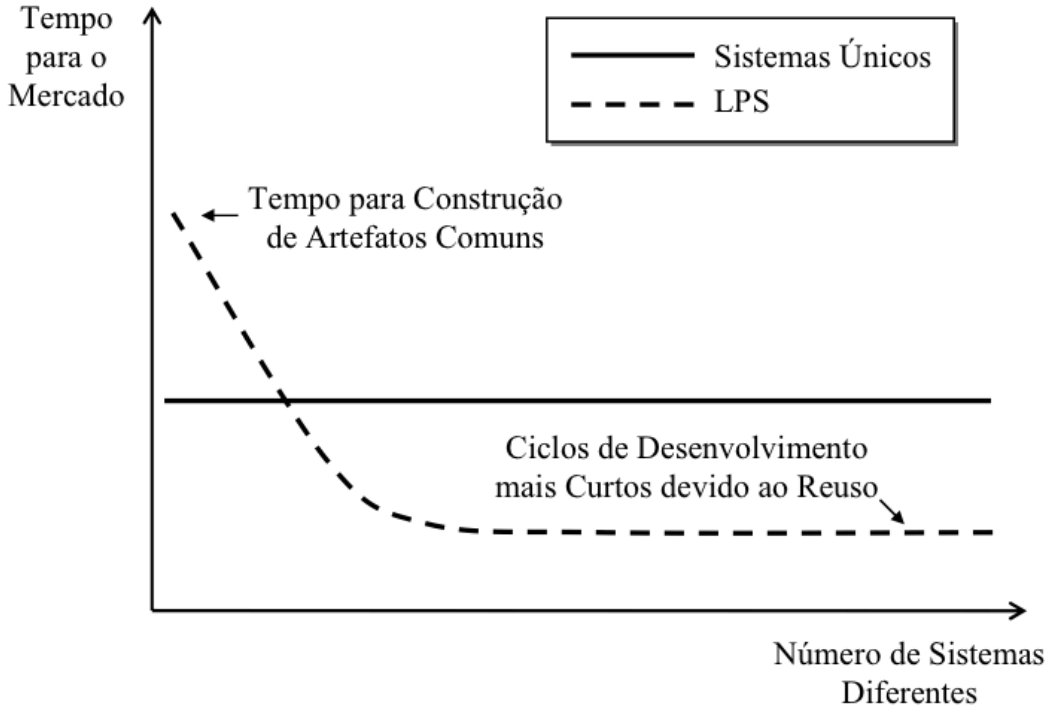


Figura 2.2. Gráfico de comparação entre o tempo de desenvolvimento para Sistemas Únicos e para LPS segundo Pohl *et al.* (2005)

2.2. Atividades Essenciais de Linha de Produto de Software

O Software Engineering Institute (SEI), através da iniciativa *Product Line Practice* (PLP), estabeleceu as atividades essenciais de uma abordagem de LPS (SEI, 2004). Na sua essência, elaborar uma linha de produto compreende o

[1] Segundo Pohl et al. (2005), o tempo de desenvolvimento varia de acordo com o escopo e complexidade do sistema, porém para efeito de comparação entre a abordagem tradicional e a abordagem de linha de produto de software, esta afirmação é suficientemente correta.

Desenvolvimento do Núcleo de Artefatos, o Desenvolvimento do Produto utilizando os recursos do núcleo de artefatos, sob gestão técnica e organizacional.

O Desenvolvimento do Núcleo de Artefatos e o Desenvolvimento de Produtos a partir do núcleo de artefatos podem ocorrer em qualquer ordem: novos produtos são construídos a partir do núcleo de artefatos ou artefatos são extraídos de produtos existentes. Muitas vezes, os produtos e artefatos essenciais são construídos em simultaneamente. A Figura 2.3 ilustra as três atividades essenciais de uma linha de produto de software segundo SEI (2004).

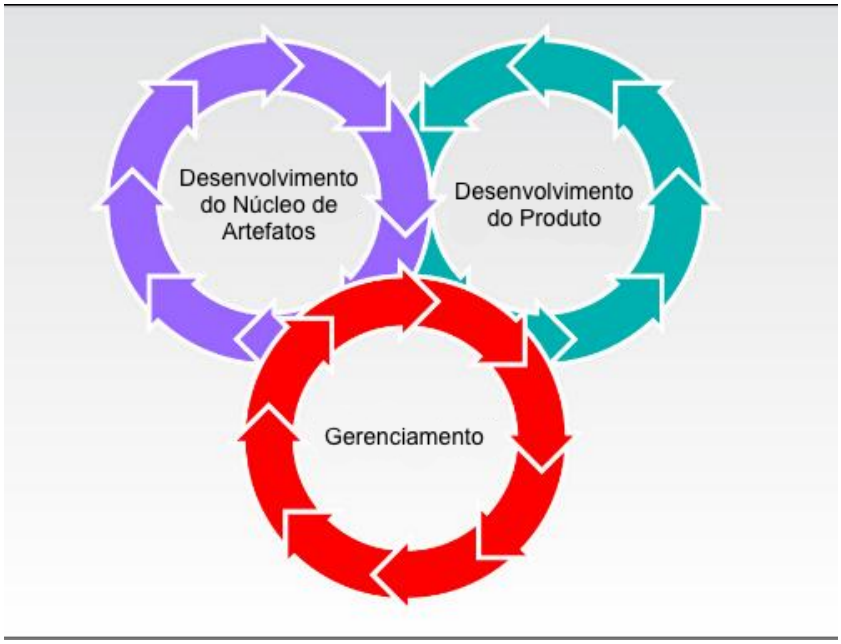


Figura 2.3. As três atividades essenciais de uma linha de produto de software (SEI, 2004)

Cada círculo representa uma das atividades essenciais do desenvolvimento de uma linha de produto de software. As três atividades estão ligadas entre si e em continuo movimento, mostrando que elas são essenciais e interativas, e dessa forma, podem ocorrer em qualquer ordem.

As setas rotativas indicam que não apenas o Núcleo de Artefatos Essencial é usado para desenvolver produtos, mas também que as revisões dos principais artefatos existentes ou mesmo novos artefatos do núcleo evoluem.

A Figura 2.3 é neutra em relação a qual a parte do desenvolvimento de uma Linha de Produto de Software é iniciado primeiro. Em alguns contextos, através dos produtos já existentes, são os artefatos - uma especificação de

requisitos genérica, a arquitetura de um artefato ou componentes de software - que depois são migrados para o núcleo de artefatos da linha de produtos. Em outros casos, os artefatos do núcleo podem ser desenvolvidos para um uso, posterior, na produção de produtos.

Existe um ciclo de feedback entre os artefatos e os produtos de uma linha de produto de software. O núcleo de artefatos é atualizado a medida que novos produtos são desenvolvidos. O uso dos artefatos do núcleo é gerenciado e os resultados são integrados com a atividade de desenvolvimento de artefato para o núcleo. Além disso, o valor dos recursos do núcleo é calculado por meio dos produtos que são desenvolvidos a partir deles. Como resultado, os artefatos principais são criados de forma mais genérica, considerando potenciais novos produtos.

O desenvolvimento de uma linha de produto de software requer uma necessidade constante de gestão, devido a necessidade de investimento em recursos de desenvolvimento e manutenção dos principais artefatos do núcleo.

Os novos produtos devem estar alinhados com os principais artefatos existentes, ou estes, devem ser atualizados para refletir os novos produtos que estão sendo desenvolvidos. A iteração é inerente ao desenvolvimento de uma linha de produto de software, isto é, transformar os artefatos essenciais e produtos.

2.2.1. Desenvolvimento do Núcleo de Artefatos

O objetivo da atividade do Desenvolvimento do Núcleo de Artefato é estabelecer as *features* para os novos produtos. O desenvolvimento de artefato do núcleo acontece em uma situação contextual de restrições e recursos existentes. Este contexto influencia em como o núcleo de artefatos é desenvolvido e a natureza de seus produtos. Os quatros fatores contextuais mais importantes, segundo o SEI (2004) são:

- Restrições de Produto: quais *features* comuns e variações existentes nos produtos irá constituir a linha de produto de software? Que *features* comportamentais está ira prover? Quais features o mercado espera que os produtos

contenham no futuro? O núcleo de artefatos deve identificar as features comuns e acomodar possíveis variações futuras com o mínimo de alteração na qualidade do produto como segurança, confiabilidade e usabilidade. Essas restrições podem ser derivadas de uma série de produtos preexistentes que servirá de base para uma linha de produto;

- Restrições de desenvolvimento: um produto novo deve ser levado ao mercado uma vez por ano, por mês ou por dia? Que tipo de padrão de indústria de software deve ser seguido durante o desenvolvimento do produto? Estas questões proveem decisões sobre quais tipo de mecanismos de variações os artefatos do núcleo devem ter e que tipo de processo de desenvolvimento deve ser usado no planejamento;

- Estratégia de Desenvolvimento: a estratégia de desenvolvimento é a abordagem geral que deve ser realizada no núcleo de desenvolvimento e produtos. A linha de produtos de software deve ser construída (i) proativamente - aquela iniciada com um conjunto de artefatos e os produtos são gerados a partir daqueles; (ii) reativamente, aquela que é iniciada a partir de um conjunto de produtos e seus componentes são generalizados para produzir os artefatos da linha de produto de software; (iii) ou usando alguma combinação dos dois? A estratégia de desenvolvimento define a arquitetura da linha de produto de software e os componentes associados, assim como, a estratégia de crescimento da linha de produto de software. De acordo com as restrições de produto e as restrições de desenvolvimento, a estratégia de desenvolvimento também é responsável por identificar os produtos que serão gerados pela LPS;

- Artefatos preexistentes: Sistemas legados e produtos existentes identificam o conhecimento no domínio de uma empresa ou define sua representação de mercado. Os produtos de uma linha de produto de software, ou pelo menos parte dela, deve ser resultante da experiência relativa de sistemas legados ou produtos existentes. Os componentes podem ser generalizados a partir de sistemas legados e podem representar a propriedade intelectual de uma organização em um determinado domínio, e dessa forma, ser incorporada ao núcleo de artefatos principais. Existem bibliotecas, *frameworks*, algoritmos, ferramentas, componentes e serviços que podem ser utilizados? Por meio da

análise detalhada, uma organização deve determinar o que vai usar. No entanto, é importante ressaltar que os artefatos a serem incorporados na LPS não devem ser limitados apenas aos desenvolvidos na organização. Software disponível externamente, como *web services*, produtos de código aberto, padrões de projeto e *frameworks* são exemplos de artefatos preexistentes que podem ser incorporados para a linha de produto da organização.

Os fatores contextuais são importantes para identificarmos as saídas do desenvolvimento do núcleo de artefatos, que segundo o SEI (2004) são o escopo da linha de produto de software, a base de artefatos do núcleo e o plano de desenvolvimento.

2.2.1.1. Escopo da Linha de Produto de Software

O escopo da linha de produto é a descrição dos produtos que a linha de produto de software irá constituir ou os que a linha de produto tem a capacidade de incluir. Estes podem consistir de uma enumeração com os nomes dos produtos, e mais tipicamente, esta descrição está classificada de acordo com as similaridades e diferenças de seus produtos. Estas similaridades e diferenças podem incluir *features* ou operações que a linha de produto de software fornecem, a plataforma no qual esta vai ser executando, e assim por diante.

O sucesso de uma linha de produto depende da definição de seu escopo. Se o escopo for muito amplo, os membros de uma linha de produtos também serão muito vago, e com isso, os artefatos do núcleo não seriam capazes de acomodar a suas variações, dessa maneira, a economia de desenvolvimento é perdida. Da mesma forma, que um escopo muito pequeno, pode não permitir que os artefatos do núcleo sejam desenvolvimento de forma genérica suficiente para acomodar futuros crescimento, e com isso, a linha de produto de software pode estagnar, dessa forma, o potencial de economia não é realizado e o retorno de investimento não acontece.

O escopo da linha deve incluir as alterações nas condições do mercado, as alterações no planejamento da organização, o aparecimento de novas oportunidades. A evolução do escopo é o ponto inicial para manter a linha de produto de software atualizada. A definição do escopo de uma linha de produto de

software é um artefato do núcleo, e dessa forma, deve ser evoluído e mantido no ciclo de vida da LPS.

2.2.1.2. Base de Artefatos do Núcleo

A base de artefatos do núcleo é composta por todos os artefatos que são essenciais dos produtos da linha de produto de software. Não são todos os artefatos que serão usados em todos os produtos da linha de produtos de software, no entanto, estes devem ser utilizados suficientemente nos produtos para que o seu desenvolvimento coordenado, manutenção e evolução sejam necessárias.

Cada artefato deve ser associado com processos que especificam como estes devem ser utilizados no desenvolvimento dos produtos atuais. O processo pode também especificar o suporte a ferramentas automatizadas para complementar o processo de inclusão na linha de produto de software.

Todos os processos devem seguir uma abordagem de implementação chamada de método de desenvolvimento. Os processos anexados a base de artefatos se tornam o plano de desenvolvimento da linha de produto de software.

Existem também artefatos de natureza não técnica como treinamento específicos para a linha de produto de software, o plano de negócios da abordagem de linha de produto de software para um conjunto de produtos, o processo de gerenciamento técnicos associados com a LPS e os riscos da construção de uma linha de produto.

2.2.1.3. Plano de Desenvolvimento

Esta saída inclui o processo a ser utilizado para a construção de produto. Como mencionado na saída anterior - Base de Artefatos do Núcleo -, cada artefato deve ter associado o processo que define como este é utilizado no desenvolvimento do produto. O conjunto desses processos associados aos artefatos, são necessários para manter a coerência do produto como um todo, definindo o desenvolvimento do produto.

O plano de desenvolvimento é responsável pela estratégia e restrições de desenvolvimento e definem o método de desenvolvimento. O método de

desenvolvimento, que é a abordagem geral que especifica os modelos, os processos e as ferramentas escolhidas para o desenvolvimento.

Esta fase também inclui o projeto em detalhes que permite a execução e o gerenciamento do processo, assim como, detalhes de cronogramas, recursos necessárias e métricas.

2.2.2. Desenvolvimento do Produto

A atividade de Desenvolvimento do Produto depende das três saídas definidas na atividade de Desenvolvimento de Artefato do Núcleo – o escopo da linha de produto, a base de artefato de núcleo e o plano de desenvolvimento – assim como a descrição individual de cada produto. Dessa forma, a existência e a disponibilidade de um produto particular pode afetar os requisitos de um produto posterior ou ainda a construção de um produto que teve uma semelhança não reconhecida com outro produto. Isto irá gerar uma atualização da base de artefatos do núcleo e prover a base para explorar essa semelhança posteriormente.

As entradas para a atividade de desenvolvimento dos produtos são:

- A descrição específica de um produto, expressada em forma de variância da descrição genérica de um produto contido no escopo da linha de produto;

- O escopo da linha de produto, que indica se é viável incluir o produto na linha de produto de software;

- Os artefatos que vão criar o produto; e

- O plano de desenvolvimento que vai detalhar como os artefatos vão ser utilizados para criar o produto.

2.2.3. Gerenciamento

O gerenciamento tem um importante papel na abordagem de uma linha de produto de software, devido a relevância da alocação de recursos e coordenação

e supervisão das atividades. O gerenciamento no nível organizacional e técnico deve estar comprometido com o esforço da linha de produto de software.

O gerenciamento organizacional identifica as restrições de desenvolvimento e determina a estratégia de implementação, e dessa forma, cria a estrutura organizacional que permite que as unidades tenham os recursos adequados para o desenvolvimento da linha de produto de software. Tal gerenciamento é definido como a autoridade responsável pelo sucesso ou falha do esforço da implementação de uma linha de produto de software.

O gerenciamento da organização determina o modelo que vai proporcionar a evolução dos artefatos do núcleo e fornece o orçamento, de acordo com o mesmo. Uma das atividades mais importantes diz respeito à criação de um plano de adoção que descreve o estado desejável da organização, ou seja, a criação de produto por meio de uma linha de produto de software e a estratégia para atingi-la.

O gerenciamento técnico coordena o desenvolvimento dos artefatos e as atividades de desenvolvimento assegurando que estas estão criando artefatos para o núcleo e que os produtos gerados estão de acordo com as atividades requisitadas. Dessa forma, devem seguir os processos da linha de produto de software e monitorar as métricas para verificar o progresso. O gerenciamento técnico define o método de desenvolvimento e fornece os elementos gerenciáveis ao plano de desenvolvimento.

O gerenciamento técnico e organizacional também contribui para a base de artefatos do núcleo ao fornecer para reuso os artefatos gerenciáveis (especialmente cronograma e orçamento) utilizados no desenvolvimento de produtos da linha de produto de software.

2.3. Abordagens para Especificação de Linha de Produto de Software

A principal atividade de uma LPS é identificar e gerenciar as semelhanças e as variações em um conjunto de artefatos como requisitos, arquiteturas, componentes e casos de testes. Na engenharia de linha de produto, a

variabilidade fornece a flexibilidade necessária para a diferenciação e a diversificação de produtos. A variabilidade refere-se a habilidade de um artefato ser configurado, customizado, estendido ou alterado para a utilização em um contexto [Bachmann e Clements, 2005].

O gerenciamento da variabilidade inclui as atividades explicitamente representadas na variabilidade de artefatos de softwares durante o ciclo de vida, gerenciamento de dependência entre os diversos tipos de variabilidade e utiliza as instanciações dessas variabilidades [Schmid e John, 2004].

Devido a relevância do Gerenciamento de Variabilidade na Engenharia de Linha de Produto, diversas abordagens foram desenvolvidas com o objetivo básico de auxiliar (ou automatizar) as tarefas envolvidas no Gerenciamento de Variabilidade nos diversos estágios do ciclo de vida de uma linha de produto [Chen *et al.*, 2009].

A revisão sistemática de Chen *et al.* (2009) descreve as diversas abordagens em Gerenciamento de Variabilidade, de acordo com a qual os artigos revisados são bastante diversos, em termos de objetivos, filosofia de planejamento, técnicas de modelagem de variabilidade, suporte de processo, etc.

A maioria das abordagens são orientadas a *features*, como *Feature-Oriented Domain Analysis* (FODA) [Kang, 1990] e suas extensões. Algumas abordagens são orientadas a arquitetura como em Hoek (2004), Koalish e Thiel (2002). Algumas abordagens são baseadas em configuração como Krueger (2002), Simmena et al. (2004), Koalish e Thiel (2002) e Asikainen *et al.* (2007). Outras abordagens estendem a UML como modelo de variabilidade como Webber e Gomma (2004) e Halmans e Pohl (2003). Algumas abordagens destacam a separação da representação de variabilidade da representação de vários artefatos da Engenharia de Linha de produto como Bachmann et al. (2004) e Muthig e Atikinson (2002). Os desenvolvedores de outras abordagens enfatizaram a importância de independência de notação e customização para facilitar a adoção, como em Schmid e John (2004). O principal objetivo de FAST [Weiss e Lai, 1999] é fornecer suporte sem a necessidade de um modelo especifico de gerenciamento de variabilidade.

A abordagem Stereotype-based Management of Variability (SMarty), permite o gerenciamento de variabilidade por meio de um perfil UML e de um processo composto por atividades e diretrizes bem definidas para identificar, delimitar e representar variabilidades em modelos UML de uma LPS utilizando os estereótipos propostos em seu perfil [Oliveira Junior *et at.*, 2010] [Oliveira Junior *et at.*, 2013] . O perfil SMartyProfile contem um conjunto de estereótipos e meta atributos para representar variabilidades em modelos UML de LPS.

Dentre essas abordagens, destacamos o modelo de *features* proposto pela abordagem de análise de domínio FODA [Kang, 1990], por ser um dos mais utilizados [Massen e Litchter, 2004] no gerenciamento de portfólio e projetos, análise de domínio e derivação de produto. Essa abordagem foi desenvolvida pela necessidade de que grandes e complexos sistemas de software requerem um claro entendimento de suas funcionalidades e recursos [Kang, 1990]. E, dessa forma, o reuso de software se torna viável apenas quando essas funcionalidades e recursos comuns da aplicação são corretamente identificados.

A definição do escopo de uma linha de produto de software é também relevante por especificar, não apenas os produtos que vão ser incluídos para geração pela LPS, mas também se uma organização deve ou não iniciar o desenvolvimento de uma LPS [Lee *et al.*, 2010]. Uma abordagem para definição de escopo de LPS é o mapa de produtos [Bayer *et al.*, 1999], o qual permite visualizar em forma tabulada os produtos gerados.

Estes artefatos das abordagens de Kang (1990) e Bayer *et al.* (1999), o modelo de *features* e o mapa de produtos, são detalhados nas duas subseções seguintes.

2.3.1. Modelo de *Features*

A FODA (*Feature Oriented Domain Analysis*) é uma abordagem orientada a funcionalidades, baseando-se na ênfase colocada no método de identificação das funcionalidades que um usuário comumente espera no domínio da aplicação [Kang, 1990]. Esta análise do domínio é parte da análise de requisitos

e do projeto alto nível e engloba uma família de produtos em um domínio, ao invés de um sistema único, produzindo um modelo de domínio parametrizável para identificar as diferenças e a arquitetura padrão para o desenvolvimento de componentes.

A abordagem é dividida nas etapas de Análise de Contexto, Modelagem de Domínio e Modelagem de Arquitetura [Kang, 1990]:

- A etapa de análise de contexto consiste na interação com usuário e especialistas do domínio para definir o escopo a ser analisado;

- A modelagem do domínio consiste na utilização das informações dos usuários e dos produtos da análise de contexto para a criação do domínio do modelo; e

- A modelagem da arquitetura consiste na utilização do domínio do modelo para o desenvolvimento do modelo de arquitetura da aplicação.

A etapa de modelagem do domínio produz os seguintes artefatos: modelo de entidade relacionamento, modelo de *features*, modelo funcional e dicionário de terminologia do domínio.

Como o interesse primário da abordagem é identificar as semelhanças das aplicações de uma família de produtos de sistemas, utilizaremos o modelo de *features* para mostrar as *features* em comum e diferentes da aplicação do domínio. Com isso, as *features* do modelo são utilizadas para generalizar e parametrizar os produtos.

Os modelos de *features* são modelos hierárquicos que fornecem os artefatos comuns e um subconjunto da variabilidade de uma linha de produto de software. Esta abordagem de orientação a *features* é baseada na ênfase colocada no método de identificação das funcionalidades que um usuário espera no domínio da aplicação.

O conceito de *features* utilizado para o modelo de *features* é de que, esta é qualquer aspecto, qualidade ou característica de software perceptível visualmente pelo usuário [Kang, 1990].

Modelos de *features* são criados utilizando *features* obrigatórias, opcionais, alternativas e inclusivas. As *features* obrigatórias são aquelas que estão contidas em todos os produtos criados pela linha de produto de software. As *features* opcionais são aquelas que podem ou não estar contidas nos produtos da LPS. As *features* alternativas são aquelas em que duas ou mais *features*, apenas uma delas pode estar contida nos produtos da LPS. E as *features* inclusiva são aquelas em que de um conjunto de *features*, pelo menos umas delas deve estar contidas nos produtos de uma LPS.

O modelo de *features* também contém elementos de restrições: implicações e exclusões. A notação de implicação representa que quando uma *feature* for selecionada, outra *feature* também deve ser selecionada. A restrição de exclusão indica que duas *features* não podem estar selecionadas em um mesmo produto já que são mutualmente exclusivas.

A **Tabela 2.1** apresenta os relacionamentos, o tipo, a semântica e a notação do modelo de *features* proposto por Kang (1990).

Tabela 2.1: Características do modelo de *features* proposto por Kang (1990)

Relacionamento	Tipo	Semântica	Notação
Relacionamento de Domínio	Mandatório	Se a *feature* pai é selecionada, a *feature* filha também deve ser selecionada	
	Opcional	Se a *feature* pai é selecionada, a *feature* filha pode ser selecionada	
	Alternativa	Se a *feature* pai é selecionada, exatamente uma *feature* filha deve ser selecionada	
	Inclusiva	Se a *feature* pai é selecionada, pelo menos uma da *feature* filha deve ser selecionada.	
Dependência	Implicação	Se uma *feature* é selecionada, a *feature* implicada deve ser selecionada, ignorando a sua posição na árvore de *features*.	
	Exclusão	Indica que ambas as *features* não podem ser selecionadas na mesma configuração de produto e que são mutualmente exclusivas	

A **Figura 2.4** apresenta um exemplo de um modelo de features para um carro. Este descreve duas *features* mandatórias: Transmissão e Motor. A primeira é uma *feature* alternativa, e dessa forma, requer que seja selecionada exatamente um modelo de transmissão; neste caso, as *features* manual ou automático. A segunda é a *feature* de motor, a qual estará presente em todos os produtos gerados pela linha de produto de software. Além das *features* mandatórias, o exemplo tem a *feature* Ar Condicionado, que é opcional, de modo que, pode estar ou não nos produtos gerados pela LPS.

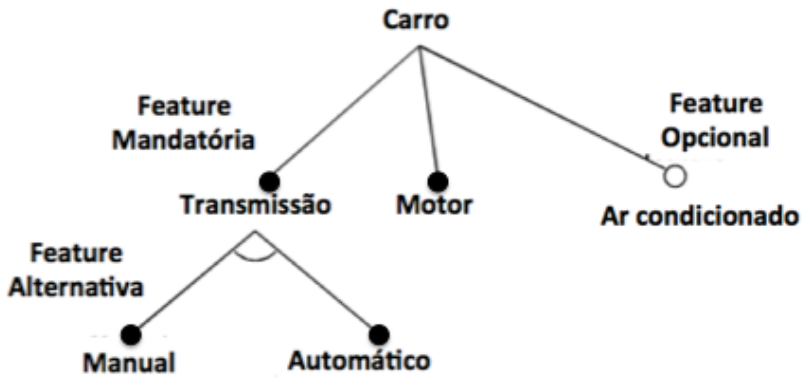


Figura 2.4. Exemplo de Modelo de *features* segundo Kang [1990]

A **Figura 2.5** apresenta o mesmo exemplo da **Figura 2.4**, com o acréscimo de um relacionamento de implicação, por meio do qual, independente da hierarquia das *features*, é possível especificar que toda vez que uma *feature* for selecionada, a outra também deve ser selecionada. Neste caso, é possível observar este comportamento no relacionamento entre a transmissão automática e o ar condicionado, a qual determina que toda vez que a *feature* de transmissão automática for selecionada, a *feature* ar condicionado também deve ser selecionada.

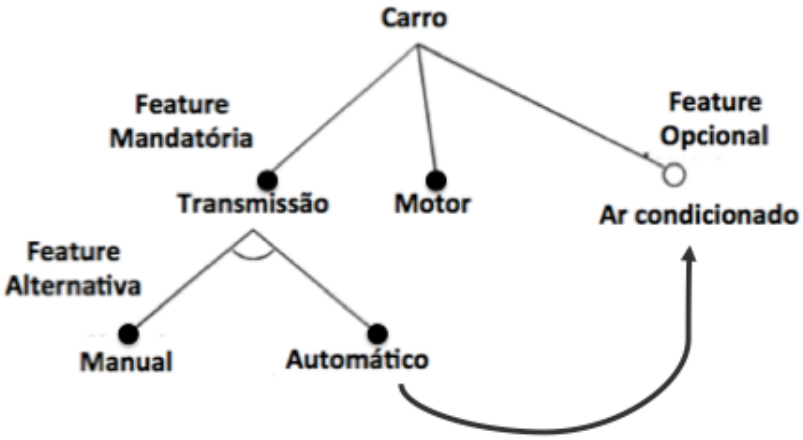


Figura 2.5. Exemplo de Modelo de *features* com implicação baseado em Kang [1990]

A **Figura 2.6** apresenta o mesmo exemplo da **Figura 2.4**, com o acréscimo um relacionamento de exclusão, através do qual, independente da hierarquia das *features*, é possível especificar que toda vez que uma *feature* for selecionada, a outra não pode ser selecionada. Neste caso, é possível observar este comportamento no relacionamento entre a transmissão manual e o ar condicionado, a qual determina que toda vez que a *feature* de transmissão manual for selecionada, a *feature* ar condicionado não pode deve ser selecionada, ou vice-versa.

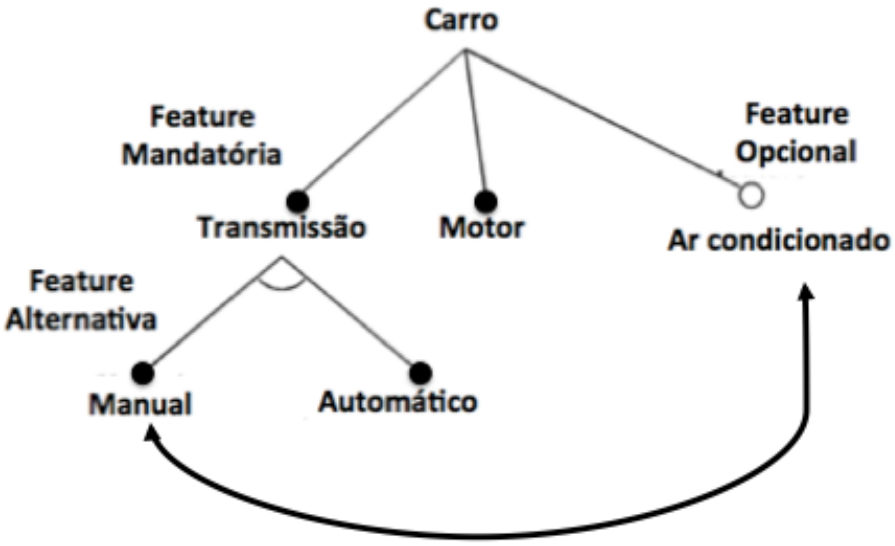


Figura 2.6. Exemplo de Modelo de *Features* com exclusão baseado em Kang [1990]

2.3.2. Mapa de Produtos

Outra abordagem para linha de produto de software é a PuLSE (*Product Line Software Engineering*) [Bayer *et al.,* 1999], tendo esta o objetivo de concepção e implantação de linhas de produto de software para uma grande variedade no contextos profissional.

A abordagem PuLSE [Bayer *et al.,* 1999] se divide em fases de implementação, componentes técnicos e componentes de suporte:

- As fases de implementação descrevem as atividades que devem ser realizadas para configurar e utilizar uma linha de produto e compreende: inicialização, construção e utilização da infraestrutura, evolução e gerenciamento;

- Os componentes técnicos proveem as informações necessárias para operacionalizar a linha de produto e são compostas de: customização, identificação de escopo, modelagem, arquitetura, instanciação e evolução e gerenciamento.

- Os componentes de suporte são pacotes de informação que permitem a melhor adaptação, evolução e implementação da linha de produto e são: os pontos de entrada dos projetos, a escala de maturidade e temas organizacionais.

Um dos componentes técnicos da metodologia consiste em identificar o escopo da linha de produto de software, na qual são identificados os produtos e as suas características. Essa etapa visa explicitamente definir a base do escopo da linha de produto de acordo com os objetivos de negócios identificados pelos interessados no projeto. Essa informação é compilada na forma de um mapa de produtos, conforme a **Tabela 2.2**.

Tabela 2.2: Exemplo de Mapa de Produto segundo Bayer *et al.* 1999

	Produto 1	Produto 2	Produto 3	Produto 4
Transmissão Manual		×		×
Transmissão Automática	×		×	
Motor	×	×	×	×
Ar Condicionado			×	×

A **Tabela 2.2** apresenta o mapa de produtos que descreve um conjunto de produtos referente ao modelo de features da **Figura 2.4**. Este especifica quatro produtos que podem ser gerados pela LPS e que estão de acordo com o modelo de *features*.

O mapa de produtos lista as funcionalidades disponíveis na primeira coluna da tabela e os produtos na primeira linha da tabela. Dessa forma, cada produto é associado a uma lista de funcionalidades de acordo com a especificação da linha de produto de software para que sejam criados os produtos da LPS.

O mapa de produtos, dessa maneira, complementa o modelo de *features* por meio da especificação de produtos, limitando o número de produtos que podem ser gerados pelo modelo de *features*.

2.4. Mapeamento Sistemático: Garantia de Qualidade em Modelos de LPS

O mapeamento sistemático da literatura consiste em um estudo secundário por meio do qual se pretende identificar e avaliar os estudos primários relevantes [Kitchenham e Charters, 2007]. Nesta subseção, será apresentado um mapeamento sistemático realizado relacionado à garantia de qualidade de modelos de Linha de Produto de Software.

O objetivo do mapeamento sistemático foi estruturado utilizando o paradigma GQM (*Goal-Question-Metric*) [Basili e Rombach, 1988] exposto na **Tabela 2.3**.

Tabela 2.3: Paradigma GQM para os objetivos do mapeamento sistemático

Analisar	publicações científicas por meio de um estudo secundário baseado em mapeamento sistemático
Com o propósito de	caracterizar técnicas de inspeção de modelos de Linha de Produto de Software
Com relação a	garantia da qualidade do produto da Linha de Produto de Software
Do ponto de vista dos	Pesquisadores
No Contexto	acadêmico e industrial com foco em inspeção de modelos de Linha de Produto de Software

Esse objetivo visa responder a questão de pesquisa principal, por meio da execução desta abordagem sistemática de revisão da literatura, especificada a seguir: *"Quais técnicas tem sido propostas e/ou utilizadas para inspeção de modelos de Linha de Produto de Software?"*. A questão de pesquisa secundária, que se busca responder por meio da execução desta abordagem sistemática de revisão da literatura, é especificada a seguir: *"Qual o contexto de aplicação das técnicas?"*

2.4.1. Escopo do Mapeamento Sistemático

O escopo do mapeamento sistemático visa definir os critérios estabelecidos para a realização da pesquisa de estudos primários. O mapeamento sistemático realizado nesse trabalho teve o seguinte escopo:

- Bibliotecas: a escolha das bibliotecas IEEE Xplore, Portal ACM e Scopus se deve ao fato de elas indexarem *journals* e anais de conferência mais relevantes em Engenharia de Software, pela acessibilidade da mesma pelo pesquisador e por não gerar custos ao mesmo. As fontes foram acessadas via Web por meio de expressões de busca baseada em palavras-chave conforme a **Tabela 2.4**.

- Idioma dos estudos primários: A escolha de estudos primários na língua inglesa se deu pela maioria dos artigos nas bibliotecas selecionadas serem escritos em inglês.

- Com o objetivo de auxiliar o desenvolvimento da pesquisa, foi utilizada a ferramenta Start (*State of Art Through Systematic Review*) [Hernandes *et al.*, 2010]. O auxílio dessa ferramenta permite ao pesquisador realizar a entrada de todas as informações referente a uma revisão sistemática e fornece um mecanismo para que o pesquisador execute a pesquisa, aceitando ou rejeitando os artigos resultantes da expressão de busca; e

- Palavras-chave: As palavras-chaves utilizadas nesta revisão foram divididas em duas categorias: linha de produto de software e inspeção de modelos de software, conforme a **Tabela 2.4**.

Tabela 2.4: Palavras-chave para pesquisa nas bibliotecas digitais

Categorias	Palavra Chave
Linha de Produto de Software	*Software Product Line*
	Product Line
	Feature Model
	SPL
Inspeção de Modelo de Software	*Inspection Technique*
	Model Inspection
	Inspection
	Software Inspection
	Verification

De acordo com os procedimentos descritos em Kitchenham e Charters (2007), foram definidos os critérios de inclusão e exclusão para os estudos primários avaliados na execução do mapeamento sistemático. Os critérios de inclusão utilizados na execução deste foram:

- Estudos primários que descrevem aplicação ou proposta de verificação de modelos de linha de produto de software

- Estudos primários que descrevem aplicação ou proposta de inspeção de modelos de linha de produto de software

Os critérios de exclusão considerados para seleção dos estudos primários neste mapeamento sistemático foram:

- Publicações que não abordem o tema de linha de produto de software;

- Publicações que não abordem o tema de verificação em modelos de linha de produto de software;

- Publicações que não abordem o tema de verificação em modelos de linha de produto de software e linha de produto software; e

- Publicações que abordem o tema de testes de modelos de linha de produto de software.

O critério de seleção para o primeiro filtro foi realizado por meio da análise do título e do *abstract* dos artigos proveniente da expressão de busca

estabelecida e que atenderam o proposto nos critérios de inclusão de estudos primários.

O segundo filtro foi realizado a partir dos estudos primários selecionados no primeiro filtro, pois, estes atenderam os critérios de inclusão previamente estabelecidos. Durante a análise para o segundo filtro foi realizada a leitura do título, *abstract* e partes da introdução e conclusão do artigo para verificar se os mesmos atendem os critérios de inclusão de estudos primários.

O critério de seleção final foi realizado a partir dos estudos primários que foram selecionados no segundo filtro. E, para estes, foi realizada a leitura completa do artigo com o objetivo de extrair os seguintes dados: Título, Autor(es), *Abstract*, Objetivo da Verificação, Modelos de Linha de Produto de Software, Artefatos Verificados, Forma de Verificação, Se Utiliza Ontologia, Se Utiliza Verificação Formal, Se Utiliza Programação Orientada à Aspectos e Se Utiliza Conceitos de *Crosscutting Concerns*.

Nos dados extraídos dos artigos, procurou-se verificar quais as principais abordagens de linhas de produto de software que estavam sendo verificadas por meio de métodos e automatizações. Dessa forma, os dados referentes à utilização de ontologia, verificação formal, programação orientada a aspectos e *crosscutting concerns*, foram extraídos com o objetivo analisar quais artefatos estavam sendo gerados em cada abordagem e quais as forma de verificação correspondente.

2.4.2. Estudos Primários Selecionados

A execução da pesquisa nas bibliotecas escolhidas, utilizando as expressões de busca baseadas nas palavras-chaves, resultou em um total de 841 estudos primários para análise dos critérios de primeiro e segundo filtros. Na **Tabela 2.5**, é discriminada a quantidade de estudos primários retornados em cada uma das bibliotecas, assim como, os estudos primários que foram aprovados.

Tabela 2.5: Estudos primários encontrados para cada filtro do mapeamento sistemático

Biblioteca Digital	Estudos Primários Retornados	Estudos Primários Aprovados no 1° filtro	Estudos Primários Aprovados no 2° filtro
IEEE Xplore	88	22	5
Portal ACM	482	28	10
Scopus	271	40	12

Os 27 artigos selecionados para a fase de extração estão listados na **Tabela 2.6**, incluindo o título do artigo, os autores e o ano de publicação.

Tabela 2.6: Artigos selecionados para extração pela execução do Mapeamento Sistemático

Título	Autores	Ano
Formal Verification of Consistency between Feature Model and Software Architecture in Software Product Line	Tonny Kurniadi Satyananda, Danhyung Lee, Sungwon Kang	2007
A Formal Approach to Verify Mapping Relation in a Software Product Line	Tonny Kurniadi Satyananda, Danhyung Lee, Sungwon Kang	2007
DECIMAL and PLFaultCAT: From Product-Line Requirements to Product Line Member Software Fault Trees	Josh Dehlinger, Meredith Humphrey, Lada Suvorov, Prasanna Padmanabhan, Robyn Lutz	2007
Verifying Architectural Variabilities in Software Fault Tolerance Techniques	Patrick H. S. Brito, Rogério de Lemos, Cecília M. F. Rubira	2009
Consistency Checking in an Infrastructure for Large-Scale Generative Programming	Axel Rauschmayer, Alexander Knapp, Martin Wirsing	2004
Applying Semantic Web Technology to Feature Modeling	Lamia Abo Zaid, Frederic Kleinermann, Olga De Troyer	2009
Semi-Automated Diagnosis of FODA Feature Diagram	Shin Nakajima	2010
Model Checking of Domain Artifacts in Product Line Engineering	Kim Lauenroth, Klaus Pohl, Simon Töhning	2009
Model Checking Lots of Systems – Efficient Verification of Temporal Properties in Software Product Lines	Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, Jean-François Raskin	2010
Resolving Feature Dependency Implementations Inconsistencies during Product Derivation	Saad Bin Abid	2010
Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints	Krzysztof Czarnecki, Krzysztof Pietroszek	2006
Flexible and Scalable Consistency Checking on Product Line Variability Models	Michael Vierhauser, Paul Grünbacher, Alexander Egyed, Rick Rabiser, Wolfgang Heider	2010
Configuration Lifting: Verification meets Software Configuration	Hendrik Post, Carsten Sinz	2008
Verify Feature Models using Protége ´-OWL	Hai Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang	2005

Towards Automated Consistency Checks of Product Line Requirements Specifications	Kim Lauenroth, Klaus Pohl	2007
Verification of Software Product Lines with Delta-oriented Slicing	Daniel Bruns, Vladimir Klebanov, Ina Schaefer	2010
Developing a Software Product Line for Train Control: A Case Study of CVL	Andreas Svendsen, Xiaorui Zhang, Roy Lind-Tviberg, Franck Fleurey, Øystein Haugen, Birger Møller-Pedersen, Gøran K. Olsen	2010
Verifying Architectural Variabilities in Software Fault Tolerance Techniques	Patrick H. S. Brito, Rogério Lemos, Cecília M. F. Rubra.	2009
Modes in component behavior specification via EBP and their application in product lines	Jan Kofroña, František Plášil, Ondrej Šery	2009
Rigorous engineering of product-line requirements: A case study in failure management	Colin Snook, Michael Poppleton, Ian Johnson	2008
Verifying feature models using OWL	Hai H. Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, Jeff Pan	2007
Formal Verification and Software Product Lines	Tomoji Kishi and Natsuko Noda	2006
Formal Semantics and Verification for Feature Modeling	Jing Sun, Hongyu Zhang, Yuan Fang Li, Hai Wang	2005
Tool-Supported Verification of Product Line Requirements	Prasanna Padmanabhan, Robyn R. Lutz	2005
Modular Verification of Open Feature Using Three-Valued Model Checking	Harry C. Li, Shriram Krishnamurthi, Kathi Fisler.	2005
Parameterized Interfaces for Open System Verification of Product Lines	Colin Blundell, Kathi Fisler, Shriram Krishnamurthi, Pascal Van Hentenryck	2004
A Methodology for the Derivation and Verification of Use Cases for Product Lines	A. Fantechi, S. Gnesi, G. Lami, and E. Nesti	2004

A execução do mapeamento sistemático resultou em um total de 27 artigos sobre técnicas de garantia de qualidade para modelos de linhas de produto de software. Destes, é possível extrair dados de 23 (85%) dos artigos que utilizam técnicas de verificação de modelos automatizada, 3 (11%) que utilizam ontologias para modelagem e verificação de modelos automatizada e 2 (7%) que utilizam inspeção em código fonte.

A extração dos dados para os 27 estudos primários selecionados, está descrita a seguir:

- Título: *Formal Verification of Consistency between Feature Model and Software Architecture in Software Product Line*
    - Autor(es): Tonny Kurniadi Satyananda, Danhyung Lee, and Sungwon Kang

- Abstract: *During software development process, software artifacts are produced. Consistency among these artifacts should be verified to ensure error-free product. In software product line development, consistency becomes more important because commonalities and variability increase the complexity of relationship among artifacts. In this paper, we present a formal approach to verification of consistency between feature model and component and connector view of software architecture. By utilizing Prototype Verification System (PVS), we introduce our model of feature description and architecture description, and illustrate the consistency verification approach using a digital watch product line example.*
- Objetivo da Verificação: Verificação de consistência entre modelo de features e de arquitetura (mais especificamente consistência de variabilidade e consistência de dependência)
- Modelo de LPS: Modelo de features.
- Artefatos Verificados: Modelo de arquitetura especificado utilizando ACME.
- Forma de Verificação: Os documentos são especificados utilizando PVS (*Prototype Verification System*). É construído um modelo matemático descrevendo a verificação de consistência em forma de teoremas, para posteriormente, utilizar o PVS prover para verificação.
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *A Formal Approach to Verify Mapping Relation in a Software Product Line*

- Autor(es): Tonny Kurniadi Satyananda, Danhyung Lee, and Sungwon Kang
- Abstract: *In software product line development, consistency among artifacts is important because commonalities and variability increase the complexity of relations among artifacts. For small scale models, the relations among elements can be easily identified and tracked by manually analyzing the descriptions of models. But when the complexity of models is high, a more systematic approach is required for identifying traceability information and verifying consistency between models. In this paper, by utilizing Formal Concept Analysis (FCA) and Prototype Verification System (PVS), we present a formal approach for identifying traceability and verifying consistency between feature model and component and connector view of software architecture.*
- Objetivo da Verificação: Verificação de consistência entre modelo de funcionalidade e modelo de arquitetura

- Modelo de LPS: Modelo de features.
- Artefatos Verificados: Modelo de arquitetura especificado utilizando ADL.
- Forma de Verificação: Os documentos são especificados utilizando PVS (*Prototype Verification System*). É construído um modelo matemático descrevendo a verificação de consistência em forma de teoremas, para posteriormente, utilizar o PVS prover para verificação.
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

• Título: *DECIMAL and PLFaultCAT: From Product-Line Requirements to Product Line Member Software Fault Trees*

- Autor(es): Josh Dehlinger, Meredith Humphrey, Lada Suvorov, Prasanna Padmanabhan and Robyn Lutz
- Abstract: *PLFaultCAT is a tool for software fault tree analysis (SFTA) during product-line engineering. When linked with DECIMAL, a product-line requirements verification tool, the enhanced version of PLFaultCAT provides traceability between product line requirements and SFTA hazards as well as semi automated derivation of the SFTA for each new product-line system previously verified by DECIMAL. The combined tool reduces the effort needed to safely reuse requirements and customize the product-line SFTA as each new system is constructed.*
- Objetivo da Verificação: Verificação de consistência entre modelo de funcionalidade e modelo de arquitetura
- Modelo de LPS: O estudo proposto no artigo utiliza como editor gráfico para construir a árvore de falhas de linha de produto.
- Artefatos Verificados: DECIMAL para documentar e registrar requisitos de linha de produto.
- Forma de Verificação: Verificação automática de requisitos de linha de produto de software.
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Não
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

• Título: *Verifying Architectural Variabilities in Software Fault Tolerance Techniques*

- Autor(es): Patrick H. S. Brito, Rogério de Lemos, Cecília M. F. Rubira
- Abstract: *This paper considers the representation of different software fault tolerance techniques as a product line architecture (PLA) for promoting the reuse of software artifact.*

The proposed PLA enables to specify a series of closely related architectural applications, which is obtained by identifying variation points associated with design decisions regarding software fault tolerance. These decisions are used to choose the appropriate technique depending on the features selected, e.g, the number of redundant resources, or the type of adjudicator. The proposed approach also comprises the formalization of the PLA, using B-Method and CSP, for systematizing the verification of fault-tolerant software systems at the architectural level. The properties verified cover two complementary contexts: the selection of the correct architectural variability for instantiating the PLA, and also the properties of the chosen fault tolerance techniques.

- Objetivo da Verificação: Verificação da consistência do modelo com a configuração de arquitetura, consistência entre variantes de funcionalidade e os pontos de variações da arquitetura de linha de produto e consistência entre a especificação comportamental do produto e o comportamento da arquitetura de referência.
- Modelo de LPS: O artigo proposto utiliza o modelo de features para a modelagem da arquitetura da linha de produto. A especificação estrutural é mapeada utilizando B-Method Machines enquanto a especificação da arquitetura de software é mapeada utilizando CSP (*Communicating Sequential Process*).
- Artefatos Verificados: Modelo de features, especificação estrutural e a especificação de arquitetura de software.
- Forma de Verificação: Verificação da arquitetura da linha de produto de software incluindo os aspectos específicos de um software tolerante a falha.
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Consistency Checking in an Infrastructure for Large-Scale Generative Programming*

  - Autor(es): Axel Rauschmayer, Alexander Knapp, Martin Wirsing
  - Abstract: *Ubiquitous computing increases the pressure on the software industry to produce ever more and error-free code. Two recipes from automated programming are available to meet this challenge: On the one hand, generative programming raises the level of abstraction in software development by describing problems in high-level domain-specific languages and making them executable. On the other hand, in situations where one needs to produce a family of similar programs, product line engineering supports code reuse by composing programs from a set of common assets (or features). AHEAD (Algebraic Hierarchical Equations for Application Design) is a*

framework for generative programming and product line engineering that achieves additional productivity gains by scaling feature composition up. Our contribution is GRAFT, a calculus that gives a formal foundation to AHEAD and provides several mechanisms for making sure that feature combinations are legal and that features in themselves are consistent.

- Objetivo da Verificação: Verificação de consistência de dependência, semântica e estrutural
- Modelos de LPS: O estudo proposto no artigo utiliza AHEAD (*Algebraic Hierarchical Equations for Application Design*) que é um framework é um framework de programação generativa e engenharia de linha de produto. E propõe o GRAFT, uma linguagem formal para analisar as funcionalidades e decompor as funcionalidades descritas em AHEAD.
- Artefatos Verificados: Verifica as funcionalidades descritas no AHEAD, o que inclui informações de artefatos como linguagem de programação e linguagens de domínio específico.
- Forma de Verificação: Realiza um conjunto de verificação consistência como "uma feature adiciona uma funcionalidade que já existia?" e "As dependências existentes estão compridas?"
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Sim
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Applying Semantic Web Technology to Feature Modeling*

  - Autor(es): Lamia Abo Zaid, Frederic Kleinermann, Olga De Troyer
  - Abstract: *Feature models are models used to capture differences and commonalities. Between software features, enabling the representation of variability within software. There are many variations of feature models and different notations are often used to represent the same information. Currently support for validating or integrating feature models is missing. In this paper, we provide an ontology framework for feature modeling which consists of an ontology that formally provides a specification for feature models. In addition, we provide means to integrate segmented feature models and provide a rule based model consistency check and conflict detection. We use SWRL rules to implement the rules and a DL reasoner to evaluate the rules and infer extra interesting information regarding the variability of the software.*
  - Objetivo da Verificação: Validação de modelos

- Modelos de LPS: O artigo proposto utiliza OWL (*web ontology language*) como linguagem de descrição para modelos de funcionalidades (F.O.D.A., FORM, FeatuRSEB) e utiliza *Semantic Web Rule Language* (SWRL) para criação regras de implementação.
- Artefatos Verificados: Modelo de features
- Forma de Verificação: Através da implementação de regras utilizando *Semantic Web Rule Language* (SWRL)
- Utiliza Ontologia: Sim
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

• Título: *Semi-Automated Diagnosis of FODA Feature Diagram*

- Autor(es): Shin Nakajima
- Abstract: *A semi-automated model diagnostic method is proposed for FODA feature diagram, a primary modeling notation used in Software Product Line Engineering. The proposed method includes a propositional logic interpretation of the feature diagram and a diagram-slicing algorithm for locating bugs. In addition to logic-based formalization of the semantics, the novelty of our approach is that it uses heuristics taking into account the diagram graph structure. Although human intelligence is always involved in removing bugs from feature diagrams, the checking and diagnosing of them can be automated to some extent.*
- Objetivo da Verificação: Verificação de consistência de modelos
- Modelos de LPS: O estudo proposto no artigo utiliza *Feature Oriented Domain Analysis* (FODA) para modelo de funcionalidade. Este, então, é formalizado utilizando lógica proposicional e, posteriormente, utiliza-se *Slicing Diagram Algorithm* para identificação de defeitos.
- Artefatos Verificados: Modelo de features
- Forma de Verificação: O método inclui uma interpretação em lógica proposicional e utiliza *Slicing Diagram Algorithm* para identificação de defeitos.
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

• Título: *Model Checking of Domain Artifacts in Product Line Engineering*

- Autor(es): Kim Lauenroth, Klaus Pohl, Simon Töhning
- Abstract: *In product line engineering individual products are derived from the domain artifacts of the product line. The reuse*

of the domain artifacts is constraint by the product line variability. Since domain artifacts are reused in several products, product line engineering benefits from the verification of domain artifacts. For verifying development artifacts, model checking is a well-established technique in single system development. However, existing model checking approaches do not incorporate the product line variability and are hence of limited use for verifying domain artifacts. In this paper we present an extended model checking approach which takes the product line variability into account when verifying domain artifacts. Our approach is thus able to verify that every permissible product (specified with I/Oautomata) which can be derived from the product line fulfills the specified properties (specified with CTL). Moreover, we use two examples to validate the applicability of our approach and report on the preliminary validation results.

- Objetivo da Verificação: Verificação se todos os possíveis produtos derivados de um artefato de domínio estão de acordo com a especificação.
- Modelos de LPS: Utiliza uma linguagem de modelo de variabilidade ortogonal desenvolvido pelo próprio grupo de pesquisa e para especificação de artefatos de domínio são utilizado I/O-Automata e *Computational Tree Logic* (CTL) .
- Artefatos Verificados: Artefatos de domínio de uma linha de produto de software
- Forma de Verificação: Verifica se cada I/O-Automata derivado satisfaz as propriedades descritas utilizando CTL
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Model Checking Lots of Systems - Efficient Verification of Temporal Properties in Software Product Lines*

  - Autor(es): Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, Jean-François Raskin
  - Abstract: *In product line engineering, systems are developed in families and differences between family members are expressed in terms of features. Formal modeling and verification is an important issue in this context as more and more critical systems are developed this way. Since the number of systems in a family can be exponential in the number of features, two major challenges are the scalable modeling and the efficient verification of system behavior. Currently, the few attempts to address them fail to recognize the importance of features as a unit of difference, or do not offer means for automated verification. In this paper, we tackle those challenges at a fundamental level. We first extend transition systems with*

features in order to describe the combined behavior of an entire system family. We then define and implement a model checking technique that allows verifying such transition systems against temporal p roperties. An empirical evaluation shows substantial gains over classical approaches.

- Objetivo da Verificação: Verificação das propriedades dos produtos de uma Linha de Produto de Software e identificar os produtos que violam essas propriedades
- Modelos de LPS: O artigo contribui/utiliza *Feature Transition System* (FTS), uma variante de sistemas de transição para descrever o comportamento de todo um sistema de família. Um implementação do modelo de verificação do modelo foi implementada utilizando Haskell.
- Artefatos Verificados: Modelo de features
- Forma de Verificação: Verifica propriedades regulares para (a) identificar se uma propriedade é satisfeita pelo FTS, que esse é um produto válido da linha de produto de software e (b) se uma propriedade é violada, o algoritmo, este sugere um contra exemplo como um produto que viola uma propriedade da linha de produto de software.
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

• Título: *Resolving Feature Dependency Implementations Inconsistencies during Product Derivation*

- Autor(es): Saad Bin Abid
- Abstract: *Features implementing the functionality in a software product line (SPL) often interact and depend on each other. It is hard to maintain the consistency between feature dependencies on the model level and the actual implementation over time, resulting in inconsistency during product derivation. We describe our initial results when working with feature dependency implementations and the related inconsistencies in actual code. Our aim is to improve consistency checking during product derivation. We have provided tool support for maintaining consistency between feature dependency implementations on both model and code levels in a product line. The tool chain supports the consistency checking on both the domain engineering and the application levels between actual code and models. We report our experience of managing feature dependency consistency in the context of an existing scientific calculator product line.*
- Objetivo da Verificação: Verificação de dependência da abordagem baseada em aspectos e dependência modelo-código.
- Modelos de LPS: O estudo proposto no artigo utiliza um plug-in para realizar uma engenharia reversa e gerar o modelos de implementação (modelo AML).

- Artefatos Verificados: Diagrama de *features*, modelo de implementação.
- Forma de Verificação: Utiliza-se EVL (*Epsilon framework Validation Language*) para verificação do modelo em relação à completude e consistência de implementação de dependência
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Sim
- Utiliza Conceitos de *Crosscutting Concerns*: Sim

- Título: *Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints*

  - Autor(es): Krzysztof Czarnecki, Krzysztof Pietroszek
  - Abstract: Feature-based model templates have been recently proposed as an approach for modeling software product lines. Unfortunately, templates are notoriously prone to errors that may go unnoticed for long time. This is because such an error is usually exhibited for some configurations only, and testing all configurations is typically not feasible in practice. In this paper, we present an automated verification procedure for ensuring that no ill-structured template in- stance will be generated from a correct configuration. We present the formal underpinnings of our proposed approach, analyze its complexity, and demonstrate its practical feasibility through a prototype implementation.
  - Objetivo da Verificação: Verificação de boa formação de instâncias resultantes
  - Modelos de LPS: O artigo proposto utiliza Object Constraint Language (OCL) para descrever as regras de boa formação e utiliza Meta Object Facility (MOF) para descrição do meta modelo. E utiliza um SAT Solver para verificação.
  - Artefatos Verificados: Modelo de features e diagrama de classes anotados
  - Forma de Verificação: Utiliza um SAT solver para verificação
  - Utiliza Ontologia: Não
  - Utiliza Verificação Formal: Sim
  - Utiliza Programação Orientada a aspectos: Não
  - Utiliza Conceitos de Crosscutting Concerns: Não

- Título: *Flexible and Scalable Consistency Checking on Product Line Variability Models*

  - Autor(es): Michael Vierhauser, Paul Grünbacher, Alexander Egyed, Rick Rabiser, Wolfgang Heider
  - Abstract: *The complexity of product line variability models makes it hard to maintain their consistency over time regardless of the modeling approach used. Engineers thus need support for detecting and resolving inconsistencies. We describe experiences of applying a tool-supported approach for*

incremental consistency checking on variability models. Our approach significantly improves the over- all performance and scalability compared to batch-oriented techniques and allows providing immediate feedback to modelers. It is extensible as new consistency constraints can easily be added. Furthermore, the approach is flexible as it is not limited to variability models and it also checks the consistency of the models with the underlying code base of the product line. We report the results of a thorough evaluation based on real-world product line models and discuss lessons learned.

- Objetivo da Verificação: Verificação de consistência em todos os níveis, desde o modelo até o código
- Modelos de LPS: O estudo proposto no artigo é independente do modelo utilizado. No caso do experimento do artigo, foram utilizado modelos em DOPLER que foram mapeados para XML Spring. Sendo realizada a verificação de consistência incremental em meta modelos DOPPLER.
- Artefatos Verificados: Modelos de Linha de Produto e de implementação independente do tipo.
- Forma de Verificação: Verificação automática de consistência do modelo
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

• Título: *Configuration Lifting: Verification meets Software Configuration*

- Autor(es): Hendrik Post, Carsten Sinz
- Abstract: *Configurable software is ubiquitous, and the term Software Product Line (SPL) has been coined for it lately. It remains a challenge, however, how such software can be verified over all variants. Enumerating all variants and analyzing them individually is inefficient, as knowledge cannot be shared between analysis runs. Instead of enumeration we present a new technique called lifting that converts all variants into a meta-program, and thus facilitates the configuration-aware application of verification techniques like static analysis, model checking and deduction-based approaches. As a side effect, lifting provides a technique for checking software feature models, which describe software variants, for consistency. We demonstrate the feasibility of our approach by checking configuration dependent hazards for the highly configurable Linux kernel which possesses several thousand of configurable features. Using our techniques, two novel bugs in the kernel configuration system were found.*
- Objetivo da Verificação: Verificação de inconsistência durante o processo de construção/empacotamento de software

- Modelos de LPS: Não utiliza modelos porém utiliza lógica proposicional para verificação linha de produtos de software.
- Artefatos Verificados: Lógica proposicional de um modelo de features, código fonte configurável e uma ferramenta de build (make file, etc).
- Forma de Verificação: Verificação em tempo de execução durante a construção/empacotamento de software.
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Verify Feature Models using Protége-OWL*

  - Autor(es): Hai Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang
  - Abstract: *Feature models are widely used in domain engineering to capture common and variant features among systems in a particular domain. However, the lack of a widely adopted means of precisely representing and formally verifying feature models has hindered the development of this area. This paper presents an approach to modeling and verifying feature diagrams using Semantic Web ontologies*.
  - Objetivo da Verificação: Verificação do modelo de funcionalidade utilizando ontologia (OWL)
  - Modelos de LPS: O artigo proposto utiliza OWL (web ontology language) como linguagem de descrição para modelos de funcionalidades.
  - Artefatos Verificados: O diagrama de features descrito através de ontologia (OWL)
  - Forma de Verificação: Verificação automática do modelo descrito utilizando OWL
  - Utiliza Ontologia: Sim
  - Utiliza Verificação Formal: Não
  - Utiliza Programação Orientada a aspectos: Não
  - Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Towards Automated Consistency Checks of Product Line Requirements Specifications*

  - Autor(es): Kim Lauenroth, Klaus Pohl
  - Abstract: *A requirements specification for an individual software system should be consistent, i.e. free of contradictions. In product line engineering, the product line requirements specification comprises all the requirements common to all products of the product line as well as the variable requirements used to derive individual products from the product line. The set of requirements (common and all the variable ones) of a product line is typically inconsistent since variable requirements can contradict each other. This is not a problem as long as contradicting requirements are not*

included in a product derived from the product line. Thus, the set of requirements realized in each individual product has to be consistent. Employing techniques used in single system development to check the consistency of product line requirements will thus pro- duce false positive results, since there can be contradiction in the product line requirements specification. In this paper we first provide a concise definition of consistency for product line requirements specifications. Based on this definition, we define a formal framework for checking consistency of product line requirements specifications. Our framework supports consistency checks in the domain engineering process. In contrast to consistency checks in single system development, it tolerates certain types of inconsistencies caused by the variability of product line requirements.

- Objetivo da Verificação: Verificação de inconsistência entre os requisitos da linha de produto de software
- Modelos de LPS: O artigo proposto é independente de modelos, podendo este ser de funcionalidades ou ortogonal. Utiliza um framework formal para verificação de consistência que é composto de uma definição formal de variabilidade, uma definição formal das propriedades que uma linguagem de requisitos deve conter para suportar verificação, uma definição formal de engenharia de linha de produto e um algoritmo para detectar possíveis contradições.
- Artefatos Verificados: Verificação de modelos que devem ser descritos em uma notação formal definida pelo framework
- Forma de Verificação: Verificação formal de inconsistência entre os requisitos da linha de produto de software
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Verification of Software Product Lines with Delta-oriented Slicing*

  - Autor(es): Daniel Bruns, Vladimir Klebanov and Ina Schaefer
  - Abstract: *Software product line (SPL) engineering is a well-known approach to develop industry-size adaptable software systems. SPL are often used in domains where high-quality software is desirable; the over- whelming product diversity, however, remains a challenge for assuring correctness. In this paper, we present delta-oriented slicing, an approach to reduce the deductive verification effort across an SPL where individual products are Java programs and their relations are described by deltas. On the specification side, we extend the delta language to deal with for- mal specifications. On the verification side, we combine proof slicing and similarity-guided proof reuse to ease the verification process.*

- Objetivo da Verificação: A verificação do resultado do reuso obtidos através da verificação de um produto considerando o que já foi verificado em outro. Utiliza uma DSL para uma especificação formal da linha de produto de software associada a um SMT solver para verificação de condição de geração. Utiliza JML (*Java Modeling Language*) para a especificação formal das propriedades dos produtos.
- Modelos de LPS: O autor utiliza uma abordagem de modelo especificado em trabalhos anteriores chamado *delta-oriented programming.*
- Artefatos Verificados: Código Fonte
- Forma de Verificação: Verificação formal através das propriedades descritas em JML
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Developing a Software Product Line for Train Control: A Case Study of CVL*

  - Autor(es): Andreas Svendsen, Xiaorui Zhang, Roy Lind-Tviberg, Franck Fleurey, Øystein Haugen, Birger Møller-Pedersen, Gøran K. Olsen
  - Abstract: *This paper presents a case study of creating a software product line for the train signaling domain. The Train Control Language (TCL) is a DSL which automates the production of source code for computers controlling train stations. By applying the Common Variability Language (CVL), which is a separate and generic language to define variability on base models, we form a software product line of stations. We discuss the process and experience of using CVL to automate the production of three real train stations. A brief discussion about the verification needed for the generated products is also included.*
  - Objetivo da Verificação: A verificação acontece de duas formas, sendo, a primeira delas referente se todos os produtos gerados pelos modelos estão de acordo com o meta modelo definido. E uma segunda verificação, juntamente com uma validação, acontece no código fonte gerado que é produzido pela CVL (*Common Variability Language*)
  - Modelos de LPS: Utiliza uma DSL chamada de TCL (*Train Control Language*) para descrever estações de trens e CVL (*Common Variability Language*) sobre a TCL para especificar as variâncias entre os modelos dos vários produtos da linha de produto de software.
  - Artefatos Verificados: O modelo descrito utilizando TCL e o modelo de variabilidade

- Forma de Verificação: Verificação se os produtos gerados automaticamente estão de conforme com os produtos da linha de produto de software
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Não
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Verifying Architectural Variabilities in Software Fault Tolerance Techniques*

  - Autor(es): Patrick H. S. Brito, Rogério Lemos, Cecília M. F. Rubra.
  - Abstract: *This technical report considers the representation of different software fault tolerance techniques as a product line architecture (PLA) for promoting the reuse of software artefacts, such as formal specifications and verification. The proposed PLA enables to specify a series of closely related applications in terms of a single architecture, which is obtained by identifying variation points associated with design decisions regarding software fault tolerance. These decisions are used to choose the appropriate technique depending on the features selected for the instance, e.g, the number of redundant resources, or the type of adjudicator. The proposed approach also comprises the formalisation of the PLA, using B-Method and CSP, for systematising the verification of fault-tolerant software systems at the architectural level. The properties verified cover two complementary contexts: the selection of the correct architectural variabilities for instantiating the PLA, and also the properties of the chosen fault tolerance techniques*.
  - Objetivo da Verificação: Utiliza CSP e B-Method para sistematização e verificação de software tolerantes à falhas utilizando PLA (*Product Line Architecture*). CSP é utilizado para especificação do comportamento e B-Method para a estrutura da PLA.
  - Modelos de LPS: modelo de *features*
  - Artefatos Verificados: A especificação em B-Method da arquitetura da linha de produto de software
  - Forma de Verificação: Verificação formal da especificação de uma linha de produto de software com técnicas de tolerância a falhas.
  - Utiliza Ontologia: Não
  - Utiliza Verificação Formal: Sim
  - Utiliza Programação Orientada a aspectos: Não
  - Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Modes in component behavior specification via EBP and their application in product lines*

- Autor(es): Jan Kofroňa, František Plášil e Ondrej Šery
- Abstract: *The concept of software product lines (SPL) is a modern approach to software development simplifying construction of related variants of a product thus lowering development costs and shortening time-to-market. In SPL, software components play an important role. In this paper, we show how the original idea of component mode can be captured and further developed in behavior specification via the formalism of Extended Behavior Protocols (EBP). Moreover, we demonstrate how the modes in behavior specification can be used for modeling behavior of an entire product line. The main benefits include (i) the existence of a single behavior specification capturing the behavior of all product variants, and (ii) automatic verification of absence of communication errors among the cooperating components taking the variability into account. These benefits are demonstrated on a part of a non-trivial case study.*
- Objetivo da Verificação: Verifica a exatidão de componentes de verificação em variantes de arquitetura
- Modelos de LPS: Utiliza um protocolo desenvolvido pelo grupo de estudo autor do artigo chamado de *Extended Behavior Protocols* (EBP). E, depois disso, utiliza uma ferramenta de verificação de modelo.
- Artefatos Verificados: EBP (utilizado para especificar o comportamento associado a um nó especifico de um componente)
- Forma de Verificação: Verificação automática do modelo EBP
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Rigorous engineering of product-line requirements: A case study in failure management*

- Autor(es): Colin Snook, Michael Poppleton, Ian Johnson
- Abstract: *We consider the failure detection and management function for engine control systems as an application domain where product line engineering is indicated. The need to develop a generic requirement set – for subsequent system instantiation – is complicated by the addition of the high levels of verification demanded by this safety-critical domain, subject to avionics industry standards. We present our case study experience in this area as a candidate method for the engineering, validation and verification of generic requirements using domain engineering and Formal Methods techniques and tools. For a defined class of systems, the case study produces a generic require- ment set in UML and an example system instance. Domain analysis and engineering produce a validated model which is integrated with the formal*

specification/verification method B by the use of our UML-B profile. The formal verification both of the generic requirement set, and of a simple system instance, is demonstrated using our U2B, ProB and prototype Requirements Manager tools. This work is a demonstrator for a tool-supported method which will be an output of EU project RODIN (This work is conducted in the setting of the EU funded Research Project: IST 511599 RODIN (Rigorous Open Development Environment for Complex Systems) http://rodin.cs.ncl.ac.uk/). The use of existing and prototype formal verification and support tools is discussed. The method, developed in application to this novel combination of product line, failure management and safety-critical engineering, is evaluated and considered to be applicable to a wide range of domains.

- Objetivo da Verificação: Verifica na instanciação das classes se os dados satisfazem as dependências dos diagramas de classe.
- Modelos de LPS: Utiliza UML juntamente com notação formal textual. Uma ferramenta transforma o modelo UML e a notação textual em um modelo unicamente textual, possibilitando a utilização de ferramentas de verificação formal
- Artefatos Verificados: Diagrama UML
- Forma de Verificação: Verificação formal de notação formal.
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Verifying feature models using OWL*

  - Autor(es): Hai H. Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, Jeff Pan
  - Abstract: *Feature models are widely used in domain engineering to capture common and variant features among systems in a particular domain. However, the lack of a formal semantics and reasoning support of feature models has hindered the development of this area. Industrial experiences also show that methods and tools that can support feature model analysis are badly appreciated. Such reasoning tool should be fully automated and efficient. At the same time, the reasoning tool should scale up well since it may need to handle hundreds or even thousands of features a that modern software systems may have. This paper presents an approach to modeling and verifying feature diagrams using Semantic Web OWL ontologies. We use OWL DL ontologies to precisely capture the inter-relationships among the features in a feature diagram. OWL reasoning engines such as FaCT++ are deployed to check for the inconsistencies of feature configurations fully automatically. Furthermore, a general OWL debugger has been developed to tackle the disadvantage of lacking debugging aids for the current OWL reasoner and to*

complement our verification approach. We also developed a CASE tool to facilitate visual development, interchange and reasoning of feature diagrams in the Semantic Web environment.

- Objetivo da Verificação: Detectar possíveis inconsistências em configurações de features.
- Modelos de LPS: Utiliza Semantic Web OWL Ontologies que são criadas a partir de modelos de features e utiliza FaCT++ para realizar automatizar a análise da representação do modelo de features
- Artefatos Verificados: O modelo de features descrito utilizando OWL
- Forma de Verificação: Verificação automática utilizando FaCT++ e ferramenta de debug OWL para indicação de erros
- Utiliza Ontologia: Sim
- Utiliza Verificação Formal: Não
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de Crosscutting Concerns: Não

- Título: *Formal Verification and Software Product Lines*

  - Autor(es): Tomoji Kishi and Natsuko Noda
  - Abstract: *Advances in embedded computing technologies have made society extremely dependent on embedded software used in automobiles, mobile phones, home electronics, and for many other applications. Consequently, the reliability of embedded software is crucial for many aspects of daily life. In the past, the development of embedded software has been implementation-centric; however, due to an increase in size and complexity of soft- ware and a reduction in development time, it is difficult to produce reliable software using conventional techniques. Therefore, the quality of embedded software becomes a matter of concern. To solve this problem, various software engineering techniques, such as analysis/design methods and reuse technologies, have been introduced. Product line engineering is one of the most advanced software practices based on these results*
  - Objetivo da Verificação: Verificação do modelo de UML especificação em linguagem formal para linha de produtos de software.
  - Modelos de LPS: É utilizado UML para especificação da linha de produto de software.
  - Artefatos Verificados: Modelos em UML que são descritos utilizando notação formal
  - Forma de Verificação: Verificação formal de modelos
  - Utiliza Ontologia: Não
  - Utiliza Verificação Formal: Sim
  - Utiliza Programação Orientada a aspectos: Não
  - Utiliza Conceitos de Crosscutting Concerns: Não

- Título: *Formal Semantics and Verification for Feature Modeling*

  - Autor(es): Jing Sun, Hongyu Zhang, Yuan Fang Li, Hai Wang
  - Abstract: *Research on features has received much attention in the domain engineering community. Feature modeling plays an important role in the design and implementation of complex software systems. However, the presentation and analysis of feature models are still largely informal. There is also an increasing need for methods and tools that can support automated feature model analysis. This paper presents a formal engineering approach to the specification and verification of feature models. A formal semantics for the feature modeling language is defined using first-order logic. It pro- vides a precise and rigorous formal interpretation for the graphical notation. In addition, further validation of the semantics using the Z/EVES theorem prover is presented. Finally, we demonstrate that the consistency of a feature model and its configurations can be automatically verified by encoding the semantics into the Alloy Analyzer. A case study of the Key Word in Context (KWIC) index systems feature model is presented to illustrate the verification process.*
  - Objetivo da Verificação: Validação da consistência do modelo de features e suas configurações através da semântica do modelo utilizando Z/EVES *theorem prover.*
  - Modelos de LPS: Utiliza lógica de primeira ordem para especificar a semântica formal dos modelos de features.
  - Artefatos Verificados: O modelo de features descrito utilizando a notação forma Z
  - Forma de Verificação: Verificação formal utilizando definições formais para cinco tipo de features e dois tipos de relação entre as features
  - Utiliza Ontologia: Não
  - Utiliza Verificação Formal: Sim
  - Utiliza Programação Orientada a aspectos: Não
  - Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Tool-Supported Verification of Product Line Requirements*

  - Autor(es): Prasanna Padmanabhan, Robyn R. Lutz
  - Abstract: *A recurring difficulty for organizations that employ a product-line approach to development is that when a new product is added to an existing product line, there is currently no automated way to verify the completeness and consistency of the new product's requirements in terms of the product line. In this paper we address the issue of requirements verification for product lines. We have implemented our approach in a requirements engineering tool called DECIMAL (DECIsion Modeling AppLication). DECIMAL is a requirements verification tool with a rich graphical user interface that automatically checks for completeness and consistency between*

a new product and the product line to which it belongs. The verification uses an SQL database server as the underlying analysis engine. The paper describes the tool and evaluates it in two applications: a virtual-reality, positional device-driver product line and the feature-interaction resolution problem.

- Objetivo da Verificação: Verificação de completude e consistência de novo produto com a linha de produto de software a qual pertence.
- Modelos de LPS: Utiliza uma ferramenta gráfica chamada DECIMAL (DECIsion Modeling AppLication) para especificação do requisitos que verifica automaticamente a completude e consistência da linha de produto de software.
- Artefatos Verificados: A especificação da linha de produto de software utilizando a ferramenta DECIMAL.
- Forma de Verificação: Verificação automática de completude, consistência, intervalo e tipo entre a especificação da linha de produto software e os produtos da linha de produto de software.
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Não
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Modular Verification of Open Feature Using Three-Valued Model Checking*

  - Autor(es): Harry C. Li, Shriram Krishnamurthi, Kathi Fisler.
  - Abstract: *Feature-oriented programming organizes programs around features rather than objects, thus better supporting extensible, product-line architectures. Programming languages increasingly support this style of pro- gramming, but programmers get little support from verification tools. Ideally, programmers should be able to verify features independently of each other and use automated compositional reasoning techniques to infer properties of a system from properties of its features. Achieving this requires carefully designed interfaces: they must hold sufficient information to enable compositional verification, yet tools should be able to generate this information automatically because experience indicates programmers cannot or will not provide it manually. We present a model of interfaces that supports automated, compositional, feature-oriented model checking. To demonstrate their utility, we automatically detect the feature-interaction problems originally found manually by Robert Hall in an email suite case study.*
  - Objetivo da Verificação: Verificação de propriedade comportamentais dos módulos orientados a features da linha de produto de software.
  - Modelos de LPS: Utiliza CTL para especificar a variabilidade. Descreve cada um dos modelos orientados a features como uma

máquina de estado com suporte para a verificação de modelo usando 3 valores propostos.
  - Artefatos Verificados: Modelos orientados a features descritos como uma máquina de estado.
  - Forma de Verificação: Verificação formal dos modelos
  - Utiliza Ontologia: Não
  - Utiliza Verificação Formal: Sim
  - Utiliza Programação Orientada a aspectos: Não
  - Utiliza Conceitos de *Crosscutting Concerns*: Não

- Título: *Parameterized Interfaces for Open System Verification of Product Lines*

  - Autor(es): Colin Blundell, Kathi Fisler, Shriram Krishnamurthi, Pascal Van Hentenryck
  - Abstract: Software product-lines view systems as compositions of features. Each component corresponds to an individual feature, and a composition of features yields a product. Feature-oriented verification must be able to analyze individual features and to compose the results into results on products. Since features interact through shared data, verifying individual features entails open system verification concerns. To verify temporal properties, features must be open to both propositional and temporal information from the remainder of the composed product. This paper addresses both forms of openness through a two-phase technique. The first phase analyzes individual features and generates sufficient constraints for property preservation. The second phase discharges the constraints upon composition of features into a product. We present the technique as well as the results of a case study on an email protocol suite.
  - Objetivo da Verificação: Verificação de consistência do modelo criado utilizando um verificador com uma performance similar ao CTL
  - Modelos de LPS: É baseado em modelos de features porém utiliza CTL para especificação da variabilidade e verificação de modelo através de verificação formal.
  - Artefatos Verificados: O modelo de features descrito em notação formal usando CTL
  - Forma de Verificação: Verificação formal em propriedades de variabilidade.
  - Utiliza Ontologia: Não
  - Utiliza Verificação Formal: Sim
  - Utiliza Programação Orientada a aspectos: Não
  - Utiliza Conceitos de *Crosscutting Concerns:* Não

- Título: *A Methodology for the Derivation and Verification of Use Cases for Product Lines*

  - Autor(es): A. Fantechi, S. Gnesi, G. Lami, and E. Nesti

- Abstract: In this paper, we present a methodology to express, in a formal way, the requirements of products belonging to a product line. We relied on a formal- ism allowing the representation of variabilities at the family level and the instantiation of them in order to move to the requirements of a single product. The proposed methodology also allows the formalization of the family constraints to be taken into account for the construction of the products belonging to it, along with the verification of the compliance to those constraints of a single product requirements document. This approach is promising due to its simplicity and effectiveness for being supported by automatic tools.
- Objetivo da Verificação: Verifica se o use case relacionado ao produto está em conformidade com a família de produtos
- Modelos de LPS: Utiliza uma notação elaborada, PLUC (Product Line Use Case), para adicionar a possibilidade de expressar regra em casos de uso relacionados a produto.
- Artefatos Verificados: Uma notação baseada em casos de uso, chamada PLUC, a qual adiciona variabilidade ao mesmo descrita em uma notaçãoo formal
- Forma de Verificação: Verificação automática dos modelos descritos em notação formal
- Utiliza Ontologia: Não
- Utiliza Verificação Formal: Sim
- Utiliza Programação Orientada a aspectos: Não
- Utiliza Conceitos de Crosscutting Concerns: Não

De acordo com o resultado do mapeamento sistemático, é possível ressaltar que apesar dos benefícios de técnicas de inspeção de modelos [Travassos *et al.*, 1999], não foi identificada nenhuma técnica de inspeção específica para modelos de linha de produto de software.

Inspeções de software visam garantir que um determinado artefato de software é completo, consistente, não ambíguo e contém o menor número de defeitos possíveis para efetivamente suportar o desenvolvimento de software. Técnicas para assegurar a qualidade como testes e inspeções são eficientes e efetivas para detectar e remover defeitos [Geppert *et al.*, 2005].

Segundo Denger e Kolb (2006), pelo fato de modelos de linhas de produto de software serem diferentes de modelos de desenvolvimento de um sistema único, técnicas padrões, utilizadas no desenvolvimento de software

tradicional, são insuficientes para verificar as características específicas de sistemas de reusabilidade.

Devido a não identificação de uma técnica de inspeção para modelos de linha de produto de software, foi proposta neste trabalho a SPLIT (*Software Product Line Inspection Technique*), um conjunto de técnicas de inspeção para linhas de produto de software que visa verificar o modelo de *features* e o mapa de produto, com o objetivo de identificar defeitos em estágios iniciais do desenvolvimento.

CAPÍTULO 3. SOFTWARE PRODUCT LINE INSPECTION TECHNIQUE – SPLIT

Uma linha de produto de software produz uma série de produtos que compartilham *features* comuns e *features* que se diferenciam de produto para produto. Sendo assim, é importante assegurar a qualidade de seus modelos, pois estes especificam vários produtos.

Segundo Denger e Kolb (2006), é necessária uma técnica de inspeção específica para linhas de produto de software, motivo pelo qual é proposto nesse capítulo um conjunto de técnicas de inspeção para linhas de produto de software (SPLIT – *Software Product Line Inspection Technique*). O seu objetivo verificar os modelos de *features* e mapa de produtos em comparação entre si e entre a especificação de requisitos de uma linha de produto de software. O documento de especificação de requisitos do software, neste caso, precisa enumerar os requisitos funcionais e não funcionais, os produtos criados pela linha de produto de software e suas restrições.

A SPLIT visa atender a necessidade técnicas especificas para Linhas de Produto de Software devido à técnicas de inspeção de modelos tradicionais de software não serem suficientes para identificar defeitos que são específicos de sistemas de reuso.

O conjunto de técnicas SPLIT é composto de três técnicas. Cada uma delas analisa um conjunto de artefatos e identifica defeitos específicos de acordo com tais artefatos. Os artefatos analisados são a especificação de requisitos do software, que serve como base para identificar defeitos, o modelo de *features* e o mapa de produtos.

Uma visão geral da SPLIT é apresentada na Figura 3.1, que identifica os artefatos de cada uma das técnicas propostas e os tipos de defeitos que são identificados. Posteriormente, cada uma das técnicas é detalhada.

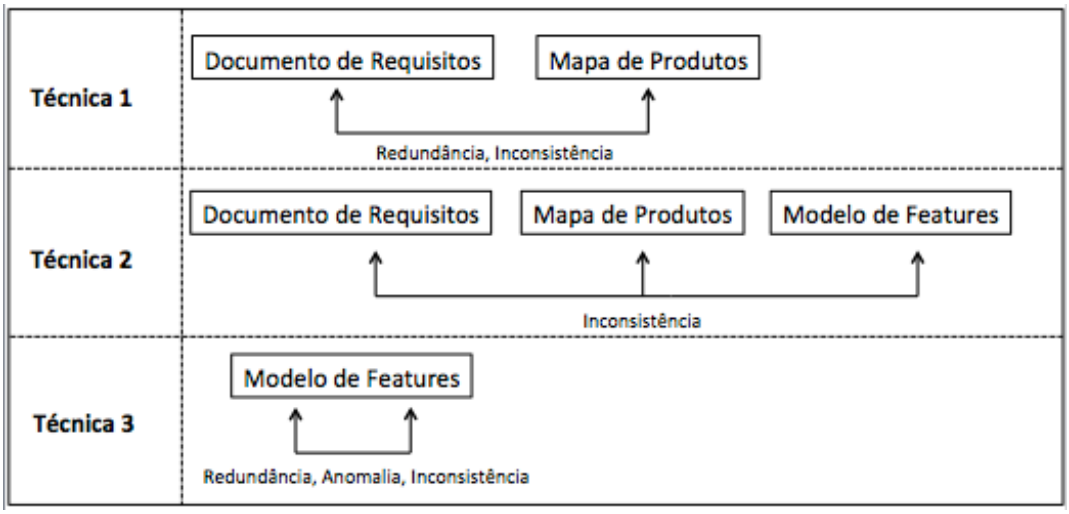


Figura 3.1. Visão Geral da SPLIT

3.1. Técnica SPLIT 1 – Documento de Requisitos x Mapa de Produtos

A Técnica 1 verifica o mapa de produtos de acordo com o documento de requisitos, identificando defeitos e classificando-os em dois tipos, conforme sua natureza:

- Redundância: o mapa de produtos apresenta produtos que contém o mesmo conjunto de *features* e que estão especificados no documento de especificação de requisitos.

- Inconsistência: o mapa de produto apresenta inconsistências quando os produtos gerados não apresentam features especificados no documento de requisitos.

A **Tabela 3.1** apresenta as questões de *checklist* da Técnica 1, incluindo o tipo de defeito que elas identificam.

Tabela 3.1: Questões que compõe o *checklist* da Técnica 1

Tipo de Defeito	Questões
Redundância	1.1. Existem **dois ou mais produtos** no mapa de produtos que apresentam o **mesmo conjunto de *features***?
Inconsistência	1.2. Existem **features** encontradas no documento de requisitos que **não estão contidas no mapa de produtos**?
	1.3. Existem **features em um produto** que pode ser gerado pelo mapa de produtos que **não estão descritas no documento de requisitos**?
	1.4. Existe algum produto que pode ser gerado pelo mapa de produtos

	que não está descrito no documento de requisitos?
	1.5. Existem features em um **produto que não pode ser gerado pelo mapa de produtos** que estão **descritos no documento de requisito**?
	1.6. Existe alguma ***feature* caracterizada como obrigatória** no documento de requisitos que **não está em todos os produtos do mapa de produtos**?

Um exemplo de utilização da Técnica SPLIT 1 pode ser encontrado na Especificação de Requisitos (Apêndice C) e no Mapa de Produtos (Apêndice D). Neste primeiro, podemos verificar que o mesmo apresenta o requisito descrito na **Figura 3.2**:

RF 4. Leitura de mensagens com suas menções

Descrição do caso de uso	O usuário pode ler as mensagens com menções a seus usuários
Classificação dos requisitos	Obrigatório
Entrada e pré-condições	
Saídas e pós-condições	
Produto	Básico Intermediário Avançado com suporte ao twitpic Avançado com suporte ao yfrog Avançado com suporte ao Instapaper Avançado com suporte ao Readlater

Figura 3.2. Texto extraído da Especificação de Requisitos (Apêndice C)

Porém, apesar de o requisito estar descrito, este não está presente no mapa de produtos descrito no Apêndice D, como pode ser visto na **Figura 3.3**.

RF 1	Criação de usuário no serviço do twitter
RF 2	Entrar no aplicativo com um usuário do twitter
RF 3	Leitura de mensagens na sua *timeline*
RF 5	Leitura de mensagens diretas
RF 6	Envio uma mensagem para o twitter
RF 7	Notificação de atualização da timeline
RF 8	Notificação de Atualização das menções
RF 9	Notificação de atualização das mensagens diretas
RF 10	Integração com o serviço de fotos twitpic
RF 11	Integração com o yofrog
RF 12	Integração com o serviço readlater
RF 13	Integração com o serviço instapaper

Figura 3.3. Texto extraído do Mapa de Produtos (Apêndice D)

Este tipo de defeito é encontrado utilizando o item 1.2 na qual é realizada a seguinte questão: “Existem features encontradas no documento de requisitos que não estão contidas no mapa de produtos? “. Esta tem como objetivo verificar todos se os itens do documento de requisitos estão contidos no mapa de produto.

3.2. Técnica SPLIT 2 – Documento de Requisitos x Mapa de Produtos x Modelo de Features

Esta técnica verifica o mapa de produtos e o modelo de *features*, de acordo com o documento de requisitos, identificando defeitos e classificando-os por inconsistência: os artefatos da linha de produto de software (mapa de produtos e modelo de *features*) apresentam produtos que não estão especificados no documento de requisitos. Esta técnica se diferencia da Técnica 1 por também analisar informações contidas apenas no modelo de *features* e, dessa forma, é possível identificar defeitos que não são identificados pela Técnica 1.

A **Tabela 3.2** apresenta as questões de *checklist* da Técnica 2, incluindo o tipo de defeito que elas identificam.

Tabela 3.2: Questões que compõe o *checklist* da Técnica 2

Tipo de Defeito	Questões
Inconsistência	2.1. Existe algum **produto** do mapa de produtos que **não pode ser gerado pelo modelo de *features***?
	2.2. Existe algum **relacionamento de implicação** descrito no documento de requisitos que **não está especificado no modelo de *features***?
	2.3. Existe algum **relacionamento de implicação** que está especificado no modelo de features mas **não está descrito no documento de requisitos**?
	2.4. Existe algum **relacionamento mutualmente exclusivo** descrito no documento de requisitos que **não está especificado no modelo de *features***?
	2.5. Existe algum **relacionamento mutualmente exclusivo** que está especificado no modelo de features mas **não está descrito no documento de requisitos**?
	2.6. Existe alguma ***feature* obrigatória** no mapa de produtos e documento de requisitos que **não está especificada como mandatória no modelo de *features***?
	2.7. Existe alguma ***feature* opcional** no mapa de produtos e documento de requisitos que **não está especificada como opcional no modelo de features**?

	2.8. Existe algum **conjunto de *features* alternativas** no mapa de produtos e documentos de requisitos que **não está especificado** como alternativa no **modelo de *features***?
	2.9. Existe algum **conjunto de *features* inclusivas** no mapa de produtos e documentos de requisitos que **não está especificada** como inclusivas no **modelo de *features***?

Um exemplo de utilização da Técnica SPLIT 2 pode ser encontrado na Especificação de Requisitos (Apêndice C) e no Modelo de Features (Apêndice E). Neste primeiro podemos verificar que o mesmo apresenta os requisitos descritos na **Figura 3.4**, os quais não apresentam nenhum tipo de implicação para nenhum outro requisito:

RF 12. Integração com o serviço readlater

Descrição do caso de uso	O usuário pode enviar o link de um tweet para o serviço readlater
Classificação dos requisitos	Opcional
Entrada e pré-condições	O tweet deve conter um link O Use Case **Integração com o serviço instapaper** (RF 13) não deve ser selecionado
Saídas e pós-condições	O usuário visualiza uma mensagem de confirmação de que o link foi enviado
Produto	Avançado com suporte ao Readlater

RF 13. Integração com o serviço instapaper

Descrição do caso de uso	O usuário pode enviar o link de um tweet para o serviço instapaper.
Classificação dos requisitos	Opcional
Entrada e pré-condições	O tweet deve conter um link
Saídas e pós-condições	O usuário visualiza uma mensagem de confirmação de que o link foi enviado O Use Case **Integração com o serviço readlater**(RF 12) não deve ser selecionado
Produto	Avançado com suporte ao Instapaper

Figura 3.4. Texto extraído da Especificação de Requisitos (Apêndice C)

No entanto, a **Figura 3.5** apresenta uma extração do modelo de features, na qual existe um relacionamento de implicação entre o requisito "Integração com o instapaper (RF 13)" para com o requisito "Integração com o readlater (RF 12)".

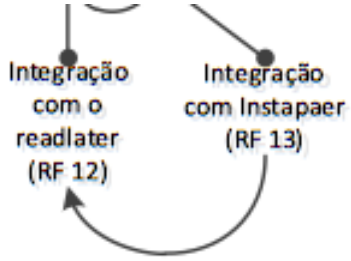


Figura 3.5. Imagem extraída do modelo de features (Apêndice E)

Este tipo de defeito é identificado utilizando o item 2.2 no qual a seguintes questão é formulada: "Existe algum relacionamento de implicação descrito no documento de requisitos que não está especificado no modelo de features?". Dessa forma, é possível identificar os defeitos entre os artefatos de uma linha de produto.

3.3. Técnica SPLIT 3 – Modelo de Features

Esta técnica verifica defeitos em modelo de *features*. Ela foi baseada em defeitos descritos em Massen e Lichter (2004), que identificam redundância, anomalias e inconsistências em modelos de *features*. Os defeitos encontrados por esta técnica podem ser classificados em três tipos:

- Redundância: o modelo de *features* contem redundância se pelo menos uma informação semântica é modelada de mais de uma forma.

- Anomalia: um modelo de *features* contém anomalias se possíveis configurações estão sendo ignoradas devido a restrições inseridas erroneamente no modelo de *features*.

- Inconsistência: um modelo de *features* contém inconsistências se o modelo incluir informações contraditórias.

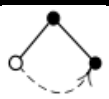
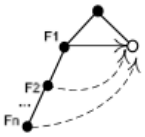
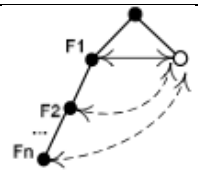
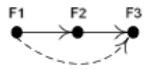
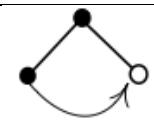
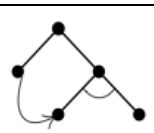
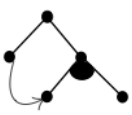
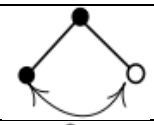
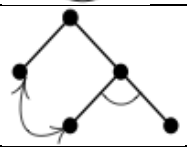
Esta técnica os seguintes conceitos apresentados em Massen e Litcher: (2004):

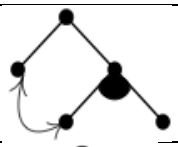
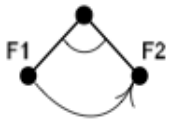
- Feature completamente obrigatória: é aquele em que todos os nós predecessores são obrigatórios; e

• Feature relativamente obrigatória: uma feature F2 é relativamente obrigatória para F1, se F2 é uma feature mandatória e todos os nós no caminho para F1 são mandatórios.

A **Tabela 3.3** apresenta as questões de *checklist* da Técnica 3, incluindo o tipo de defeito que elas identificam.

Tabela 3.3: Questões que compõe o checklist da Técnica 3

Tipo de Defeito	Item	Notação
Redundância	3.1. Existe uma ***feature* completamente obrigatória** que apresenta uma **implicação**?	
	3.2. Existe uma ***feature* F2 relativamente obrigatória** para uma feature F1 e que apresenta uma **implicação**?	
	3.3. Existem **duas *features* filhas** de uma mesma feature pai que **são ao mesmo tempo alternativa e mutuamente exclusiva**?	
	3.4. Existe um ***feature* com relacionamento de implicação por múltiplas features** F1, ..., Fn, sendo que F1 é pai de F2 e que F2,...,Fn são **relativamente obrigatórias** para F1?	
	3.5. Existe um ***feature* com relacionamento mutualmente exclusivo** por múltiplas features F1, ..., Fn, sendo que F1 é pai de F2 e que F2,...,Fn s**ão relativamente obrigatórias** para F1?	
	3.6. Existe uma *feature* F1 que tem um relacionamento de implicação para F2 que tem uma implicação para F3?	
Anomalia	3.7. Existe uma ***feature* opcional** que tem um relacionamento de **implicação por uma *feature* completamente obrigatória**?	
	3.8. Existe uma ***feature* que é alternativa** em relação a *feature* pai e que tem um **relacionamento de implicação por uma *feature* completamente obrigatória**?	
	3.9. Existe uma ***feature* que é inclusiva** em relação a *feature* pai e que tem um **relacionamento de implicação por uma *feature* completamente obrigatória**?	
	3.10. Existe uma ***feature* opcional** que tem um relacionamento **mutualmente exclusivo com uma *feature* completamente obrigatória**?	
	3.11. Existe uma ***feature* alternativa** que tem um relacionamento **mutualmente exclusivo com uma *feature* completamente obrigatória**?	

	3.12. Existe uma ***feature*** **inclusiva** que tem um **relacionamento mutualmente exclusivo** com uma *feature* completamente obrigatória?	
Inconsistência	3.13. Existem **duas *features*** com relacionamento **mutualmente exclusivo** e que são completamente **obrigatórias**?	
	3.14. Existe uma ***feature*** **relativamente obrigatória** para a feature F1 e estas features apresentam um relacionamento **mutualmente exclusivo**?	
	3.15. Existem **duas *features*** **filhas alternativas** e que apresentam um relacionamento de **implicação**?	
	3.16. Existem **duas *features*** que apresentam **simultaneamente** um relacionamento de **implicação** e **mutualmente exclusivos**?	

Um exemplo de defeito encontrado é apresentado na **Figura 3.6**. Neste defeito, uma feature mandatória "Postar uma mensagem (RF6)" está implicando em uma feature opcional "Notificar a atualização na linha do tempo (RF7)". Devido a isto, todos os produtos da linha de produto de software deveriam apresentar a *feature* "Notificar a atualização na linha do tempo (RF7)", que sendo uma *feature* opcional, resultaria em perdas de possíveis configurações.

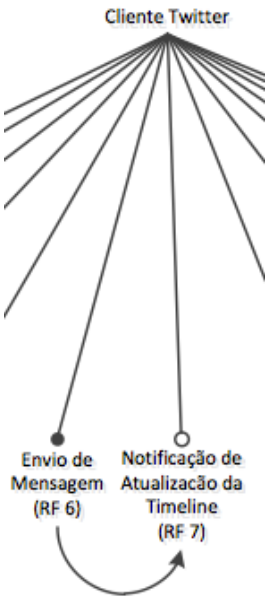
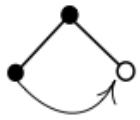


Figura 3.6. Exemplo de defeito no modelo de *feature* do cliente Twitter utilizado no estudo de viabilidade identificado ao utilizar o conjunto de técnicas SPLIT

O defeito apresentado na Figura 3.6 pode ser encontrado pelo conjunto de técnicas SPLIT, pois este está especificado pela Técnica 3, que verifica isoladamente o modelo de features. Um trecho extraído da técnica SPLIT que verifica o defeito apresentado na **Figura 3.6**, é apresentado na **Tabela 3.4**.

Tabela 3.4: Um trecho da técnica de inspeção de linha de produto de software (SPLIT)

3.7. Existe uma *feature* completamente obrigatória que tem um relacionamento de implicação por uma *feature* completamente obrigatória?	

O texto integral do conjunto de técnica de inspeção para modelos de linha de produto de software é apresentado no Apêndice A.

CAPÍTULO 4. ESTUDO DE VIABILIDADE

De acordo com Mafra *et al.* (2006), um estudo de viabilidade tem como objetivo analisar se a aplicação de uma tecnologia é viável, se atende de forma razoável aos objetivos inicialmente definidos, de forma a justificar (ou não) a continuação da pesquisa. O estudo de viabilidade do conjunto de técnicas de inspeção em modelos de linha de produto de software (SPLIT), proposto neste trabalho, teve como objetivo verificar se esta alcança as metas de encontrar defeitos em estágios iniciais do processo de desenvolvimento de software.

O objetivo do estudo de viabilidade foi especificado utilizando o paradigma GQM (*Goal Question Metric*), proposto por Basili e Rombach (1998), conforme especificado na **Tabela 4.1**:

Tabela 4.1: Objetivo do Estudo de Viabilidade segundo GQM [Basili e Rombach, 1998]

Analisar	Um conjunto de técnicas de inspeção para modelos de linha de produto de software proposto
Com o propósito de	Caracterizar
Com relação a	Ao número de defeitos encontrados comparados com o número de defeitos encontrados por uma abordagem de inspeção baseada em tipos de defeitos
Do ponto de vista dos	Pesquisadores
No Contexto	Uma inspeção de modelos de linhas de produto de software por estudantes de graduação

4.1. Planejamento do Estudo

O planejamento do estudo foi realizado visando analisar o conjunto de técnicas de inspeção quantitativamente, por meio do número de defeitos encontrados, comparado com uma abordagem de inspeção baseada em tipos de defeitos.

A comparação com uma abordagem de inspeção baseada em tipos de defeitos, se deu pelo fato de, apesar da realização de um mapeamento sistemático, nenhuma outra técnica de inspeção para especificações de Linha de Produto de Software ter retornado no mesmo. Com isso, esta forma de abordagem de inspeção baseada em defeitos, visa apresentar vantagens em cima de uma inspeção adhoc, pois nesta, os tipos de defeitos que podem ser encontrado pelos participantes são especificados para os mesmos.

Dessa forma, foram formuladas as seguintes hipóteses:

- H0: Não existe diferença no número médio de defeitos encontrados em modelos de linha de produto de software utilizando a SPLIT e uma abordagem de inspeção baseada em tipos de defeitos.

- HA1: O número médio de defeitos encontrados em modelos de linha de produto de software utilizando uma abordagem de inspeção baseada em tipos de defeitos é maior que o número de defeitos encontrados utilizando a SPLIT.

- HA2: O número médio de defeitos encontrados em modelos de linha de produto de software utilizando a SPLIT é maior que o número de defeitos encontrados utilizando uma abordagem de inspeção baseada em tipos de defeitos.

4.2. Instrumentação

O experimento foi composto do seguinte conjunto de artefatos: Termo de Consentimento Livre e Esclarecido (Apêndice B), especificação de uma linha de produto de software, a documentação da SPLIT para o grupo de participantes que a utilizou na execução e um questionário de avaliação. A especificação da linha de produto de software é composta de uma especificação de requisitos (Apêndice C), um mapa de produtos (Apêndice D) e um modelo de features (Apêndice E), sendo que:

- o artefato documento de requisitos é a especificação de uma linha de produto de software para um cliente de Twitter, que contém 13 features de

populares clientes de Twitter, incluindo 6 features mandatórias, 3 features opcionais e 3 features alternativas. Esta linha de produto de software gera um total de 6 produtos, sendo que estes apresentam um conjunto de features comuns e features distintas.

- o mapa de produtos para a linha de produto de software do cliente de Twitter lista as features disponíveis no documento de requisitos e as associa, de acordo com a disponibilidade, em cada um dos produtos. Esta enumera seis diferentes clientes de Twitter da linha de produto, mapeando as features de cada um dos produtos; e

- o modelo de features da linha de produto de software foi especificado de acordo com o documento de requisitos, e por meio do mesmo, é possível gerar os seis clientes de Twitter definidos no mapa de produtos. Neste modelo de features foram inseridos defeitos pelos pesquisadores baseados em defeitos existentes em modelos de features listados em Massen e Lichter (2004).

4.3. O projeto do experimento

Os participantes foram divididos em dois grupos (grupo A e grupo B) para inspecionar o mesmo conjunto de artefatos de uma linha de produto de software. Os integrantes do grupo A revisaram os artefatos utilizando uma abordagem de inspeção baseada em tipos de defeito e os integrantes do grupo B revisaram o conjunto de técnicas SPLIT proposto neste trabalho de pesquisa. Como nenhum participante tinha experiência prévia com modelos de linha de produto de software, os participantes foram selecionados aleatoriamente para cada um dos grupos. Cada grupo era composto por dez alunos de graduação, escolhidos por conveniência, na disciplina de Modelagem e Projeto de Sistemas dos cursos de Sistemas de Informação e Ciência da Computação da Universidade Federal do Amazonas.

4.4. Preparação

Os participantes assinaram um Termo de Consentimento Livre e Esclarecido (TCLE) e participaram de um treinamento que abordou os conceitos

gerais e especificações de linha de produto de software, incluindo modelo de *features* e mapa de produtos.

Após o treinamento geral sobre linha de produto de software, os participantes receberam um treinamento para a execução do experimento. Os grupos foram divididos em duas salas distintas. Os integrantes do grupo A receberam um treinamento sobre técnicas de inspeção de modelo que descrevia os tipos de defeitos que deveriam ser encontrados pelos inspetores durante a execução do estudo. Os integrantes do grupo B receberam um treinamento acerca do conjunto de técnicas SPLIT proposto neste trabalho.

Os treinamentos tiveram diferentes instrutores, pois ocorreram paralelamente, evitando a comunicação entre os grupos. No entanto, os materiais dos dois treinamentos foram preparados por ambos os instrutores, com o objetivo de garantir uma equivalência em relação ao conhecimento sobre inspeção de modelos de software entre os dois grupos.

4.5. Execução

O experimento, compreendendo o treinamento de cada grupo, teve sua execução limitada em duas horas. Os integrantes dos grupos A e B executaram o experimento simultaneamente, em salas separadas, para evitar qualquer comunicação entre os participantes dos diferentes grupos.

O experimento foi acompanhado pelos pesquisadores, que permaneceram um em cada sala durante a execução do experimento para evitar qualquer troca de informação entre os participantes, assim como, fornecer auxílio durante a execução do experimento, caso algum participantes tivesse dúvidas sobre o processo de execução do experimento.

4.6. Resultados

Ao final do experimento, o formulário com a lista de defeitos e o questionário de avaliação foram recolhidos para a análise. O número de defeitos encontrados por participante para cada tipo de inspeção é apresentado na **Tabela 4.2** e na **Tabela 4.3**:

Tabela 4.2: Número de defeitos encontrados usando uma abordagem de inspeção baseada em tipos de defeitos (Grupo A)

Participante	Número de Defeitos Encontrados
Participante 01	4
Participante 02	9
Participante 03	7
Participante 04	8
Participante 05	4
Participante 06	9
Participante 07	1
Participante 08	3
Participante 09	7
Participante 10	7

Tabela 4.3: Número de defeitos encontrados usando SPLIT (Grupo B)

Participante	Número de Defeitos Encontrados
Participante 11	10
Participante 12	11
Participante 13	12
Participante 14	8
Participante 15	10
Participante 16	13
Participante 17	8
Participante 18	10
Participante 19	8
Participante 20	10

Um exemplo de defeito encontrado na realização do experimento é apresentado na **Figura 4.1**, onde um trecho do modelo de features do cliente de Twitter é apresentado. Neste defeito, uma *feature* mandatória "Envio de mensagem (RF6)" está implicando em uma *feature* opcional "Notificação de atualização de *Timeline* (RF7)". Devido a isto, todos os produtos da linha de produto de software deveriam apresentar a *feature* "Notificação de atualização de *Timeline* (RF7)", que sendo uma *feature* opcional, resultaria em perdas de possíveis configurações.

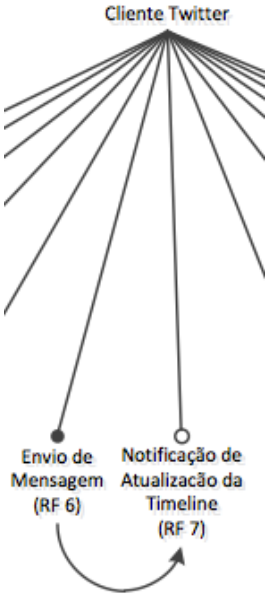
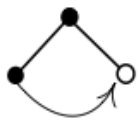


Figura 4.1. Exemplo de defeito no modelo de *feature* do cliente Twitter utilizado no estudo de viabilidade identificado ao utilizar o conjunto de técnicas SPLIT

O defeito apresentado na **Figura 4.1** pode ser encontrado pelo conjunto de técnicas SPLIT, pois este está especificado pela Técnica 3, que verifica isoladamente o modelo de features. Um trecho extraído da técnica SPLIT que verifica o defeito apresentado na **Figura 4.1**, é apresentado na **Tabela 4.4**.

Tabela 4.4: Um trecho da SPLIT

3.7. Existe uma *feature* completamente obrigatória que tem um relacionamento de implicação por uma *feature* completamente obrigatória?	

4.7. Análise Quantitativa

A análise quantitativa do resultado do experimento foi realizada com a aplicação do teste Mann-Whitney U. Ele foi usado em dois grupos (que utilizaram diferentes técnicas de inspeção) para comparar, diferentes participantes em cada condição, e sem pressuposto na distribuição dos dados. Foi utilizado um critério (α) – a probabilidade de incorretamente rejeitar a hipótese nula – de 0,05 devido ao pequeno tamanho da amostra [Dyba et al. 2006].

Pelo fato de o fator U obtido no teste estatístico ser 0,001 (menor que o critério de significância utilizado), é possível rejeitar a hipótese nula. Isso quer dizer que não foi encontrada evidência de que, o número de defeitos encontrados pelo participantes utilizando a SPLIT com relação ao número de defeito encontrado pelos participantes que utilizaram uma abordagem de inspeção baseada em defeito sejam equivalentes.

A **Tabela 4.5** apresenta uma comparação em relação ao número de defeitos encontrados pelos inspetores com a abordagem de inspeção baseada em tipos de defeito e o conjunto de técnicas SPLIT utilizando a média e os valores de desvio padrão. Esta sugere que o número de defeitos encontrados pelos inspetores que utilizaram o conjunto de técnica de inspeção proposta nesse trabalho foi 70% maior que os inspetores que utilizaram uma abordagem de inspeção baseada em tipos de defeitos. Assim como, um menor desvio padrão mostra que o número de defeitos encontrados pelos inspetores que utilizaram a SPLIT apresenta uma variância menor.

Tabela 4.5: Análise de comparação de defeitos encontrados

Tipo de inspeção	**Tamanho da Amostra**	**Número de Defeitos Encontrados**		
		Média	**Desvio Padrão**	**% do Desvio Padrão**
Abordagem de inspeção baseada em tipos de defeitos	10	5,9	2,7	46%
SPLIT	10	10	1,7	17%

A **Figura 4.2** apresenta o gráfico bloxplot com o número de defeitos encontrados por cada tipo de inspeção. Neste podemos verificar que a mediana para inspetores que utilizaram a técnica de inspeção SPLIT é maior que os inspetores que utilizaram a abordagem baseada em tipo de defeito.

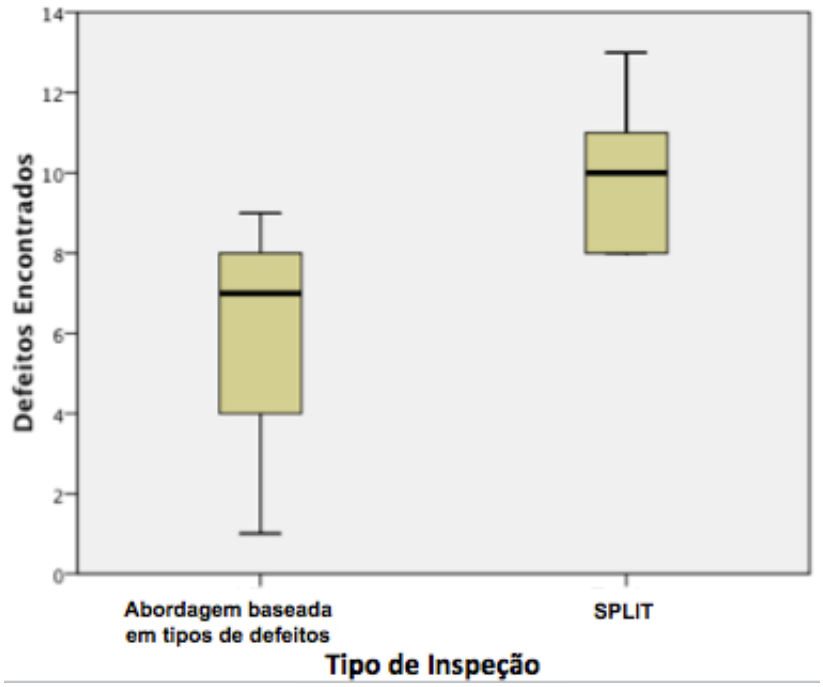


Figura 4.2. Boxplot para o número de defeitos encontrados por tipo de inspeção por participante

Estes resultados rejeitam a hipótese nula H0, de que não existe diferença entre o número médio de defeitos encontrados pelos inspetores que utilizaram uma abordagem de inspeção baseada em tipos de defeitos para com aqueles que utilizaram o conjunto de técnicas de inspeção proposta neste trabalho. De acordo com a análise do gráfico boxplot e do teste Mann-Whitney, a hipótese alternativa HA2 é válida para esse experimento. Este resultado sugere que o número de defeitos encontrados em modelos de linha de produto de software ao utilizarmos a SPLIT é maior que usando uma abordagem de inspeção baseada em defeitos.

4.8. Ameaças à Validade

As ameaças à validade foram identificadas e categorizadas para o experimento, de acordo com Wohlin *et al*. (2000), como segue:

- Validade Interna: Classificou-se como o principal risco para uma interpretação imprópria dos resultados, a diferença de treinamento entre os dois grupos. O grupo A teve um treinamento em uma abordagem de inspeção baseada em tipos de defeitos e o grupo B teve um treinamento no conjunto de técnicas SPLIT. Com o objetivo de minimizar esta ameaça, o material do treinamento foi

preparado pelos dois instrutores com o objetivo de minimizar a diferença entre o conhecimento dos dois grupos, e dessa forma, fornecer o material mais equivalente possível sobre defeitos em especificações de Linha de Produto de Sofware.

- Validade Externa: As duas principais ameaças à validade externas foram classificadas em respeito a generalização de resultado: os participantes do treinamento são alunos de graduação e a especificação de software utilizada no experimento não é uma aplicação real. Apesar de não serem inspetores reais, pode-se considerar os benefícios de estudos que utilizam alunos como participantes, de acordo com Carver et al. (2003). Em relação à segunda ameaça, buscou-se basear o cliente de Twitter utilizado neste experimento com features encontradas em clientes de Twitter reais.

- Validade de conclusão: A principal ameaça está relacionada ao tratamento e o resultado deste experimento, por utilizar uma pequena amostra de dados para a análise estatística. Isto não é ideal pois, por vezes, há falta de uma representação estatística para o fenômeno. Porém, este é um problema de difícil solução em experimentos em Engenharia de Software [Conte et al., 2007].

- Validade de Construção: Para o experimento, foi utilizado o número de defeitos encontrados por inspetores para medir e comparar os dois diferentes tipos de inspeção. Esta é uma métrica comum utilizada para avaliar técnicas de inspeção, como podemos observar em Basili e Rombach (1998) e Shull et al. (2000). Como o tempo de execução do experimento foi limitado, não foi utilizado nenhum indicador de defeitos encontrados por tempo.

CAPÍTULO 5. ESTUDO DE CASO

O estudo de caso ciclo de vida foi realizado após a execução e análise de resultados do estudo de viabilidade, e visou analisar a aplicação do conjunto de técnicas de inspeção em modelos de linhas de produto de software (SPLIT) em especificações de uma linha de produto real.

Por este trabalho estar inserido no contexto de uma pesquisa conduzida pelo Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos (INCT-SEC) [INCT-SEC, 2012], por meio da cooperação com a Universidade Federal de São Carlos (UFSCar). Para este estudo, foi fornecido pelos pesquisadores Prof. Ph.D. Valter Vieira de Camargo e pela pesquisadora Prof. Dra. Rosângela Delloso Penteado um modelo de *features* (Apêndice F) e um mapa de produtos de linhas de produto de software real (Apêndice G), de veículos robóticos móveis. Os modelos de linha de produto de software de veículos robóticos móveis, foram fornecidos por meio de um mapa de produtos e um modelo de features. A partir do qual, foram extraídos os requisitos para compor a especificação de requisitos que foi utilizada no estudo de caso.

Este estudo de caso se diferencia do estudo de viabilidade, previamente descrito, que utilizava um exemplo criado com o propósito de ser utilizado na execução do experimento. De acordo com Mafra *et al*. (2006), um estudo de caso deve ser conduzido com o propósito de caracterizar a aplicação da tecnologia no contexto de um ciclo de vida de desenvolvimento, não produzido especificamente para utilização no experimento.

O objetivo do estudo de caso é similar ao que foi planejado para o estudo de viabilidade, porém este utilizou-se uma linha de produto de software real. O objetivo foi especificado utilizando o paradigma GQM (*Goal Question Metric*), proposto por Basili e Rombach (1998), conforme apresentado na **Tabela 5.1**:

Tabela 5.1: Objetivo do Estudo de Caso segundo GQM [Basili e Rombach, 1998]

Analisar	Um conjunto de técnicas de inspeção em modelos de linha de produto de software proposto
Com o propósito de	Caracterizar
Com relação a	Ao número médio de defeitos encontrados comparados com o número médio de defeitos encontrados por uma abordagem de inspeção baseada em tipos de defeitos
Do ponto de vista dos	Pesquisadores
No contexto	De modelos de uma linha de produto de software real de veículos robóticos móveis no contexto do INCT-SEC por estudantes de graduação e pós graduação

5.1. Planejamento do Estudo

O planejamento do estudo foi realizado visando caracterizar o conjunto de técnicas de inspeção em modelos de linha de produto de software (SPLIT) quantitativamente, por meio do número médio de defeitos encontrados, comparados com uma abordagem de inspeção baseada em tipos de defeitos.

Dessa forma, foram formuladas as seguintes hipóteses:

- H0: Não existe diferença no número médio de defeitos encontrados em modelos de linha de produto de software utilizando a SPLIT e uma abordagem de inspeção baseada em tipos de defeitos.

- HA1: O número médio de defeitos encontrados em modelos de linha de produto de software utilizando uma abordagem de inspeção baseada em tipos de defeitos é maior que o número de defeitos encontrados utilizando a SPLIT.

- HA2: O número médio de defeitos encontrado em modelos de linha de produto de software utilizando a SPLIT é maior que o número médio de defeitos encontrados utilizando uma abordagem de inspeção baseada em tipos de defeitos.

5.2. Instrumentação

O estudo de caso utilizou os seguintes artefatos: Termo de Consentimento Livre e Esclarecido (TCLE), apresentado no Apêndice H; especificação de uma linha de produto de software; e a documentação da SPLIT para um grupo que a utilizou na execução. A especificação da linha de produto de software é composta por uma especificação de requisitos, um mapa de produtos e um modelo de *features*.

A principal diferença entre a instrumentação deste estudo e do estudo anterior, está no fato de que este utilizou um problema real, ou seja, uma especificação de linha de produto de software da indústria (Apêndice I).

Para fins de realização deste experimento, foram omitidas algumas *features* do produto, para que a sua execução ocorresse no tempo limite de 2 horas. Dessa forma, o modelo de *features* utilizado no estudo de caso é apresentado no Apêndice J e o mapa de produto no Apêndice K.

O artefato documento de requisitos é uma linha de produto de software para um veículo robótico móvel que contem 17 *features*. Esta linha de produto de software gera um total de 7 produtos, sendo que estes apresentam um conjunto de *features* comuns e *features* distintas. O documento de requisitos descreve cada *feature* da linha de produto de software, especificando seus relacionamentos com outras *features*; as classificando como mandatórias, opcionais ou alternativas; e enumerando as *features* presente em cada produto. O mapa de produtos para a linha de produto de software do veiculo robótico móvel lista todas as *features* disponíveis do documento de requisitos e as associa, de acordo com a disponibilidade, em cada um dos produtos.

No modelo de *features* e no mapa de produtos fornecidos pelos pesquisadores da UFSCar, foram inseridos defeitos baseados em defeitos existentes em modelos de *features* listados em Massen e Lichter (2004).

5.3. O projeto do experimento

O estudo de caso teve duas execuções: a primeira aconteceu na Universidade Estadual de Maringá (UEM) e a segunda na Universidade Federal do Amazonas (UFAM). Os participantes da UEM eram alunos de graduação e pós-graduação da turma de Sistemas Distribuídos. Esta execução aconteceu devido a cooperação entre esta e a Universidade Federal do Amazonas por meio do INCT-SEC. Os participantes da UFAM eram alunos do 3º ano da turma de Modelagem e Projeto de Sistemas da Universidade Federal do Amazonas (UFAM).

Na execução na UEM, os participantes do experimento foram divididos em dois grupos, de acordo com a experiência em inspeção de software e conhecimento em linhas de produto de software. Na execução na UFAM, os participantes foram divididos aleatoriamente, pois os mesmos apresentavam experiência prévia devido à participação em experimentos de inspeção de modelos e usabilidade de software. Porém, não apresentavam conhecimento sobre linhas de produto de software.

Da mesma forma que o estudo de viabilidade, os participantes foram divididos em dois grupos para inspecionar um mesmo conjunto de artefatos de uma linha de produto de software: cada integrante do grupo A avaliou os artefatos utilizando uma abordagem de inspeção baseada em tipos de defeito e cada integrante do grupo B avaliou o conjunto de técnicas SPLIT proposto neste trabalho.

5.4. Preparação

Os participantes assinaram um Termo de Consentimento Livre e Esclarecido (TCLE) e participaram de um treinamento que abordou os conceitos gerais e especificações de linha de produto de software, incluindo modelo de *features* e mapa de produtos. Após o treinamento, os participantes receberam um treinamento para a execução do experimento. As duas execuções referentes ao estudo de caso, ocorreram em local e datas diferentes e da mesma forma, com os grupos divididos em duas salas de aula distintas, o grupo A recebeu um

treinamento sobre técnicas de inspeção de modelo que descrevia os tipos de defeitos que deveriam ser encontrados pelos inspetores durante a execução do estudo. O grupo B recebeu um treinamento acerca do conjunto de técnicas SPLIT proposta neste trabalho.

Os treinamentos tiveram diferentes instrutores, pois ocorreram paralelamente, evitando a comunicação entre os grupos. No entanto, os materiais dos dois treinamentos foram preparados por ambos os instrutores, com o objetivo de garantir uma equivalência em relação ao conhecimento sobre inspeção de modelos de software entre os dois grupos.

5.5. Resultados

Ao final de cada execução do experimento, os formulários com a lista de defeitos encontrados por cada participante foram recolhidos para a análise do experimento. O número de defeitos encontrados por participante, incluindo a experiência em inspeção de modelos de software e conhecimento sobre linha de produto de software, por cada tipo de inspeção é apresentado na **Tabela 5.2** e na **Tabela 5.3**.

Esses dados são referentes às duas execuções do experimento, a primeira que ocorreu na Universidade Estadual de Maringá (UEM) e a segunda que ocorreu na Universidade Federal do Amazonas (UFAM), e foram consolidados nas tabelas a seguir:

Tabela 5.2: Número de defeitos encontrados em um abordagem de inspeção baseada em defeitos (Grupo A)

Participante	**Conhecimento LPS**	**Experiência em Avaliação/Inspeção de SW**	**Número de Defeitos Encontrados**
Participante 01	Estudo de LPS	Nenhum	11
Participante 02	Nenhum	Nenhum	8
Participante 03	Nenhum	Nenhum	11
Participante 04	Projeto Indústria	Projeto Indústria	11

Participante	Conhecimento LPS	Experiência em Avaliação/Inspeção de SW	Número de Defeitos Encontrados
Participante 05	Noções Palestra/Leitura	Estudo sobre Inspeção	9
Participante 06	Nenhum	Nenhum	10
Participante 07	Projeto Acadêmico	Estudo sobre Inspeção	14
Participante 08	Nenhum	Nenhum	10
Participante 09	Nenhum	Nenhum	5
Participante 10	Nenhum	Nenhum	9
Participante 11	Nenhum	Estudo sobre inspeção	6
Participante 12	Nenhum	Estudo sobre inspeção	12
Participante 13	Nenhum	Estudo sobre inspeção	15
Participante 14	Nenhum	Estudo sobre inspeção	8
Participante 15	Nenhum	Estudo sobre inspeção	13
Participante 16	Nenhum	Estudo sobre inspeção	10

Tabela 5.3: Número de defeitos encontrados usando SPLIT (Grupo B)

Participante	Conhecimento LPS	Experiência em Avaliação/Inspeção de SW	Número de Defeitos Encontrados
Participante 01	Noções Palestra/Leitura	Nenhum	20
Participante 02	Noções Palestra/Leitura	Nenhum	14
Participante 03	Nenhum	Nenhum	7
Participante 04	Nenhum	Nenhum	21
Participante 05	Nenhum	Nenhum	16
Participante 06	Nenhum	Nenhum	14
Participante 07	Noções Palestra/Leitura	Nenhum	7

Participante	Conhecimento LPS	Experiência em Avaliação/Inspeção de SW	Número de Defeitos Encontrados
Participante 08	Estudo de LPS	Nenhum	17
Participante 09	Nenhum	Nenhum	9
Participante 10	Nenhum	Nenhum	10
Participante 11	Nenhum	Estudo sobre inspeção	19
Participante 12	Nenhum	Estudo sobre inspeção	18
Participante 13	Nenhum	Estudo sobre inspeção	15
Participante 14	Nenhum	Estudo sobre inspeção	12
Participante 15	Nenhum	Estudo sobre inspeção	13
Participante 16	Nenhum	Estudo sobre inspeção	20

5.6. Análise Quantitativa

A análise quantitativa do resultado do experimento foi realizada com a aplicação do teste Mann-Whitney U. Este teste não-paramétrico é equivalente ao teste t de Student. Ele foi usado em dois grupos (técnicas de inspeção) para comparar, diferentes participantes em cada condição, e sem pressuposto na distribuição dos dados.

Foi utilizado um critério (α) – a probabilidade de incorretamente rejeitar a hipótese nula – de 0,05 devido ao pequeno tamanho da amostra [Dyba et al. 2006].

Pelo fato de o fator U obtido no teste estatístico ser 0.007 (menor que o critério de significância utilizado), é possível rejeitar a hipótese nula. Isto quer dizer que não foi encontrada evidência de que, o número médio de defeitos encontrados pelo participantes utilizando a SPLIT para aqueles participantes que utilizaram uma abordagem de inspeção baseada em defeito, sejam equivalentes.

A **Tabela 5.4** apresenta uma comparação em relação ao número médio de defeitos encontrados pelos inspetores com a abordagem de inspeção baseada em tipos de defeito e o conjunto de técnicas SPLIT utilizando a média e os valores de desvio padrão. A **Tabela 5.4** sugere que o número de defeitos encontrados pelos inspetores que utilizaram o conjunto de técnica de inspeção proposta nesse trabalho foi 43% maior que os inspetores que utilizaram uma abordagem de inspeção baseada em tipos de defeitos.

Tabela 5.4: Análise de comparação de defeitos encontrados

Tipo de inspeção	**Tamanho da Amostra**	**Número de Defeitos Encontrados**		
		Média	**Desvio Padrão**	**% do Desvio Padrão**
Abordagem de inspeção baseada em tipos de defeitos	16	10,1	2,6	26%
SPLIT	16	14,5	4,5	31%

A **Figura 5.1** apresenta o gráfico bloxplot com o número de defeitos encontrados por cada tipo de inspeção. Neste podemos verificar que a mediana para inspetores que utilizaram a técnica de inspeção SPLIT é maior que os inspetores que utilizaram a abordagem baseada em tipo de defeito.

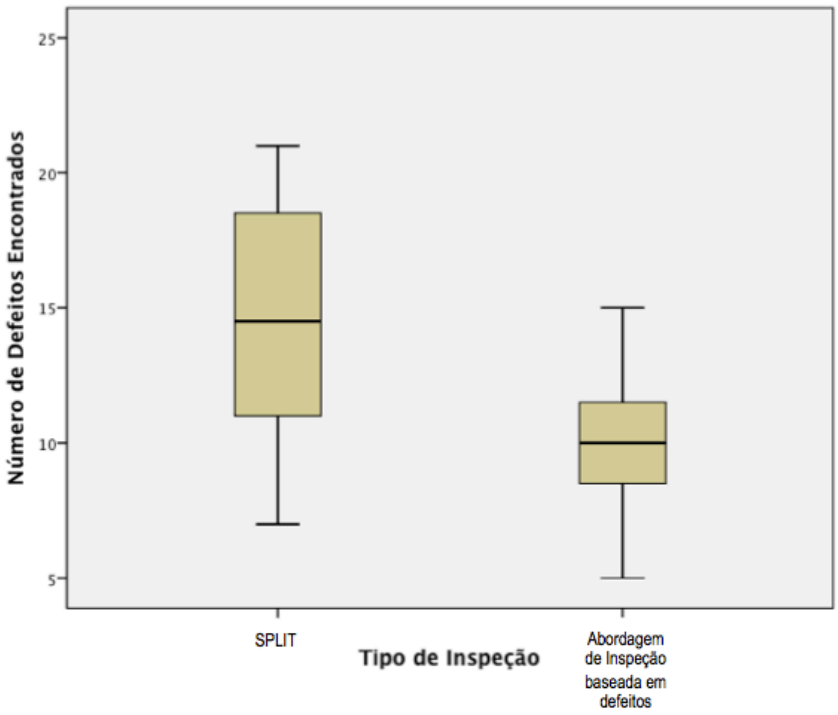


Figura 5.1. Boxplot para o número de defeitos encontrados por tipo de inspeção por participante

Estes resultados sugerem rejeitar a hipótese nula H0, de que não existe diferença entre o número de defeitos encontrados pelos inspetores que utilizaram uma abordagem de inspeção baseada em tipos de defeitos para com aqueles que utilizaram o conjunto de técnicas de inspeção proposta neste trabalho. De acordo com a análise do gráfico boxplot e do teste Mann-Whitney, a hipótese alternativa HA2 deve ser aceita para esse experimento. Esse resultado sugere que o número de defeitos encontrados em modelos de linha de produto de software ao utilizarmos a SPLIT é maior que usando uma abordagem de inspeção baseada em defeitos.

5.7. Ameaças à Validade

Da mesma forma que no estudo de viabilidade, as ameaças à validade foram identificadas e categorizadas para o experimento, de acordo com Wohlin *et al*. (2000), como segue:

- Validade Interna: Classificou-se como o principal risco para uma interpretação imprópria dos resultados, a diferença de treinamento entre os dois grupos. O grupo A teve um treinamento em uma abordagem de inspeção baseada em tipos de defeitos e o grupo B teve um treinamento no conjunto de técnicas

SPLIT. Com o objetivo de minimizar esta ameaça, o material do treinamento foi preparado pelos dois instrutores com o objetivo de minimizar a diferença entre o conhecimento dos dois grupos, e dessa forma, fornecer o material mais equivalente possível sobre defeitos em especificação de Linha de Produto de Sofware. Outro risco considerado para a validade interna diz respeito as duas execuções do experimento, que ocorreram em datas e locais diferentes. Da mesma forma que para a ameaça do estudo anterior, buscou-se minimizar esta por meio da utilização do mesmo material nas duas execuções, assim como, os treinamentos ministrados pelos mesmos pesquisadores para evitar que o tratamento dos participantes fosse diferente nas duas execuções.

- Validade Externa: As duas principais ameaças à validade externas foram classificadas em respeito a generalização de resultado pelo fato de os participantes do treinamento serem alunos de graduação e pós-graduação. Apesar de não serem inspetores com experiência, pode-se considerar os benefícios de estudos que utilizam alunos como participantes, de acordo com Carver et al. (2003).

- Validade de conclusão: A principal ameaça está relacionada ao tratamento e o resultado deste experimento, por utilizar uma pequena amostra de dados para a análise estatística. Isto não é ideal pois, por vezes, há falta de uma representação estatística significante para o fenômeno. Porém, este é um problema de difícil solução em experimentos em Engenharia de Software [Conte et al., 2007].

- Validade de Construção: Para o experimento, foi utilizado o número médio de defeitos encontrados por inspetores para medir e comparar os dois diferentes tipos de inspeção. Esta é uma métrica comum utilizada para avaliar técnicas de inspeção, como podemos observar em Basili e Rombach (1998) e Shull et al. (2000). Como o tempo de execução do experimento foi limitado, não foi utilizado nenhum indicador de defeitos encontrados por tempo.

CAPÍTULO 6. CONCLUSÃO

Linha de produto de software é uma abordagem que permite a organizações desenvolver uma série de produtos similares em um mesmo domínio de aplicação, reduzindo o desenvolvimento e o custo de manutenção e aumentando a produtividade [Denger e Kolb, 2006]. Porém, assim como o desenvolvimento tradicional de software, esta abordagem deve ter seus modelos avaliados para a melhoria da qualidade de software.

Neste trabalho foi proposto e avaliado um conjunto de técnicas de inspeção (SPLIT) para avaliar especificações de linhas de produto de software. Esta visa atender a necessidade devido às técnicas de inspeção de modelos tradicionais de software não serem suficientes para identificar defeitos que são específicos de sistemas de reuso [Denger e Kolb, 2006].

A SPLIT é composta de três técnicas que tem como objetivo encontrar defeitos em modelos de *features* e mapa de produtos baseados em requisitos de uma linha de produto de software.

6.1. Resultados Alcançados

Para a avaliação do conjunto de técnicas propostas foi executado primeiramente um estudo de viabilidade, utilizando um exemplo criado com fins específicos de utilização durante o experimento. Por meio da análise desse estudo, foi possível observar que era possível encontrar um número maior de defeitos utilizando as técnicas SPLIT, quando comparada com uma abordagem de inspeção baseada em tipos de defeitos nos estágios iniciais de desenvolvimento.

Em seguida, foi executado um segundo experimento, desta vez utilizando a especificação de uma linha de produto de software da indústria, para complementar os resultados encontrados no estudo de viabilidade. A análise deste resultado corroborou o resultado do estudo anterior, no qual foi identificado pelos inspetores que utilizaram a SPLIT, um número maior de defeitos do que inspetores que utilizaram uma abordagem baseada em tipos de defeitos.

As principais contribuições deste trabalho são as seguintes:

- A realização de um Mapeamento Sistemático que visou abordar processos para garantia de qualidade em modelos de linha de produto de software, que descreveu:
    - Uma grande concentração de pesquisa em verificação de modelos automatizada, que constitui 85% dos estudos primários sobre o tema
    - Não foram identificadas técnicas de inspeção para modelos de linha de produto de software
- A elaboração de um conjunto de técnicas de inspeção em modelos de linha de produto de software (SPLIT), que visa:
    - Encontrar um número maior de defeitos em estágios iniciais do desenvolvimento, possibilitando a redução de custos e a melhoraria da qualidade de software; e
    - Auxiliar a utilização de técnicas de linha de produto em sistemas embarcados críticos, no contexto do INCT-SEC

O conjunto de técnicas de inspeção em modelos de linha de produto de software e o estudo de viabilidade foram publicados na conferência SEKE (*Software Engineering and Knowledge Engineering*) em Cunha *et al.* (2012). Para o estudo de caso, apresentado neste trabalho, está sendo desenvolvido um artigo para *journal* para submissão no IET Software.

6.2. Trabalhos Futuros

A realização deste trabalho de pesquisa levou ao desenvolvimento de um conjunto de técnicas de inspeção para modelos de linha de produto de software. Os resultados desta pesquisa possibilitam os seguintes trabalhos futuros:

- A replicação e a execução de novos estudos experimentais que permitam aumentar a significância dos resultados obtidos e reduzir as ameaças à validade à conclusão desta pesquisa;

- A realização de um estudo experimental comparativo com a técnica de inspeção de modelos de *features* em Mello *et al.* (2012), a qual foi desenvolvida após a realização do mapeamento sistemático e paralelamente ao desenvolvimento da SPLIT; e

- A extensão da técnica proposta para que englobe outros artefatos de especificação de modelos de linha de produto de software, proposta por outras abordagens de LPS que não foram avaliadas neste trabalho de pesquisa.

REFERÊNCIAS BIBLIOGRÁFICAS

Asikainen, T.; Soininen, T.; Männistö, T.: "A Koala- Based Approach for Modelling and Deploying Configurable Software Product Families," in Software Product-Family Eng (PFE-5): Springer, pp. 225-249. 2004.

Asikainen, T.; Männistö, T.; Soininen T.: Kumbang: A domain ontology for modeling variability in software product families, Advanced Engineering Informatics, vol. 21, pp. 23-40, 2007.

Bachmann, F.; Goedicke M.; Leite, J; Nord R.; Pohl K.; Ramesh B.; Vilbig A.: "A Meta-model for Representing Variability in Product Family Development," in Software Product-Family Eng (PFE-5): Springer, 2004, pp. 66-80.

Bachmann, F.; Clements, P.: Variability in Software Product Lines, Software Engineering Institute, Pittsburgh, USA, Technical Report CMU/SEI-2005-TR-012, 2005.

Basili, V.R.; Rombach, H.D.: “The TAME project: towards improvement-oriented software environments", Software Engineering, IEEE Transactions on, vol.14, no.6, pp.758, 773. 1988.

Bayer, J.; Flege, O.; Knauber, P.; Laqua, R.; Muthig, D.; Schmid, K.; Widen, T.; DeBaud, J.: PuLSE: A Methodology to Develop Software Product Lines". In Proceedings of the 1999 Symposium on Software Reusability (SSR '99). ACM, New York, NY, USA, 122-131.

Carver, J.; Jaccheri, L.; Morasca, S.; Shull, F.: "Issues in Using Students in Empirical Studies in Software Engineering Education”, Software Metrics Symposium, 2003. Proceedings. Ninth International, vol., no., pp.239, 249, 3-5 Sept. 2003.

Chen L.; Babar, M. A.; Ali N.: Variability Management in Software Product Lines: A Systematic Review. In Proceedings of the 13th International

Software Product Line Conference (SPLC '09). Carnegie Mellon University, Pittsburgh, PA, USA, 81-90. 2009.

Clements, P.; Northrop, L.: “Software Product Lines: Practices and Patterns”. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 2001.

Conte, T.; Massollar, J.; Mendes, E.; Travassos, G.H.: "Usability Evaluation Based on Web Design Perspectives," Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on, vol., no., pp.146, 155, 20-21 Sept. 2007.

Cunha, R; Conte, T.; Almeida, E. S.; Maldonado, J. C.: “A Set of Inspection Techniques on Software Product Line Models”. In 24th International Conference on Software Engineering & Knowledge Engineering (SEKE 2012), Redwood City, USA. Skokie, USA: Knowledge Systems Institute Graduate School, 2012. v. 1. p. 657-662.

Cunha, R.; Conte, T.: S”PL Models Quality Assurance – a Mapping Study.” Technical Report USES-TR-2011-004. Disponível em: www.icomp.ufam.edu.br /uses (2011).

Denger, C; Kolb, R.: “Testing and inspecting reusable product line components: first empirical results”. In Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering (ISESE '06). ACM, New York, NY, USA, 184-193.

Dyba, T.; Kampenes, V.; Sjoberg, D.: “A Systematic Review of Statistical Power in Software Engineering Experiments”. Information and Software Technology 48 (8): 745-755. Elsevier, 2006.

Geppert, B.; Krueger, C.; and Trew, T;: Proceedings of the Second International Workshop on Software Product Line Testing (SPLiT 2005), Rennes, France, Sep. 2005.

Gomaa, H.: “Designing software product lines with UML: from use cases to pattern-based software architectures”. Boston: Addison-Wesley, 2005.

Halmans, G.; Pohl, K.: Communicating the variability of a software-product family to customers, Software and Systems Modeling, vol. 2, pp. 15-36, 2003.

Hernandes, E.; Zamboni, A.; Di Thommazo, A.: “Avaliação da ferramenta StArt utilizando o modelo TAM e o paradigma GQM”. Proceedings of 7th Experimental Software Engineering Latin American Workshop (ESELAW 2010), pp 30 – 39.

Heymans, P.; Trigaux, J. C.: “Software product line: state of the art”. Technical report for PLENTY project, Institut d.’Informatique FUNDP, Namur, 2003.

Hoek, A. V. D.: Design-time Product Line Architectures for Any-time Variability, Sci. Comput. Program., vol. 53, pp. 285-304, 2004.

Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos (INCT-SEC). Disponível em http://www.inct-sec.org/ (20/12/2012).

Kang, K.: “Feature-oriented domain analysis (FODA) - feasibility study”. Technical Report CMU/SEI-90-TR-21, SEI/CMU, Pittsburgh, 1990.

Kitchenham, B; Charters, S.: “Guidelines for performing Systematic Literature Reviews in Software Engineering”. EBSE Technical Report: EBSE-2007-01, 2007.

Kitchenham, B.; Budgen, D.; Brereton, P.: “The value of mapping studies – A participant-observer case study”. In: Proceedings of Evaluation and Assessment of Software Engineering - EASE´2010, Keele, UK, v. 56, pp.638-651.

Krueger, C.: "Variation Management for Software Production Lines," in Softw Product Lines (SPLC2): Springer, pp. 107-108. 2002.

Lee, J., Kang, S., Lee D.: “A Comparison of Software Product Line Scoping Approaches”. In International Journal of Software Engineering and Knowledge Engineering. Vol. 20, No. 5. 637-663, 2010.

Leite, J. C. S. D. P.; Doorn, J. H.; Hadad, G. D. S.; Kaplan, G. N.: "Scenario inspections." Requirements Engineering, v. 10, n. 1, pp. 1-21, 2005.

Mafra, S., Barcelos, R., Travassos, G. H., 2006. "Aplicando uma Metodologia Baseada em Evidência na Definição de Novas Tecnologias de Software". In: 20th Brazilian Symposium on Software Engineering (SBES 2006), v.1, pp.239–254.

Massen, T.; Lichter, H.H.: "Deficiencies in Feature Models". In Workshop on Software Variability Management for Product Derivation - Towards Tool Support, in conjunction with 3rd Software Product Line Conference (SPLC'04), 2004

Mello, R.M.; Teixeira, E.N.; Schots, M.; Werner, C.M.L.; Travassos, G.H.: "Checklist-Based Inspection Technique for Feature Models Review," Software Components Architectures and Reuse (SBCARS), 2012 Sixth Brazilian Symposium on, vol., no., pp.140-149, 23-28 Sept. 2012.

Muthig, D; Atkinson, C.: "Model-Driven Product Line Architectures," in Software Product Lines (SPLC2): Springer, 2002, pp. 79-90.

Oliveira Junior, E. A.; Gimenes, I. M. S.; Maldonado, J. C.: Systematic Management of Variability in UML-based Software Product Lines. Journal of Universal Computer Science, 16(17): 2374–2393. 2010.

Oliveira Junior, E. A.; Gimenes, I. M. S.; Maldonado, J. C.: Systematic Evaluation of Software Product Line Architectures. Journal of Universal Computer Science, 19(1): 25–52. 2013.

Pohl, K.; Böckle, G.; Van Der Linden, F.: "Software Product Line Engineering: Foundations, Principles and Techniques". Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Schmid, K.; John I.: A Customizable Approach to Full Lifecycle Variability Management, Sci. Comput. Program. vol. 53, pp. 259-284, 2004.

SEI. Software Engineering Institute. “A Framework for Software Product Line Practice, Version 5.0”. Disponível em: http://www.sei.cmu.edu/productlines/tools/framework (acessado em 20/12/2012).

Shull, F.; Rus, I.; Basili, V.R.: “How Perspective-Based Reading Can Improve Requirements Inspections”. Computer, vol.33, no.7, pp.73, 79, July 2000

Shull, F.; Carver, J.; Travassos, G.: “An Empirical Methodology for Introducing Software Processes”, Em: Proceedings of the Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9), pp. 288-296, 2001.

Sinnema, M.; Deelstra, S.; Nijhuis, J.; Bosch J.: "COVAMOF: A Framework for Modeling Variability in Software Product Families," in Software Product Lines (SPLC3): Springer, pp. 197-213. 2004.

Thiel, S.; Hein, A.: "Systematic Integration of Variability into Product Line Architecture Design," in Softw Product Lines (SPLC2): Springer, pp. 67-102. 2002.

Travassos, G. H.; Shull, F.; Fredericks, M.; Basili, V.: “Detecting defects in object-oriented designs: using reading techniques to increase software quality”. In Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '99), A. Michael Berman (Ed.). ACM, New York, NY, USA, 47-56.

Travassos, G.H.; Shull, F.; Carver, J.; Basili, V.R.: Reading Techniques for OO Design Inspections. University of Maryland Technical Report CS-TR-4353. April 2002 (version 3), http://drum.lib.umd.edu/bitstream/1903/ 1193/1/CS-TR-4353.pdf.

Webber, D.L.; Gomaa, H: Modeling variability in software product lines with the variation point model, Sci. Comput. Program., vol. 53, pp. 305-331, 2004.

Weiss, D.M; Lai, C.T.R.: Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M.C.; Regnell, B.; Wesslén, A.: "Experimentation in Software Engineering: An Introduction". Kluwer Academic Publishers, Norwell, MA, USA, 2000.

Apêndice A. Técnica de Inspeção para Modelos de Linha de Produto de Software (SPLIT)

1. Modelo de Features

Relacionamento	Tipo	Semântica	Notação
Relacionamento de Domínio	Mandatório	Se a *feature* pai é selecionada, a *feature* filha também deve ser selecionado	
	Opcional	Se a *feature* pai é selecionada, a *feature* filha pode ser selecionada	
	Alternativa	Se a *feature* pai é selecionada, exatamente uma *feature* filha deve ser selecionada	
	Inclusivas	Se a *feature* pai é selecionada, pelo menos uma da *feature* filha deve ser selecionada.	
Dependência	*Implicação*	*Se uma feature é selecionada, a feature implicada deve ser selecionada, ignorando a sua posição na árvore de feature*	
	Exclusão	*Indica que ambas as feature não podem ser selecionadas na mesma configuração de produto e que são mutualmente exclusivas*	

Instruções Iniciais

1. Leia o documento de Requisitos.
2. Identifique cada *feature* no formulário **Lista de *features*.**

2. Técnica 1 – Requisitos x Mapa de Produtos

Esta técnica tem como objetivo analisar o **documento de requisitos e o mapa de produtos** com o objetivo de encontrar defeitos que podem ser classificados como segue:

- **Redundância**: O mapa de produtos apresenta redundância quando gera produtos com as mesmas *features*.
- **Inconsistência**: O mapa de produtos apresenta inconsistência quando informações não correspondem ao documento de requisitos.

O *checklist* para cada classificação de defeito está disposto abaixo:

Tipo de Defeito	Item
Redundância	1.1. Existem **dois ou mais produtos** no mapa de produtos que apresentam o **mesmo conjunto de *features***?
Inconsistência	1.2. Existem **features** encontradas no documento de requisitos que **não estão contidas n o mapa de produtos**?
	1.3. Existem **features em um produto** que pode ser gerado pelo mapa de produtos que **não estão descritos no documento de requisitos**?
	1.4. Existe algum produto que pode ser gerado pelo mapa de produtos que não está descrito no documento de requisitos?
	1.5. Existem features em um **produto que não pode ser gerado pelo mapa de produtos** que estão **descritos no documento de requisito**?
	1.6. Existe alguma ***feature* caracterizada como obrigatória** no documento de requisito que **não está em todos os produtos do mapa de produtos**?

Instruções de Aplicação

1. Leia o Documento de Requisitos
2. Leia a Mapa de Produtos
3. Faça as perguntas do *Checklist*
4. Para cada resposta afirmativa a uma pergunta, registre o problema no Formulário de Relato de Defeitos

3. Técnica 2 – Requisitos x Mapa de Produtos x Modelo de *Features*

Esta técnica tem como objetivo analisar o **documento de requisitos, mapa de produtos e modelo de *features*** com o objetivo de encontrar defeitos que podem ser classificados como segue:

- **Inconsistência**: é caracterizada quando os artefatos (documento de requisitos, mapa de produtos e modelo de *features*) incluem informações contraditórias entre si.

O *checklist* para cada classificação de defeito pela Técnica 2 é:

Tipo de Defeito	Item
Inconsistência	2.1. Existe algum **produto** do mapa de produtos que **não pode ser gerado pelo modelo de *features***?
	2.2. Existe algum **relacionamento de implicação** descrito no documento de requisitos que **não está especificado no modelo de *features***?
	2.3. Existe algum **relacionamento de implicação** que está especificado no modelo de features mas **não está descrito no documento de requisitos**?
	2.4. Existe algum **relacionamento mutualmente exclusivo** que **não está especificado no modelo de *features***?
	2.5. Existe algum **relacionamento mutualmente exclusivo** que está especificado no modelo de features mas **não está descrito no documento de requisitos**?
	2.6. Existe alguma ***feature* obrigatória** no mapa de produtos e documento de requisitos que **não está especificada como mandatória no modelo de *features***?
	2.7. Existe alguma ***feature* opcional** no mapa de produtos e documento de requisitos que **não está especificada como opcional no modelo de features**?
	2.8. Existe algum **conjunto de *features* alternativas** no mapa de produtos e documentos de requisitos que **não está especificado** como alternativa no **modelo de *features***?
	2.9. Existe algum **conjunto de *features* inclusivas** no mapa de produtos e documentos de requisitos que **não está especificada** como inclusivas no **modelo de *features***?

Instruções de Aplicação

1. Leia o Documento de Requisitos.
2. Leia a Mapa de Produtos.
3. Leia o Modelo de *Features.*
4. Faça as perguntas do *checklist*
5. Para cada resposta afirmativa a uma pergunta, registre o problema no Formulário de Relato de Defeitos.

4. Técnica 3 – Modelo de Features

Esta técnica tem como objetivo analisar o **modelo de features** com o objetivo de encontrar defeitos que podem ser classificados de acordo com MASSEN e LICHTER (2004), como segue:

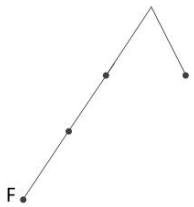
- **Redundância**: Um modelo de features contém redundância se pelo menos uma informação semântica é modelada de mais de uma maneira. Este tipo de defeito é considerado leve.

- **Anomalias**: Um modelo contém anomalia caso possíveis configurações estejam sendo perdidas, pela restrição da variabilidade ao utilizar features completamente obrigatórias. Este tipo de defeito é considerado médio.

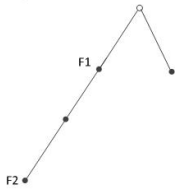
- **Inconsistência**: Um modelo contém inconsistência caso inclua informações contraditórias. Este tipo de defeito é considerado severo.

Para caracterização dos defeitos, [MASSEN e LICHTER, 2004] propões dois novos conceitos:

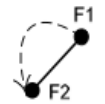
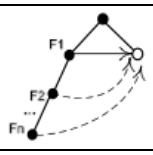
- Feature completamente obrigatória: é aquele em que todos os nós predecessores são obrigatórios

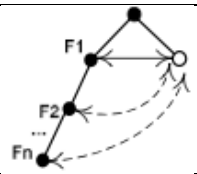
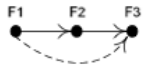
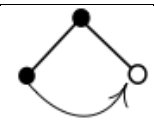
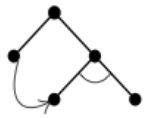
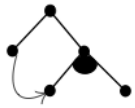
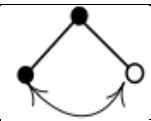
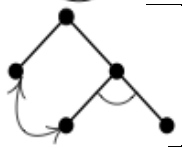
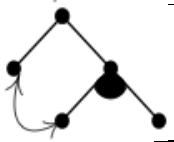
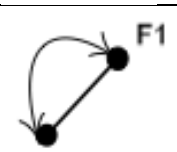
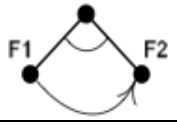


- Feature relativamente obrigatória: uma feature F2 é relativamente obrigatória para F1, se F2 é uma feature mandatória e todos os nós no caminho para F1 são mandatórios.



O *checklist* para cada classificação de defeito pela Técnica 3 esta disposto a seguir:

Tipo de Defeito	Item	
Redundância	3.1. Existe uma ***feature* completamente obrigatória** que apresenta uma **implicação**?	
	3.2. Existe uma ***feature* F2 relativamente obrigatória** para uma feature F1 e que apresenta uma **implicação**?	
	3.3. Existem **duas *features* filhas** de uma mesma feature pai que **são ao mesmo tempo alternativa e mutuamente exclusiva**?	
	3.4. Existe um ***feature* com relacionamento de implicação por múltiplas features** F1, ..., Fn, sendo que F1 é pai de F2 e que F2,...,Fn são **relativamente obrigatórias** para F1?	

	3.5. Existe um ***feature*** **com relacionamento mutualmente exclusivo** por múltiplas features F1, ..., Fn, sendo que F1 é pai de F2 e que F2,...,Fn s**ão relativamente obrigatórias** para F1?	
	3.6. Existe uma ***feature*** **F1** que tem um relacionamento de **implicação para F2 que tem uma implicação para F3**?	
Anomalia	3.7. Existe uma ***feature*** **opcional** que tem um relacionamento de **implicação por uma *feature* completamente obrigatória**?	
	3.8. Existe uma ***feature*** **que alternativa** em relação a *feature* pai e que tem um **relacionamento de implicação por uma *feature* completamente obrigatória**?	
	3.9. Existe uma ***feature*** **que inlcusiva** em relação a *feature* pai e que tem um **relacionamento de implicação por uma *feature* completamente obrigatória**?	
	3.10. Existe uma ***feature*** **opcional** que tem um relacionamento **mutualmente exclusivo com uma *feature* completamente obrigatória**?	
	3.11. Existe uma ***feature*** **alternativa** que tem um relacionamento **mutualmente exclusivo com uma *feature* completamente obrigatória**?	
	3.12. Existe uma ***feature*** **inclusiva** que tem um **relacionamento mutualmente exclusivo** com uma *feature* completamente obrigatória?	
Inconsistência	3.13. Existem **duas *features*** com relacionamento **mutualmente exclusivo** e que são completamente **obrigatórias**?	
	3.14. Existe uma ***feature*** **relativamente obrigatória** para a feature F1 e estas features apresentam um relacionamento **mutualmente exclusivo**?	
	3.15. Existem **duas *features* filhas alternativas** e que apresentam um relacionamento de **implicação**?	
	3.16. Existem **duas *features*** que apresentam **simultaneamente** um relacionamento de **implicação** e **mutualmente exclusivos**?	

Instruções de Aplicação

1. Leia o Modelo de *Features.*
2. Faça as perguntas do *checklist*
3. Para cada resposta afirmativa a uma pergunta, registre o problema no Formulário de Relato de Defeitos.

Apêndice B. Termo de Consentimento Livre e Esclarecido (Estudo de Viabilidade)

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO (TCLE)
ESTUDO DE VIABILIDADE

OBJETO DO ESTUDO: *Utilização de Técnica de Inspeção para Modelos de Linha de Produto de Software*

Profª. / Orientadora:	Tayana Conte D.Sc.	
Pesquisador:	Rafael Normando Cunha	**Data:** 22.06.2011
Participante:		

Prezado(a) Senhor(a),
Uma pesquisa sobre utilização de Técnica de Inspeção para Modelos de Linha de Produto de Software está sendo desenvolvido por um aluno do curso de Pós-graduação em Informática da Universidade Federal do Amazonas, sob a orientação da Profª. Tayana Conte. Você foi previamente selecionado pelo seu conhecimento na área de Engenharia de Software e está sendo convidado a participar desta pesquisa. A pesquisa tem por objetivo o uso de técnicas de inspeção com o propósito de verificar a conformidade de modelos de linha de produto de software. Essa pesquisa será feita a partir de dados coletados em trabalho prático. Embora o trabalho prático faça parte da disciplina, você tem o direito de **não** permitir a utilização dos dados do seu trabalho na pesquisa.

1) **Procedimento**
   O estudo será realizado com data e hora marcada com o grupo de participantes da disciplina de Tópicos em Engenharia de Software da graduação, sendo que serão formados dois grupos com o objetivo de realizar a inspeção de modelos de linha de produto de software. Para participar deste estudo solicito a sua especial colaboração em: (1) executar o roteiro de atividades elaborado para este estudo; (2) responder ao questionário de avaliação; (3) permitir que os dados coletados sejam analisados e utilizados na pesquisa.
2) **Tratamento de possíveis riscos e desconfortos**
   Serão tomadas todas as providências durante a coleta de dados de forma a garantir a sua privacidade e seu anonimato.
3) **Benefícios e Custos**
   Espera-se que, como resultado deste estudo, você possa aumentar seus conhecimentos sobre linha de produto de software, de maneira a contribuir para o aumento da qualidade das atividades com as quais você trabalhe ou possa vir a trabalhar. Este estudo também contribuirá com resultados importantes para a pesquisa de um modo geral na área de Engenharia de Software. Você não terá nenhum gasto ou ônus com a sua participação no estudo e também não receberá qualquer espécie de reembolso ou gratificação devido à autorização dos seus dados **na pesquisa**.
4) **Confidencialidade da Pesquisa**
   Toda informação coletada neste estudo é confidencial e seu nome não será identificado de modo algum, a não ser em caso de autorização explícita para este fim. Quando os dados forem coletados, seu nome será removido dos mesmos e não será utilizado em nenhum momento durante a análise ou apresentação dos resultados.
5) **Participação**
   Sua participação neste estudo é muito importante e voluntária, pois requer a sua aprovação para utilização dos dados coletados neste estudo. Você tem o direito de não querer participar ou de sair deste estudo a qualquer momento, sem penalidades. Em caso de você decidir se retirar do estudo, favor notificar o pesquisador(a) responsável.
   Os pesquisadores responsáveis pelo estudo poderão fornecer qualquer esclarecimento sobre o mesmo, assim como tirar dúvidas, bastando entrar em contato pelos seguintes endereços de e-mails:
   Pesquisador: Rafael Normando Cunha – rafael.cunha@gmail.com - (92) 9221-6115
   Especialista/Orientadora: Tayana Conte – tayanaconte@gmail.com
6) **Declaração de Consentimento**
   Declaro que li e estou de acordo com as informações contidas neste documento e que toda linguagem técnica utilizada na descrição deste estudo de pesquisa foi explicada satisfatoriamente, recebendo respostas para todas as minhas dúvidas. Confirmo também que recebi uma cópia deste Termo (TCLE), compreendo que sou livre para não autorizar a utilização dos meus dados neste estudo em qualquer momento, sem qualquer penalidade. Declaro ter mais de 18 anos e concordo de espontânea vontade em participar deste estudo.

Assinatura:		**Assinatura do Pesquisador:**

E-mail para contato:

Apêndice C. Especificação de Requisitos (Estudo de Viabilidade)

Descrição do sistema
O aplicativo tem como objetivo fornecer para o usuário uma forma de interagir com o serviço do twitter possibilitando ao mesmo que sejam executadas funcionalidades básicas como leitura de mensagens e criação de usuários; e funcionalidades mais avançadas como integração com outros serviços como readlater e twitpic.

A aplicação vai apresentar as seguintes versões:

- básica
- intermediário
- Avançada com suporte ao twitpic.
- Avançada com suporte ao yfrog
- Avançada com suporte ao Instapaper
- Avançada com suporte ao Readlater

Dessa maneira, busca-se adequar um produto para cada segmento de usuário.

1. Classificação dos requisitos

Os requisitos funcionais são classificados como obrigatório e opcional, como uma forma de prover aos desenvolvedores informações sobre quais casos de usos devem estar presente em todas as versões.

2. Requisitos funcionais

RF 1. Criação de usuário no serviço do twitter

Descrição do requisito	O usuário ao entrar no aplicativo deve poder criar um novo usuário para o serviço.
Classificação	Obrigatório
Entrada e pré-condições	O aplicativo deve receber como entrada o nome do usuário, senha e e-mail.
Saídas e pós-condições	
Produto	Básico Intermediário Avançado com suporte ao twitpic Avançado com suporte ao yfrog Avançado com suporte ao Instapaper Avançado com suporte ao Readlater

RF 2. Entrar no aplicativo com um usuário do twitter

Descrição do caso de uso	O usuário pode entrar no aplicativo utilizando um nome de usuário o qual tenha sido criado previamente, validando-o com uma senha
Classificação dos requisitos	Obrigatório
Entrada e pré-condições	O aplicativo deve receber como entrada o nome do usuário e a senha
Saídas e pós-condições	
Produto	Básico Intermediário Avançado com suporte ao twitpic Avançado com suporte ao yfrog Avançado com suporte ao Instapaper Avançado com suporte ao Readlater

RF 3. Leitura de mensagens na sua timeline

Descrição do caso de uso	O usuário após entrar na aplicativo poder ler as mensagens dos usuários os quais segue.
Classificação dos requisitos	Obrigatório
Entrada e pré-condições	
Saídas e pós-condições	
Produto	Básico Intermediário Avançado com suporte ao twitpic Avançado com suporte ao yfrog Avançado com suporte ao Instapaper Avançado com suporte ao Readlater

RF 4. Leitura de mensagens com suas menções

Descrição do caso de uso	O usuário pode ler as mensagens com menções a seus usuários
Classificação dos requisitos	Obrigatório
Entrada e pré-condições	
Saídas e pós-condições	
Produto	Básico Intermediário Avançado com suporte ao twitpic Avançado com suporte ao yfrog Avançado com suporte ao Instapaper Avançado com suporte ao Readlater

RF 5. Leitura de mensagens diretas

Descrição do caso de uso	O usuário pode ler as mensagens diretas para o seu nome de usuários
Classificação dos requisitos	Obrigatório
Entrada e pré-condições	
Saídas e pós-condições	
Produto	Básico Intermediário Avançado com suporte ao twitpic Avançado com suporte ao yfrog Avançado com suporte ao Instapaper Avançado com suporte ao Readlater

RF 6. Envio uma mensagem para o twitter

Descrição do caso de uso	O usuário pode enviar uma mensagem para o twitter para que seja disponibilizada na *timelime* dos seus seguidores
Classificação dos requisitos	Obrigatório
Entrada e pré-condições	
Saídas e pós-condições	
Produto	Básico Intermediário Avançado com suporte ao twitpic Avançado com suporte ao yfrog Avançado com suporte ao Instapaper Avançado com suporte ao Readlater

RF 7. Notificação de atualização da timeline

Descrição do caso de uso	O usuário é notificado no momento em que algum usuário que segue envia uma mensagem
Classificação dos requisitos	Opcional
Entrada e pré-condições	
Saídas e pós-condições	
Produto	Intermediário Avançado com suporte ao twitpic Avançado com suporte ao yfrog Avançado com suporte ao Instapaper Avançado com suporte ao Readlater

RF 8. Notificação de Atualização das menções

Descrição do caso de uso	O usuário é notificado no momento em que algum usuário menciona o seu nome de usuário
Classificação dos requisitos	Opcional
Entrada e pré-condições	
Saídas e pós-condições	
Produto	Intermediário Avançado com suporte ao twitpic Avançado com suporte ao yfrog Avançado com suporte ao Instapaper Avançado com suporte ao Readlater

RF 9. Notificação de atualização das mensagens diretas

Descrição do caso de uso	O usuário é notificado no momento em que algum usuário que segue envia uma mensagem direta
Classificação dos requisitos	Opcional
Entrada e pré-condições	
Saídas e pós-condições	
Produto	Intermediário Avançado com suporte ao twitpic Avançado com suporte ao yfrog Avançado com suporte ao Instapaper Avançado com suporte ao Readlater

RF 10. Integração com o serviço de fotos twitpic

Descrição do caso de uso	O usuário pode enviar fotos diretamente do aplicativo para o serviço de fotos twitpic utilizando o seu usuário do twitter
Classificação dos requisitos	Opcional
Entrada e pré-condições	O usuário deve escolher uma foto para se enviada para o twitpic O Requisito Funcional **Integração com o yofrog**(RF 11) não deve ser selecionado
Saídas e pós-condições	O usuário visualiza uma mensagem de confirmação de que a foto foi enviada
Produto	Avançado com suporte ao twitpic

RF 11. Integração com o yofrog

Descrição do caso de uso	O usuário pode enviar fotos diretamente do aplicativo para o serviço de fotos yfrog utilizando o seu usuário do twitter
Classificação dos requisitos	Opcional
Entrada e pré-condições	O usuário deve escolher uma foto para ser enviada para o yfrog
Saídas e pós-condições	O usuário visualiza uma mensagem de confirmação de que a

	foto foi enviada O Use Case **Integração com o serviço de fotos twitpic**(RF 10) não deve ser selecionado
Produto	Avançado com suporte ao yfrog

RF 12. Integração com o serviço readlater

Descrição do caso de uso	O usuário pode enviar o link de um tweet para o serviço readlater
Classificação dos requisitos	Opcional
Entrada e pré-condições	O tweet deve conter um link O Use Case **Integração com o serviço instapaper** (RF 13) não deve ser selecionado
Saídas e pós-condições	O usuário visualiza uma mensagem de confirmação de que o link foi enviado
Produto	Avançado com suporte ao Readlater

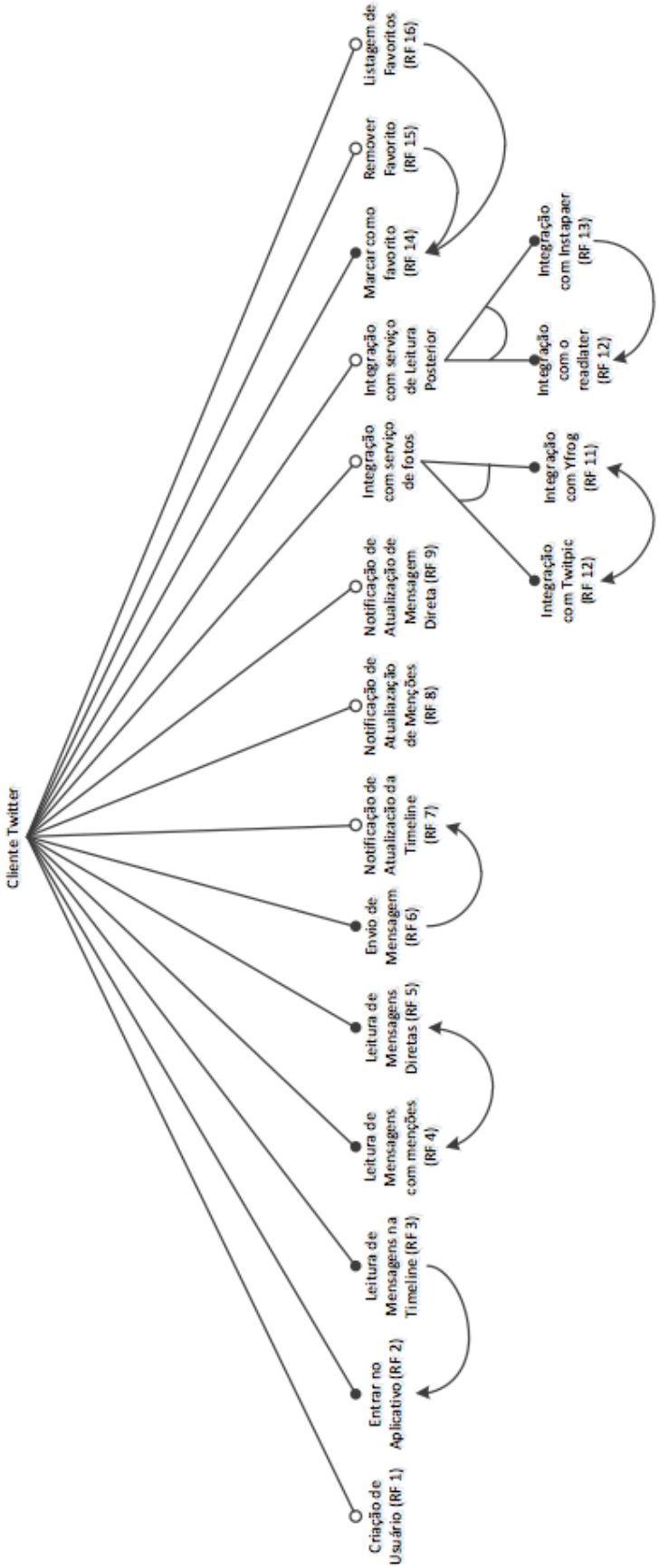
RF 13. Integração com o serviço instapaper

Descrição do caso de uso	O usuário pode enviar o link de um tweet para o serviço instapaper.
Classificação dos requisitos	Opcional
Entrada e pré-condições	O tweet deve conter um link
Saídas e pós-condições	O usuário visualiza uma mensagem de confirmação de que o link foi enviado O Use Case **Integração com o serviço readlater**(RF 12) não deve ser selecionado
Produto	Avançado com suporte ao Instapaper

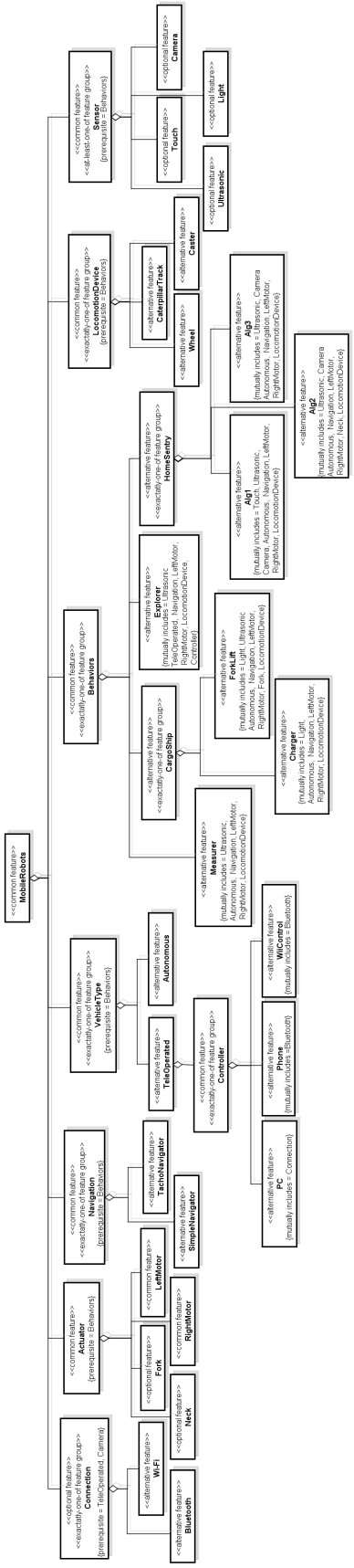
Apêndice D. Mapa de Produtos (Estudo de Viabilidade)

		Básico	Intermediário	Avançado com suporte ao serviço twitpic	Avançado com suporte ao serviço yfrog	Avançado com suporte ao serviço Instapaper	Avançado com suporte ao serviço readlater
RF 1	Criação de usuário no serviço do twitter	X	X	X	X	X	X
RF 2	Entrar no aplicativo com um usuário do twitter	X	X	X	X	X	X
RF 3	Leitura de mensagens na sua *timeline*	X	X	X	X	X	X
RF 5	Leitura de mensagens diretas	X	X	X	X	X	X
RF 6	Envio uma mensagem para o twitter	X	X	X	X	X	X
RF 7	Notificação de atualização da timeline		X	X	X	X	X
RF 8	Notificação de Atualização das menções		X	X	X	X	X
RF 9	Notificação de atualização das mensagens diretas		X	X	X	X	X
RF 10	Integração com o serviço de fotos twitpic			X			
RF 11	Integração com o yofrog				X		
RF 12	Integração com o serviço readlater					X	X
RF 13	Integração com o serviço instapaper						

Apêndice E. Modelo de Features (Estudo de Viabilidade)



Apêndice F. Modelo de Features do Estudo de Caso (Completo)



Apêndice G. Mapa de Produtos do Estudo de Caso (Completo)

Product	Sensor				Connection		Actuator			Navigation		VehicleType				LocomotionDevice		
	Ultrasonic	Touch	Light	Camera	Bluetooth	Wifi	Neck	Fork	RightMotor	LeftMotor	SimpleNavigator	TachoNavigator	Autonomous	Teleoperated Controller			Wheel	CaterpillarTrack	Caster
														PC	Phone	WiiControl			
Explorer1	x			x		x			x	x	x			x			x		
Explorer2	x			x		x			x	x		x		x			x		
Explorer3	x			x		x			x	x	x			x				x	
Explorer4	x			x		x			x	x		x		x				x	
Explorer5	x			x		x			x	x	x			x					x
Explorer6	x			x		x			x	x		x		x					x
Explorer7	x			x		x			x	x	x			x			x		
Explorer8	x			x		x			x	x		x		x			x		
Explorer9	x			x		x			x	x	x			x				x	
Explorer10	x			x		x			x	x		x		x				x	
Explorer11	x			x		x			x	x	x			x					x
Explorer12	x			x		x			x	x		x		x					x
Explorer13	x			x	x				x	x	x				x		x		
Explorer14	x			x	x				x	x		x			x		x		
Explorer15	x			x	x				x	x	x				x			x	
Explorer16	x			x	x				x	x		x			x			x	
Explorer17	x			x	x				x	x	x				x				x
Explorer18	x			x	x				x	x		x			x				x
Explorer19	x			x	x				x	x	x					x	x		
Explorer20	x			x	x				x	x		x				x	x		
Explorer21	x			x	x				x	x	x					x		x	
Explorer22	x			x	x				x	x		x				x		x	
Explorer23	x			x	x				x	x	x					x			x
Explorer24	x			x	x				x	x		x				x			x
Measurer1	x								x	x	x		x				x		
Measurer2	x								x	x		x	x				x	x	
Measurer3	x								x	x	x		x						x
Measurer4	x								x	x		x	x				x		
Measurer5	x								x	x	x		x					x	
Measurer6	x								x	x		x	x						x
Charger1			x						x	x	x		x				x		
Charger2			x						x	x		x	x					x	
Charger3			x						x	x	x		x						x
Charger4			x						x	x		x	x				x		
Charger5			x						x	x	x		x					x	
Charger6			x						x	x		x	x						x
ForkLift1	x		x				x	x	x	x		x	x				x		
ForkLift2	x		x				x	x	x	x	x		x					x	
ForkLift3	x		x				x	x	x	x		x	x						x
ForkLift4	x		x				x	x	x	x	x		x				x		
ForkLift5	x		x				x	x	x	x		x	x					x	
ForkLift6	x		x				x	x	x	x	x		x						x
HomeSentryAlg1a	x	x		x					x	x		x	x				x		
HomeSentryAlg1b	x	x		x					x	x	x		x					x	
HomeSentryAlg1c	x	x		x					x	x		x	x						x
HomeSentryAlg1d	x	x		x					x	x	x		x				x		
HomeSentryAlg1e	x	x		x					x	x		x	x					x	
HomeSentryAlg1f	x	x		x					x	x	x		x						x
HomeSentryAlg2a	x			x			x		x	x		x	x				x		
HomeSentryAlg2b	x			x			x		x	x	x		x					x	
HomeSentryAlg2c	x			x			x		x	x		x	x						x
HomeSentryAlg2d	x			x			x		x	x	x		x				x		
HomeSentryAlg2e	x			x			x		x	x		x	x					x	
HomeSentryAlg2f	x			x			x		x	x	x		x						x
HomeSentryAlg3a	x			x					x	x		x	x				x		
HomeSentryAlg3b	x			x					x	x	x		x					x	
HomeSentryAlg3c	x			x					x	x		x	x						x
HomeSentryAlg3d	x			x					x	x	x		x				x		
HomeSentryAlg3e	x			x					x	x		x	x					x	
HomeSentryAlg3f	x			x					x	x	x		x						x

Apêndice H. Termo de Consentimento Livre e Esclarecido (Estudo de Caso)

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO (TCLE)			
Profª. / Coordenadora:	Tayana Conte D.Sc.	**Grupo de Pesquisa:**	USES – Usabilidade e Engenharia de Software
Participante:			**Data:**
E-mail para Contato:			

Prezado(a) Senhor(a),

O grupo de pesquisa USES eventualmente realiza estudos experimentais para caracterizar/avaliar uma determinada tecnologia de software. Estes estudos são conduzidos por alunos de Pós-graduação em Informática da Universidade Federal do Amazonas (UFAM), sob a orientação da Profª. Tayana Conte. Você foi previamente selecionado pelo seu conhecimento/experiência e está sendo convidado a participar desta pesquisa. Essa pesquisa será feita com base em dados coletados a partir de trabalhos práticos. Embora o trabalho prático faça parte da disciplina, você tem o direito de **não** permitir a utilização dos dados do seu trabalho na pesquisa.

7) Procedimentos

O estudo será realizado com data e hora marcada com os participantes pré-selecionados. Para participar do estudo normalmente será aplicado um formulário de caracterização de perfil, a fim de identificar seu nível de conhecimento/experiência. Em seguida, o estudo será executado de forma individual ou em grupos formados, seguindo sempre o planejamento do estudo feito pelo pesquisador(a) responsável. Caso seja necessário, ao final do estudo será solicitado ao participante que responda um questionário de avaliação sobre a tecnologia de software que está sendo caracterizada/avaliada.

8) Tratamento de possíveis riscos e desconfortos

Serão tomadas todas as providências durante a coleta de dados de forma a garantir a sua privacidade e seu anonimato.

9) Benefícios e Custos

Espera-se que, como resultado deste estudo, você possa aumentar seus conhecimentos, de maneira a contribuir para o aumento da qualidade das atividades com as quais você trabalhe ou possa vir a trabalhar. Este estudo também contribuirá com resultados importantes para a pesquisa de um modo geral para o grupo de pesquisa USES. Você não terá nenhum gasto ou ônus com a sua participação no estudo e também não receberá qualquer espécie de reembolso ou gratificação devido à autorização dos seus dados **na pesquisa**.

10) Confidencialidade da Pesquisa

Toda informação coletada neste estudo é confidencial e seu nome não será identificado de modo algum, a não ser em caso de autorização explícita para este fim. Quando os dados forem coletados, seu nome será removido dos mesmos e não será utilizado em nenhum momento durante a análise ou apresentação dos resultados.

11) Participação

Sua participação neste estudo é muito importante e voluntária, pois requer a sua aprovação para utilização dos dados coletados neste estudo. Você tem o direito de não querer participar ou de sair deste estudo a qualquer momento, sem penalidades. Em caso de você decidir se retirar do estudo, favor notificar o pesquisador(a) responsável. Os pesquisadores responsáveis pelo estudo poderão fornecer qualquer esclarecimento sobre o mesmo, assim como tirar dúvidas.

Coordenadora do Grupo de Pesquisa USES: Profª. Tayana Conte – tayanaconte@gmail.com
Professor Responsável pela Pesquisa UEM: Edson A. Oliveira Junior - edson@din.uem.br
Pesquisador: Rafael Normando Cunha – rafael.cunha@gmail.com

12) Declaração de Consentimento

Declaro que li e estou de acordo com as informações contidas neste documento e que toda linguagem técnica utilizada na descrição deste estudo de pesquisa foi explicada satisfatoriamente, recebendo respostas para todas as minhas dúvidas. Confirmo também que recebi uma cópia deste Termo (TCLE), compreendo que sou livre para não autorizar a utilização dos meus dados neste estudo em qualquer momento, sem qualquer penalidade. Declaro ter mais de 18 anos e concordo de espontânea vontade em participar deste estudo.

_____ _____

Profa. Tayana Conte
Coordenadora do Grupo de Pesquisa USES

Assinatura do Participante

Apêndice I. Especificação de Requisitos (Estudo de Caso)

1. Descrição

Este documento tem como objetivo descrever os requisitos de uma linha de produto de software para um veículo robótico móvel, possibilitando ao mesmo que seja configurado para o fim a qual vai ser utilizado e, dessa forma, criar os seguintes produtos: Explorer1, Explorer10, Explorer11, Measurer1, Forklift1, HomeSentryAlg1a, HomeSentryAlg1f.

2. Requisitos

Esta seção enumera os requisitos funcionais que compõem a linha de produto de software.

2.1. Sensores

Os sensores tem como característica prover conhecimento do ambiente para o veículo robótico móvel, e com isso, possibilita a sua locomoção através da identificação de obstáculos em sua trajetória, realizando cálculos para que esta seja recalculada baseada na leitura dos sensores. Dessa maneira, é necessário que o veículo apresente pelo menos um dos sensores disponíveis:

2.1.1. Ultrassom

O sensor de ultrassom permite que o veículo robótico identifique obstáculo através de um sonar. Esta funcionalidade está presente nas seguintes especificações de produto:

- Explorer1
- Explorer10
- Explorer11
- Measurer1
- Forklift1
- HomeSentryAlg1a
- HomeSentryAlg1f

2.1.2. Contato

O sensor de contato permite que o veículo robótico identifique obstáculos através do contato ao encontrar objetos em sua trajetória, possibilitando que seja calculada uma nova trajetória para o veículo. Esta funcionalidade esta presente nas seguintes especificações de produto:

- HomeSentryAlg1a
- HomeSentryAlg1f

2.1.3. Luz

O sensor de luz permite que o veículo robótico detecte a pontos de luz, possibilitando que sua trajetória seja recalculada baseada na quantidade de luz que um objeto esteja emitindo. Esta funcionalidade esta presente nas seguintes especificações de produto:

- Forklift1
- HomeSentryAlg1f

2.1.4. Câmera

A câmera permite que sejam realizadas análises de imagens para identificação de obstáculos ao redor do veículo robótico, e com isso, seja realizado cálculos para determinar a trajetória desejada. Esta funcionalidade esta presente nas seguintes especificações de produto:

- Explorer1
- Explorer10
- Explorer11
- Measurer1
- HomeSentryAlg1a

2.2. Conexões

As conexões tem como objetivo prover ao veículo robótico conexões uma forma de comunicação para o envio de dados coletados e controle remoto. As conexões do veículos dependem que o mesmo apresente uma câmera para prover a coleta de dados baseada em imagem e que apresente uma forma de operação remota para controlar o veículo. E o veículo deve apresentar apenas uma das conexões disponíveis.

2.2.1. Bluetooth

A conexão por bluetooth permite que o veículo robótico envie dados e que seja controlado remotamente baseado no padrão Bluetooth versão 2.0. Esta funcionalidade está presente no produto:

- Explorer10

2.2.2. Wi-fi

A conexão por wi-fi permite que o veículo robótico envie dados e que seja controlado remotamente baseado no padrão de redes sem fio 802.11g definido pelo IEEE. Esta funcionalidade esta presente na seguinte especificação de produto:

- Explorer1

2.3. Tipos de Veículos

Os veículos robóticos podem divididos em dois tipos: os veículos operados remotamente e os veículos autônomos. O primeiro são aqueles que podem ser operados por usuários remotamente baseado em imagens adquiridas através da câmera que o veículo apresenta. Já o segundo são aqueles que não dependem de usuários, e com isso, deslocam-se de maneira autômona no ambiente. Para tal fim, é necessário que um funcionalidade de comportamento esteja selecionada. Os tipos de veículos são exclusivos entre si, e com isso, os produtos devem apresentar apenas um deles.

2.3.1. Autônomo

Os veículos autônomos não requerem operação remota por usuário, e com isso, dependem exclusivamente dos seus sensores para mapear o ambiente que o cerca com os obstáculos, para que sua rota seja calculada, com o objetivo de alcançar o fim para qual foi destinado. Esta funcionalidade está presente nas seguintes especificações de produto:

- Measurer1
- Forklift1
- HomeSentryAlg1a
- HomeSentryAlg1f

2.3.2. Operados remotamente

Os veículos operados remotamente são aqueles que requerem intervenção do usuário para que seu objetivo seja concluído. Os veículos operados remotamente podem ser categorizados de acordo com o tipo de controle que utilizam, sendo necessário, que apenas um deles esteja presente.

2.3.2.1. Controles

Os controles dos veículos operados remotamente devem ser selecionados de acordo com a finalidade para qual o veículo se destina, e com isso, prover ao usuário a maneira adequada para controle do veículo. É necessário que apenas um dos controles esteja presente:

2.3.2.1.1. Computador

O controle por computador permite que o veículo seja controlado remotamente por um computador por um usuário. Esta funcionalidade esta presente na especificação de produto:

- Explorer1

2.3.2.1.2. Telefone

O controle por telefone permite que o veículo seja controlado remotamente por um telefone celular por um usuário. Esta funcionalidade esta presente na seguinte especificação de produto:

- Explorer10

2.3.2.1.3. Controle do Wii

O controle do Wii permite que o veículo seja controlado remotamente por um controle do vídeo game Wii por um usuário, e com isso, os comandos podem ser enviados através de gestos realizado pelo usuário. Esta funcionalidade está presente nas seguintes especificações de produto:

- Explorer11

2.4. Dispositivos de Locomoção

Os dispositivos de locomoção tem como proposito prover ao veículo robótico uma forma de locomoção que deve ser escolhida de acordo com o objetivo e ambiente para qual este se destina. E cada veículo criado deve ter apenas um dos dispositivos de locomoção disponível.

2.4.1. Rodas

Este dispositivo de locomoção utiliza rodas para o deslocamento do veículo no ambiente. Esta funcionalidade está presente nas seguintes especificações de produto:

- Explorer1
- Measurer1
- Forklift1
- HomeSentryAlg1a

2.4.2. Trilho (Esteiras)

Este dispositivo de locomoção utiliza trilho (esteiras) para o deslocamento do veículo no ambiente. Esta funcionalidade está presente nas seguintes especificações de produto:

- Explorer10
- Explorer11

2.4.3. Rodas não direcionais

Este dispositivo de locomoção utiliza rodas não direcionais para o deslocamento do veículo no ambiente. Esta funcionalidade está presente na seguinte especificação de produto:

- HomeSentryAlg1f

2.5. Comportamento

O comportamento tem como proposito prover funcionalidades ao veículo robótico para que esta possa executar o propósito para qual foi criado. Dessa maneira, apenas um comportamento deve estar presente em cada veículo criado pela linha de produto de software.

2.5.1. Medidor

Esta funcionalidade permite que o veículo realize cálculo de distância utilizando o sensor de ultrassom. Dessa maneira, esta funcionalidade requer que o sensor de ultrassom esteja presente, assim como, que este seja autônomo. Esta funcionalidade está presente nas seguintes especificações de produto:

- Measurer1
- HomeSentryAlg1a

2.5.2. Transporte de Carga

O transporte de carga permite ao veículo transportar um objeto para realizar determinada atividade. Neste caso, o veículo pode transportar uma bateria externa ou transportar um objeto qualquer de um local a outro.

2.5.2.1. Carregador

Esta funcionalidade permite que o carregue a bateria de um outro objeto externo através de uma bateria que está acoplada ao veículo. Esta funcionalidade requer que o sensor de luz esteja presente, assim como, que este seja autônomo. Esta funcionalidade está presente na especificação de produto:

- HomeSentryAlg1f

2.5.2.2. Empilhadeira

Esta funcionalidade permite que o veículo transporte objeto de um local A para um local B através de um braço mecânico que segura o objeto no ponto A e o solta no ponto B. Esta funcionalidade requer que o sensor de luz esteja presente e que um tipo de dispositivo de locomoção esteja presente. Esta funcionalidade está presente na seguinte especificação de produto:

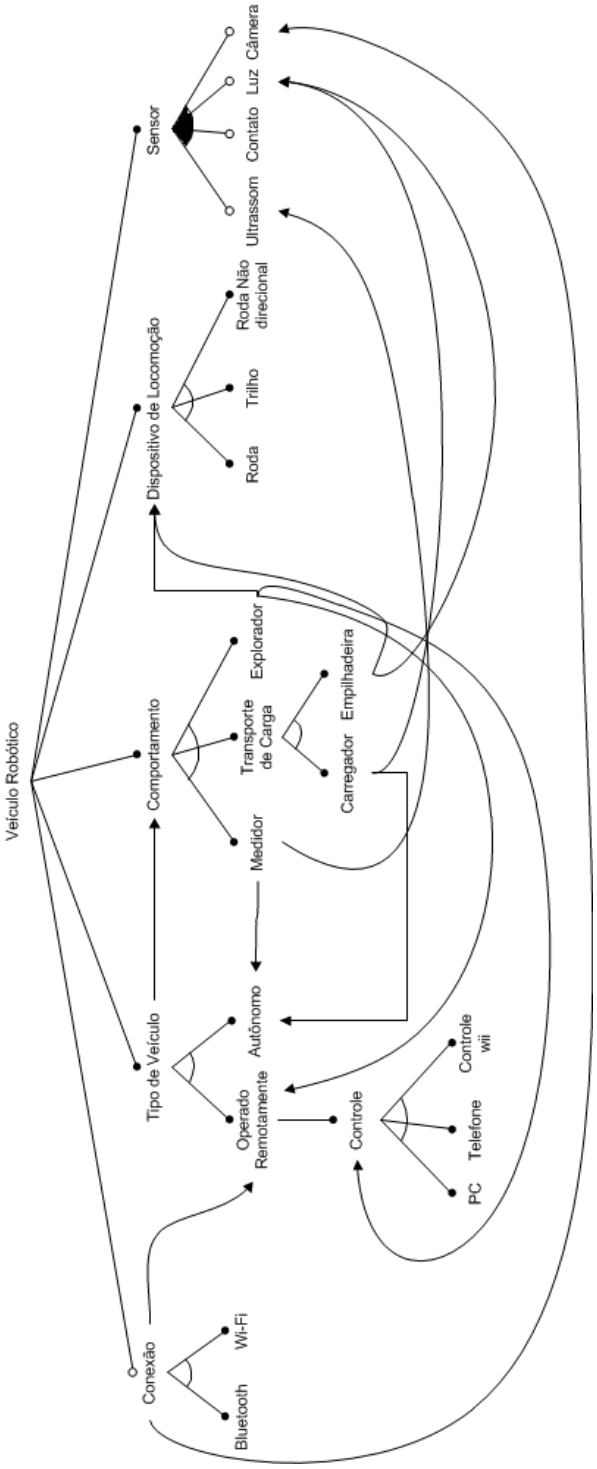
- Forklift1

2.5.3. Explorador

Esta funcionalidade permite que o veículo se desloque livremente com o objetivo de explorar à área pela qual vai se locomover. Esta funcionalidade requer que pelo menos um tipo de dispositivo de locomoção esteja presente. Esta funcionalidade está presenta nas seguires especificações de produto:

- Explorer1
- Explorer10
- Explorer11

Apêndice J. Modelo de Features do Estudo de Caso (Reduzido)



Apêndice K. Mapa de Produtos do Estudo de Caso (Reduzido)

	Explorer1	Explorer10	Explorer11	Measurer1	Forklift1	HomeSentryAlg1a	HomeSentryAlg1f
Ultrassom	X	X	X	X	X	X	
Contato						X	X
Luz					X		X
Câmera	X	X	X			X	
Bluetooth		X					
Wi-fi	X						
Autônomo				X	X	X	X
PC	X						
Controle do Wii			X				
Roda	X			X	X	X	
Trilho		X					
Roda não direcional			X				X
Medidor				X	X	X	
Carregador							X
Empilhadeira					X		
Explorador	X	X	X				