GUSTAVO ANDRADE DO VALE

AVALIAÇÃO DA MANUTENIBILIDADE DE SISTEMAS DE SOFTWARE DERIVADOS DE LINHAS DE PRODUTOS DE SOFTWARE

LAVRAS - MG
2013

GUSTAVO ANDRADE DO VALE

AVALIAÇÃO DA MANUTENIBILIDADE DE SISTEMAS DE SOFTWARE DERIVADOS DE LINHAS DE PRODUTOS DE SOFTWARE

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação para obtenção do título de Bacharel em Sistemas de Informação.

Área de Concentração:
Engenharia de Software

Orientador:
Prof. Dr. Heitor Augustus Xavier Costa

Coorientador:
Esp. Ramon Simões Abílio

LAVRAS - MG
2013

Ficha Catalográfica preparada pela Divisão de Processo Técnico da Biblioteca
Central da UFLA

Vale, Gustavo Andrade do

Avaliação da Manutenibilidade de Sistemas de Software Derivados de Linhas de Produtos de Software / Gustavo Andrade do Vale. Lavras - Minas Gerais, 2013. 79 p.

Monografia de Graduação - Universidade Federal de Lavras. Departamento de Ciência da Computação.

1. Linha de Produtos de Software. 2. Qualidade de Software. 3. Manutenção de Software. I. VALE, G. A. do. II. Universidade Federal de Lavras. III. Análise da Manutenibilidade de Sistemas de Software Desenvolvidos Utilizando Linhas de Produtos de Software.

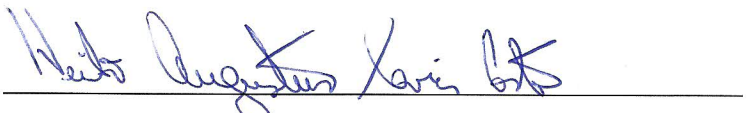
GUSTAVO ANDRADE DO VALE

AVALIAÇÃO DA MANUTENIBILIDADE DE SISTEMAS DE SOFTWARE DERIVADOS DE LINHAS DE PRODUTOS DE SOFTWARE

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação para obtenção do título de Bacharel em Sistemas de Informação.

APROVADA em 20 de Agosto de 2013

Prof. Dr. Eduardo Magno Lages Figueiredo	UFMG
Prof. Dr. André Pimenta Freire	UFLA
Esp. Ramon Simões Abílio	(Coorientador)



Prof. Dr. Heitor Augustus Xavier Costa
(Orientador)

LAVRAS
MINAS GERAIS - BRASIL

“Don’t Worry be Happy”

Bob Marley

“O sucesso é ir de fracasso em fracasso sem perder entusiasmo”

Winston Churchill

AGRADECIMENTOS

À compreensão de todos pelas ausências que muitas vezes nos impede de dar atenção ao que realmente vale a pena.

Às dificuldades que enfrentei se não fosse por elas, eu não teria saído do lugar. Agradeço as críticas auxiliaram na minha construção e aos conselhos que muitas vezes reduziram a caminhada.

A Jesus que sempre estiveste comigo, sempre soubeste de meus medos, sempre fortaleceste meus sonhos, sempre vigiaste meus passos, sempre me colocaste no colo… Sempre presente. Presente em mim, no meu amigo, nos meus mestres, nos meus pensamentos… Que, com a minha história, eu possa melhorar tua obra.

Ao meu pai obrigado por acreditar e incentivar os meus sonhos. À minha mãe por iluminar e facilitar meu caminho. A minha irmã por me ajudar em caminhos desconhecidos. A minha madrinha, que mesmo distante, me ajudou sempre que possível. Aos meus amigos Rotieh, Marcelo, Tico, Fred, Gian, Rafael, Alysson, Júlio, João Marcos, Bruno, Flávia, Verônica, Sarah e todos os amigos e conhecidos que me acompanharam em algum momento nessa jornada. Essa homenagem é pequena diante da grandeza do que fizeram por mim.

Agradeço também, meus tios, avós e demais familiares pelo apoio e incentivo. A minha namorada, Fernanda e a seus familiares pelo acolhimento e carinho. Aos meus mestres, que pela sua presença

marcaram minha vida e em um simples gesto ou até mesmo num olhar transmitiram palavras. Por tudo o que me ajudaram a ser, pela imensa alegria presente ao final dessa jornada, a vocês, o meu simples, mas eterno obrigado.

AVALIAÇÃO DA MANUTENIBILIDADE DE SISTEMAS DE SOFTWARE DERIVADOS DE LINHAS DE PRODUTOS DE SOFTWARE

RESUMO

Em uma Linha de Produtos de Software (LPS) pode-se desenvolver vários produtos com pequenas diferenças, chamado de variabilidade de produtos. Gerenciar variabilidade é tarefa difícil, assim como determinar o que vai ser reutilizável em um próximo produto. Realizar modificações em uma linha de produtos é mais complexo que realizar alterações em sistemas únicos, pois uma mudança pode causar impactos em vários produtos, e quando muitos módulos são modificados os testes devem ocorrer em diversos produtos. No entanto, a estratégia de linha de produtos de software tem se mostrado vantajosa em relação às demais técnicas de desenvolvimento. O presente trabalho tem por objetivo principal avaliar a manutenibilidade de sistemas de software gerados em uma LPS. Neste trabalho, são apresentados 7 critérios para se conseguir uma LPS com maior capacidade de manutenção. Associadas a cada critério, estão algumas diretrizes para auxiliar na manutenção do software. Esses critérios e diretrizes foram aplicados na LPS TankWar. A avaliação ocorreu em seis produtos e utilizou-se o "Índice de Manutenibilidade" como referência, que envolve outras medidas como o Volume de Halstead e a Complexidade Ciclomática. Os resultados foram positivos e o índice de manutenibilidade aumentou cerca de 13 pontos.

Palavras-chave: Linha de Produtos de Software, Desenvolvimento Orientado a Características, Manutenibilidade, Índice de Manutenibilidade.

MAINTAINABILITY EVALUATION OF SOFTWARE SYSTEMS DERIVED FROM SOFTWARE PRODUCT LINES

ABSTRACT

Software Product Lines (SPL) can generate several products with small differences. This characteristic is called product variability. Managing variability is as difficult as to determine what part of a software will be reusable another product. Making changes to a product line is not as simple as making changes to single systems, because a change can impact on various products. Besides, when many modules are modified the process of testing should occur in various products. However, when compared SPL and unique systems, SPL has proved advantageous over other techniques. This work aims at evaluating the maintainability of software systems generated from a SPL. In this work 7 criteria to achieve higher software maintainability are shown. Associated to each criterion are some guidelines to assist in the activities of software maintenance. These criteria and guidelines were applied to SPL TankWar. The evaluation involved six products and the Maintainability Index was used as reference. The index involves other measures such as Halstead's Volume and Cyclomatic Complexity. The results were positive and the maintainability index increased about 13 points.

Keywords: Software Product Line, Feature Oriented Development, Maintainability, Maintainability Index.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

Linha de Produtos de Software (LPS) consiste em uma abordagem para projeto e implementação de sistemas de software cujo objetivo é promover o reúso sistemático e em larga escala de componentes [Brownsword; Clements, 1996; Weiss; Lai, 1999; Clements; Northrop, 2002]. Esses componentes podem ser definidos como características (*features*) – permitem diferenciar os produtos e podem ser definidas como um módulo ou um conjunto de módulos de uma aplicação com funções coerentes, bem definidas, independentes e combináveis [Boucher *et al.*, 2010]. Uma vez que um sistema é decomposto em características, pode-se ter diferentes versões dele mediante a (re)combinação de certas características [Boucher *et al.*, 2010]. Entre os principais pontos a serem gerenciados em uma LPS estão as diferenças entre os produtos, conhecidas por variabilidades [Schobbens *et al.*, 2006]. Gerenciar variabilidade é difícil por causa da sua complexidade; para auxiliar esse gerenciamento, modelos de características foram criados os quais permitem modelar uma LPS. Esses modelos podem ser estruturados em forma de árvores e possuem diferentes tipos de características, operadores lógicos e restrições entre características, conhecidas como restrições transversais [Schulze *et al.*, 2012].

Para implementação de LPSs com a orientação a características, podem ser utilizadas tecnologias baseadas, por exemplo, em anotações ou em composições. Entre as tecnologias baseadas em composição, pode ser destacado o AHEAD (*Algebric Hierarchial Equations for Application Design*) [Batory, 2004]. Nessa tecnologia, o núcleo comum da LPS é tratado como constante e as características como funções de refinamento [Batory, 2004]. Como apoio para o desenvolvimento utilizando AHEAD, foi desenvolvido o AHEAD *Tools Suite* (ATS) [Batory, 2004]. No ATS, estão disponíveis ferramentas para

implementação e composição de características, como a linguagem Jak (superconjunto do Java), jampack e mixin (composição) e jak2java (transformação de código Jak para Java).

Um dos objetivos da indústria é desenvolver sistemas de software com qualidade, com o mínimo de erros, defeitos e falhas possível. No entanto, qualidade é um conceito subjetivo e dizer que um software tem ou não qualidade é tarefa difícil e envolve outros conceitos como o de qualidade interna e o de qualidade externa [ISO/IEC 25000, 2005]. Qualidade interna envolve requisitos que os desenvolvedores irão perceber, tais como, testabilidade, manutenibilidade, reusabilidade, adaptabilidade, escalabilidade e suporte. A qualidade externa é percebida pelo usuário final [Zanlorenci *et al.*, 2003]. A manutenibilidade é objeto de vários estudos, pois a manutenção de um sistema de software é inevitável [Lehman; Belady, 1985]. Da mesma forma, a manutenção de uma LPS se faz necessária e essencial, sendo mais complexa do que em sistemas comuns, pois a alteração em um módulo pode impactar em vários produtos. Sendo assim, ela merece atenção da academia, que pode auxiliar no desenvolvimento de LPS manuteníveis.

1.1. Motivação

Cada vez mais, clientes desejam sistemas de software em prazos mínimos, com baixo custo e que atendam necessidades específicas. Assim como produtos em geral, por exemplo, carros, bicicletas, roupas e sapatos, deixaram de ser produzidos de forma manual, o desenvolvimento de sistemas de software segue a tendência de ser cada vez mais automatizado e genérico. A estratégia de LPS se difere do modelo tradicional de desenvolvimento de software, pois combina desenvolvimento genérico e desenvolvimento customizado, o que resulta em sistemas desenvolvidos em larga escala.

A manutenção em LPSs se faz necessária, bem como em produtos desenvolvidos de forma tradicional. Realizar manutenção em uma LPS é ainda mais complexo, pois a alteração em um módulo pode impactar em vários produtos. LPSs têm sido estudadas, com mais intensidade nos últimos 13 anos, desde a criação da Conferência em Linha de Produtos de Software (*Software Product Line Conference* - *SPLC*[1]). Apesar disso, a quantidade de ferramentas e de diretrizes que ajudam na manutenção ou no aumento da capacidade de manutenção de uma LPS é reduzido. Isso indica a existência de um campo aberto para pesquisa relacionada à manutenibilidade em LPS.

1.2. Objetivo

Neste trabalho, o objetivo principal é avaliar o impacto da manutenção no código fonte das características dos produtos que compõem uma LPS, seguindo os critérios e diretrizes elaborados. Para atingir este objetivo, foram necessários atingir objetivos secundários:

- Definir critérios de manutenibilidade com caracterização dos possíveis problemas que o código fonte das características de uma LPS pode ter;
- Elaborar diretrizes de manutenibilidade com possíveis formas de tratar os problemas identificados;
- Identificar LPSs desenvolvidas com AHEAD;
- Pesquisar medidas para avaliar a manutenibilidade de um sistema de software;
- Pesquisar, instalar e testar ferramentas para medição e análise de código fonte escrito com a linguagem de programação Java;
- Realizar manutenção utilizando os critérios definidos e as diretrizes elaboradas de manutenibilidade.

[1] http:// www.splc.net/

1.3. Método de Desenvolvimento

A metodologia de pesquisa é um conjunto de métodos, técnicas e procedimentos cuja finalidade é viabilizar a execução da pesquisa que tem como resultado um novo produto, processo ou conhecimento [Jung, 2009].

1.3.1. Tipos de Pesquisa

Uma pesquisa pode ser classificada em (Figura 1-1) [Jung, 2009]:

- Quanto a Natureza: i) Pesquisa Básica (gerar conhecimento sem finalidade de aplicação) e ii) Pesquisa Aplicada (gerar conhecimento com finalidade de aplicação);
- Quanto aos Objetivos: i) Exploratória (descobrir/inovar); ii) descritiva (como?); e iii) explicativa (por que?);
- Quanto as Abordagens: i) Quantitativa e ii) Qualitativa;
- Quanto aos Procedimentos: i) Survey; ii) Pesquisa-Ação; iii) Estudo de Caso Único ou Múltiplos; iv) Operacional; e v) Experimental.

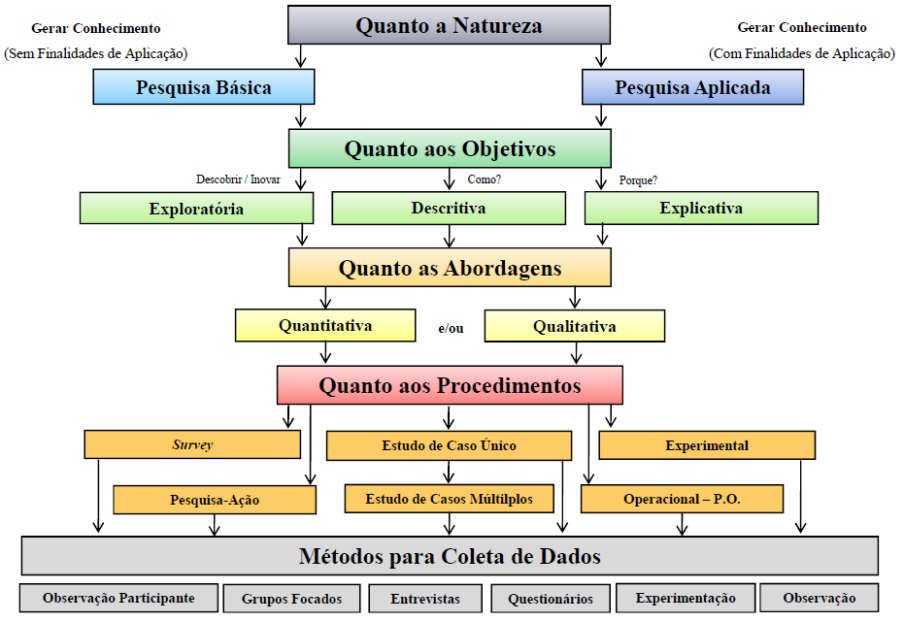


Figura 1-1 - Tipos de Pesquisa Científica [Jung, 2009]

Quanto à natureza, este trabalho classifica-se como uma **pesquisa aplicada**, pois o problema tratado possui finalidade de aplicação. Com relação aos objetivos, pode ser caracterizado como uma **pesquisa exploratória**, pois avalia a qualidade de sistemas de software com ênfase no requisito de manutenibilidade de uma LPS. Além disso, pode ser caracterizado como um trabalho com objetivos de **pesquisa descritiva**, visto que a estratégia de LPS é explicada detalhadamente, bem como o aumento da manutenibilidade de uma LPS por meio de critérios e de diretrizes de manutenibilidade. Quanto à sua abordagem, a deste trabalho é **qualitativa**, pois foi analisada a estrutura do código fonte e a qualidade dos comentários, e **quantitativa,** pois os resultados são medidos e avaliados utilizando medidas de software. Quanto aos procedimentos, pode ser caracterizado como um **estudo de caso**, pois os critérios e as diretrizes são utilizados em uma LPS. A coleta de dados é por meio

da **observação**, visto que um domínio foi modelado com a observação de seus requisitos necessários.

1.3.2. Procedimentos Metodológicos

O presente trabalho foi desenvolvido por meio de uma pesquisa bibliográfica em base de artigos, relatórios técnicos, monografias, dissertações e teses científicas e de reuniões periódicas com os orientadores. Foram pesquisados trabalhos relacionados ao tema abordado e utilizados para a fundamentação teórica para a realização deste trabalho.

Para escolher a LPS a ser estudada, foi realizada uma revisão da literatura utilizando parte da técnica Revisão Sistemática da Literatura [Kitchenham, *et al.*, 2009] para encontrar LPSs desenvolvidas com a tecnologia AHEAD. Dentre as LPSs encontradas, foi realizada uma análise de seu código fonte e a contagem da quantidade de linhas, quantidade de características, entre outras medidas. Com a análise desses dados, foi escolhida uma LPS para um estudo de caso. Em seguida, uma análise mais detalhada do código fonte foi realizada para melhor compreensão da LPS; com essa análise, conseguiu-se contabilizar a quantidade de linhas de código de cada método e a quantidade de linhas de comentários da LPS, por exemplo.

Em seguida, padrões de codificação, problemas estruturais de código fonte e métodos de refatoração foram estudados. Assim, problemas dirigidos à legibilidade e à complexidade da LPS estudada poderiam ser apontados (definição de critérios) e caminhos para solucionar/minimizar os problemas foram descritos (elaboração de diretrizes). Na etapa seguinte, uma pesquisa foi realizada em busca de medidas que permitissem medir a manutenibilidade de um sistema de software. Como o objetivo principal do trabalho não é criar uma medida optou-se por utilizar medidas existentes e, assim, o Índice de

Manutenibilidade (*Maintainability Index* - MI) foi escolhido. O MI tem sido amplamente utilizado na indústria e na academia, inclusive no contexto de LPS [Aldekoa *et al.*, 2008]. Para calcular o MI, é necessário obter o valor de outras medidas, como o Volume de Hastead e a quantidade de linhas de código. Assim, algumas ferramentas computacionais foram analisadas, instaladas e testadas para capturar o valor das medidas necessárias para o cálculo do MI.

A próxima etapa foi definir quais os produtos seriam analisados para então realizar a manutenção na LPS, ou seja, avaliar a LPS quanto aos critérios e utilizar as diretrizes quando o critério não fosse atendido completamente. Os critérios e as diretrizes foram elaborados a partir da detecção de possíveis problemas estruturais e da complexidade do código fonte das características da Linha de Produtos em estudo. Na próxima etapa, foram utilizadas as diretrizes em duas fases: i) a LPS legada foi replicada em projetos diferentes e, em cada replicação, um critério foi avaliado e as diretrizes foram utilizadas; e ii) os critérios foram avaliados e as diretrizes utilizadas de forma acumulativa sob a mesma LPS. A cada utilização das diretrizes no código fonte da LPS, os produtos foram testados para analisar se algum efeito colateral foi propagado. Quando algum efeito foi detectado, a refatoração foi desfeita e refeita ou buscou-se outro caminho para aplicá-la. Ao fim de cada fase, o valor de MI foi calculado e analisado e comparações e discussões foram realizadas.

1.4. Estrutura do Trabalho

Este trabalho está organizado da seguinte forma.

Alguns trabalhos relacionados são brevemente descritos no Capítulo 2.

Uma visão geral sobre linha de produtos de software é apresentada no Capítulo 3.

Definições e conceitos de qualidade de software são brevemente explicados no Capítulo 4.

Os critérios e as diretrizes são justificados e descritos no Capítulo 5.

O estudo de caso, a aplicação dos critérios e diretrizes e as discussões são descritos no Capítulo 0.

Conclusões, contribuições, limitações existentes e sugestões de trabalhos futuros são apresentadas no Capítulo 7.

2. TRABALHOS RELACIONADOS

A qualidade de software nos últimos 20 anos tem ganhado cada vez mais importância. Dentre os aspectos de qualidade, a característica "Manutenibilidade" tem se destacado e ganhado espaço nas empresas que desenvolvem sistemas de software. Essa característica tem sido estudada e avaliada desde as fases iniciais do processo de desenvolvimento de sistemas de software. Medidas e critérios para avaliação da manutenibilidade desses sistemas estão presentes na literatura e abordam, por exemplo, artefatos de código, projeto e modelos.

A mineração de dados foi utilizada com o objetivo de avaliar a manutenibilidade de um sistema de acordo com a norma ISO/IEC 9126 [Antonellis *et al.*, 2007]. Para atingir esse objetivo, foi proposta uma metodologia dividida em três etapas: i) elementos e métricas são extraídos do código fonte; ii) pesos foram atribuídos aos indicadores selecionados para referir sua importância na avaliação da manutenibilidade do sistema de acordo com a ISO/IEC 9126; e iii) foi aplicada uma técnica de agrupamento da mineração de dados (algoritmo de *k-means*) sobre os valores de manutenibilidade. Foram utilizadas nove métricas para verificar as cinco subcaracterísticas de manutenibilidade. A avaliação foi feita no Apache Geronimo[2], um servidor de aplicações de código aberto. Apesar da avaliação de ter sido realizada em Java, foi afirmado que essa metodologia pode ser aplicada em outras linguagens, tais como, C, C++, Cobol e Borland Delphi.

Um conjunto de procedimentos e diretrizes foi proposto com objetivo de avaliar a qualidade de sistemas [Aguayo *et al.*, 2005]. Esse conjunto consiste em

[2] http://geronimo.apache.org/

um *checklist* que avalia as características definidas na norma ISO/IEC 9126. Para avaliar a manutenibilidade, foi necessário um esquema em que são consideradas (i) uma avaliação da versão inicial e (ii) uma avaliação da versão modificada, entre as quais se estabelece um prazo às modificações. Após o conhecimento da versão inicial e das modificações no produto de software, a serem realizadas pelo mantenedor em tempo limitado, foi possível estimar a manutenibilidade do produto em avaliação. O tempo deve ser estabelecido de acordo com a quantidade de modificações recomendadas e com as exigências vigentes do mercado a que se destina o sistema. Essas modificações são estabelecidas por um método proposto chamado MEDE-PROS.

Nessa mesma linha, em um terceiro trabalho, um conjunto de critérios de manutenibilidade foi proposto para verificar e modificar o Modelo de Implementação [Santos, 2007]. Para isto, foram analisados sistemas de software manuteníveis orientados a aspectos para detectar informações que os faziam ter esta característica. O conjunto de critérios foi aplicado em um sistema de domínio bancário utilizando a linguagem AspectJ. Esse conjunto foi construído sob a luz da ISO/IEC 9126.

Seguindo essa mesma linha de raciocínio, um conjunto de critérios para auxiliar na manutenibilidade foi proposto com o objetivo de criar/avaliar metodologias que permitam o desenvolvimento de sistemas de software manuteníveis [Deraman; Layzell, 1995]. Os critérios foram divididos em duas categorias: i) primários, ajudam a determinar a facilidade de manutenção durante o tempo de vida do sistema, especialmente para manutenção adaptativa; e ii) secundários, auxiliam a determinar a qualidade do sistema produzido, o que ajuda a satisfazer os requisitos dos usuários, em especial dizem respeito a manutenção perfectiva e corretiva. Entre os critérios primários estão a modelagem do mundo real, a explicitação e a modularidade. Entre os critérios

secundários estão a uniformidade, a prototipação, o envolvimento de usuário e a documentação. Esses critérios foram utilizados para avaliar cinco metodologias para o desenvolvimento de sistemas.

Um conjunto de diretrizes foi criado com o objetivo de manter a qualidade de forma consciente na engenharia de LPS [Etxeberria *et al.*, 2008]. Neste estudo, são considerados cinco principais pontos para manter a qualidade de LPSs: i) qualidade na modelagem da variabilidade; ii) qualidade de projeto (*design*) ; iii) avaliação da arquitetura; iv) qualidade na implementação; e v) qualidade de teste. Após determinar esses pontos, 16 métodos são apresentados divididos por propriedades como objetivo, tipos de atributos e técnicas de avaliação. Pode-se concluir que existem vários métodos, mas existe uma lacuna entre uma abordagem e outra, tornando-se necessários estudos sobre o tema.

Um sistema de software orientado a objetos foi refatorado utilizando padrões de projeto e boas práticas de programação, tais como, o padrão *Factory*, padrão *Template*, padrão *Singleton*, codificação fluente e lógica de casos de uso [Ferreira, 2012]. Após essa refatoração, os dois sistemas de software, legado e novo, tiveram a manutenibilidade avaliada utilizando como referência o índice de manutenibilidade. Com os dados desse índice algumas análises foram feitas, com respeito a eficiência, a funcionalidade, a manutenibilidade, entre outras.

Os trabalhos relacionados encontrados abordam formas de avaliar a manutenibilidade de um sistema de acordo com a norma ISO/IEC 9126, seja utilizando técnicas, como mineração de dados [Antonellis *et al.*, 2007], propondo critérios [Aguayo *et al.*, 2005; Santos, 2007], ou trabalhos que não abordam a ISO/IEC 9126 nem a ISO/IEC 25000, seja critérios criados para incluir/avaliar a manutenibilidade em metodologias de desenvolvimento [Deraman; Layzell, 1995] e [Ferreira, 2012], ou o trabalho que objetiva manter a

qualidade de forma consciente na engenharia de LPS [Etxeberria *et al.*, 2008]. Dos trabalhos listados, apenas no penúltimo é apresentada uma avaliação em linhas de produtos de software, mas é sobre os métodos existentes sem tratar pontos específicos. O último utiliza o índice de manutenibilidade como referência para avaliar a manutenibilidade, medida utilizada também neste trabalho.

Com isso, este trabalho diferencia-se dos demais por se tratar da avaliação da manutenibilidade do código fonte de sistemas de software derivados de uma LPS, utilizar a ISO/IEC 25000, especificadamente pela ISO/IEC 25010, e avaliar sistemas de software desenvolvido em linguagem orientada a características. Além disso, realiza a avaliação em vários produtos de software derivados de uma LPS.

3. LINHAS DE PRODUTOS DE SOFTWARE

3.1. Considerações Iniciais

Desenvolver sistemas de software utilizando conceitos da estratégia de Linha de Produtos de Software (LPS) pode ser entendido como a produção de sistemas em larga escala de forma personalizada. Uma LPS consiste em analisar quais as funções que um sistema pode ter em determinado domínio e introduzir as possibilidades analisadas em uma plataforma. Com isso, pode-se desenvolver um sistema de forma "rápida" e com menos esforço, reutilizando ativos de sistemas desenvolvidos anteriormente, desde que eles pertençam a uma mesma família (possuam pontos em comum) [Long, 2002].

Entretanto, a estratégia de LPS não se limita apenas ao reúso de partes de software. A sua ideia básica é o trabalho sobre um grupo de sistemas que compartilham um conjunto comum e gerenciado de características (*features*). Essas características satisfazem necessidades específicas de um segmento e são desenvolvidas a partir de um aglomerado comum de artefatos reutilizáveis. Artefatos reutilizáveis abrangem os tipos de artefatos de desenvolvimento de sistemas, tais como modelos de requisitos, modelos de arquitetura, componentes de software e planos de teste. Esses artefatos devem ser reutilizados de forma consistente e sistemática a fim de construir aplicativos robustos [Silva *et al.*, 2011]. Embora a estratégia de LPS possua passos similares aos da Engenharia de Software clássica, ela é simplista e iterativa, pois enfatiza os processos: i) planejamento: aquisição dos ativos base; ii) desenvolvimento: equivalente a execução do PMBoK [PMBOK, 2013]; e iii) gerenciamento: responsável por manter a ordem e os prazos dos processos anteriores.

Produtos pertencentes a uma mesma linha/família devem ter características em comum entre os produtos pertencentes ao domínio modelado. Essas características em comum são chamadas de plataforma base. Outro ponto a ser gerenciado é a diferença entre produtos, conhecidas como variabilidades. Variabilidade está relacionada à LPS e chama a atenção da academia e da indústria por causa da complexidade do seu gerenciamento [Schobbens *et al.*, 2006]. Em geral, as variabilidades estão presentes em LPS para satisfazer necessidades de diferentes *stakeholders* [Pohl *et al.*, 2005]. Algo interessante é o gerenciamento em LPS envolver o desenvolvimento de ativos base podendo adicionar funções quando o produto estiver em desenvolvimento. Essa tarefa traz benefícios, pois agrega valor ao produto final, porém cuidados devem ser tomados para que o produto planejado não sofra modificações. É importante lembrar que, diferentemente do desenvolvimento tradicional, a estratégia de LPS trabalha com reusabilidade sempre que possível, portanto cabe aos responsáveis pelo planejamento detectar a real possibilidade de reúso para evitar atrasos e/ou fracassos.

Um breve histórico de LPS é apresentado na Seção 3.2. Aspectos importantes para a adoção de LPS são mostrados na Seção 3.3. Algumas vantagens e desvantagens em adotar a LPS são discutidas na Seção 3.4. O processo de desenvolvimento abordando a Engenharia de Domínio e a Engenharia de Aplicação é tratado na Seção 3.5. Atividades essenciais como desenvolvimento e gestão de ativos base e de produtos são consideradas na Seção 3.6. Abordagens de construção de uma LPS são explicadas na Seção 3.7. Breve conceitos de Orientação a Características e AHEAD são apresentados na Seção 3.8.

3.2. Histórico

A ideia de desenvolvimento baseado em reúso surgiu no final da década de 1960, próximo da criação de padrões para o desenvolvimento estruturado de software [Mcllroy *et al.*, 1969]. Entretanto, naquela época, muitos projetos incorriam em fracassos por tentar utilizar partes prontas porque o custo era superior ao desenvolvimento de projetos que iniciavam do zero. Com essa quantidade de fracassos, as empresas abandonaram a ideia de reúso; por outro lado, a academia estudava meios para que o reúso fosse viável e, em 1972, surgiu o conceito de "Família de Produtos" [Parnas, 1972]. Esse conceito ganhou destaque e, na década de 1980, surgiram ideias de Domínio de Aplicação e de Análise de Engenharia de Domínio, permitindo introduzir conceitos de Modelos de Características para análise de domínio [Neighbors, 1988].

Em 1990, a Análise de Domínio ganhou repercussão ligada a tecnologia de orientação a características. Essa união aparentou ser confiável e robusta tornando a notação FODA (*Feature-Oriented Domain Analysis*) mundialmente conhecida, sendo utilizadas em estudos relacionados à Análise de Domínio e à Modelagem de Características [Kang, 1990]. A Análise de Domínio e a Modelagem de Características tornaram-se parte fundamental do processo de desenvolvimento de uma LPS [Pohl *et al.*, 2005]. Assim como a maioria dos produtos deixaram de ser produzidos de forma manual, como carros e máquinas com a revolução industrial, pode-se pensar que a tendência da produção de software é ser cada vez mais genérica e em larga escala. Entretanto, na LPS, pode-se desenvolver de forma personalizada, o que faz com que vários produtos específicos sejam gerados de uma única plataforma, simultaneamente. Isso implica em uma produção em larga escala com produtos personalizados.

3.3. Aspectos Importantes

A adoção de LPS é diferente em alguns sentidos em relação a adoção de outros tipos de tecnologias e processos, pois requer bastante tempo, principalmente, em estágios iniciais [Long, 2002]. Essa adoção requer mudança na forma de desenvolver sistemas, deixando a ideia de sistema único para incorporar o desenvolvimento de vários produtos a partir de uma plataforma [Northrop, 2004]. Ela consiste em (i) ter ativos base, processos de suporte e estruturas organizacionais, (ii) desenvolver produtos a partir de ativos base de forma a atingir os objetivos de negócio e (iii) instituir mecanismos para melhorar e estender o esforço de produção de sistemas que façam sentido [Silva *et al.*, 2011].

No entanto, dependendo do cenário, atingir as metas de adoção podem se tornar uma atividade complexa. Para evitar complexidades adicionais durante a fase de adoção, uma organização que pretende investir em LPS deve ter objetivos claros [Bühne *et al.*, 2003]. Para atingir seus objetivos, a organização deve selecionar uma ou mais estratégias de adoção que especificam como os objetivos serão abraçados às práticas de LPS [Silva *et al.*,2011]. Neste sentido, adotar LPS torna-se individualizado para a organização, o contexto específico parece desempenhar papel significativo na decisão de adoção [Bühne *et al.*, 2003]. Com isso, o processo deve ser sistêmico e planejado para obter sucesso.

3.4. Vantagens e Desvantagens

Em dois trabalhos foram encontrados vantagens da estratégia de linha de produtos de software. Em um deles [Silva *et al.*, 2011], a adoção de LPS pode trazer benefícios (i) Organizacionais, (ii) de Engenharia de Software e (iii) de Negócio (Tabela 3-1). Em outra visão [Cohen, 2003], os benefícios são (i) Tangíveis (Tabela 3-2), benefícios que podem ser medidos diretamente, por

exemplo, redução do *time-to-market* e redução de defeitos, e (ii) Intangíveis (Tabela 3-3), benefícios que os desenvolvedores relatam e não são fáceis de serem mensurados, por exemplo, satisfação do cliente e satisfação dos desenvolvedores. Em ambas as visões, os benefícios podem ser relacionados, porém não é trivial realizar essa relação, pois alguns do benefícios da segunda visão podem implicar em mais de um benefício da primeira visão. Por exemplo, o benefício tangível **Produtividade** implica em benefícios de **Engenharia de Software**, **Organizacional** e **de Negócio**.

Tabela 3-1 - Benefícios Organizacionais, de Engenharia de Software e de Negócio (Fonte: [Cohen, 2003])

Benefício	Descrição
Organizacionais	Envolvem atividades relacionadas a organização, tais como, produtos com qualidade, facilidade em treinar pessoas, melhor compreensão de um domínio. Essas atividades resultam em serviços com qualidade elevada e confiança e fidelidade dos clientes.
Engenharia de Software	Possuem vantagens direcionadas a produção de sistemas de software, tais como, facilidade no reúso de requisitos e de seus componentes, melhor análise de requisitos, facilidade em decidir a comunalidade dos produtos, estabelecimento de padrões de programação e controle de qualidade.
Negócio	Estão ligados à produtividade e ao lucro da organização. Quando as LPS são bem planejadas e estruturadas, os processos são agilizados, a receita da empresa pode ser aumentada e o tempo de produção é reduzido, por exemplo. Essas atividades implicam em eficiência de negócios.

Tabela 3-2 - Alguns Benefícios Tangíveis (Fonte: [Cohen, 2003])

Benefício	Descrição
Lucratividade	Os ativos base permitem que a organização produza produtos para um segmento específico de mercado. O benefício é observado no aumento da participação de mercado e da lucratividade da organização.
Qualidade	Redução na quantidade de defeitos reportados é comum em sistemas desenvolvidos em LPS. A qualidade pode ser mensurada em termos de redução do tempo para reparo e da redução do efeito da geração de novos defeitos a partir de correções executadas.
Performance	A utilização de ativos base aumenta a performance em relação ao desenvolvimento tradicional, especialmente com o aumento da maturidade da linha, o que faz com que os ativos base estejam cada vez mais eficientes para aplicar em algoritmos otimizados.

Tabela 3-2 - Alguns Benefícios Tangíveis (Fonte: [Cohen, 2003]) (cont.)

Benefício	Descrição
Tempo de integração	O tempo de integração no desenvolvimento incremental é completado mais rápido do que em porções sem ativos ou um aplicativo.
Volume de Código	A quantidade de objetos projetados para subsistemas utilizando ativos base é menor do que os estimados para sistemas individuais, resultando em redução semelhante no tamanho real de código fonte.
Produtividade	A equipe de desenvolvimento, o custo total e o cronograma são reduzido. O sistema possui flexibilidade nos atendimentos das solicitações de modificações dos clientes.

Tabela 3-3 - Alguns Benefícios Intangíveis (Fonte: [Cohen, 2003])

Benefício	Descrição
Desgaste de profissionais	Menor desgaste dos profissionais, o que resulta em uma redução do volume de negócios de membros da equipe.
Aceitabilidade dos desenvolvedores	Após um treinamento inicial, os desenvolvedores relatam satisfação em trabalhar com a abordagem baseada em ativos base e arquitetura comuns.
Satisfação Profissional	Desenvolvedores reportam que o trabalho braçal já foi realizado (desenvolvimento de ativos base), assim eles podem se concentrar em atividades mais interessantes, como o aperfeiçoamento e/ou inovação de elementos específicos.
Satisfação do Cliente	Os ativos base reduzem os riscos, aumentando a previsibilidade da entrega e diminuindo a taxa de defeitos. Esses fatores afetam positivamente o cliente introduzindo-o a preferir produtos derivados de uma LPS.

Para enfatizar algumas vantagens na adoção de LPS, dois exemplos de sucesso são apresentados: i) da Cummins Inc.; e ii) da CelsusTech. Em 2002, a Cummins Inc. era a maior produtora de motores diesel para veículos, de pequenos caminhões a equipamentos de grande porte, tais como, equipamentos de perfuração, locomotivas e guinchos. Essa empresa foi capaz de reduzir o tempo de produção de sistemas de software para um motor diesel de cerca de um ano para aproximadamente uma semana, pois 75% dos novos projetos provinham de ativos existentes [Clements *et al.*, 2003]. A CelsusTech é uma empresa subcontratada da marinha sueca para o desenvolvimento de sistemas para embarcações militares. Ao fim da implantação dos sistemas, a empresa constatou que a participação do software no custo total foi reduzido de 65% para

20%. A mão de obra envolvida no desenvolvimento de software reduziu uma equipe de 200 pessoas para menos de 50 pessoas. O tempo de entrega passou de anos para poucos meses. Entre 70% e 80% dos sistemas eram compostos de componentes do repositório de ativos. Além disso, houve aumento na qualidade dos sistemas e na satisfação dos clientes [Brownsword; Clements, 1996].

Uma das desvantagens em utilizar LPS é a mudança nos procedimentos e nos padrões de desenvolvimento de sistemas, pois essa mudança envolve a questão da resistência das pessoas em aprender algo novo, o que pode levar projetos ao fracasso. Outra desvantagem relacionada a mudanças é o fato de poucos engenheiros de software possuírem visão global sobre a arquitetura da LPS. A adoção de LPS requer um especialista que conheça o domínio da aplicação. Esse especialista deve ter autonomia, responsabilidade e experiência em lidar com pessoas e dentro da empresa [Cohen, 2002].

Os custos acumulados, quando são considerados o desenvolvimento tradicional e o desenvolvimento com LPS, são apresentados na Figura 3-1. Nessa figura, pode-se perceber que o planejamento e a gestão em LPS devem ser a longo prazo. De fato, investimentos em LPS terão retorno em médio prazo, pois em média três produtos precisam ser desenvolvidos para que a vantagem real possa ser vista [Kernighan; Ritchie, 2002]. Projetar arquiteturas em LPS torna-se difícil, por causa da falta de parâmetros para avaliar as arquiteturas e desenvolver e explorar LPS. Fundamentado nas possíveis dificuldades na adoção de LPS, em um estudo [Kernighan; Ritchie, 2002], um questionário foi utilizado, em cuja compilação das respostas, pode-se notar que a resistência organizacional gerada pela adoção de LPS é o fator mais crítico dentre outros problemas (Tabela 3-4).

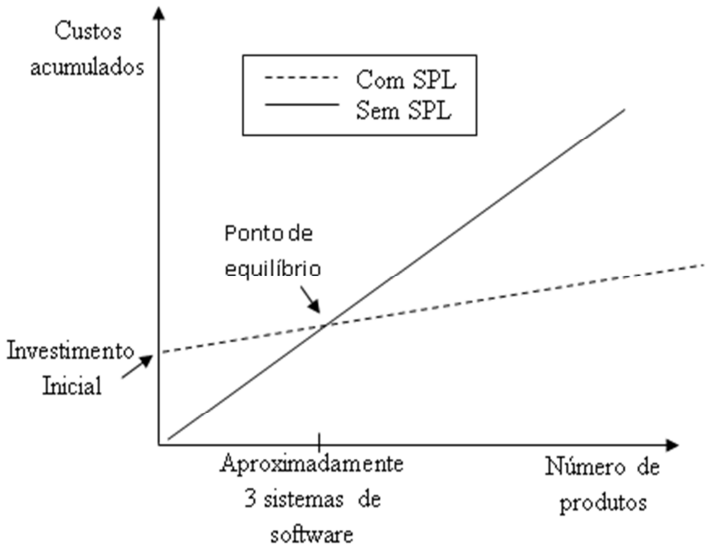


Figura 3-1 - Custos do Desenvolvimento Com e Sem LPS (Fonte: [Pohl *et al*., 2005; Clements; Northrop, 2002])

Tabela 3-4 - Problemas na Adoção de LPS (Fonte: [Kernighan; Ritchie, 2002])

Problemas	Índice
Resistência Organizacional	52%
Resistência da Gestão	36%
Resistência dos Desenvolvedores	32%
Preocupações com Grandes Investimentos	45%
Falta de Pessoal Devidamente Treinado	29%
Incapacidade de medir o impacto	19%
Preocupação com o tempo de um projeto	18%

3.5. Processos da LPS

Uma LPS está organizada em dois processos (Figura 3-2) [Pohl *et al.*, 2005]: i) Engenharia de Domínio; e ii) Engenharia de Aplicação. Essa organização implica na diferenciação dos objetivos para construir uma plataforma robusta e para construir aplicações específicas, respectivamente, em curto espaço de tempo. Esses processos devem interagir para que ambos beneficiem-se e tornem o desenvolvimento eficaz.

Na Engenharia de Domínio, são definidos a plataforma de reutilização, os pontos em comum (comunalidade) e a variabilidade da LPS. A plataforma refere-se aos artefatos de software (por exemplo, requisitos, *design* e teste) - chamados de ativos base. Na Engenharia de Aplicação, são derivadas aplicações concretas a partir da plataforma definida na Engenharia de Domínio. Ela assegura a correta instanciação e explora a variabilidade da LPS de acordo com as necessidades da aplicação final. Esses processos seguem quatro passos: i) Análise; ii) Arquitetura; iii) Implementação; e iv) Testes. Uma análise sobre o potencial reúso realizada após a definição dos requisitos do produto e do plano de negócio, o que permite iniciar os processos.

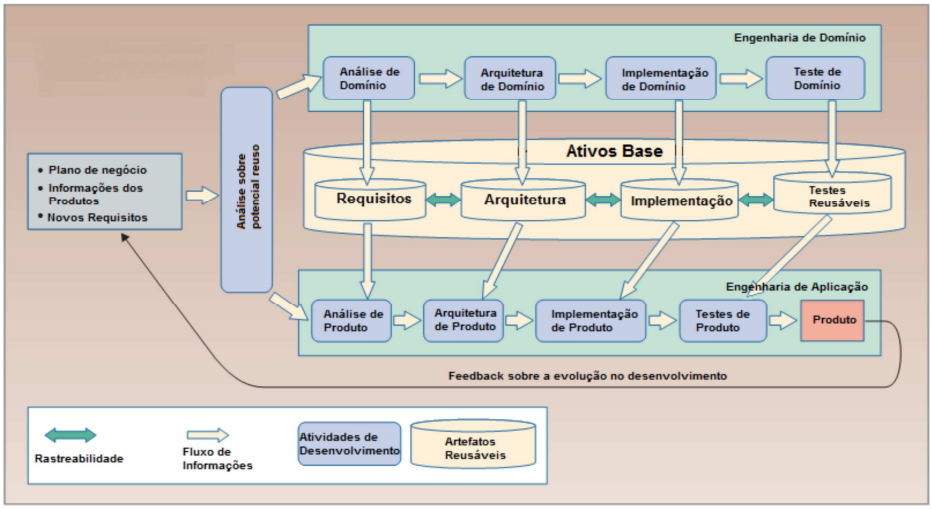


Figura 3-2 - Ciclo de Vida: Engenharia de Domínio e Engenharia de Aplicação (Fonte: [Silva *et al.*, 2011])

3.6. Atividades Essenciais

Em LPS, há três atividades essenciais e altamente interativas que se misturam às práticas de negócio e à tecnologia (Figura 3-3) [Linden *et al.*, 2007; Long, 2002]: i) Desenvolvimento de Ativos Base; ii) Desenvolvimento de Produtos; e iii) Gerenciamento.

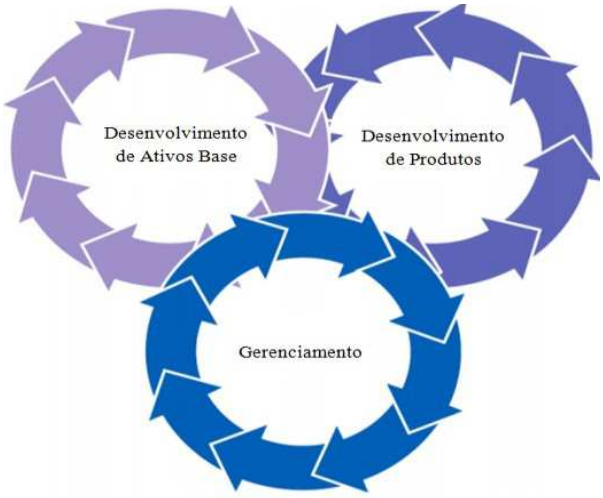


Figura 3-3 - Atividades Essenciais (Fonte: [Long, 2002])

3.6.1. Desenvolvimento de Ativos Base

O Desenvolvimento de Ativos Base (Figura 3-4) é uma atividade na forma de ciclo que resulta em (i) Escopo da Linha de Produção, (ii) Ativos Base e (iii) Plano de Produção. Essas três saídas do desenvolvimento de ativos base em conjunto compõem a plataforma da LPS [Linden *et al.*, 2007] e fornecem suporte ao desenvolvimento da LPS. Por isso, é importante eles serem bem planejados. O objetivo desta atividade é definir os aspectos comuns e a variabilidade da LPS.

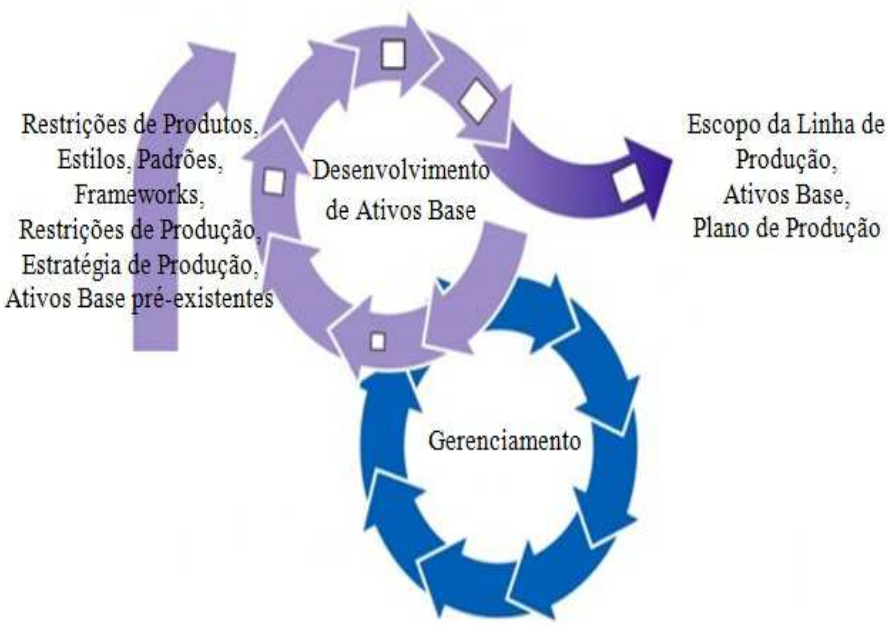


Figura 3-4 - Atividade: Desenvolvimento de Ativos Base (Fonte: [Long, 2002])

As setas rotatórias sugerem a não existência de um momento específico para adicionar uma restrição ou novos padrões no desenvolvimento. Entradas como restrições transversais, estilos e padrões afetam diretamente as saídas dessa atividade. Em alguns segmentos, os produtos existentes são a base para os Ativos Base; em outros, estes podem ser desenvolvidos do "zero" para futuro reúso. Ativos base incluem, mas não estão limitados à, arquitetura e sua documentação, especificações, componentes de software, ferramentas como geradores de componentes ou aplicação, modelos de desempenho, cronogramas, orçamentos, planos de teste, casos de teste, planos de trabalho e processo descrições [Long, 2002].

Embora possa ser possível a criação de Ativos Base a serem utilizados nos produtos sem quaisquer adaptações, em muitos casos, algumas adaptações são necessárias para torná-los mais utilizáveis no contexto mais amplo de uma LPS.

Logo, mecanismos de variação dos principais Ativos Base utilizados ajudam a controlar as adaptações necessárias e suportar as diferenças entre os sistemas [Bachmann; Clements, 2005]. Essas adaptações devem ser planejadas antes do desenvolvimento e facilitadas para a equipe de desenvolvimento sem colocar em risco as propriedades existentes dos Ativos Base.

3.6.2. Desenvolvimento de Produtos

Nessa atividade (Figura 3-5), os sistemas são desenvolvidos a partir dos Ativos Base, utilizando o plano de produção para satisfazer as exigências da LPS. Os insumos essenciais dessa atividade são (i) Requisitos, (ii) Escopo da Linha de Produtos, (iii) Ativos Base e (iv) Plano de Produção [Ahmed *et al.*, 2009]. De posse do plano de produção, que detalha como os Ativos Base serão utilizados para construir um sistema, o engenheiro de software pode montar as partes da LPS. A atividade de desenvolvimento de produtos possui como entrada a saída da atividade de desenvolvimento de Ativos Base. A atividade de desenvolvimento de produtos geram produtos. Se bem gerenciados, esses produtos serão condizentes com os planejados na atividade anterior.

As setas de rotação indicam iterações entre as partes envolvidas. Por exemplo, a existência e a disponibilidade de um determinado produto pode afetar os requisitos de produtos subsequentes. Outro exemplo, um produto, que tem comunalidades previamente não reconhecidas em relação a outro produto da LPS, criará a necessidade de atualizar os Ativos Base e fornecerá fundamento para explorar essa comunalidade em futuros produtos [Long, 2002]. Além disso, deve-se ter *feedback* sobre quaisquer problemas ou deficiências encontradas com os Ativos Base.

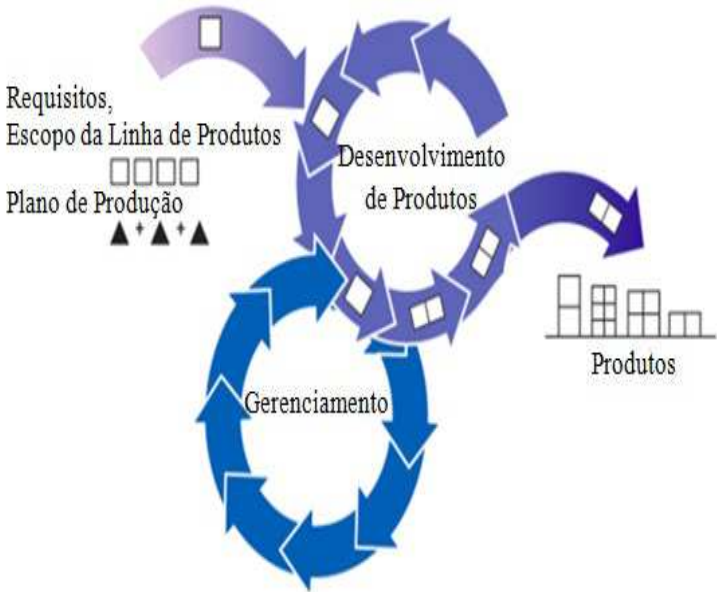


Figura 3-5 - Atividade: Desenvolvimento de Produtos (Fonte: [Long, 2002])

3.6.3. Gerenciamento

Na atividade Gerenciamento, são realizadas tarefas vitais (técnicas e organizacionais) ao sucesso da institucionalização da LPS em uma organização, pois é fornecida e coordenada a infraestrutura necessária [Ahmed *et al.*, 2009]. Nessa atividade, são supervisionadas a construção dos Ativos Base e as atividades para o desenvolvimento do sistema, a fim de garantir que os grupos que constroem os Ativos Base e os que constroem os sistemas estejam plenamente envolvidos nas atividades individuais, acompanhando o processo definido e o progresso da LPS [Long, 2002].

O conjunto de Ativos Base e o plano de como eles são utilizados para construir os sistemas não são gerados sem previamente estudar o ambiente e caracterizar o negócio. Portanto, deve existir investimento organizacional. A gestão deve dirigir, controlar e garantir a plena utilização dos Ativos Base.

Nessa atividade, o gerenciamento está mais relacionado às práticas de negócios do que às práticas técnicas [Long, 2002].

3.7. Abordagens de Construção de LPS

Existem três abordagens principais de desenvolvimento em LPS (Figura 3-6) [Chen *et al.*, 2005; Krueger, 2001]: i) Proativa; ii) Reativa; e iii) Extrativa. Na abordagem Proativa, o trabalho realizado é no desenvolvimento dos Ativos Base como primeiro passo para o desenvolvimento da LPS. Essa abordagem considera os sistemas a serem gerados previamente, fazendo um planejamento inicial completo. Na abordagem Reativa, é iniciado o desenvolvimento, considerando os Ativos Bases, e uma versão da LPS é desenvolvido, ocorrendo uma evolução da LPS por realizar incrementos à medida que novas funções ou requisitos sejam necessários. Na abordagem Extrativa, são analisados os produtos compostos em um domínio, ou seja, dentro de um contexto, quais são as semelhanças e as variabilidades que podem existir em uma LPS. Ao final, uma versão inicial de uma LPS é desenvolvida [Chen *et al.*, 2005].

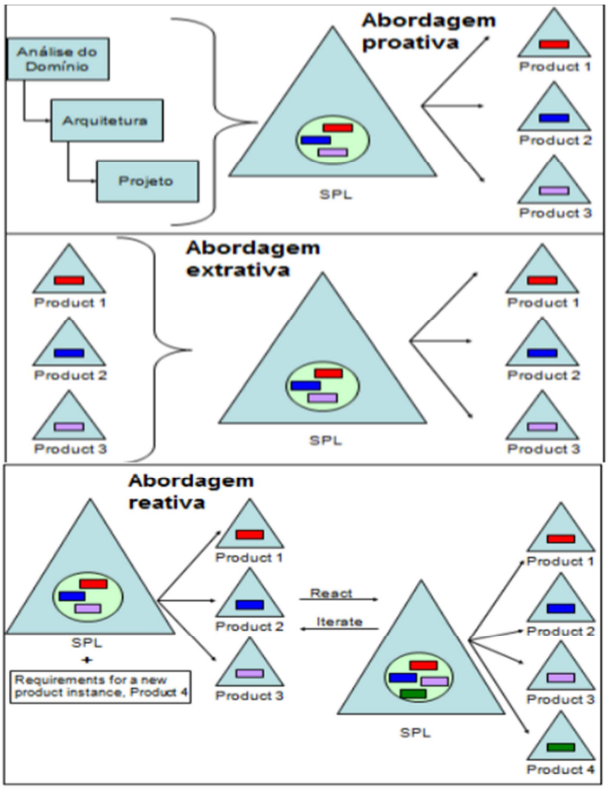


Figura 3-6 - Abordagens em LPS (Fonte: [Krueger, 2001])

3.8. Orientação a Características e AHEAD

A Programação Orientada a Características (POC) é considerada uma técnica moderna para modularização e separação de características. Em sua definição, sistemas devem ser sistematicamente construídos por meio de definição e de composição de características [Batory, 2004; Liu; Batory, 2006]. É uma tecnologia criada para síntese de sistemas de software em LPS, na qual características são utilizadas para distinguir os sistemas de uma mesma família de produtos [Batory, 2004]. A ideia em POC é ter produtos sistematicamente construídos por meio da definição e da composição de características, sendo que essas devem ser tratadas como abstrações de primeira classe no projeto de

sistemas. Portanto, os produtos devem ser implementados em unidades de modularização sintaticamente independentes. Além disso, deve ser possível combinar módulos que representam características de forma flexível, sem que haja perda de recursos de verificação estática de tipos.

Na POC, classes são utilizadas para implementar funções básicas de um sistema. As extensões, as variações e as adaptações dessas funções constituem características e são implementadas em módulos sintaticamente independentes das classes do programa. Os módulos criados em POC podem ser refinados de modo incremental, inserindo ou modificando métodos e atributos ou modificando a hierarquia de tipos. Comparando com pacotes em Java, os quais encapsulam um conjunto de classes, refinamentos encapsulam fragmentos de múltiplas classes. Um pacote com três classes `c1`, `c2` e `c3` (na vertical) e os seus refinamentos `r1`, `r2` e `r3` (na horizontal) são apresentados na Figura 3-7. As linhas tracejadas indicam que os refinamentos adicionam transversalmente comportamento extra nas classes `c1`, `c2` e `c3`. Uma composição com os refinamentos `r1`, `r2` e `r3` produz as classes `c1`, `c2` e `c3` completas, ou seja, com as variações possíveis.

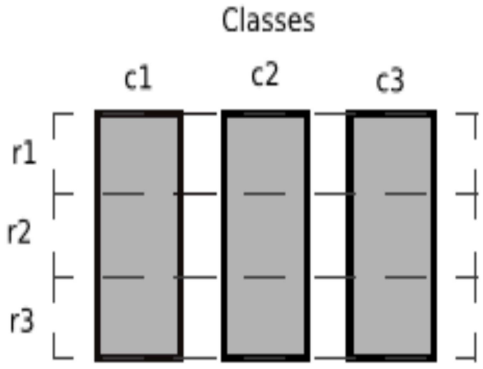


Figura 3-7 - Classes e Seus Refinamentos [Santos, 2009]

Uma das tecnologias para implementar os conceitos de POC é AHEAD (*Algebraic Hierarchical Equations for Application Design*) [Batory, 2004]. O AHEAD *Tool Suite* (ATS) é um conjunto de ferramentas que permite refinamentos de código fonte e de outros artefatos, por exemplo, `JavaDocs`, `makefiles` e máquinas de estados [Batory, 2004]. A linguagem Jakarta, um de seus principais componentes, permite a implementação de características em unidades sintaticamente independentes, denominadas refinamentos. Por meio desses refinamentos, é possível adicionar campos e métodos em classes do programa base e/ou adicionar comportamento extra em métodos existentes. A principal ferramenta do ATS é o *composer*, o qual recebe a aplicação base e um conjunto de características como entrada e gera como saída uma aplicação base com a composição de suas características. Em um arquivo `.jak`, são definidos os refinamentos, os quais são identificados pela palavra reservada `refines`. Um refinamento que insere um atributo contendo uma mensagem em português na classe `Language` é apresentado na Figura 3-8. Em ATS, características são implementadas em diretórios, assim como os pacotes em Java. Esses diretórios contêm refinamentos das classes do programa base, sendo uma composição de características é uma composição de diretórios [Batory, 2004].

```
1  layer Portuguese;
2  public refines class Language {
3      public final static String Die = “Morreu...”;
4  }
```

Figura 3-8 - Exemplo de Refinamento em Jakarta

Para compilar o conteúdo desses diretórios, é necessário converter os arquivos `.jak` para `.java` utilizando a ferramenta `jak2java`. As diferenças entre Jak e Java são sutis, por exemplo, na diretiva `super` do AHEAD (`.jak`), são informados o método, os tipos e os valores dos parâmetros; na diretiva `super` do Java, são informados o método e os valores dos parâmetros (Tabela 3-5).

Tabela 3-5 - Transformação jak2java[3]

Arquivo .jak	Tradução para .java
new void foo() {...}	void foo() {...}
overrides int bar() {...}	int bar() {...}
SoUrce extLayer “C:!...!midsm.jak”;	
Super(int, String).mymethod(5, “e”);	super.mymethod(5, “e”);
Super(int, String) (5, “e”);	super (5, “e”);

3.9. Considerações Finais

Uma LPS consiste no reúso planejado, viável, lucrativo e em grande escala de um conjunto comum e gerenciado de funções (características) que satisfazem necessidades específicas de um segmento (domínio), desenvolvidos a partir de um conjunto de ativos base e de forma determinada. Uma LPS deve possuir aplicações (produtos) feitas sobre uma única plataforma base a ser mantida e com componentes específicos para a LPS e não apenas o desenvolvimento baseado em componentes [Clements; Northrop, 2002]. A arquitetura deve ser definida para suportar variabilidade e não apenas uma arquitetura configurável que reduzirá custos. Versões anteriores de um produto podem ser instâncias válidas da LPS [Clements; Northrop, 2002].

O processo de Engenharia de Domínio leva mais tempo para ser planejado e executado e engloba a atividade de desenvolvimento de ativos base. A Engenharia de Aplicação é semelhante a atividade de desenvolvimento de produtos. A atividade de gerenciamento na maioria das vezes é responsável pelo sucesso ou insucesso do projeto. Apesar da Engenharia de Domínio se assemelhar com a atividade desenvolvimento de ativos base, não quer dizer que na Engenharia de Domínio não há atividades para desenvolvimento de produtos.

Embora as organizações sejam diferentes em termos da natureza de seus produtos, de mercado, da missão, dos objetivos de negócio, da estrutura

[3] http://www.cs.utexas.edu/~schwartz/ATS/fopdocs/j2j.html/

organizacional, da cultura, das políticas, das disciplinas de processo de software e assim por diante, atividades essenciais aplicam-se em qualquer situação, uma vez que representam o nível mais alto de generalidade que envolve os aspectos mais importantes sobre o desenvolvimento em LPS. Em geral, as organizações realizam essa divisão de responsabilidades de várias maneiras.

Algumas organizações têm equipes dedicadas a cada função e outras utilizam as mesmas pessoas para ambas. Isso depende da disponibilidade orçamentária e da estratégia entre outros aspectos. Não há atividade primária, isto é, em alguns contextos, os produtos existentes são "quebrados" em Ativos Base, enquanto, em outros, os Ativos Base podem ser desenvolvidos para uso futuro [McGregor *et al.*, 2002]. Em muitos casos, a gestão é a atividade responsável pelo sucesso/fracasso do produto de uma LPS [Long, 2002]. A abordagem de construção proativa utiliza de conceitos que podem estar vinculados com produtos se assemelhando a abordagem extrativa. No entanto, a abordagem proativa não visa à extração de produtos para compor a LPS, mas de funções.

4. QUALIDADE DE SOFTWARE

4.1. Considerações Iniciais

Dizer que um sistema de software possui ou não qualidade é relativo, pois ele pode ter alta qualidade para um desenvolvedor/usuário e ter baixa qualidade para outro desenvolvedor/usuário. Isso acontece em decorrência do conceito de qualidade de software ser subjetivo. Além disso, pode-se dizer que uma das principais metas da Engenharia de Software é produzir sistemas com qualidade. Para isso, normas e modelos foram criados para a avaliação de qualidade de projeto, de processo e de sistemas de software, tais como, CMMI, ISO/IEC 15504, MPS-BR, ISO/IEC 9126 e ISO/IEC 25000. As duas últimas são destinadas a avaliar a qualidade de sistemas de software. A ISO/IEC 25000 é uma revisão e atualização da ISO/IEC 9126 e da ISO/IEC 14598; elas são inter-relacionadas, mas têm conflitos. Essa revisão permitiu melhor organização e completude para descrever qualidade de sistemas de software.

Encontrar ou criar parâmetros para medir a qualidade é essencial para garantir a qualidade de um software. As medidas podem ajudar desde o processo de análise de custo às atividades de teste e de manutenção. Além disso, elas não estão restritas apenas para desenvolvedores e gerentes de projetos, pois clientes e mantenedores de sistemas estão interessados na medição de diferentes características de projetos, de processos e de sistemas de software.

Conceitos e fatores de qualidade são apresentados na Seção 4.2. Questões de qualidade de sistemas de software estão resumidos na Seção 4.3. Breve comparação entre os conceitos de medição, medida e métrica relacionadas a qualidade de software e manutenção e manutenibilidade são vistos na Seção 4.4.

O cálculo para encontrar o valor da medida Índice de Manutenibilidade é apresentado na Seção 4.5.

4.2. Conceitos e Fatores de Qualidade

Qualidade de software consiste na existência de características que podem ser associadas a seus requisitos [Petrasch, 1999]. Desenvolver sistemas de software com qualidade é um desafio a ser enfrentado pelos engenheiros de software, pois se trata de questões abstratas, sendo difícil encontrar um parâmetro para medir os pontos de qualidade [Pressman, 2010]. Por outro lado, a qualidade é percebida pelos envolvidos em um processo de software (desenvolvimento ou manutenção). Qualidade pode ser definida como conveniência para uso, quando os requisitos e as expectativas dos clientes são considerados, pois eles podem utilizar o mesmo produto de maneira diferente [Juran; Gryna Jr., 1970], e adequação ao uso conforme as exigências, quando as funções do produto podem ser realizadas de maneira adequada [Rothery, 1993]. Na ISO/IEC 25010, qualidade de software é definida como o grau em que um sistema de software satisfaz necessidades explícitas e implícitas quando utilizado sob condições específicas [ISO/IEC 25010, 2010].

Analisando propriedades como correção, facilidade de uso, desempenho, legibilidade, pode-se organizá-las em dois grupos [Zanlorenci *et al*., 2003]: i) fatores externos; e ii) fatores internos. Os fatores de qualidade externos são aqueles detectados principalmente pelo cliente ou eventuais usuários. O desempenho, a facilidade de uso, a correção, a confiabilidade, a extensibilidade são exemplos de fatores externos. Os fatores de qualidade internos são aqueles relacionados à visão de um programador, particularmente aquele que assumirá tarefas de manutenção. Modularidade, legibilidade, portabilidade, acoplamento e coesão são exemplos de fatores internos. Os fatores mais relevantes no desenvolvimento de software são os externos, pois, uma vez que o objetivo do

desenvolvimento de software é satisfazer ao cliente, são esses fatores que possuem papel importante na avaliação do produto. No entanto, os fatores internos vão garantir o alcance dos fatores externos e reduzir serviços inevitáveis de manutenção [Zanlorenci *et al.*, 2003].

4.3. Qualidade de Sistemas de Software

A qualidade de um sistema (ou produto) de software está relacionada com a qualidade do processo de desenvolvimento/manutenção utilizado para desenvolvê-lo/mantê-lo e com a qualidade do projeto. No entanto, um processo ou um projeto que possua nível elevado de qualidade não garante que o produto tenha o mesmo nível de qualidade. A qualidade de sistemas de software é tratada por normas internacionais relacionadas que fornecem suporte ao processo de avaliação de um produto de software. Dentre essas normas, estão a ISO/IEC 9126, a ISO/IEC 14598 e a ISO/IEC 25000.

A ISO/IEC 25000 - Requisitos de Qualidade e Avaliação de Produtos de Software (*Software Product Quality Requirements* - SQuaRE) é a evolução das normas ISO/IEC 9126 e ISO/IEC 14598. Ela é constituída por divisões que detalham tópicos relacionados à especificação e à avaliação da qualidade de sistemas de software. Essas divisões são [ISO/IEC 25000, 2005]: (i) ISO/IEC 2500n - Gestão da Qualidade; (ii) ISO/IEC 2501n - Modelo de Qualidade; (iii) ISO/IEC 2502n - Medição da Qualidade; (iv) ISO/IEC 2503 - Requisitos de Qualidade; (v) ISO/IEC 2504n - Avaliação da Qualidade; e, (vi) ISO/IEC 25051 a ISO/IEC 25099 - Divisão e Extensão.

Os modelos de qualidade externa e interna, presentes na ISO/IEC 9126-1, foram combinados em um único modelo, presente na ISO/IEC 25010, denominado Qualidade de Produto [ISO/IEC 25010, 2010]. Esse modelo é composto por 8 características, subdivididas em 31 subcaracterísticas,

relacionadas às propriedades estáticas e dinâmicas de sistemas de software (Figura 4-1) sendo elas:

- Adequação Funcional: Capacidade de o sistema atender às necessidades implícitas e explícitas, quando utilizado em situações específicas. Subcaracterísticas:
    - ✓ Completude Funcional - grau em que o conjunto de funções do sistema realiza as tarefas especificadas alcançando os objetivos do utilizador;
    - ✓ Correção Funcional - capacidade de um produto ou sistema fornecer resultados corretos com o grau de precisão necessário;
    - ✓ Adequação Funcional - capacidade do conjunto de funções do sistema facilitar a realização de tarefas específicas e objetivas;
- Compatibilidade: Capacidade de um produto, sistema ou componente trocar informações com outros produtos, sistemas ou componentes e/ou realizar suas funções necessárias, ao compartilhar o mesmo hardware ou ambiente de software. Subcaracterísticas:
    - ✓ Coexistência - grau em que um produto pode desempenhar as suas funções de forma eficiente ao compartilhar de um ambiente e recursos comuns com outros produtos, sem produzir impacto negativo em qualquer outro produto;
    - ✓ Interoperabilidade - grau em que dois ou mais sistemas, produtos ou componentes podem trocar informações e utilizar as informações trocadas;
- Usabilidade: Grau ao qual um produto ou sistema pode ser utilizado por usuários específicos atingindo os objetivos especificados com eficácia, eficiência e satisfação em um contexto de uso específico. Subcaracterísticas:
    - ✓ Reconhecimento - grau em que os usuários podem reconhecer se um produto ou sistema é adequado para as suas necessidades;

- ✓ Apreensibilidade - grau ao qual um produto ou sistema pode ser utilizado por usuários específicos para atingir os objetivos especificados de aprendizagem, permitindo utilizar o produto ou sistema com eficácia, eficiência, sem risco e com satisfação em um determinado contexto de uso;
- ✓ Operabilidade - grau ao qual um produto ou sistema possui atributos que o tornam fácil de operar e controlar;
- ✓ Proteção Contra Erros do Usuário - capacidade de um sistema proteger os usuários de cometer erros;
- ✓ Estética da Interface do Usuário - grau em que uma interface de usuário permite a interação de forma agradável e satisfatória;
- ✓ Acessibilidade - grau ao qual um produto ou sistema pode ser utilizado por pessoas com a maior variedade de características e capacidades para atingir um objetivo específico em um determinado contexto de uso;

- Confiabilidade: Capacidade de um sistema, produto ou componente executar as funções necessárias em condições específicas por um período de tempo determinado. Subcaracterísticas:
  - ✓ Maturidade - grau em que um sistema satisfaz as necessidades de fiabilidade em operação normal;
  - ✓ Disponibilidade - grau em que um sistema, produto ou componente está operacional e acessível quando for necessário utilizá-lo;
  - ✓ Tolerância a Falhas - grau em que um sistema, produto ou componente funcionam como pretendido, apesar da presença de falhas de hardware ou software;
  - ✓ Recuperabilidade - capacidade de recuperar dados diretamente afetados e restabelecer o estado desejado para o sistema caso ocorra uma interrupção ou falha;

- Segurança: Grau ao qual um produto ou sistema protege a informação e os dados, de modo que pessoas ou outros sistemas tenham o grau de acesso aos dados apropriado para seus tipos e níveis de autorização. Subcaracterísticas:
    - ✓ Confidencialidade - grau em que um produto ou sistema garante que os dados são acessíveis somente a usuários ou sistemas autorizados;
    - ✓ Integridade - grau em que um sistema, produto ou componente impede o acesso ou modificação não autorizados de programas e dados;
    - ✓ Não Repúdio - grau em que a ocorrência de ações ou eventos pode ser provada, de modo que os eventos ou ações não podem ser repudiados posteriormente;
    - ✓ Responsabilidade - grau em que as ações de uma entidade podem ser atribuídas exclusivamente à entidade;
    - ✓ Autenticidade - grau em que a identidade de um indivíduo ou recurso pode ser provada se reivindicado;
- Manutenibilidade: Grau de eficácia e eficiência com que um produto ou sistema pode ser modificado pelos mantenedores. Subcaracterísticas:
    - ✓ Modularidade - grau em que um sistema ou programa de computador é composto de componentes discretos, de modo que a alteração em um componente tenha impacto mínimo sobre os outros componentes;
    - ✓ Reusabilidade - grau em que um componente pode ser utilizado em mais de um sistema ou na construção de outros componentes;
    - ✓ Analisabilidade - capacidade de avaliar o impacto de uma alteração nos componentes de um produto ou sistema, diagnosticar as deficiências ou causas de falhas de um produto ou identificar partes a serem modificadas de forma eficiente e eficaz;

- ✓ Modificabilidade - grau ao qual um produto ou sistema pode ser modificado de forma eficaz e eficiente sem a introdução de defeitos ou degradação da qualidade do produto existente;
- ✓ Testabilidade - grau de eficácia e eficiência com que se pode estabelecer os critérios de teste para um sistema, produto ou componente e com que os testes podem ser realizados para determinar se esses critérios foram cumpridos;

- Portabilidade: Grau de eficácia e de eficiência com o qual um sistema, produto ou componente pode ser transferido para outro hardware, software ou ambiente operacional. Subcaracterísticas:
  - ✓ Adaptabilidade - grau em que um produto ou um sistema pode ser adaptado de forma eficiente e eficaz para um hardware, software ou ambiente operacional diferente;
  - ✓ Instabilidade ou Capacidade de Instalação - grau de eficácia e eficiência com que um produto ou sistema pode ser instalado e/ou desinstalado com êxito em um ambiente especificado;
  - ✓ Capacidade de Substituir ou Portabilidade - grau ao qual um produto pode ser substituído por outro produto de software especificado para o mesmo fim.
- Eficiência de Desempenho: Capacidade do sistema atingir um bom desempenho em relação à quantidade de recursos utilizados sob condições estabelecidas:
  - ✓ Comportamento em Relação ao Tempo - tempo de processamento e resposta consumidos por um produto ou sistema para que suas funções atendam aos requisitos;
  - ✓ Utilização de Recursos - quantidade de recursos utilizados por um produto ou sistema para realizar as suas funções e satisfazer os requisitos;

- ✓ Capacidade - limite máximo em que um determinado parâmetro de um produto ou sistema pode atender aos requisitos.

Figura 4-1 - Qualidade de Produto (Fonte: [ISO/IEC 25010, 2010])

4.4. Medição da Manutenibilidade de Software

Uma das maiores dificuldades presentes na Engenharia de Software é como medir a qualidade de um sistema de software. Ao contrário de outras disciplinas de engenharia, onde os produtos gerados apresentam características físicas, tais como, peso, altura, tensão de entrada e tolerância a erros, a medição da qualidade de sistemas de software é relativamente novo. Nesse contexto, os termos medição, medidas e métricas são comumente confundidos e o mesmo conceito pode ser designado por termos diferentes. Outros dois termos que são confundidos são manutenção e manutenibilidade, eles são descritos no final desta seção.

Medição é uma técnica ou método que aplica métricas de software a projetos, a produtos ou a processos da Engenharia de Software para alcançar objetivos pré-definidos [Basili *et al.*, 1994] ou é o processo pelo qual números ou símbolos são associados a atributos de entidades do mundo real de tal forma a descrevê-los de acordo com regras claramente definidas [Fenton; Pfleeger, 1996]. Contudo, ela dá apoio à gestão e a melhoria de projetos, de processos e de sistemas de software. Além disso, ela permite a viabilidade de planos de projeto e é útil na monitoração do cumprimento das atividades do projeto em relação aos planos. Com isso, pode-se concluir que medição de software é uma disciplina fundamental na avaliação da qualidade de sistemas de software e da capacidade de processo organizacionais.

Medida é um número ou símbolo atribuído a uma entidade pelo mapeamento a fim de caracterizar um atributo e deve especificar o domínio (atributo), faixa de valores ou símbolos, bem como as regras para realizar o mapeamento [Fenton; Pfleeger, 1996]. As medidas de um software podem ser as mais variadas e a escolha deve obedecer a determinados critérios, por exemplo: i) a pertinência da medida (interoperabilidade); ii) o custo da medida (percentual reduzido do custo do software); e iii) a utilidade (possibilidade de comparação de medidas). Além disso, as medidas, tais como custo, aderência ao cronograma, segurança e manutenibilidade, podem ser aplicadas a processos e a sistemas de software [Basili e*t al.*, 1994].

Métrica é definida como a combinação de duas ou mais medidas ou atributos e/ou método de medição e da escala de medida definidos [ISO/IEC/IEEE 24765, 2010]. O termo "métrica" deve ser utilizado com cuidado, pois, matematicamente, métrica é uma regra utilizada para descrever a distância entre dois pontos. Por exemplo, pode-se verificar a corretude do programa P como uma medida do tamanho ao qual um programa satisfaz sua

especificação S e definir uma métrica M (S,P) na qual S e P são produtos. Essa métrica retorna o quão distante o programa P está de sua especificação S [Fenton; Pfleeger, 1996].

Novas medidas são propostas ou adaptadas à medida que tecnologias são criadas, por exemplo, medidas para sistemas desenvolvidos com a tecnologia de orientação a objetos, podem ser diferentes das medidas utilizadas para sistemas desenvolvidos com a tecnologia de orientação a aspectos ou a características. Isso mostra a complexidade da medição de sistemas software, visto que existe uma quantidade de medidas que podem ser aplicadas para avaliação da manutenibilidade. Nesse contexto avaliar a manutenibilidade em sistemas orientados a características é tarefa difícil, pois o número de medidas para esses sistemas é reduzido e, uma análise crítica deve ser feita para avaliar se as muitas medidas de sistemas orientados a objetos são válidas para avaliar a manutenibilidade em sistemas orientados a características. No entanto, antes de definir a medida a ser utilizada é necessário diferenciar manutenção de manutenibilidade.

O processo de desenvolvimento de sistemas de software está completo depois dos sistemas terem sido entregues e utilizados pelos usuários. Após isso, qualquer mudança é considerada manutenção. Como mudanças são inevitáveis ao longo da vida de um sistema, mecanismos devem ser previstos para avaliar, controlar e fazer essas modificações. A preocupação pela manutenção aconteceu com expressividade na década de 80, quando foi proposto um conjunto de leis denominado de Leis de Lehman [Lehman; Belady, 1985]. A primeira delas diz que a manutenção de sistemas de software é inevitável. Em geral, sistemas são desenvolvidos sem a preocupação com o seu tempo de vida útil, não sendo projetados para facilitar a sua manutenção [Pressman, 2010]. Dessa forma,

observa-se que o problema não está no sistema, mas na forma como ele foi desenvolvido.

A manutenção de software pode ser [Pfleeger, 2009; Pressman, 2010; Sommerville, 2010] (Tabela 4-1): i) adaptativa; ii) corretiva; iii) perfectiva/evolutiva; e, iv) preventiva. A manutenção adaptativa diz respeito às adaptações tecnológicas que o sistema deve passar para continuar útil aos seus usuários. A manutenção corretiva refere-se ao diagnóstico e a correção de um ou mais erros não detectados durante a fase de testes e descobertos após a entrega do software. A manutenção evolutiva refere-se à adição de funções ao sistema. A manutenção preventiva refere-se às alterações realizadas com o objetivo de melhorar a confiabilidade e manutenibilidade do software.

Tabela 4-1 - Tipos de Manutenção

Tipo de Manutenção	Motivo da Manutenção
Adaptativa	Realizada quando o produto de software precisa ser adaptado à novas tecnologias (hardware e software) implantadas no ambiente operacional.
Corretiva	Realizada quando são corrigidos erros não identificados durante o Fluxo de Trabalho de Teste.
Evolutiva	Realizada quando o produto de software deve englobar novos requisitos ou melhorias decorrentes da evolução na tecnologia de implementação empregada.
Preventiva	Realizada quando o produto de software é alterado para aumentar sua manutenibilidade e/ou confiabilidade. Este tipo de manutenção é relativamente raro em ambientes de desenvolvimento. Modificações realizadas neste tipo de manutenção não afetam o comportamento funcional do produto de software.

Para verificar a facilidade com que um sistema pode ser alterado/modificado, pode-se avaliar a sua manutenibilidade. Manutenibilidade é o grau de eficácia e eficiência com que um sistema pode ser modificado pelos mantenedores e pode ser interpretada como uma capacidade inerente do sistema

para facilitar as atividades de manutenção ou a qualidade em uso vivida pelos mantenedores [ISO/IEC 25010, 2010].

Assim, manutenção faz parte do ciclo de vida de um sistema, podendo ser adaptativa, corretiva, evolutiva ou preventiva, enquanto manutenibilidade é uma característica de qualidade do software.

4.5. Índice de Manutenibilidade

O índice de manutenibilidade (*Maintainability Index* - MI) foi elaborado, por Oman em 1994, e modificado pelo SEI em 1997. Seu o objetivo é medir quantitativamente a manutenibilidade de um sistema e para ajudar na diminuição dos custos de manutenção. Para o cálculo do valor do MI, algumas medidas são utilizadas, tais como, as medidas de Halstead e Complexidade Ciclomática de McCabe. A fórmula para calcular o valor do MI é [SEI, 1997]:

$$MI = 171 - \left(5{,}2 * \ln(aveV)\right) - (0{,}23 * aveV(g')) - (16{,}2 * \ln(aveLOC)) + (50 * \sin(\sqrt{2{,}4 perCM}))$$

Equação 4-1 - Fórmula para Calcular o Valor do MI

sendo *aveV* = Volume de Halstead por módulo, *aveV(g')* = Complexidade Ciclomática de McCabe por módulo, *aveLOC* = Linhas de código por módulo e *perCM* = Percentual de linhas de comentários do produto por módulo. No entanto, para identificar o impacto do MI, é necessário conhecer a influência das expressões *aveV*, *aveV(g')*, *aveLOC* e *perCM* no MI:

- Medidas de Halstead. As medidas de Halstead são compostas por cinco medidas: i) `n`, vocabulário do programa; ii) `N`, extensão do programa; iii) `V`, volume; iv) `D`, dificuldade; e v) `E`, esforço. Para compor essas equações, algumas medidas são utilizadas: i) `n1`, quantidade de operadores distintos;

ii) `n2`, quantidade de operandos distintos; iii) `N1`, quantidade de operadores; e iv) `N2`, quantidade de operandos (Tabela 4-2). No cálculo do valor do MI, as medidas Dificuldade e Esforço não são utilizadas. Na fórmula para calcular o valor do MI, a parte que envolve as medidas de Halstead está no fator $5{,}2 * \ln(aveV)$, sendo *aveV* = Volume de Halstead dividido pela quantidade de módulos do produto;

Tabela 4-2 - Medidas de Halstead

Descrição	Fórmula
Quantidade de operadores distintos	n1
Quantidade de operandos distintos	n2
Quantidade de operadores	N1
Quantidade de operandos	N2
Vocabulário do programa	n = n1 + n2
Extensão do programa	N = N1 + N2
Volume	V = N * (log2n)
Dificuldade	D = (n1 / 2) * (N2 / n2)
Esforço	E = D * V

- Complexidade Ciclomática. A Complexidade Ciclomática afeta diretamente os custos de manutenção. Ela é determinada pela quantidade de pontos de decisão adicionais em um método para a entrada de um método. Os pontos de decisão incluem 'se' (*if*), 'enquanto' (*while*), 'para' (*for*) e as chamadas de métodos [PMD, 2013]. Em geral, a quantidade que varia entre 1 e 4 denota baixa complexidade, entre 5 e 7 denota complexidade moderada, 8 e 10 denota alta complexidade e acima de 11 complexidade muito alta [PMD, 2013]. Na fórmula para calcular o valor do MI, a Complexidade Ciclomática é utilizada no fator $0{,}23 * aveV(g')$;
- Quantidade de Linhas de Código. A influência da quantidade de linhas de código (LOC) no cálculo do valor do MI é representada pelo fator $(16{,}2 * \ln(aveLOC))$. Esse fator tem influência similar ao das Medidas de Halstead, pois utiliza a função do logaritmo neperiano. No caso, o que muda é o valor da constante;

- Percentual de Linhas de Comentários. A influência do percentual de linhas de comentários (CLOC) no cálculo do valor do MI é representada pelo fator $(50 * \sin(\sqrt{2,4perCM}))$. Esse é o único fator que contribui para o aumento do valor do MI, os demais decrementam. No entanto, a utilização de comentários deve ser feita de maneira objetiva e clara, pois, se o percentual de linhas de comentários dividido pela quantidade de módulos for muito alto, o produto aparentemente vai estar com valor do MI alto, mas não mostra a real manutenibilidade do sistema.

4.6. Considerações Finais

Neste capítulo, foram apresentados alguns conceitos importantes para a avaliação da qualidade de sistemas de software. O primeiro conceito que deve ser entendido é o de qualidade de software. Na apresentação desse conceito, é ressaltada a importância de ter sistemas de software com qualidade alta por causa, principalmente, das exigências dos clientes. A qualidade de software pode ser organizada em qualidade de processos, de projeto e de sistemas de software. Este capítulo é direcionado para qualidade de sistemas de software pela sua importância e complexidade. Pôde-se observar que existem várias normas e modelos de qualidade relacionados à avaliação da qualidade de um sistema de software. Esses documentos estão em constante evolução, por exemplo, a revisão e atualização das normas ISO/IEC 9126 e ISO/IEC 14598 incorporadas na ISO/IEC 25000.

Entre as características de qualidade de sistemas, a manutenibilidade tem sua importância, pois a manutenção de software é inevitável. Além disso, desenvolvedores têm dado valor a essa característica, pois os custos referentes a manutenção podem chegar, muitas vezes, a atingir metade do valor final do desenvolvimento do sistema.

A definição de parâmetros robustos para realizar a avaliação se faz necessária para obter avaliação criteriosa e condizente e, por causa da confusão dos termos medição, medida e métricas, a apresentação e a diferenciação desses termos foi necessária. O termo "medida" deve ser utilizado no lugar do tradicional termo "métrica", visto que a definição matemática de uma métrica não condiz com o contexto no qual ela é utilizada dentro da Engenharia de Software. De acordo com as definições apresentadas, "medida" é o calculo de um valor para um atributo de um projeto, processo ou sistema de software e, quando tem-se pelo menos dois valores para o mesmo atributo, pode-se utilizar uma métrica para compará-los. Dessa forma, o termo "medida" é utilizado no lugar do termo "métrica" neste trabalho.

5. CRITÉRIOS E DIRETRIZES

5.1. Considerações Iniciais

A manutenção de software é necessária e gera custos para qualquer empresa, por isso, quanto maior a manutenibilidade de um sistema de software espera-se que menores serão os gastos com manutenção. Assim, a elaboração de critérios e diretrizes se faz necessária para nortear engenheiros de software e desenvolvedores a realizar manutenções e aumentar a manutenibilidade do sistema de software ou da Linha de Produtos de Software (LPS). E, posteriormente reduzir os custos com manutenção uma vez que a manutenção de software (adaptativa, corretiva, evolutiva e preventiva) ocorre em toda vida útil de um software.

Os critérios representam possíveis problemas e as diretrizes são formas de resolver ou minimizar os problemas identificados. Os critérios propostos devem ser utilizados de forma a aumentar a manutenibilidade de sistemas de software/LPS desenvolvidos com a tecnologia AHEAD. Eles podem ser aplicados na fase de desenvolvimento ou manutenção do sistema/LPS. Neste capítulo são definidos os critérios e as diretrizes aplicáveis no código fonte das características da LPS com a finalidade de aumentar a legibilidade e reduzir a complexidade do código.

5.2. Critérios Definidos e Diretrizes Elaboradas

Os critérios foram definidos e as diretrizes foram elaboradas para avaliar e manter o código fonte de uma LPS. Caso o código fonte não atenda a um critério, as diretrizes relacionadas ao critério devem ser utilizadas na realização da manutenção do código fonte. Foram definidos sete critérios apresentados nas próximas subseções. Cada critério possui descrição, justificativa em utilizá-lo e

um conjunto de diretrizes a serem seguidas para transformar o código fonte a fim de atender ao critério. Por exemplo, o critério 7 é "A escrita do código segue as convenções para código fonte em Java", cuja justificativa é "Convenções de código fonte melhoram a leitura do sistema de software facilitando aos engenheiros de software compreender o código fonte.". No caso do código fonte não atender ao critério, a seguinte diretriz deve ser seguida: "Inspecionar o código fonte das características adequando-o às convenções definidas no "*Java Code Conventions*" [Oracle, 2013]".

5.2.1. Critério 1

Descrição: O objetivo do refinamento está explicito no código fonte?

Justificativa: A presença de um comentário de documentação, no código fonte, indicando o objetivo da implementação do refinamento facilita o entendimento/análise do código.

Diretrizes: 1- Acrescentar no início da classe/refinamento/método um comentário indicando o objetivo da implementação. 2- Reescrever o comentário caso exista um, mas não é descrito o objetivo da classe/refinamento/método, ainda que esteja explícito o que o código faz.

5.2.2. Critério 2

Descrição: Os métodos de uma classe possuem excesso de responsabilidade?

Justificativa: Um método que possui muita responsabilidade é difícil de refinar.

Diretrizes: 1- Encontrar os métodos passíveis de refatoração (pode-se utilizar a quantidade de linhas de código por método). 2- Refatorar esse método

para torná-lo mais conciso. 3- Utilizar os métodos de refatoração: *Extract Method*, *Replace Temp with Query* e *Decompose Conditional* [Fowler *et al.*, 1999].

5.2.3. Critério 3

Descrição: Existem clones de código fonte na implementação das características?

Justificativa: Clones de código podem causar efeitos negativos no sistema de software reduzindo sua manutenibilidade e introduzindo erros.

Diretrizes: 1- Encontrar os clones de código e verificar o motivo de sua existência (por exemplo, falha durante a programação ou limitação da tecnologia. 2- Refatorar o código para eliminar os clones, quando possível. 3- Utilizar os métodos de refatoração: *Extract Method*, *Pull Up Field* e *Form Template Method* [Fowler *et al.*, 1999].

5.2.4. Critério 4

Descrição: Os mecanismos "*new*" e "*overrides*" são utilizados para indicar explicitamente métodos novos e métodos sobrescritos?

Justificativa: Na cadeia de refinamentos, um método pode ser refinado várias vezes e novos métodos podem ser incluídos pelos refinamentos. A indicação explícita de novos métodos e métodos sobrescritos auxilia na manutenção, indicará ao programador para estar atento nos métodos presentes na cadeia de refinamento.

Diretrizes: 1- Localizar os novos métodos e acrescentar "*new*". 2- Localizar os métodos sobrescritos e acrescentar "*overrides*".

5.2.5. Critério 5

Descrição: As classes de uma característica possuem excesso de responsabilidade?

Justificativa: O excesso de responsabilidade em uma classe pode indicar a existência de *God class*. Classes com excesso de responsabilidade podem gerar problemas na cadeia de refinamentos durante a evolução da LPS, pois ampliam a possibilidade de refinamentos indevidos o que aumenta a quantidade de responsabilidade da classe. Além disso, uma classe com métodos e atributos, indevidamente, públicos ou protegidos podem facilitar o acoplamento.

Diretrizes: 1- Encontrar as classes que possuem excesso de responsabilidade (por exemplo, excesso de métodos e/ou atributos públicos ou protegidos). 2- Refatorar as classes. 3- Utilizar o mecanismo de refatoração: *Extract Class* [Fowler *et al.*, 1999].

5.2.6. Critério 6

Descrição: Os dados de uma classe estão encapsulados?

Justificativa: Na cadeia de refinamentos, pode ser necessário acessar atributos para acessar seus valores. O mecanismo de composição *mixin* compõe as características utilizando uma sequência de heranças, portanto os atributos devem estar visíveis para as subclasses. Deve-se implementar o encapsulamento, mantendo atributos privados e provendo métodos assessores para diminuir o acoplamento.

Diretrizes: 1- Encontrar as classes que possuem atributos públicos ou protegidos. 2- Refatorar as classes. 3- Utilizar os métodos de refatoração:

Encapsulate Field, *Remove Setting Method*, *Move Method* e *Hide Method* [Fowler *et al.*, 1999].

5.2.7. Critério 7

Descrição: A escrita do código segue as convenções para código fonte Java?

Justificativa: Convenções de código fonte melhoram a leitura do sistema de software facilitando aos engenheiros de software compreender o código fonte. A escolha das convenções de código Java se justifica por AHEAD ser baseado em Java.

Diretrizes: 1- Inspecionar o código fonte das características adequando-o às convenções definidas no "*Java Code Conventions*" [Oracle, 2013].

5.3. Considerações Finais

Com os critérios definidos e as diretrizes elaboradas acredita-se que após a avaliação e manutenção do código fonte utilizando os critérios seguindo as diretrizes obtém-se como resultado um sistema de software/LPS com maior capacidade de manutenção. No qual, atributos como legibilidade, analisabilidade e modificabilidade serão melhorados, pois os critérios e diretrizes foram elaborados, principalmente, para aumentar a legibilidade e reduzir a complexidade do código fonte.

6. AVALIAÇÃO DA LINHA DE PRODUTOS DE SOFTWARE TANKWAR

6.1. Considerações Iniciais

Neste capítulo, é apresentada a Linha de Produtos de Software (LPS) TankWar e as ferramentas utilizadas como auxilio da medição. Além disso, o processo de avaliação e seus resultados são apresentados e analisados. A LPS TankWar foi desenvolvida utilizando a tecnologia AHEAD e, durante o processo de composição de características, o código fonte dos produtos derivados dessa LPS são transformados em Java.

Os produtos em Java são avaliados por meio de medição de suas propriedades, por exemplo, quantidade de linhas de código, para verificar o impacto dos critérios e das diretrizes na manutenibilidade de produtos da LPS TankWar. A medição aconteceu nesses produtos, pois ainda não há ferramentas computacionais para medir propriedades de código fonte desenvolvido com AHEAD. Os critérios e as diretrizes foram utilizados nos artefatos de código fonte que compõem cada característica e a medição foi realizada em seis produtos derivados da LPS TankWar, nos quais foi calculado o Índice de Manutenibilidade.

A descrição, a escolha e o mapeamento da LPS TankWar são abordados na Seção 6.2. As ferramentas utilizadas são descritas na Seção 6.3. A escolha dos produtos da LPS é explicada na Seção 6.4. A aplicação de cada critério na LPS legada e a avaliação do impacto são discutidos na Seção 6.5. A avaliação e a aplicação dos critérios e o seguimento das diretrizes, de forma acumulativa[4],

[4] Os critérios e as respectivas diretrizes foram utilizados por sucessivas adições sem perda ou eliminação correspondente.

são mostrados na Seção 6.6. A comparação da LPS legada com a LPS nova é realizada na Seção 0.

6.2. LPS TankWar

6.2.1. Escolha da LPS

Para a escolha da LPS a ser utilizada como estudo de caso, foram seguidos alguns passos da técnica de pesquisa Revisão Sistemática da Literatura (RSL). Foi realizada uma pesquisa em três máquinas de busca (IEEE[5], Elsevier ScienceDirect[6] e Scopus[7]) utilizando a *String*: ("*Feature Oriented*" OR "*Feature-Oriented"* OR "*feature oriented*" OR "*feature-oriented*") AND ("*Development*" OR "*development*") AND ("Software" OR "software"). A escolha dessas três máquinas de busca se deve ao fato de serem comumente utilizadas em RSL [Abílio *et al.*, 2012; Laguna; Crespo, 2012; Nascimento *et al.*, 2012;]. Duas restrições foram adicionadas: i) idioma em inglês; e ii) busca apenas em título, palavras-chave e resumo. Como resultado, foram encontrados 246 artigos. Desse total, 72 foram obtidos no IEEE, 16 no Elsevier e 158 no Scopus (Tabela 6-1). Foi realizada a leitura do título, do resumo e das palavras-chave desses artigos com o objetivo de selecionar trabalhos que envolvessem os temas: LPS, Orientação a Características e desenvolvimento de software. Com isso, 27 artigos foram selecionados e lidos por completo, sendo 11 do IEEE, 2 do Elsevier e 14 do Scopus.

[5] Disponível em: http://ieeexplore.ieee.org/
[6] Disponível em: http://www.sciencedirect.com
[7] Disponível em: http://www.scopus.com

Tabela 6-1 - Pesquisa para Encontrar LPSs desenvolvidas com Orientação a Características

Fonte	Artigos Encontrados	Artigos Lidos
IEEE	72	11
Elsevier	16	2
Scopus	158	14
Total	246	27

Durante a leitura dos artigos, foi possível identificar 24 sistemas de software. No entanto, como a LPS deveria ser desenvolvida em AHEAD (decisão de pesquisa), as seguintes LPS foram selecionadas para análise (Tabela 6-2): `DesktopSearcher`, `Devolution`, `EPL`, `GPL`, `TankWar` e `WebStore`. Uma dessas LPSs é o `DesktopSearcher`, utilizado para encontrar arquivos locais em computadores. O modelo de características dessa LPS possui 22 características, sendo 6 abstratas e 16 concretas, com a possibilidade de gerar 462 produtos. Além disso, essa LPS possui 45 classes e 3.782 linhas, incluindo linhas de código, de comentários e em branco.

Um teste foi realizado em cada LPS e apenas a LPS WebStore apresentou problemas na execução de produtos gerados, nas demais os produtos foram gerados e executados sem problemas. Com isso, essa LPS foi descartada para a realização deste trabalho. A LPS EPL possui poucas de linhas (139) e a Devolution pode gerar poucos produtos (29). A LPS TankWar foi escolhida por estar entre as duas LPS com o maior número de características totais e concretas, 37 e 31, respectivamente, e por ser possível gerar quase 40.000 produtos, além do número total de linhas estar acima de 5.500.

Tabela 6-2 - LPSs Desenvolvidas com AHEAD

	LPS					
Especificidade	**Desktop Searcher**	**Devolution**	**EPL**	**GPL**	**Tank War**	**Web Store**
Domínio	Sistema de busca local	Cliente de *e-mails* e mensagens instantâneas	Avaliação de expressões	Biblioteca de grafos	Jogo	Loja virtual
Quantidade de Características	22	14	23	66	37	19
Quantidade de Características Abstratas	6	3	11	30	6	6
Quantidade de Características Concretas	16	11	12	36	31	13
Quantidade de Produtos	462	29	1.024	42.113	39.060	64
Quantidade de Classes	45	70	21	75	88	87
Quantidade de Linhas	3.782	6.564	139	2.878	5.640	2.391

6.2.2. Descrição Geral do TankWar

O TankWar é um jogo desenvolvido por estudantes da Universidade de Magdeburg, situada na Alemanha, como uma LPS para aderir às exigências de portabilidade comum a jogos de dispositivos móveis [Schulze *et al.*, 2010]. Essa LPS é de tamanho médio (aproximadamente 5.000 LOC) e composta por 37 características [Schulze *et al.*, 2012]. Além disso, podem ser gerados produtos de software para computadores pessoais (PC - *Personal Computer*) e para dispositivos móveis (Handy). As versões desenvolvidas para PC apresentam melhor qualidade gráfica, pois dispositivos móveis, em geral, possuem restrições gráficas e/ou de memória. Entre as principais variações de produtos permitidas, foram constatadas:

- possibilidade de alternar entre dois idiomas (inglês e alemão);
- escolha do nível de dificuldade (fácil ou difícil);

- escolha de tamanho de tela, no máximo três opções;
- habilitação de som;
- escolha da quantidade de tipos de tanque, o usuário vai poder selecionar 1, 2 ou 3;
- escolha de ferramentas para incorporar ao tanque, no máximo seis ferramentas;
- armazenamento de resultados, ao armazenar o resultado, um *ranking* é acrescentado ao jogo.

A tela inicial de um produto gerado no idioma inglês e com a opção de gravar o placar (opção *score* torna isso explícito) é apresentada na Figura 6-1. A variabilidade de produtos é expressa pelo modelo de características (Figura 6-2) em que as possibilidades citadas são facilmente encontradas, por exemplo, o tipo de plataforma PC ou Handy. Além disso, esse modelo possui sintaxe que deve ser respeitada, sendo expressa por diferentes componentes como operadores lógicos, hierarquia do modelo e restrições transversais. Por exemplo, a primeira restrição transversal deixa explícito que, para utilizar umas das seguintes características `fuer_PC`, `Re_fuer_PC` ou `Sound_fuer_pc`, a característica `PC` deve ser escolhida.

Figura 6-1 - Tela Inicial de um Produto da LPS TankWar

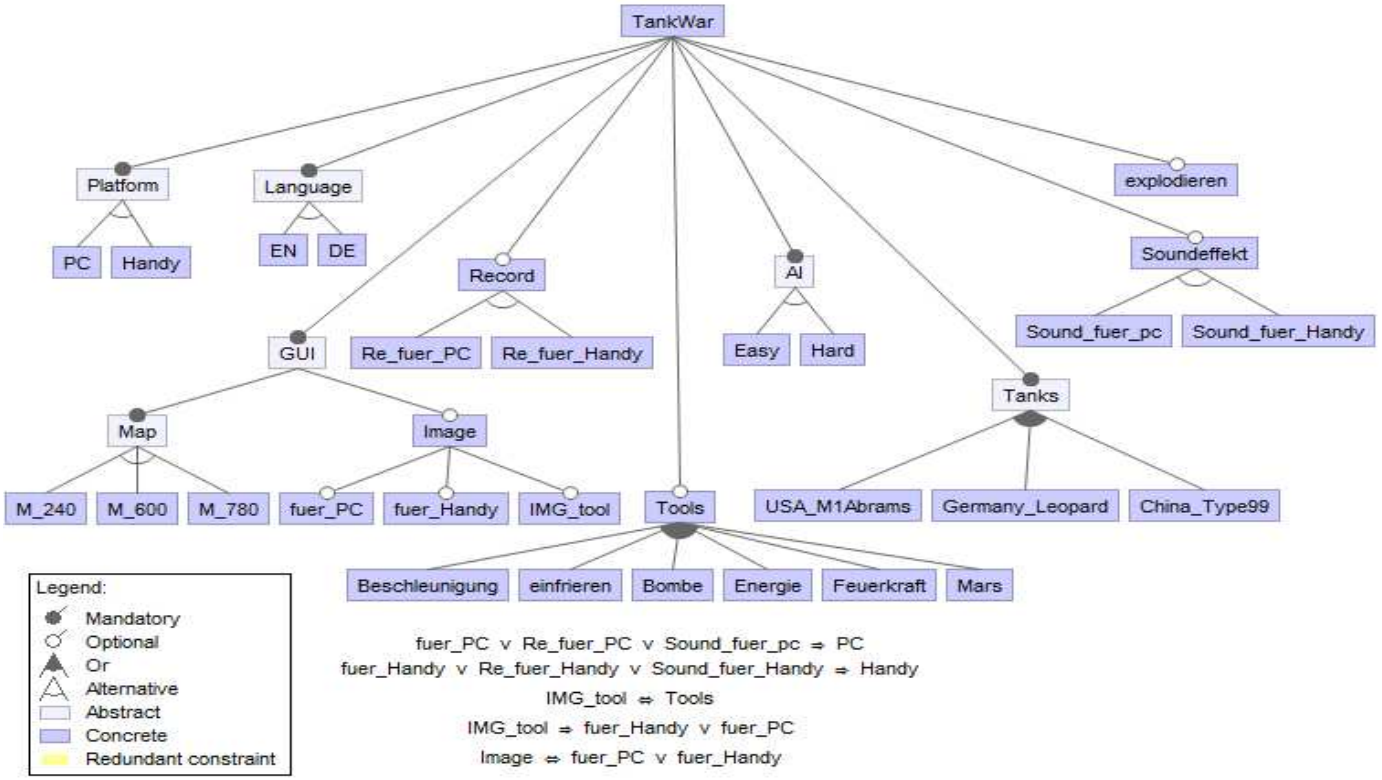


Figura 6-2 - Modelo de Características da LPS TankWar

6.2.3. Caracterização da LPS TankWar

Uma caracterização da LPS TankWar foi realizada para conhecê-la/medi-la a fim de auxiliar na utilização dos critérios e das diretrizes e na comparação com a nova versão a ser gerada (Tabela 6-3). Essa LPS possui 5.640 linhas, incluindo código fonte (4.865 linhas - 86,3%), comentários (de documentação e gerais) (105 linhas - 1,9%) e em branco, para aumentar a legibilidade, (670 linhas - 11,9%). As linhas referentes à documentação possuem blocos de comentários que informam quais são os parâmetros do construtor da classe, sendo que elas não aparecem em todos os artefatos de código, apenas naqueles referentes à característica `TankWar` (característica raiz). Essa é a maior característica em relação à quantidade de linhas de código (1.075, cerca de 22% das linhas de código). De modo geral, as linhas referentes aos comentários gerais não ajudam no entendimento do código fonte e representam pouco mais de 1% das linhas da LPS TankWar. Dessas linhas, 12 são linhas de código fonte comentadas, 22 são comentários para separar blocos de código, tais como "//---", e 27 apresentam comentários sem relevância, por exemplo, 4 comentários com apenas duas barras (//) ou comentários simples como "//*modify the position of Explodes*" no qual foram utilizadas 5 linhas de comentários (uma palavra em cada linha).

Tabela 6-3 - Caracterização da LPS TankWar

Medidas	Quantidade
Linhas	5.640
Linhas de Código	4.865
Linhas de Comentários de Documentação	44
Linhas de Comentários Gerais	61
Classes (Constantes)	29
Interfaces (Constantes)	1
Refinamentos de Classes	56
Refinamentos de Interfaces	2
Métodos	308
Métodos em Refinamentos de Classe	97

Tabela 6-3 - Caracterização da LPS TankWar (cont.)

Medidas	Quantidade
Importações	87
Atributos	236
Constantes	53

A LPS TankWar é composta por 29 classes, 1 interface e 58 refinamentos, sendo 56 de classe e 2 de interfaces, o que totaliza 88 elementos. Desses elementos, com relação ao modificador de acesso, 1 é padrão (*default* - 1,1%), 85 são públicos (*public* - 96,6%), 2 são protegidos (*protected* - 2,3%) e 0 privado (*private* - 0,%). Existem 308 métodos, sendo 97 (25%) pertencem a refinamentos de classe, e 87 importações (*imports*), sendo 8 importações (9,19%) em refinamentos de classe. Existem 236 atributos com respectivo modificador de acesso 52 padrão (22,0%), 20 públicos (8,5%), 36 privados (15,3%), 128 protegidos (54,2%). Além disso, foram encontradas 53 constantes, no entanto elas não tinham o modificador `final` apesar de seguir parcialmente os padrões de programação (estar em caixa alta).

Dentre os problemas encontrados na LPS-l TankWar destacam-se: a) redundância de partes do código (clones de código); b) métodos extensos; c) demora em entender o contexto do código; d) demora em localizar defeitos; e) no código, incerteza sobre o tamanho do escopo de uma função; f) código mal endentado prejudicando a legibilidade; g) falta de padrão na declaração de constantes; e h) atributos declarados não utilizados.

6.3. Apoio Ferramental

Para o desenvolvimento deste trabalho, o `Eclipse Juno` (versão 4.2) foi utilizado com o acréscimo dos *plug-ins*: `FeatureIDE`, `Google CodePro Analytix` e `PMD`. Além do Eclipse, as ferramentas `CLOC` e `JMeld` foram utilizadas.

O `Eclipse`[8] é uma plataforma/ambiente estruturado de código aberto extensível. Ele é utilizado para desenvolvimento integrado, implantação e gerenciamento de software, é multilinguagem e pode aumentar sua funcionalidade por meio de *plug-ins*. O *plug-in* `FeatureIDE`[9] é utilizado para desenvolvimento de sistemas de software orientados a características. Esse *plug-in* foi utilizado para a modelagem, a implementação e a geração de produtos de software da LPS TankWar. O modelo de características disponível no *plug-in* foi utilizado para a modelagem, a implementação foi feita com a tecnologia AHEAD (artefatos de código fonte com extensão .jak) e a geração dos produtos com os arquivos de configuração. O *plug-in* `Google CodePro Analytix`[10] foi utilizado para analisar propriedades do código fonte de um sistema de software, por exemplo, repetição de código, dependências entre módulos e cobertura de código. Algumas das funções disponíveis na interface gráfica desse *plug-in* permitem apresentar análises no formato de gráficos e de tabelas e gerar relatórios no padrão HTML (*HyperText Markup Language*). Esse plug-in foi utilizado como auxiliar no cálculo das medidas do código fonte dos produtos de software derivados pela LPS TankWar. O *plug-in* `PMD`[11] é semelhante ao `Google CodePro Analytix`. Com esse *plug-in*, pode-se verificar código fonte Java e destacar potenciais problemas, tais como, possíveis erros, código morto, expressões complicadas e código duplicado. Neste trabalho, esse *plug-in* foi utilizado como auxiliar na aplicação dos critérios e o cálculo das medidas do código fonte dos vários produtos de software derivados pela LPS TankWar.

[8] http://www.eclipse.org/
[9] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/deploy
[10] http://dl.google.com/eclipse/inst/codepro/latest/3.7
[11] http://sourceforge.net/projects/pmd/files/pmd-eclipse/update-site/

A ferramenta CLOC[12] foi utilizada para contabilizar linhas em branco, linhas de comentários e linhas de código fonte. O CLOC foi escrito em Perl e é portátil, pois executa em modo texto em diferentes sistemas operacionais (Linux, Windows e MAC OS). No entanto, ele possui uma limitação, linhas que possuem código e comentários são consideradas como linhas de código. A ferramenta JMeld[13] permite realizar comparação visual e mesclagem de código. Ela foi construída para ser rápida com grandes arquivos e executar em qualquer sistema operacional.

6.4. Escolha dos Produtos

A LPS TankWar oferece a possibilidade de gerar 39.060 produtos distintos. A realização da análise do índice de manutenibilidade nesses produtos não se justifica. Portanto, optou-se por estabelecer três critérios para escolher os produtos derivados da LPS TankWar: i) um produto derivado com maior quantidade de características para execução do programa; ii) um produto derivado com menor quantidade de características possíveis para execução do programa; e iii) um produto derivado com quantidade de características intermediário. Apesar da LPS TankWar possuir 31 características concretas, a quantidade máxima de características que um produto pode ter é 23 por causa das restrições transversais e dos operadores lógicos; por exemplo, a dificuldade do jogo pode ser fácil (*Easy*) ou difícil (*Hard*), mas não ambas. A quantidade mínima de características que um produto pode ter é 6, pois ele deve ser composto por algumas características obrigatórias, por exemplo, tamanho do gráfico, nível e idioma.

[12] http://cloc.sourceforge.net/
[13] Mais informações em: http://keeskuip.home.xs4all.nl/jmeld/index.htm

A escolha da plataforma deve ocorrer no início da seleção dos produtos e a escolha de uma plataforma implica na exclusão de três características dependentes, pois elas só executam em determinada plataforma. Com intuito de expandir o estudo, foram gerados três tipos de produtos para cada plataforma totalizando a geração de seis produtos para análise do índice de manutenibilidade. Além disso, cabe ressaltar que uma característica concreta (`Soundeffekt`) não possui linhas de código, apenas arquivos de som (.wav). Com isso, os dois produtos com a quantidade máxima de características possuem 23 características; entre esses produtos, há quatro características distintas. Os dois produtos com a quantidade mínima possuem 6 características. Com esses 4 produtos, consegue-se abranger todas as características previstas na LPS TankWar. Os produtos com quantidade intermediária de características foram escolhidos de forma que os tipos de características existentes na LPS TankWar estivessem presentes e a quantidade de características não fosse menor do que a metade da quantidade de características concretas (31 / 2 = ~16). As 16 características foram escolhidas considerando a funcionalidade do jogo e a quantidade de linhas de código da característica, no caso de características alternativas.

Os produtos derivados da LPS TankWar escolhidos para análise como estudo de caso foram denominados `Produto1MinimoPC` (1.838 linhas de código), `Produto2MinimoHandy` (1.802 linhas de código), `Produto3IntermediarioPC` (3.051 linhas de código), `Produto4IntermediarioHandy` (3.197 linhas de código), `Produto5MaximoPC` (3.330 linhas de código) e `Produto6MaximoHandy` (3.476 linhas de código).

6.5. Avaliação e Aplicação das Diretrizes dos Critérios Separadamente

Nesta seção, são apresentados o Índice de Manutenibilidade (*Maintainability Index* - MI) dos seis produtos derivados da LPS TankWar escolhido para análise, após a utilização de diretrizes dos critérios avaliados. Em especial, há discussão relatando essa utilização e é calculado o MI dos produtos. As medidas que compõem o MI e suas siglas são mostradas na Tabela 6-4. A medida Complexidade Ciclomática é representada pela sigla "CC".

Tabela 6-4 - Medidas para Calcular o Índice de Manutenibilidade

Medidas	Siglas
Quantidade de operadores distintos	n1
Quantidade de operandos distintos	n2
Quantidade de operadores	N1
Quantidade de operandos	N2
Vocabulário do programa	N
Extensão do programa	N
Volume	V
Complexidade Cliclomática	CC
Quantidade de linhas (*Total Number of Lines*)	TNOL
Quantidade de linhas de código (*Lines of Code*)	LOC
Quantidade de linhas de comentário (*Comments Line of Code*)	CLOC
Quantidade de módulos (*Number of Modules*)	NOM

6.5.1. Produtos Gerados da LPS TankWar

Os valores das medidas foram obtidos com a utilização de duas ferramentas: o *plug-ins* `Google CodePro Analytix` e a `CLOC`. Os valores para as medidas n1, n2, N1, N2, N, N, V, CC e TNOL foram calculados pelo *plug-in* `Google CodePro Analytix` com o valor das medidas *Number of Unique Operators, Number of Unique Operands, Number of Operators, Number of Operands, Program Vocabulary, Program Length, Program Volume, Average Cyclomatic Complexity, Number of Lines,* respectivamente. Os valores para as medidas LOC, CLOC e NOM foram calculados pela ferramenta CLOC com o

valor das medidas *code, comment, files*, respectivamente. O MI foi calculado utilizando os valores dessas medidas aplicados na Equação 4-1. O valor dessas medidas é apresentado na Tabela 6-5. Por exemplo, o valor das medidas CC, TNOL, LOC, CLOC, NOM, MI para o produto `Produto1MinimoPC` é 3,28; 2.209; 1.838; 67; 16; e 81,0228, respectivamente. Da mesma forma, o valor das mesmas medidas para o produto `Produto2MinimoHandy` é 3,4; 2.190; 1.802; 71; 14; e 81,2680, respectivamente.

Tabela 6-5 - Valor das Medidas da LPS TankWar

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	526	508	885	924	925	964
N1	2.700	2.673	4.362	4.547	4.599	4.784
N2	4.972	4.863	8.345	8.633	9.072	9.360
n	560	543	919	959	959	999
N	7.672	7.536	12.707	13.180	13.671	14.144
V	70.039,85	68.463,11	125.086,7	130.553	135.416,54	140.935,63
CC	3,28	3,4	3,6	3,59	3,24	3,25
TNOL	2.209	2.190	3.805	4.006	4.287	4.495
LOC	1.838	1.802	3.051	3.197	3.330	3.476
CLOC	67	71	202	208	269	275
NOM	16	14	20	20	20	20
MI	81,0228	81,2680	79,0768	77,7905	79,6694	78,4091

6.5.2. Produtos Gerados Após Utilizar as Diretrizes

Ao não atender completamente um critério, as características do produto devem "sofrer" uma transformação de modo a atendê-lo. Para realizar essa transformação, é sugerido um conjunto de diretrizes, associado ao critério, para ser utilizado. Nas próximas seções, são apresentadas a utilização das diretrizes e o cálculo das medidas apresentadas na Tabela 6-4.

6.5.2.1. Utilização das Diretrizes do Critério 1

Para aplicar o critério 1, alterações foram feitas no código fonte por meio de inspeção manual com intuito de documentar classes e métodos. Quando encontrado um refinamento, foi explicado o motivo do refinamento. Por exemplo, o refinamento da classe `Tank` da característica `Beschleunigung` estava sem comentário e, por isso, foram adicionados comentários no início da classe (linhas 1, 2 e 3 apresentados na Figura 6-3). Além disso, os métodos receberam comentários (linhas 10 a 12 apresentados na Figura 6-3).

Tank.jak[Beschleunigung]

```
1  /**
2   * O objetivo dessa classe e adicionar o efeito da ferramenta beschleunigung (aceleracao) no tanque
3   */
4
5  public refines class Tank {
6
7      protected long beschleunigungTimer;
8      protected boolean beschleunigung = false;
9
10     /**
11      * O objetivo desse metodo e introduzir funcoes de controle da ferramenta
12      */
13
14     protected void toolKontroller(){
15         Super().toolKontroller();
16         if (tankManager.status == GameManager.PAUSE || tankManager.status == GameManager.EXIT) {
17             if (beschleunigung) {
18                 beschleunigungTimer += elapsedTime;
19             }
20         }
21         if (beschleunigung && System.currentTimeMillis() - beschleunigungTimer > 10000) {
```

Figura 6-3 - Exemplo da Utilização das Diretrizes do Critério 1

O valor das medidas para os produtos é apresentado na Tabela 6-6. Apenas as medidas TNOL e CLOC apresentaram valores diferentes em relação à LPS TankWar Legada. Por exemplo, para o produto `Produto1MinimoPC`, o valor para a medida TNOL passou de 2.209 para 2.629 e o CLOC de 67 para 486. Ou seja, a quantidade de linhas de comentário aumento e, consequentemente, a quantidade de linhas aumentou. Com isso, o valor da medida MI aumentou para todos os produtos, o que significa que houve melhora na manutenibilidade dos produtos.

Tabela 6-6 - Valores das Medidas Após a Utilizar as Diretrizes do Critério 1

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	526	508	885	924	925	964
N1	2.700	2.673	4.362	4.547	4.599	4.784
N2	4.972	4.863	8.345	8.633	9.072	9.360
n	560	543	919	959	959	999
N	7.672	7.536	12.707	13.180	13.671	14.144
V	70.039,85	68.463,11	125.086,7	130.553	135.416,54	140.935,63
CC	3,28	3,4	3,6	3,59	3,24	3,25
TNOL	2.629	2.625	4.574	4.842	5.279	5.547
LOC	1.838	1.802	3.051	3.197	3.330	3.476
CLOC	486	506	963	1.042	1.248	1.327
NOM	16	14	20	20	20	20
MI	99,5746	95,8331	93,2645	92,2631	91,2041	90,2397

6.5.2.2. Utilização das Diretrizes do Critério 2

Para aplicar o critério 2, a quantidade de linhas de código dos métodos foi utilizada como valor de referência. No entanto, não há consenso na quantidade de linhas em que um método é considerado extenso. Na literatura, foram encontrados diferentes intervalos de valores para indicar quando um módulo é considerado extenso; por exemplo, em um dos estudos [Bolton; Johnston, 2009], métodos com quantidade de linhas superior a 200 (LOC + linhas de comentários + linhas em branco) são extensos. Em outro estudo [Evanco; Verner, 2001], é indicado que um método deve ter entre 80 e 100 linhas. Entretanto, para este trabalho, decidiu-se tratar métodos com mais de 50 linhas, pois os métodos da LPS TankWar, em sua maioria, não tinham linhas em branco e de comentários. A LPS TankWar é composta por 308 métodos, dos quais 14 possuem mais que 50 LOC. A refatoração desses métodos seguiu os seguintes passos: i) os métodos extensos foram detectados (por meio do mapeamento realizado e apresentado na subseção 0); e ii) tentou-se refatorá-los.

O valor das medidas para os produtos é apresentado na Tabela 6-7. Em relação a LPS TankWar Legada, o valor das medidas n2, N1, N2, n, N, V, CC, TNOL e LOC de todos os produtos sofreram alterações. Entre essas alterações, apenas o valora da medida CC foi modificada para melhor, ou seja, reduzida. Essas alterações são em decorrência da criação de métodos, pois, com métodos novos, a quantidade de operadores e de operandos aumentou, fazendo com que o valor das medidas extensão (N), vocabulário (n) e volume (V) aumentasse. Além disso, houve aumento da quantidade de linhas código e, consequentemente, da quantidade de linhas.

Por exemplo, o valor da medida CC do `Produto1MinimoPC` foi alterada de 3,28 para 3,17. Essa redução ocorreu porque métodos longos foram refatorados resultando em métodos com menor quantidade de linhas e pontos de decisão. O método `tankMalen`, da classe `Tank` da característica `Image`, possui um comando *switch/case*, no qual são tratados 10 "casos" (*cases*) distintos. Em três desses casos, há 3 comandos "se" (*if*) para cada caso. Esses comandos foram refatorados em um método para cada *case*. Com isso, houve redução de 3 pontos de decisão por caso (*case*), totalizando 9 pontos. Além disso, com a refatoração, a quantidade de linhas de código do método `tankMalen` foi reduzida de 54 para 40.

Tabela 6-7 - Valores das Medidas Após a Utilizar as Diretrizes do Critério 2

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	532	514	895	934	935	974
N1	2.691	2.661	4.361	4.543	4.598	4.780
N2	4.971	4.859	8.361	8.646	9.088	9.373
n	566	549	929	969	969	1.009
N	7.662	7.520	12.722	13.189	13.686	14.153
V	70.066,37	68.436,98	125.433	130.839,52	135.769,94	141.228,68
CC	3,17	3,29	3,48	3,48	3,15	3,17
TNOL	2.217	2.202	3.848	4.054	4.330	4.543

Tabela 6-7 - Valores das Medidas Após a Utilizar as Diretrizes do Critério 2 (cont.)

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
LOC	1.836	1.804	3.066	3.216	3.345	3.495
CLOC	67	71	202	208	269	275
NOM	16	14	20	20	20	20
MI	81,0162	81,2025	78,8542	77,5429	79,4637	78,1794

6.5.2.3. Utilização das Diretrizes do Critério 3

Para aplicar o critério 3, inicialmente, foi utilizado o visor `CPD` do *plug-in* `PMD` que permite realizar buscas por códigos clonados. No entanto, foram obtidos resultados incoerentes apresentando falso-positivos e falso-negativos. Com isso, decidiu-se verificar cada linha de saída do *plug-in* e, além disso, fazer varredura manual para comparar classes com a ferramenta `JMeld`. Em alguns casos, pode-se dizer que foi falha do programador, por exemplo, a classe `Menu` das características `PC` e `Handy` são similares e possuem juntas 559 LOC (303 e 256 LOC, respectivamente). Essas duas características têm a característica abstrata `Platform` como característica mãe. A refatoração empregada foi: i) tornar a característica `Platform` em uma característica concreta; ii) criar a classe `Menu` nessa característica concreta; iii) remover os clones adicionando métodos e partes de código nessa classe; e iv) refinar acrescentando as partes de código que se diferem nas características `PC` e `Handy` em suas respectivas classes `Menu`. Com isso, houve redução da quantidade de linhas para 396 linhas (`Menu`(`Platform`) 186 LOC + `Menu`(`PC`) 127 LOC + `Menu`(`Handy`) 83 LOC), o que implica diminuição de 163 LOC. Essa refatoração foi realizada em diversas classes fornecendo redução total de 644 LOC na LPS TankWar. Outro caso ocorreu com as 4 classes `Maler`, `Tank`, `TankManager` e `Tool` de cada uma das características `Beschleunigung`, `Bombe`, `Einfrieren`, `Energie`, `Feuerkraft` e `Mars`. Essas classes são similares, mas não se pôde refatorar o código, pois a mudança interferiria na separação de interesses.

O valor das medidas para os produtos é apresentado na Tabela 6-8. Em relação com a LPS TankWar Legada, o valor das medidas dos produtos, com exceção das medidas n1 e NOM, foram modificadas (variações dos valores presentes na Tabela 6-5 e na Tabela 6-8). De forma geral, a remoção de clones aconteceu entre classes de características que não podem compor o mesmo produto, por exemplo, as características `PC` e `Handy` e as características `fuer_PC` e `fuer_Handy`. Além disso, como foi criado um novo refinamento, alguns métodos não declarados em uma classe passaram a ser, mesmo que essa declaração fosse vazia. Essas declarações ocorreram para atender necessidades da linguagem sob o refinamento de uma constante. Com isso, linhas de código foram adicionadas e, com novos métodos declarados, novos operadores e operandos surgiram. A quantidade de linhas de comentário aumentou, pois, quando dois métodos similares eram encontrados em características diferentes, aquele com comentários foi mantido. O único benefício para ambos, LPS TankWar e produtos, foi a redução da CC que no produto `Produto6MaximoHandy`, por exemplo, caiu de 3,25 para 3,07.

Tabela 6-8 - Valores das Medidas Após a Utilizar as Diretrizes do Critério 3

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	531	513	892	934	932	974
N1	2.716	2.675	4.382	4.566	4.619	4.803
N2	5.071	4.941	8.479	8.769	9.206	9.496
n	565	548	929	969	969	1.009
N	7.787	7.616	12.861	13.335	13.825	14.299
V	71.189,58	69.290,61	126.743,46	137.087,03	137.087,03	142.685,58
CC	2,94	3,06	3,3	3,34	3,02	3,07
TNOL	2.324	2.287	3.965	4.167	4.447	4.649
LOC	1.889	1.847	3.117	3.276	3.396	3.555
CLOC	69	71	207	209	274	276
NOM	16	14	20	20	20	20
MI	80,2927	80,2928	78,4977	76,9038	79,0826	77,6007

6.5.2.4. Utilização das Diretrizes do Critério 4

Para aplicar o critério 4, foi efetuada inspeção manual nos refinamentos de classe para adição dos mecanismos "*new*" e "*overrides*" na declaração dos métodos. O valor das medidas para os produtos é apresentado na Tabela 6-9. Em comparação com a LPS TankWar Legada, a utilização da diretrizes não alterou o valor das medidas. A não alteração é justificada pelo fato dos dois mecanismos terem sido adicionados na própria declaração do método. Com isso, não foram adicionadas linhas de código e de comentários. A utilização das diretrizes do critério 4 torna a declaração explícita de métodos novos ou sobrescritos facilitando a leitura e entendimento do código fonte.

Tabela 6-9 - Valores das Medidas Após a Utilizar as Diretrizes do Critério 4

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	526	508	885	924	925	964
N1	2.700	2.673	4.362	4.547	4.599	4.784
N2	4.972	4.863	8.345	8.633	9.072	9.360
n	560	543	919	959	959	999
N	7.672	7.536	12.707	13.180	13.671	14.144
V	70.039,85	68.463,11	125.086,7	130.553	135.416,54	140.935,63
CC	3,28	3,4	3,6	3,59	3,24	3,25
TNOL	2.209	2.190	3.805	4.006	4.287	4.495
LOC	1.838	1.802	3.051	3.197	3.330	3.476
CLOC	67	71	202	208	269	275
NOM	16	14	20	20	20	20
MI	81,0228	81,2680	79,07685	77,7905	79,6694	78,4091

6.5.2.5. Utilização das Diretrizes do Critério 5

Para verificar o critério 5, foi utilizado o resultado da regra *GodClass* do *plug-in* `PMD` para detectar classes com excesso de responsabilidade. Houve retorno da regra nos produtos `Produto3IntermediarioPC`, `Produto4IntermediarioHandy`, `Produto5MaximoPC` e `Produto6MaximoHandy`, o qual sinalizou para as classes `Menu`,

`TankManager`, `Maler` e `Tank` com média ponderada de 4,6; 3,6; 1,7 e 1,7, respectivamente. A classe `Menu` foi refatorada de forma que três dos seus principais métodos (métodos `add` sobrecarregado) foram removidos e adicionados em nova classe chamada `Increments`. Essa classe estende a classe `Menu` por estabelecer relacionamento de generalização. Com isso, parte da responsabilidade da classe `Menu` foi retirada e o *plug-in* `PMD` não mais a acusou como *GodClass*. Para as outras três classes, nada foi feito, por não ser trivial a sua refatoração, o que poderia afetar negativamente a manutenibilidade da LPS TankWar.

O valor das medidas para os produtos é apresentado na Tabela 6-10. Em comparação com a LPS TankWar Legada, a utilização das diretrizes modificou o valor das medidas n2, N1, n e NOM em 1 ponto para mais ou para menos e o valor das medidas N2, N, V, CC, TNOL, LOC e NOM em pequena expressão. Além disso, o valor das medidas CLOC e n1 não foram alteradas. A NOM foi alterada por causa da criação da classe `Increments`.

Tabela 6-10 - Valores das Medidas Após a Utilizar as Diretrizes do Critério 5

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	527	509	886	925	926	965
N1	2.701	2.674	4.363	4.548	4.600	4.785
N2	4.972	4.882	8.352	8.652	9.079	9.379
n	561	544	920	960	960	1.000
N	7.689	7.556	12.715	13.200	13.679	14.164
V	70.132,66	68.664,86	125.185,4	130.770,95	135.516,35	141.155,36
CC	3,26	3,38	3,59	3,58	3,23	3,25
TNOL	2.217	2.206	3.813	4.022	4.295	4.511
LOC	1.840	1.812	3.053	3.207	3.332	3.486
CLOC	67	71	202	208	269	275
NOM	17	15	21	21	21	21
MI	81,4607	81,6062	79,4015	78,0463	79,9918	78,6646

6.5.2.6. Utilização das Diretrizes do Critério 6

Para verificar o critério 6, foi realizada inspeção manual nas classes para encontrar atributos. Em seguida, cada atributo foi analisado e, caso fosse possível, o modificador de acesso era alterado. Em alguns casos, a alteração do modificador de acesso foi simples, apenas reduzindo sua visibilidade de público para privado, por exemplo. Em outros casos, foi necessário criar métodos (*get* e *set*) que recebiam e/ou alteravam o valor do atributo. Com a utilização das diretrizes, foram alterados 96 modificadores de acesso dos atributos, eles eram 8 públicos, 55 protegidos e 33 padrão (*default*). Metade dos atributos públicos tornaram-se protegidos, 1 atributo protegido não estava sendo utilizado e foi removido (atributo `tankScore2` da classe `TankManager` da característica `Record`) e os atributos restantes tornaram-se privados. Com a mudança do modificador de acesso desses atributos, 59 métodos foram adicionados a LPS TankWar. Na mudança do modificador de acesso de 14 atributos, 2 métodos tiveram que ser criados (*get* e *set*), em outros 31 atributos apenas um método teve que ser criado e nos demais nenhum método foi criado.

O valor das medidas para os produtos é apresentado na Tabela 6-11. Em comparação com a LPS TankWar Legada, após utilizar as diretrizes, o valor das medidas sofreram alterações em todos os produtos (variações dos valores presentes na Tabela 6-5 e na Tabela 6-11), com exceção do valor das medidas n1, CLOC e NOM. A adição de métodos fez com que a quantidade de operadores e de operandos aumentasse, consequentemente, o volume aumentou. Além disso, a quantidade de linhas de código aumentou e, consequentemente, a quantidade de linhas aumentou. O valor da medida CC diminuiu, pois a LPS TankWar nova versão possui mais métodos com menos quantidade de linhas, fator que ajuda a reduzir essa medida. A quantidade de linhas de comentários permaneceu a mesma, pois os métodos novos não foram documentado. Como a

maioria dos métodos foram criados em características fundamentais para o funcionamento do jogo, as medidas dos produtos com quantidade menor de características tiveram maior impacto.

Tabela 6-11 - Valores das Medidas Após a Utilizar as Diretrizes do Critério 6

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	587	566	948	984	988	1.024
N1	2.889	2.856	4.620	4.796	4.889	5.065
N2	5.113	4.994	8.489	8.767	9.216	9.494
N	621	601	982	1.019	1.022	1.059
N	8.002	7.850	13.109	13.563	14.105	14.559
V	74.246,15	72.465,08	130.297,94	135.534,22	141.010,21	146.295
CC	2,57	2,7	3,05	3,11	2,84	2,89
TNOL	2.449	2.419	4.050	4.239	4.533	4.729
LOC	2.022	1.978	3.237	3.375	3.516	3.654
CLOC	67	71	202	208	269	275
NOM	16	14	20	20	20	20
MI	77,9956	78,2734	77,1660	76,0451	77,8878	76,7772

6.5.2.7. Utilização das Diretrizes do Critério 7

Para verificar o critério 7, foi realizada modificação manual no código fonte seguindo a convenção de código Java. Os pontos dessa convenção seguidos foram: i) organização das declarações; ii) endentações; iii) uso correto de comentários; iv) declarações, envolvendo quantidade de declarações por linha; v) instruções (*statements*); e, vi) uso correto de linhas em branco. A nomenclatura não foi levada em conta, pois, na implementação da LPS TankWar, houve a mistura de inglês e alemão e a alteração tornaria o processo lento e complicado, além de não modificar em nada o valor da medida MI.

O valor das medidas para os produtos é apresentado na Tabela 6-12. Em comparação com a LPS TankWar Legada, após utilizar as diretrizes, apenas o valor das medidas N2, N, V, TNOL e LOC apresentaram alterações. A alteração do valor das medidas N2, N e V foram mínimas, porém o valor da medida

TNOL sofreu alteração mais "sensível" em todos os produtos, pois havia várias declarações de atributos na mesma linha no código fonte. Além disso, em geral, não havia linhas em branco entre seções/declarações de classes e métodos. Por isso, a quantidade de linhas aumentou em 291, 312, 665, 745, 718, 791, para os produtos `Produto1MinimoPC`, `Produto2MinimoHandy`, `Produto3IntermediarioPC`, `Produto4IntermediarioHandy`, `Produto5MaximoPC` e `Produto6MaximoHandy`, respectivamente. As alterações nas declarações dos atributos impactaram na quantidade de linhas, aumentando-as, e optou-se por não remover os comentários, apenas enquadrá-los nas convenções Java.

Tabela 6-12 - Valores das Medidas Após a Utilizar as Diretrizes do Critério 7

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	526	508	885	924	925	964
N1	2.700	2.673	4.362	4.547	4.599	4.784
N2	4.972	4.863	8.345	8.633	9.072	9.360
N	560	543	919	959	959	999
N	7.673	7.537	12.708	13.181	13.672	14.145
V	70.048,98	68.472,2	125.096,5	130.562,9	135.426,5	140.945,6
CC	3,28	3,4	3,6	3,59	3,24	3,25
TNOL	2.500	2.502	4.470	4.751	5.005	5.286
LOC	1.829	1.809	3.040	3.215	3.320	3.495
CLOC	67	71	202	208	269	275
NOM	16	14	20	20	20	20
MI	79,4961	79,4016	76,9125	75,3545	77,5495	76,0539

6.5.3. Análise do MI Após Utilizar Diretrizes

O valor das variações do MI de cada produto após utilizar as diretrizes é apresentado na Tabela 6-13. Esse valor foi obtido pela diferença entre o MI de cada produto posteriormente a utilização de cada conjunto de diretrizes e o MI da cada produto da LPS TankWar Legada. Por exemplo, o MI do produto `Produto3IntermediarioPC` é 79,0768; após utilizar as diretrizes do

critério 7 nesse produto, o MI é 76,9125 o que implica em uma alteração de -2,1643 pontos. Ao analisar os valores dessa tabela, o impacto no MI dos produtos pode ser classificado em três categorias:

- **impacto positivo**: produtos gerados da LPS TankWar Legada após utilizar as diretrizes dos critérios 1 e 5. Considerando a utilização das diretrizes do critério 1, esse impacto aconteceu por causa da adição dos comentários e, consequentemente, das linhas adicionadas. A adição de comentários de documentação no código fonte melhorou em 18,55; 14,56; 14,18; 14,47; 11,53 e 11,83 pontos o MI dos produtos. Considerando a utilização das diretrizes do critério 1, a maioria das medidas foi alterada de forma que prejudicaram o MI; apesar disso, o impacto no MI foi positivo para os produtos por causa do aumento da quantidade de módulos (classes). Esse aumento, *per si*, aumenta o valor do MI, pois, no cálculo do MI, as medidas V, CC e LOC são divididas pela quantidade de módulos. No entanto, o valor da medida V é muito maior que as demais medidas, além de sofrer maiores variações. Por exemplo, no produto `Produto1MinimoPC`, os valores das medidas V, CC e LOC na LPS TankWar Legada são: 70.039,85; 3,28; e 1.838, respectivamente. Após utilizar as diretrizes do critério 5, o valor dessas medidas mudaram para 70.132,66; 3,26; e 1.840 e as variações foram de 93 pontos, 0,02 pontos e 2 linhas, respectivamente;

Tabela 6-13 - Variação do MI Após Utilizar Diretrizes Separadamente em Relação à LPS TankWar Legada

	Produtos					
Critérios	**1**	**2**	**3**	**4**	**5**	**6**
1	18,5517	14,5651	14,1876	14,4726	11,5347	11,8306
2	-0,0066	-0,0654	-0,2226	-0,2475	-0,2056	-0,2296
3	-0,7301	-0,9752	-0,5790	-0,8866	-0,5867	-0,8083
4	0	0	0	0	0	0
5	0,4378	0,3382	0,3246	0,2557	0,3223	0,2555
6	-3,0272	-2,9945	-1,9108	-1,7453	-1,7815	-1,6318
7	-1,5266	-1,8664	-2,1643	-2,4359	-2,1198	-2,3551

- **impacto neutro**: produtos gerados da LPS TankWar Legada após utilizar as diretrizes do critério 4. O valor de MI dos produtos continuaram os mesmos dos produtos da LPS TankWar Legada, pois as diretrizes do critério 4 consistem em adicionar os mecanismos "*new*" e "*overrides*". Como no cálculo do valor de MI não há o fator quantidade de caracteres, nada foi alterado.;
- **impacto negativo**: produtos gerados da LPS TankWar Legada após utilizar as diretrizes dos critérios 2, 3, 6 e 7. Em relação aos produtos gerados após utilizar as diretrizes do critério 2, a redução pode ser explicada, pois, na refatoração de métodos extensos, foram gerados vários métodos que, consequentemente, aumentaram o volume e as medidas referentes a contagem de linhas. Com isso, apesar da medida CC ter sido reduzida, o MI foi suavemente reduzido em: 0,0066; 0,0654; 0,2226; 0,2475; 0,2056; 0,2296, em ordem crescente, respectivamente (Tabela 6-13). A utilização dessas diretrizes se faz útil, pois na orientação a características métodos extensos são difíceis de refinar. Em relação aos produtos gerados após utilizar as diretrizes do critério 3, o valor de MI dos produtos foram reduzidos em comparação com a LPS TankWar Legada. Com essa utilização, todas as medidas foram modificadas em todos os produtos, exceto as medidas n1 e NOM. Apesar do valor de MI ter sido reduzido em 0,7301; 0,9752; 0,579; 0,8866; 0,5867; e 0,8083 para cada produto, respectivamente (Tabela 6-13), esse resultado pode ser explicado, pois a maioria das remoções de clones ocorreu em características que não podem estar contidas no mesmo produto e foram movidas para uma característica superior hierarquicamente no modelo de características. Com isso, ao invés de reduzir a quantidade de linhas, elas foram aumentadas. Em relação aos produtos gerados após utilizar as diretrizes do critério 6, o valor de MI foi reduzido, pois, na alteração do modificador de acesso, alguns métodos foram

criados, o gerou impacto negativo para quase todas as medidas. Os produtos com menor quantidade de características tiveram maior impacto negativo, pois a maioria das modificações (maior quantidade de métodos criados) ocorreram na característica `TankWar` (característica raiz). Em relação aos produtos gerados após utilizar as diretrizes do critério 7, o valor de MI foi reduzido, uma das ações que contribuiu na redução foi manter apenas uma declaração de atributo por linha. Como a LPS TankWar Legada não seguia esse padrão, a quantidade de linhas de código do produto `Produto6MaximoHandy` passou de 3.476 para 3.495 (acréscimo de 19 linhas). Outra ação que colaborou na redução foi adicionar linhas em branco. No produto `Produto6MaximoHandy`, a quantidade de linhas passou de 4.495 para 5.296 (acréscimo de 801 linhas), o que ocasionou a diminuição do percentual de linhas de comentário do código.

6.6. Avaliação dos Produtos Acumulando a Utilização das Diretrizes

Nesta seção, são apresentados os resultados da utilização das diretrizes em uma mesma LPS de forma acumulativa. Dessa forma, foi possível mensurar quanto as diretrizes contribuem no MI. Além disso, é apresentada uma comparação entre (i) a soma dos valores do MI de cada produto utilizando as diretrizes de forma separada e (ii) o valor de MI dos seis produtos gerados após utilizar as diretrizes de maneira sequencial acumulada.

6.6.1. Ordem de Avaliação dos Critérios e Utilização das Diretrizes

Com a verificação dos critérios separadamente, foram identificadas algumas situações que podem contribuir para agilizar a verificação dos critérios em conjunto e, com isso, foi estabelecida uma ordem (Figura 6-4). Observou-se a interferência dos resultados da utilização das diretrizes de um critério sobre a

das diretrizes de outro critério. Por exemplo, o critério 7 deve ser verificado antes do critério 1. A ordem de avaliação dos critérios e a justificativa dessa ordem são:

- **Critério 7**: estabelecer um padrão do codificação pode evitar retrabalho na utilização das diretrizes de outros critérios;
- **Critério 1**: documentar classes de métodos ajuda no entendimento do código e nas refatorações;
- **Critério 4**: deixar os refinamentos de métodos explícitos chama atenção do programador, caso alguma refatoração acontecer nesses refinamentos;
- **Critério 2**: refatorar métodos extensos podem ajudar na refatoração de classes e clones de código;
- **Critério 3**: reduzir clones de código pode evitar retrabalho na refatoração de classes;
- **Critério 6**: ter uma classe com os dados encapsulado pode facilitar sua modificação;
- **Critério 5**: conhecer o código e ter os dados encapsulados para refatorar uma classe é importante, pois é uma tarefa mais complexa e, por isso, deve ser realizada por último.

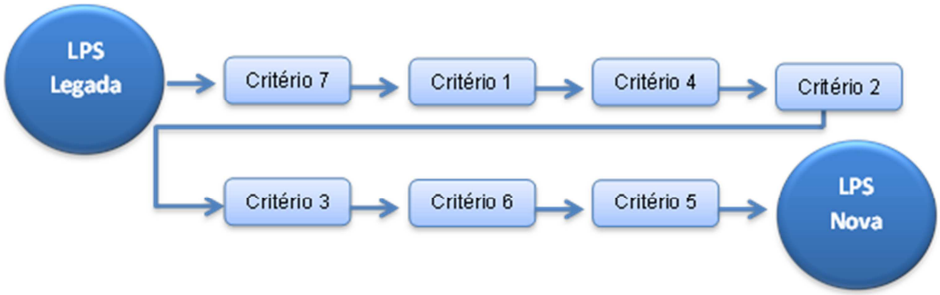


Figura 6-4 - Ordem de Avaliação dos Critérios

6.6.2. Utilização das Diretrizes Acumulativamente

Ao não atender um critério, o produto deve "sofre" uma transformação de modo a atendê-lo. Para realizar essa transformação, é sugerido um conjunto de diretrizes, associado ao critério, para ser utilizado. Nas próximas seções, são apresentadas a utilização das diretrizes e o cálculo das medidas apresentadas na Tabela 6-4, porém essa utilização foi realizada de maneira acumulativa. Para verificar o impacto da utilização das diretrizes de forma acumulada em relação aos impactos individuais, foi realizada a subtração entre (i) a soma das variações individuais do valor de MI dos critérios em questão (Tabela 6-13) e (ii) a soma das variações individuais do valor de MI acumulado.

6.6.2.1. Critérios 7 e 1

Iniciando a verificação dos critérios, foram utilizadas as diretrizes dos critérios 7 e 1, nessa ordem. O valor das medidas para os produtos resultantes são apresentados na Tabela 6-14. Pode-se observar que o valor de MI do produto `Produto1MinimoPC` é 99,8291. Os resultados do impacto da utilização das diretrizes de forma acumulada em relação aos impactos individuais foram 1,7811; 2,7289; 2,0588; 2,3027; 2,4653; e 2,6695 para os produtos. Esses valores representam impacto positivo.

Tabela 6-14 - Valores das Medidas Após Utilizar as Diretrizes dos Critérios 7 e 1

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	526	508	885	924	925	964
N1	2.700	2.673	4.362	4.547	4.599	4.784
N2	4.973	4.863	8.346	8.633	9.073	9.360
N	560	543	919	959	959	999
N	7.673	7.536	12.708	13.180	13.672	14.144
V	70.048,98	68.463,11	125.096,54	130.553	135.426,45	140.935,63
CC	3,28	3,4	3,6	3,59	3,24	3,25
TNOL	2.927	2.935	5.231	5.568	5.989	6.326
LOC	1.834	1.811	3.043	3.207	3.322	3.486

Tabela 6-14 - Valores das Medidas Após Utilizar as Diretrizes dos Critérios 7 e 1 (cont.)

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
CLOC	488	508	970	1.048	1.270	1.348
NOM	16	14	20	20	20	20
MI	99,8291	96,6957	93,1591	92,1300	91,5496	90,5542

6.6.2.2. Critérios 7, 1 e 4

Em seguida, o critério 4 foi verificado. Após utilizar as diretrizes desse critério de maneira acumulativa, o valor de MI não foi alterado (Tabela 6-15), ou seja, a variação permaneceu constante (1,7811; 2,7289; 2,0588; 2,3027; 2,4653; e 2,6695).

Tabela 6-15 - Valores das Medidas Após Utilizar as Diretrizes dos Critérios 7, 1 e 4

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	526	508	885	924	925	964
N1	2.700	2.673	4.362	4.547	4.599	4.784
N2	4.973	4.863	8.346	8.633	9.073	9.360
N	560	543	919	959	959	999
N	7.673	7.536	12.708	13.180	13.672	14.144
V	70.048,98	68.463,11	125.096,54	130.553	135.426,45	140.935,63
CC	3,28	3,4	3,6	3,59	3,24	3,25
TNOL	2.927	2.935	5.231	5.568	5.989	6.326
LOC	1.834	1.811	3.043	3.207	3.322	3.486
CLOC	488	508	970	1.048	1.270	1.348
NOM	16	14	20	20	20	20
MI	99,8291	96,6957	93,1591	92,1300	91,5496	90,5542

6.6.2.3. Critérios 7, 1, 4 e 2

Continuando a verificação com o critério 2, apenas métodos com mais de 50 linhas de código foram refatorados. O valor das medidas para os produtos resultantes são apresentados na Tabela 6-16. Os resultados do impacto da utilização das diretrizes de forma acumulada em relação aos impactos

individuais foram 1,7700; 2,5435; 2,3164; 2,5603; 2,6070; e 2,8264 para os produtos. Esses valores representam impacto positivo.

Tabela 6-16 - Valores das Medidas Após Utilizar as Diretrizes dos Critérios 7, 1, 4 e 2

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	532	514	893	934	933	974
N1	2.691	2.664	4.355	4.542	4.592	4.779
N2	4.972	4.862	8.350	8.643	9.077	9.370
N	566	549	927	969	967	1.009
N	7.663	7.526	12.705	131.185	13.669	14.149
V	70.075,51	68.491,58	125.225,88	130.799,85	135.560,55	141.188,77
CC	3,17	3,29	3,48	3,48	3,15	3,17
TNOL	2.953	2.960	5.288	5.628	6.046	6.386
LOC	1.834	1.811	3.054	3.219	3.333	3.498
CLOC	512	532	1.008	1.086	1.308	1.386
NOM	16	14	20	20	20	20
MI	99,8114	96,4448	93,1940	92,1400	91,4856	90,4813

O aumento do valor de MI dos produtos pode ser explicado pelo fato da utilização das diretrizes dos critérios anteriores. Por exemplo, na refatoração do método `menuBehandeln` da classe `Maler` da característica `PC`, o método `statusGamesStart` foi criado e adicionado seguindo as convenções de código Java (critério 7) e comentários de documentação foram introduzidos. Na refatoração do método `koordinateAktualisieren` da classe `Tank` da característica `TankWar`, 6 métodos foram criados e, em cada método gerado, as diretrizes dos critérios 7, 1 e 4 foram utilizadas. Como a maioria dos novos métodos foram criados sem ser em características básicas, os produtos mais completos tiveram maior impacto no valor de MI.

6.6.2.4. Critérios 7, 1, 4, 2 e 3

O critério 3 foi verificado de forma acumulada. O valor das medidas para os produtos resultantes são apresentados na Tabela 6-17. Os resultados do

impacto da utilização das diretrizes de forma acumulada em relação aos impactos individuais foram 1,9532; 2,6274; 2,6668; 3,1308; 2,8358; e 3,0834. Esses valores representam impacto positivo.

Tabela 6-17 - Valores das Medidas Após Utilizar as Diretrizes dos Critérios 7, 1, 4, 2 e 3

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	536	516	900	940	940	980
N1	2.701	2.663	4.370	4.557	4.607	4.794
N2	5.053	4.923	8.467	8.757	9.194	9.484
n	570	551	934	975	974	1.015
N	7.754	7.586	12.837	13.314	13.801	14.278
V	70.986,45	69.077,42	126.666,3	132.198,1	137.013,3	142.598,2
CC	2,91	3,05	3,25	3,3	2,98	3,04
TNOL	3.099	3.069	5.473	5.801	6.231	6.559
LOC	1.876	1.849	3.110	3.290	3.389	3.559
CLOC	572	5.866	1.093	1.164	1.393	1.526
NOM	16	14	20	20	20	20
MI	99,2644	95,5534	92,9653	91,8238	91,1277	89,9300

6.6.2.5. Critérios 7, 1, 4, 2, 3 e 6

Em seguida, o critério 6 foi verificado, porém as alterações de atributos das classes `Maler` não foram feitas nas características `PC` e `Handy`, pois o código clonado foi movido para a classe `Maler` da característica `Platform` ao utilizar as diretrizes do critério anterior. Assim, ao invés de alterar o modificador de acesso de 6 atributos e criar 12 métodos, o modificador de acesso de 3 atributos foram alterados e 6 métodos foram criados. O valor das medidas para os produtos resultantes são apresentados na Tabela 6-18. Os resultados do impacto da utilização das diretrizes de forma acumulada em relação aos impactos individuais foram 2,7224; 2,7222; 3,6029; 3,9806; 3,5462; e 3,8756. Esses valores representam impacto positivo.

Tabela 6-18 - Valores das Medidas Após Utilizar as Diretrizes dos Critérios 7, 1, 4, 2, 3 e 6

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	593	570	957	994	997	1.034
N1	2.891	2.849	4.625	4.805	4.894	5.074
N2	5.188	5.052	8.605	8.889	9.332	9.616
n	627	605	991	1.029	1.031	1.069
N	8.079	7.901	13.230	13.694	14.226	14.690
V	75.072,66	73.011,49	131.674,76	137.036,23	142.399,82	147.811,45
CC	2,39	2,52	2,83	2,91	2,67	2,75
TNOL	3.535	3.488	5.915	6.226	6.673	6.984
LOC	2.045	2.012	3.281	3.455	3.560	3.734
CLOC	731	739	1.255	1.320	1.555	1.620
NOM	16	14	20	20	20	20
MI	97,0086	92,6537	91,9906	90,9283	90,0566	89,0902

6.6.2.6. Critérios 7, 1, 4, 2, 3, 6 e 5

Por fim, o critério 5 foi verificado, mas, no retorno da regra `GodClass` do *plug-in* `PMD`, para detectar classes com excesso de responsabilidade, há diferentes resultados para as classes `Menu`, `TankManager`, Maler e `Tank`, sendo eles em média ponderada, 3,4; 2,5; 1,1 e 1, respectivamente. Esse fato mostra que a aplicação dos demais critérios resultou em alterações no excesso de responsabilidade dessas quatro classes de forma positiva. O valor das medidas para os produtos resultantes são apresentados na Tabela 6-19. Os resultados do impacto da utilização das diretrizes de forma acumulada em relação aos impactos individuais foram 5,9092; 6,9538; 6,7841; 7,4341; 6,8398; e 7,3981. Esses valores representam impacto positivo.

Tabela 6-19 - Valores das Medidas Após Utilizar as Diretrizes dos Critérios 7, 1, 4, 2, 3, 6 e 5

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
n1	34	35	34	35	34	35
n2	594	571	958	995	998	1.035
N1	2.892	2.850	4.626	4.806	4.895	5.075

Tabela 6-19 - Valores das Medidas Após Utilizar as Diretrizes dos Critérios 7, 1, 4, 2, 3, 6 e 5

	Produtos					
Medida	**1**	**2**	**3**	**4**	**5**	**6**
N2	5.195	5.059	8.612	8.896	9.339	9.623
n	628	606	992	1.030	1.032	1.070
N	8.087	7.909	13.238	13.702	14.234	14.698
V	75.165,59	73.105,26	131.773,26	137.135,48	142.499,8	147.911,78
CC	2,39	2,51	2,83	2,91	2,66	2,74
TNOL	3.550	3.503	5.930	6.241	6.688	6.999
LOC	2.052	2.019	3.288	3.462	3.567	3.741
CLOC	735	743	1.259	1.324	1.559	1.624
NOM	17	15	21	21	21	21
MI	97,0086	92,6537	91,9906	90,9283	90,0566	89,0902

6.7. Análise das Linhas de Produtos

Nesta seção, são apresentadas uma comparação entre o mapeamento realizado na LPS TankWar Legada e na LPS TankWar Nova e uma análise sobre as alterações do valor do MI de cada produto das LPSs. A LPS TankWar Nova é resultante da LPS TankWar Legada após a utilização das diretrizes. Com as alterações, ela passou a ter 7.450 linhas no total, incluindo linhas de código fonte (4.442 linhas - 59,6%), de documentação (1.460 linhas - 19,6%), de comentários (22 linhas - 0,3%) e em branco (1.526 linhas - 20,5%). (Tabela 6-20).

A LPS TankWar Nova é composta por 92 classes, sendo que três continuaram do tipo interface. A quantidade de refinamentos de classes e de interfaces continuou proporcionalmente equivalente (68% das classes) com alteração de 58 para 63. A quantidade de métodos foi alterado para 354, sendo 139 (39,26%) pertencem a refinamentos de classe. A quantidade de importações (*imports*) foi alterado em menor escala de 87 para 88. Essa alteração aconteceu por causa das modificações ocasionadas pela utilização das diretrizes do critério 3 (remoção dos códigos clonados).

Tabela 6-20 – Valores das Medidas Utilizadas para Mapear da LPS TankWar Legada e da LPS TankWar Nova

Medidas	LPS (quantidade)	
	Legada	Nova
Linhas Totais	5.640	7.450
Linhas de Código	4.865	4.442
Linhas de Comentários de Documentação	44	1.460
Linhas de Comentários	61	22
Classes (Constantes)	29	33
Interfaces (Constantes)	1	1
Refinamentos de Classes	56	61
Refinamentos de Interfaces	2	2
Métodos	308	354
Métodos em Refinamentos de Classe	97	139
Importações	87	88
Atributos	236	208
Constantes	53	51

A quantidade de atributos foi alterado de 236 para 208, por causa da utilização das diretrizes do critério 3, quando os clones de código foram alterados para classes de características superiores hierarquicamente no modelo de características (alguns atributos foram removidos). Esse fato também aconteceu com as duas constantes removidas, elas movidas da classe `Maler` das características `PC` e `Handy` para a classe `Maler` da característica `Platform`. Em comparação com o modificador de acesso dos atributos houve maior grau de encapsulamento, pois a quantidade de modificadores padrão (*default*) reduziu de 52 para 16, de públicos reduziu de 20 para 16, de protegidos (*protected*) reduziu de 128 para 84 e de privados (*private*) aumentou de 36 para 110. O valor do MI do produto `Produto1MinimoPC` derivado da LPS TankWar nova aumentou de 81,0228 para 97,0086 (Figura 6-5).

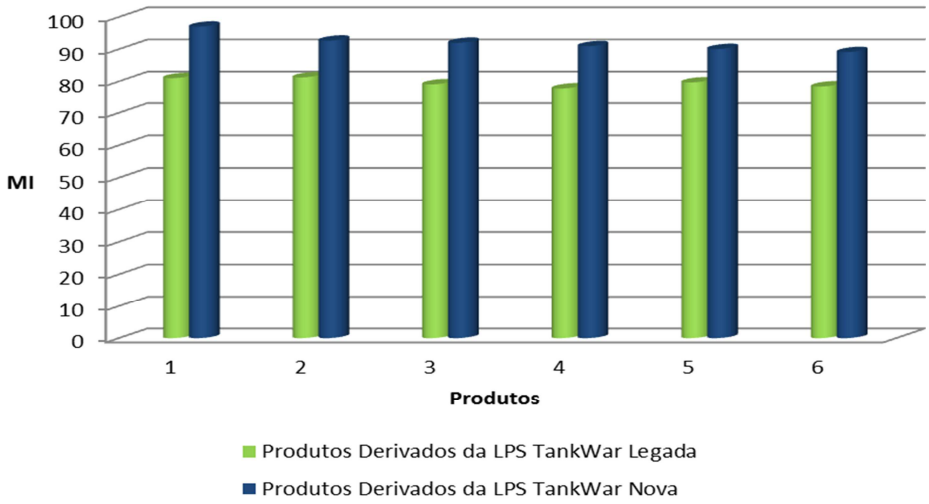


Figura 6-5 - Variação do valor do MI da LPS TankWar Legada e da LPS TankWar Nova nos Seis Produtos

Os resultados obtidos foram positivos em todos os produtos, principalmente, pela percepção da deficiência de documentação da LPS TankWar Legada e definição dos critérios e elaboração das diretrizes para solucionar esse problema. Quando as diretrizes foram utilizadas de forma acumulativa, por exemplo, os novos métodos passaram a ser documentados e os resultados foram melhores. Isso pode comprovar a importância e a influência da documentação do código fonte na equação do MI, pois, com um código bem documentado, melhora-se o seu entendimento e torna-se menos árdua a manutenção.

6.8. Considerações Finais

No Índice de Manutenibilidade, utiliza-se uma combinação de medidas que mensuram diversos fatores de um sistema como o Volume (V) e a Complexidade Ciclomática (CC). Com os resultados, pode-se perceber que, quando as diretrizes foram utilizadas de forma acumulada, melhores resultados foram obtidos, pois a quantidade de linhas de código foi reduzida. Essa

quantidade não está relacionada ao tamanho ou ao idioma dos nomes dos métodos ou dos atributos, ou seja, não necessariamente se a codificação de um sistema possuir menos caracteres significará que ele possui menos código, ela está relacionada à quantidade de comandos e de atributos que o código possui.

A adoção dos critérios pode ser determinante na melhoria da legibilidade do código, pois o entendimento de um comando simples é menos árduo do que ter que interpretar a lógica desse comando; por isso, a utilização das diretrizes dos critérios 1, 4 e 7 são importantes. Além de melhorar a legibilidade do código, a utilização das diretrizes ajudou na redução da complexidade, pois reflete que, em uma eventual manutenção, seja ela perfectiva ou não, o código fonte estará mais difícil de ser alterado, de ser entendido e com menos risco de ocorrerem efeitos colaterais. Realizar a manutenção em uma LPS é ainda mais complicado do que realizar a manutenção em sistemas únicos, tendo em vista que a alteração em um módulo pode alterar uma função não utilizada no produto escolhido.

Para a LPS TankWar Nova, a utilização das diretrizes do critério 1 foi o que mais causou impacto no valor do MI, pois LPS TankWar Legada estava praticamente sem comentários. A utilização das diretrizes na LPS TankWar Legada não reflete em benefícios diretamente para os produtos derivados, contudo o resultado final foi positivo e essa utilização na ordem sugerida pode ajudar a reduzir trabalhos/esforços posteriores.

7. CONSIDERAÇÕES FINAIS

7.1. Conclusões

O desenvolvimento de sistemas de software deve ocorrer cada vez mais rápido para atender a demanda do mercado. Muitas vezes, esses sistemas possuem pequenas variações e podem ser desenvolvidos em linhas de produtos de software (LPS) com utilização de características. Os benefícios de construir sistemas utilizando LPS vão além do desenvolvimento rápido, pois pode-se promover a reutilização sistemática e em larga escala de componentes. Se bem feita, essa reutilização pode aumentar a qualidade e a manutenibilidade dos sistemas. Entre as tecnologias disponíveis para a implementação de uma LPS, foi escolhida AHEAD, uma tecnologia orientada a características. As características permitem diferenciar os produtos e podem ser definidas como um módulo ou conjunto de módulos de uma aplicação com funções coerentes, bem definidas, independentes e combináveis [Boucher *et al.*, 2010].

O objetivo principal, deste trabalho, foi avaliar o índice de manutenibilidade (*Maintainability Index* - MI) de sistemas de software derivados de uma LPS. Esse índice envolve várias medidas, como o volume de Halstead, a complexidade ciclomática e a quantidade de linhas de código. A avaliação ocorreu em seis produtos gerados da LPS TankWar, um jogo desenvolvido na Universidade de Magdeburg, com aproximadamente 5.000 linhas de código e 37 características. Com essa LPS, podem ser derivados em torno de 40.000 produtos, porém avaliar o MI de todos os produtos é uma tarefa árdua. Os produtos foram escolhidos de forma a utilizar todas as características da LPS.

A metodologia de desenvolvimento aconteceu da seguinte forma. Baseado nos sete critérios definidos, foi realizada uma análise e, quando um critério não foi atendimento, as suas diretrizes eram utilizadas para aumentar o valor do MI. As comparações para identificar a melhoria na manutenção dos sistemas eram feitas nos valores do MI antes e depois da utilização das diretrizes. Essa utilização foi feita de duas formas: i) as diretrizes de cada critério foram utilizadas de maneira separada na LPS TankWar Legada; e ii) as diretrizes foram utilizadas de forma acumulada, em ordem determinada de critérios. Os resultados foram satisfatórios e o valor do MI, na segunda forma de utilização, aumentou em 15,9858; 11,3857; 12,9138; 13,1378; 10,3872; 10,6823 pontos nos produtos `Produto1MinimoPC`, `Produto2MinimoHandy`, `Produto3IntermediarioPC`, `Produto4IntermediarioHandy`, `Produto5MaximoPC` e `Produto6MaximoHandy`, respectivamente. Esse valores foram obtidos na relação do valor do MI da LPS TankWar Nova e da LPS TankWar Legada. O resultado foi positivo, pois, com a utilização das diretrizes, algumas das principais deficiências da LPS TankWar Legada foram sanadas, por exemplo, falta de comentários de documentação (critério 1). Ao considerar o critério 1, uma análise sobre a qualidade dos comentários deve ser realizada, pois um sistema de software com muitos caracteres e/ou linhas de comentários não necessariamente torna-se mais fácil de ser entendido, caso esses comentários não relatem o objetivo da classe, dos métodos ou dos respectivos refinamentos.

Nessa análise, pôde-se perceber que, com a utilização das diretrizes dos critérios 2, 3, 6 e 7, de maneira separada, o MI foi reduzido, pois a avaliação aconteceu nos produtos e as alterações na LPS. Em alguns casos, como na redução de clones, a quantidade de linhas de código foi reduzida na LPS e acrescida nos produtos, por causa dos refinamentos de classes e de métodos

existentes. Com isso, pode-se concluir que o valor do MI dos produtos não varia de maneira direta e proporcional com o valor do MI da LPS TankWar. Além disso, realizar manutenção em uma LPS não é tão trivial como alterar sistemas únicos, pois alterações em vários módulos podem gerar efeitos colaterais em diversos produtos e testes devem ocorrer em vários produtos e não apenas em um produto.

Portanto os objetivos foram cumpridos, pois a LPS TankWar foi escolhida entre as linhas de produtos identificadas em uma revisão da literatura. Em uma busca em trabalhos que avaliaram a manutenibilidade de sistemas de software sejam eles orientados a objetos, aspectos ou características o Índice de Manutenibilidade foi escolhido e utilizado na avaliação das duas versões da LPS estudada (Legada e Nova). Ferramentas para medir o software foram identificadas, tais como, `Google CodePro Analytix` e `PMD` com intuito de reduzir o esforço manual e garantir a precisão das medidas. Com o resultado do estudo de caso, no qual os critérios foram avaliados e as diretrizes aplicadas, pôde-se perceber que eles fornecem ajuda efetiva para aumentar a manutenibilidade dos sistemas de software da LPS.

7.2. Contribuições

O presente trabalho apresenta a reunião de algumas LPS encontradas por meio de uma revisão na literatura. Critérios foram definidos e diretrizes foram elaboradas para avaliar e manter sistemas de software derivados de LPS. Além disso, foi sugerida uma ordem de aplicação dos critérios a fim de evitar retrabalho e tornar o processo de utilização das diretrizes mais eficiente.

7.3. Limitações

Esse trabalho possui algumas limitações, por exemplo, (i) os critérios foram definidos apenas com os problemas detectados na LPS estudada, apesar de tratar problemas genéricos, (ii) as avaliações do MI foram realizadas nos produtos e as alterações na LPS e (iii) a avaliação da manutenibilidade utiliza uma única medida, mesmo que ela seja composta por outras.

7.4. Trabalhos Futuros

Algumas sugestões de trabalhos futuros:

- Desenvolver uma ferramenta para medir a manutenibilidade de LPSs desenvolvidas por tecnologias baseadas em composição;
- Acrescentar outras medidas para avaliar a manutenibilidade de uma LPS;
- Reaplicar o estudo em outras LPS para comparar os resultados;
- Definir outros critérios e respectivas diretrizes para pontos distintos em relação às subcaracterísticas da característica de Manutenibilidade da norma ISO/IEC 25000.

REFERÊNCIAS BIBLIOGRÁFICAS

Abílio, R.; Teles, Pedro; Costa, H.; Figueiredo, E. **A Systematic Review of Contemporary Metrics for Software Maintainability** In: 6th Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software, pp. 130-139, 2012.

Aguayo, M. T. V.; Guerra, A. C.; Colombo, R. M. T. **Avaliação da Manutenibilidade de Produtos de Software**. In: Conferência IADIS Ibero-Americana www/internet 2005, pp. 432-436, 2005.

Ahmed, F.; Campbell, P.; Lagharid, M. S. **Cognitive Factors in Software Product Line Engineering**. In: 11th International Conference on Computer Modelling and Simulation, pp. 352-355, 2009.

Aldekoa, G.; Trujillo, S.; Sagardui, G.; Díaz, O.; **Quantifying Maintainability in Feature Oriented Product Lines**. In: 12th European Conference on Software Maintenance and Reengineering, pp. 243-247, 2008.

Antonellis, P.; Antoniou, D.; Kanellopoulos, Y.; Makris, C.; Theodoridis, E.; Tjortjis, C.; Tsirakis, N. **A Data Mining Methodology for Evaluating Maintainability According to ISO/IEC-9126 Software Engineering - Product Quality Standard**. In: 11th IEEE Conference on Software Maintenance and reengineering, pp. 21-23, 2007.

Batory, D. **Feature-Oriented Programming and the AHEAD Tool Suite**. In: 26th International Conference on Software Engineering, pp. 702-703, 2004.

Bolton, G., Johnston, S. **IfSQ Level-2 A Foundation-Level Standard for Computer Program Source Code**. Second Edition, 63p., 2009.

Boucher, Q.; Classen, A.; Faber, P.; Heymans, P. **Introducing TVL, a Text-Based Feature Modelling Language**. In: 4th International Workshop on Variability Modelling of Software-intensive Systems, pp. 159-162, 2010.

Brownsword, L.; Clements, P. **A Case Study in Successful Product Line Development**. Technical Report no. CMU/SEE-96-TR-016, Carnegie-Mellon Software Engineering Institute, 95p., 1996.

Bühne, S.; Chastek, G.; Käkölä, T.; Knauber, P.; Northrop, L. M.; Thiel, S. **Exploring the Context of Product Line Adoption**. In: 5th International Workshop on Product Family Engineering, pp. 19-31, 2003.

Chen, Y.; Gannod, G. C.; Collofello, J. S. **A Software Product Line Process Simulator**. In: 6th International Workshop Process Simulation and Modeling, pp. 385-409, 2005.

Clements, P.; Baas L.; Kazman R. **Software Architecture in Practice**. Second Edition. SEI Series in Software Engineering. Addison-Wesley, 2003.

Clements, P.; Northrop, L. M. **Software Product Lines: Practices and Patterns**. Addison-Wesley. Vol. 3, 563p., 2002.

Cohen, S. **Predicting when Software Product Lines Pays**. In: Technical Note CMU/SEE-2003-TN-017, 34p., 2003.

Deraman, A.; Layzell, P.J. **Software Design Criteria for Maintainability**. Pertanika Journal of Science and Technology, volume 3, pp. 1-18, 1995.

Etxeberria, L., Sagardui, G., Belategi, L. **Quality Aware Software Product Line Engineering**. In: Journal of the Brazilian Computer Society, volume 14, pp. 57-69, 2008.

Evanco, W.; Verner, J. **Revisiting Optimal Software Components**. In: 12th ESCOM Conference, pp. 114-120, 2001.

Fenton, N. E.; Pfleeger, S. L. **Software Metrics: A Rigorous and Practical Approach**. 2nd ed., PWS Pub. Co., 656p., 1996.

Ferreira, I. A.; **Análise do Impacto da Aplicação de Padrões de Projeto na Manutenibilidade de um Sistema Orientado a Objetos**. Monografia de Graduação do Departamento de Ciência da Computação da Universidade Federal de Lavras, 78p., 2012.

Fowler, M. Beck, K. Brant, J., Opdyke, W. Roberts, D. **Refactoring: Improving the Design of Existing Code**. Addison-Wesley, ISBN: 0-201-485672, 464p., 1999.

ISO/IEC 25000: 2005. **Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE**. International Standards Organization, 2005.

ISO/IEC 25010: 2010, **Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE)**, 2010.

ISO/IEC/IEEE 24765:2010. Systems and Software Engineering - Vocabulary. 2010.

Jung, C. F. **Metodologia Aplicada a Projetos de Pesquisa: Sistemas de Informação & Ciência da Computação**. Taquara. 2009. Disponível em: <http://www.jung.pro.br/moodle/mod/resource/view.php?id=102>

Juran, J. M.; Gryna Jr., F. M. **Quality Planning an Analysis From Product Development Through Use**. McGraw-Hill, 684p., 1970.

Kang, K. **Feature-Oriented Domain Analysis Feasibility Study**. Technical report, no. CMU/SEI-90-TR-21 Carnegie Mellon University, 163p., 1990.

Kitchenham, B.; Brereton, O. P.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. **Systematic literature reviews in software engineering - A systematic literature review**. Information and Software Technology, vol.51, n.1, pp. 7-15, 2009.

Krueger, C. W. **Easing the Transition to Software Mass Customization**. In: 4th International Workshop on Product Family Engineering, pp. 282-293, 2001.

Laguna, M. A.; Crespo, Y. **A Systemtic Mapping study on software product line evoluton: From Legacy System Reenginnering to Product Line Refactoring** In: Science of Computer Programming, pp. 1010-1034, 2012.

Lehman, M. M.; Belady, L. A. **Program Evolution: Processes of Software Change**. Academic Press, 538p., 1985.

Linden, F. J. van der; Schmid, K.; Rommes, E. **Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering**. Springer-Verlag, 333p., 2007.

Long, C. A. **Software Product Lines: Practices and Patterns**. In: IEEE Software, Vol.19(4), pp.131-132, 2002.

McGregor, J. D.; Northrop, L.M.; Jarrad, S.; Pohl, K. **Guest editors' introduction: Initiating software product lines**. IEEE Software, Vol. 19, pp. 24-27, 2002.

Mcllroy, M. D.; Buxton, J.M.; Naur, P.; Randell, B. **Mass-Produced Software Components**. In: Software Engineering Concepts and Techniques, pp. 88-98, 1969.

Nascimento, L. M.; Viana, D. L.; Silveira Neto, P. A. M.; Martins D. A. O.; Garcia, V. C.; Meira, S. R. L. **A Systematic Mapping Study on Domain-Specific Languages**. In: 7th International Conference on Software Engineering Advances, pp. 179-187, 2012.

Neighbors, J. M. **Report on the Domain Analysis Working Group Session**. In: Workshop on Software Reuse, eds. G. Booch and L. Williams, Rocky Mountain Inst. of Software Engineering (RMISE), 3p., 1988.

Northrop, L. M. **Software Product Line Adoption Roadmap**. CMU/SEI2004- TR- 022 Technical Note, 54p., 2004.

Oracle. **Code Conventions for the Java TM Programming Language**. Disponível em: http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html/, Acessado: 18/07/2013.

Parnas, D. L. **A Course on Software Engineering Techniques**. In: ACM SIGCSE, Second Technical Symposium, pp. 154-159, 1972.

Petrasch, R. **The Definition of Software Quality: A Practical Approach**. In: 10th International Symposium on software reliability Engineering, pp. 33-34, 1999.

Pfleeger, S. L.; Atlee, J. M. **Software Engineering: Theory and Practice**. 4 edition. Prentice Hall, 792p., 2009.

PMBoK, **A Guide to the Project Management Body of Knowledge**. Project Management Institute, Inc. ed. 5, 616p., 2013.

PMD. **Programming Mistake Detector**. Disponível em: <http://pmd.sourceforge.net/pmd-5.0.4/rules/java/codesize.html#CyclomaticComplexity> Acessado 15/07/2013.

Pohl, K.; Böckle, G.; Linden, F. J. van der. **Software Product Line Engineering: Foundations, Principles, and Techniques**. Springer-Verlag, 494p., 2005.

Pressman, R. S. **Software Engineering - A practitioner's Approach**. ed. 7, 860p., 2010.

Rothery, B. **ISO 9000**. Makron Books, 268p., 1993.

Santos, R. P. **Critérios de Manutenibilidade para Construção e Avaliação de Produtos de Software Orientados a Aspectos**. Monografia de Graduação do Departamento de Ciência da Computação da Universidade Federal de Lavras. 103p., 2007.

Schobbens, P. Y.; Heymans, P.; Trigaux, J. C. **Feature Diagrams: A Survey and a Formal Semantics**. In: 14th IEEE International Requirements Engineering Conference, pp. 139-148, 2006.

Schulze, S.; Apel, S.; Kästner, C. **Code Clones in Feature-Oriented Software Product Lines**. In: International Conference on Generative Programming and Component Engineering, pp. 103-112, 2010.

Schulze, S.; Thüm, T.; Kuhlemann, M.; Saake, G. **Variant-Preserving Refactoring in Feature-Oriented Software Product Lines**. In: 6th Workshop on Variability Modeling of Software-Intensive Systems, VaMoS '12, pp. 73-81, 2012.

Segura, S.; Hierons, R. M.; Benavides, D.; Ruiz-Cortés, A. **Automated Metamorphic Testing on the Analyses of Feature Models**. In: Information and Software Technology, volume 53(3), pp. 245-258, 2011.

SEI – Software Engineering Institute. **Software Technology Reference Guide - A Prototype**. Carnegie Mellon University Pittsburg, Pennsylvania 15213, 436p. , 1997.

Silva, A.; Anselmo, P.; Garcia, V.; Muniz, P. **Linhas de Produto de Software: Uma Tendência da Indústria**. In: V Encontro Regional de Informática Ceará - Piauí (ERCEMAPI 2011), Cap. 1, Ed. Sociedade Brasileira de Computação. 241p., 2011.

Sommerville, I. **Software Engineering**. 6th Edition. International Computer Science Serires, Addison-Wesley, 742p., 2010.

Weiss, D. M.; Lai, C. T. R. **Software Product-Line Engineering: A Family-Based Software Development Process**. In: Addison-Wesley, 426p., 1999.

Zanlorenci, E. P.; Burnett, R. C. **Abordagem de Engenharia de Requisitos em Software Legado**. Workshop em Engenharia de Requisitos, pp. 270-284, 2003.