Revista Brasileira de Informática na Educação – RBIE
Brazilian Journal of Computers in Education
(ISSN online: 2317-6121; print: 1414-5685)
http://br-ie.org/pub/index.php/rbie

Submission: 10/Dec/2016; 1st round notif.: 06/Feb/2017; New version: 29/Jun/2017; 2nd round notif.: 21/Sep/2017
Camera ready: 27/Oct/2017; Edition review: 12/Nov/2017; Available online: 12/Nov/2017; Published: 01/Jan/2018

Uma Linha de Produto de Software para Desenvolvimento de Jogos Educativos

Title: A Software Product Line for Development of Educational Games

Carlos Alberto Correia Lessa Filho
Instituto de Computação/UFAL
carloswgama@gmail.com

Arturo Hernandez Dominguez
Instituto de Computação/UFAL
arturohd@ic.ufal.br

Resumo

Estudos têm revelado que cerca de 90% do desenvolvimento de jogos não chegam a serem finalizados devido a fatores como tempo, custo ou falta de conhecimento específico. Essas dificuldades estão presentes tanto em empresas de jogos, quanto por desenvolvedores de jogos independentes. Por outro lado, os jogos têm se mostrado um forte aliado no ensino por motivar os alunos aos estudos. Diante destas duas informações obtidas através de pesquisas, este artigo tem como objetivo propor uma Linha de Produto de Software para o desenvolvimento de jogos educativos, visando facilitar o desenvolvimento de jogos desse gênero. A análise realizada envolveu o desenvolvimento de quatro jogos com o objetivo de avaliar a viabilidade, complexidade do código, qualidade, esforço, técnicas e ferramentas utilizadas. Baseando-se nos resultados, pode-se observar vantagens como diminuição no tempo de desenvolvimento, redução da Complexidade Ciclomática e direcionamento do desenvolvimento ao usar uma Linha de Produto no desenvolvimento de jogos educativos.
Palavras-Chave: *jogos educacionais, Linha de Produto de Software, motivação*

Abstract

Studies have revealed that about 90% of development of game fail to be completed due to factors such as time, cost or lack of specific knowledge. These difficulties are present in game companies, as well as by independent game developers. On the other hand, the games have proved to be a strong ally in teaching to motivate students to study. Faced with these two informations, obtained through research, this article aims to propose a Software Product Line for the development of educational games in order to facilitate the development of educational games. The analysis involved the development of four games in order to assess the feasibility, code complexity, quality, effort, techniques and tools used. Based on the results, one can realize advantages such as reduction in development time, Cyclomatic Complexity and development direction when using a Software Product Line in the development of educational games.
Keywords: *educational games, Software Product Line, motivation*

Cite as: Lessa Filho, C. A. C & Dominguez, A. H. (2018). A Software Product Line for Development of Educational Games (Uma Linha de Produto de Software para Desenvolvimento de Jogos Educativos). Brazilian Journal of Computers in Education (Revista Brasileira de Informática na Educação - RBIE), 26(1), 01-29. DOI: 10.5753/RBIE.2018.26.01.1

1 Introdução

Um grande problema enfrentado na educação do Brasil é o alto índice de desistência dos estudos por parte dos adolescentes. Tal fato preocupante possui uma relação forte com a falta de motivação aos estudos. A metodologia tradicional de ensino realizada entre professores e estudantes, por meio da transmissão do conteúdo pelo professor ao estudante de forma passiva e decorativa, tem demonstrado insatisfação pelos estudantes, de modo que, cerca de 40% dos estudantes entre 15 e 17 anos que deixam os estudos, informam como motivo do afastamento, o fato da escola ser desinteressante e não motivadora (Santos, Silva Neto, Silva Junior, & Bittencourt, 2015).

Para contornar essa realidade, pesquisas têm sido realizadas com foco em elevar a motivação aos estudos por partes dos alunos, não visando apenas a continuidade nos estudos, como também, levar o aprendiz a aprimorar seu conhecimento (Yoon & Kim, 2015). Algumas destas pesquisas buscaram a motivação por meio do *Problem-Based-Learning* (PBL), enquanto outras buscaram a motivação por meio da diversão do aprendiz. A busca da diversão unida ao aprendizado, por sua vez, apresentou um novo papel ao uso de jogos eletrônicos.

Deste modo, os jogos podem ser um forte contribuente para o aprendizado. Entretanto, apesar dos jogos representarem um dos maiores mercados financeiros (Savi & Dra, 2008), ainda existe um desafio para quem desejar construir seu próprio jogo. A maioria dos jogos que têm seus desenvolvimentos iniciados de forma independente não chega a ser concluído.

Em um seminário realizado em 2009 na Universidade Bunkyo Gakuin em Tóquio, que contou com a participação do grupo *International Game Developers Association* (IGDA) e do *game designer*, responsável por jogos de sucesso, Kenta Cho, foi observado que cerca de 90% dos projetos de jogos independentes iniciados não chegam a ser concluídos (Cho, 2009). A grande desistência no desenvolvimento de jogos independentes ocorre principalmente devido a três fatores que citaremos a seguir: falta de novas ideias durante o processo de desenvolvimento; muito tempo gasto ao longo da produção; o fato do jogo não aparentar mais ser tão interessante pelos desenvolvedores, quanto parecia no início de seu desenvolvimento.

Esta realidade e as dificuldades encontradas durante o desenvolvimento de jogos são ainda pior no cenário de jogos educativos devido à falta de ferramentas ou métodos que dêem suporte aos seus desenvolvimentos. Uma pesquisa realizada no ano de 2015 (Rocha & Bittencourt, 2015) demonstra que os jogos voltados para abordagens mais sérias, também conhecidos como Jogos Sérios, apresentam algumas dificuldades durante o seu desenvolvimento. Entre essas dificuldades é possível citar: a qualidade da metodologia na criação do jogo; a reusabilidade e estender os jogos sérios e seus artefatos; e como avaliar diferentes pontos do desenvolvimento deste tipo de jogo.

Visando uma alternativa para tentar diminuir alguns desses problemas como a falta de ideias nos requisitos do jogo, estratégias a serem adotadas, diminuição de custos e tempo, pode-se optar por seguir estratégias de Linha de Produto de Software (LPS). Uma LPS, segundo Käköla e Leitner (Käkölä & Leitner, 2014), é uma metodologia validada industrialmente para o desenvolvimento intensivo de sistemas e serviços de softwares com menor custo, maior velocidade, maior qualidade e satisfação do usuário final. Além de proporcionar uma maior validade para a implementação do jogo, diminuir o esforço, tempo e mão-de-obra, o uso de uma LPS também abrange as etapas de: elaboração da ideia com os requisitos; definição de soluções de problemas com a definição do projeto; e a validação do desenvolvimento através de testes.

Diante desses problemas no desenvolvimento de jogos, unidos à carência de ferramentas voltadas ao desenvolvimento de jogos educativos e cientes de uma possível alternativa como a escolha de uma Linha de Produto, este trabalho tem como objetivo desenvolver uma LPS

voltada ao desenvolvimento de jogos educativos que possibilite o suporte ao desenvolvimento de um jogo em várias de suas etapas. Para avaliar a LPS desenvolvida neste trabalho e o uso de LPS no desenvolvimento de jogos, foi realizado um estudo de caso envolvendo o desenvolvimento de quatro jogos baseados em jogos educativos existentes, objetivando o processo de construção de artefatos. A análise realizada tem como foco avaliar a viabilidade de construção de jogos através da LPS proposta; identificar as vantagens do desenvolvimento de jogos através de uma LPS por meio de métricas como número de linhas de código, tempo de desenvolvimento e complexidade; avaliar o uso de ferramentas de suporte ao desenvolvimento de LPS e técnicas de implementação da variabilidade, assim como facilidades e dificuldades encontradas no uso.

2 Referencial Teórico

Os jogos eletrônicos tiveram sua história iniciada por volta da década de 50 (Rogers, 2010), onde poucos tinham acesso a esse tipo de entretenimento digital. Os primeiros desenvolvedores de jogos eram estudantes de universidades como MIT ou funcionários de instalações militares que possuíam supercomputadores.

O primeiro jogo oficial na história dos jogos eletrônicos é o *Tennis for Two* desenvolvido por William Higinbotham no Laboratório de Brookhaven (Wolf, 2008). Este jogo possuía uma interface gráfica simples, sendo representada unicamente através de um ponto de luz indo de um canto a outro da tela, como uma partida de tênis. Alguns anos mais tarde, em 1962, com o lançamento do famoso jogo, *Spacewar*, desenvolvido no MIT, os jogos começaram a ter maior visibilidade com os diversos públicos. Novas ideias de desenvolvimento para jogos foram surgindo, e no ano de 1971 surgiu a primeira máquina de Arcade, utilizadas em estabelecimentos comerciais, dando continuidade no ano seguinte com o lançamento do primeiro *vídeo game* de casa, *Mangnavox Odyssey*.

Não demorou muito para que os jogos educativos começassem a fazer parte desse universo digital. Não há uma informação precisa sobre qual seria o primeiro jogo educativo eletrônico criado, contudo acredita-se que o "*Logo Programming*" (Linguagem Logo), o jogo da tartaruga que forma figuras geométricas, tenha sido o primeiro jogo educativo eletrônico (Online Universities, 2012).

Alguns anos depois, os jogos educativos eletrônicos disparam com a chegada do *Apple* II nas casas das pessoas. Novos jogos ganharam espaços, como o "*Lemonade Stand*", que se trata de um jogo de uma barraca de venda de limões, onde a criança pode aprender um pouco sobre matemática e economia.

Além da Linguagem Logo, o jogo "*Where in the World Is Carmen Sandiego?*" foi lançado originalmente em 1983, permitindo abrir as portas para outros jogos educativos fazerem parte das salas de aula, como uma ferramenta complementar ao ensino. O jogo "*Where in the world is Carmen Sandiego?*" (Carmen Sandiego, 2015) obteve de forma rápida grandes fãs entre professores, pais e estudantes por incentivar o jogador a aprender diversos conteúdos e colocar seus conhecimentos em prática, para conseguir capturar a ladra Carmen Sandiego que viaja por diversos países do mundo. Mais de 300 mil escolas dos Estados Unidos utilizaram o jogo em sua grade curricular e o jogo obteve mais de 90 prêmios educacionais.

Na década de 90, um novo termo chamado *edutainment*, que se trata da junção de *education* e *entertaiment* foi introduzido. O termo buscava tratar de meios de entretenimento com foco na educação, de modo que um usuário desta ferramenta pudesse aprender um determinado conteúdo sem que percebesse que estava passando por um processo de aprendizagem. No Brasil, ficou conhecido como edutenimento. Esse movimento contribuiu para a produção de mais jogos

educativos e para classificação de jogos já produzidos anteriormente, como foi o caso do famoso *"Where in the World Is Carmen Sandiego?"* (Molina, 2013).

O uso do *edutainment* ainda é bastante discutido por autores. Há autores que são a favor desses jogos educativos por motivar aos estudantes o aprendizado tradicional sem perceber que está estudando. Também há autores que veem isso como uma forma negativa, por gerar a impressão que o processo de estudar é algo negativo e que deve ficar escondido atrás do entretenimento (Ariffin, Oxley, & Sulaiman, 2014).

Essa discussão é o principal ponto que difere um jogo sério e um jogo *edutainment*, onde um jogo sério também se preocupa com o processo de aprendizado, porém sem se atentar tanto com o entretenimento do estudante com o jogo, deixando clara a ideia de ser um processo de aprendizagem.

Não muito distante do surgimento do movimento *edutainment*, houve o lançamento do jogo *SimCity*, baseado na construção e administração de cidades, que também passou a ser utilizado como exemplo nas salas de aula. Muitos desses jogos educativos que fizeram parte dos primeiros 30 anos dos jogos eletrônicos serviram de inspiração para a produção de novos jogos educativos, tanto por despertar interesse dos estudantes, quanto também pelas instituições de ensino que buscavam novos meios de motivar os seus alunos aos estudos.

Embora os jogos educativos eletrônicos estejam presentes desde meados do século passado, apenas no ano de 2000, o termo *Game-Based Learning* (GBL) começou a ser utilizado (Ariffin et al, 2014). Foi através desta metodologia não tradicionalista e com o uso de recursos presentes na vida de muitos estudantes, que os jogos começaram a ser tratados como uma ferramenta que pode permitir o aumento no interesse dos estudos pelos alunos, com maior facilidade comparada com as metodologias tradicionais (Luo, Wei, & Zhang, 2010). Ao analisar formas de ensino, tal representação já costumava ser utilizada sem ser por meios eletrônicos, onde professores ensinavam a matemática com a repartição de maçãs, por ser algo do cotidiano do estudante e de fácil compreensão. O uso de jogos permite executar esse método de forma mais ampla e divertida pelos estudantes.

A vantagem dos jogos educativos digitais ao se comparar com os jogos educativos não digitais está na maior praticidade para realizar as demonstrações de conteúdos de forma didática e lúdica. A união entre computador e jogos educativos, além de possibilitar novas formas de realizar e avaliar o aprendizado presencialmente, também possibilita o Ensino a Distância (EaD), de forma que o professor, mesmo sem a presença física, tenha a condição de ensinar um determinado conteúdo e avaliar a performance dos seus estudantes (Ruben, 1999).

Alguns jogos educativos também podem ter funções que vão além de transmitir conhecimento e motivar aos estudos, como também aprimorar as habilidades através de treinamento, como é o caso do jogo *Virtual Reality Dance Training System* (Chan, Leung, & Tang, 2011). Nesse, o jogador deverá imitar os movimentos apresentados por um professor virtual, recebendo em seguida o *feedback* dos movimentos realizados. Esse e outros jogos similares apresentam um sistema que coleta as informações dos movimentos realizados pelo jogador durante a partida e ao final ou durante a execução dos movimentos, é exibido na tela o desempenho do jogador, para que este possa aprimorar suas habilidades.

Uma técnica utilizada nos jogos eletrônicos que contribuem para o desenvolvimento do raciocínio está presente na forma como o jogador necessita assimilar a informação e tomar decisões rapidamente, precisando passar por um processo que o leve a deduzir as regras do jogo (Prensky, 2001).

Apesar das vantagens apresentadas no uso de jogos educativos eletrônicos, também é possível observar que tais jogos apresentam temas específicos, de forma que dificulte o seu uso em diferentes conteúdos. Diante dessa situação, professores, estudantes ou colaboradores podem

precisar criar seus próprios jogos para atender as suas reais necessidades. Porém, o desenvolvimento de jogos como foi citado na introdução deste trabalho, pode ser uma tarefa complexa por vários fatores.

2.1 Jogos Educativos Construcionistas

Os jogos educativos construcionistas se assemelham aos jogos educativos digitais citados na Seção 2, entretanto com o diferencial de usar o conceito do construcionismo desenvolvido por Seymour Papert.

O construcionismo de Seymour Papert é baseado no construtivismo de Jean Piaget, o qual analisa o processo de aprendizado baseado na evolução biológica do ser, incluído os estágios do desenvolvimento cognitivo. Entretanto, para Papert, para que se possa obter o conhecimento completo sobre um determinado conteúdo, é necessário construir um objeto concreto, também chamado de artefato, sobre o que está sendo estudado (Papert & Harel, 1991).

A construção do artefato concreto permitirá ao estudante se deparar com situações inesperadas o qual não passaria caso estudasse apenas o conteúdo abstrato passado em sala de aula. Ao se deparar com falhas durante a construção do artefato, o estudante iniciará um processo de reflexão para compreender o motivo pelo qual seu planejamento não funcionou como esperado e o que ocasionou o comportamento do artefato. Após realizar as correções necessárias, o estudante terá um conhecimento mais aprofundado do conteúdo estudando.

Entretanto, a construção de um artefato concreto sobre o conteúdo estudado, pode não ser tão fácil de realizar no mundo real, de modo que os jogos podem facilitar esses processos.

Os jogos construcionistas podem abordar diversos assuntos como setores financeiros e administrativos com a construção e administração de uma empresa ou cidade desde sua fundação, como ocorre no jogo *SimCity* (MAXIS, 2016); biologia com a construção de animais virtuais através da combinação de genes, como o jogo *Gene2* (Morelato, Guima, & Borges, 2008); ou até mesmo programação como o jogo *Code Hunter* da equipe da Microsoft (Microsoft Research, 2017).

2.2 Engines

Para facilitar o desenvolvimento de jogos eletrônicos no âmbito da programação, o desenvolvedor pode optar pelo uso de *engines*, também conhecidos como motores.

Jogos costumam simular o mundo real como iluminação, efeitos de tiros ou fenômenos físicos, seguindo o mesmo conceito. Tais fatos e efeitos são implementados pelas *engines* dos jogos. A atração de um jogo, segundo Clua e Bittencourt (2004), normalmente está atrelada ao desafio imposto por ele. O funcionamento do jogo será governado através de um conjunto de regras, que definem o que os jogadores podem fazer e o que eles não podem realizar dentro desse mundo virtual. As *engines* de jogos são uma peça importante no desenvolvimento de jogos digitais. Uma *engine* de jogo possui uma arquitetura de software que possibilita a interligação de um conjunto de componentes.

As *engines* de jogos são plataformas adequadas para proporcionar a flexibilidade, extensão e reusabilidade de algoritmos, métodos e técnicas que usem com eficiência o processador em complexos ambientes. Tem como finalidade a criação e desenvolvimento de jogos com funções como renderização, física, detecção e colisão, e um eficiente sistema de gerenciamento de memória (Cadavid, Ibarra, & Salcedo, 2014). As *engines* responsáveis pela física são capazes de implementarem efeitos de reflexão, refração e difração, podendo ser combinadas com outras *engines* para se obter resultados mais satisfatórios, além de poder exportar os jogos para diferentes plataformas.

Algumas destas *engines* também podem ser utilizadas para gerar outros tipos de aplicativos que não sejam jogos como ambientes de simulação, que permite usuários verificarem como uma atividade será executada durante uma tarefa ou missão.

A seguir, serão apresentadas algumas das principais *engines* utilizadas para o desenvolvimento de jogos.

• **Unity 3D** - é uma *engine* de jogos desenvolvida pela empresa *Unity Tecnologies*, utilizada não apenas para desenvolvimento de jogos, como também para criar animações em tempo real. Os jogos desenvolvidos podem ser executados em diversas plataformas, como Windows, Mac ou Iphone (Bae & Kim, 2014). É uma solução intuitiva com um grande poder de customização, sendo possível desenvolver jogos 2D e 3D, entre as linguagens Javascript e C#, além de usar *workflows* intuitivos. A plataforma também conta com uma grande comunidade ativa, tutoriais para facilitar a aprendizagem, além de serviços como mecanismo de propaganda, servidores *Multiplayer*, análise de comportamento de jogadores e relatório de desempenho (Unity Technologies, 2016).

• **GameMaker** - é uma *engine* desenvolvida pela YoyoGames, permitindo o desenvolvimento do jogo sem a necessidade de programar, usando recursos como *Drag and Drop* (YoYoGames, 2016). Devido ao fato de ser uma ferramenta que permite o desenvolvimento de jogos sem conceitos complexos de programação, esta *engine* também costuma ser utilizadas por professores para ensinarem seus estudantes conceitos básicos de lógica e desenvolvimento de aplicativos, podendo ser visto exemplos em turmas de Ensino Fundamental no ensino de informática, como também em cursos de graduação em Ciências da Computação (Mello & Rebouças, 2015; Catete, Peddycord, & Barnes, 2015).

• **Construct** - é uma *engine* baseada no desenvolvimento de jogos 2D para HTML5, o qual não exige do desenvolvedor conhecimentos em programação. Assim como o GameMaker, o *Construct* 2 é uma ferramenta que possui um poderoso sistema de eventos baseado na lógica da aplicação. Com o *Construct* o desenvolvedor poderá realizar comportamentos baseados na física, movimentação de personagens, efeitos visuais e exportar o seu jogo em HTML5, que poderá ser publicado em ambientes como Chrome Web Store, Facebook, Kongregate, Newgrounds e Firefox Marketplace e até mesmo para *smartphones* (Sudarmilah, 2013).

• **Scratch** - é uma *engine* desenvolvida inicialmente para jovens entre 8 e 16 anos, mas passou a ser utilizadas por pessoas de diversas idades, incluindo crianças mais novas com seus pais (MIT Media Lab, 2016). A *engine* possibilita o desenvolvimento de jogos, estórias, animações, através de blocos de comandos baseados em lógica de programação. Por se tratar de uma *engine* voltada para um público mais novo, muitas pesquisas têm sido realizadas com o uso e desenvolvimento de jogos educativos durante o ensino de computação nas escolas, resultando em uma contribuição positiva por incentivar o pensamento criativo, trabalhos colaborativos e a pensar em forma sistemática, solucionando problemas através de um modo divertido de aprendizado (Von Wangenheim, 2014).

2.3 Linha de Produto de Software

Apesar das *engines* serem ferramentas poderosas que disponibilizam recursos como controle de física, colisão, renderização e iluminação, entre outros recursos como gerenciamento de áudio e imagens, ainda são voltadas para a programação e desenvolvimento dos jogos. Dessa forma, não envolvem outras etapas como definição da ideia, levantamento de requisitos e testes.

Em contra partida, a LPS é uma técnica que engloba todas as etapas do desenvolvimento de um *software*, podendo solucionar não apenas os problemas relacionados à implementação do jogo como custo e tempo, como também definição do projeto e testes de qualidades, evitando problemas futuros como *bugs*. Por se tratar de uma técnica mais ampla comparada apenas ao uso

de *engines*, uma LPS pode conter o uso de *engines* como o Unity 3D, durante a etapa de implementação do domínio e da aplicação.

O desenvolvimento de um jogo ou um *software* para uma empresa pode ser um processo muito demorado e de grandes custos ao longo de seu desenvolvimento. Entretanto, muitas empresas trabalham com o desenvolvimento de *softwares* no mesmo domínio, de forma que se houver a possibilidade de reutilizar recursos e etapas durante o desenvolvimento, esse processo pode se tornar mais rápido e barato.

O conceito de Linha de Produto de Software teve origem no setor automobilístico com a Linha de Produção desenvolvido por Henry Ford. Tinha como finalidade simplificar a montagem dos veículos reduzindo o esforço humano, através de especialização nos movimentos de cada etapa do desenvolvimento. Desta forma, aumenta a produtividade sem elevar os custos na produção e mantém o mesmo padrão em todos os veículos (Wood Junior, 1992).

Entretanto, a produção em massa da Linha de Produção não é um sistema que permitia a customização dos veículos produzidos e nem todas as pessoas possuíam a mesma necessidade. Por outro lado, a ideia de gerar uma produção em massa e customizada representaria um grande custo caso voltasse a ser adotada a ideia de uma produção em massa para cada customização possível. Diante dessa situação, deu-se início a produção de plataformas comuns (Pohl, Böckle, & Linden, 2005). As plataformas comuns são plataformas que possuem todas as características comuns daquele tipo de objeto que pretende ser criado, com o diferencial que essa plataforma permite adicionar com facilidade outros recursos específicos que podem variar de acordo com a necessidade de cada cliente. No exemplo dos carros, a plataforma comum pode ser representada contendo as características básicas comuns e obrigatórias em todos os carros, enquanto as variabilidades poderiam ser representadas como componentes opcionais ou alternativos como: ar-condicionado (opcional); e marcha automática ou manual (alternativo).

O conceito de Linhas de Produtos de Softwares segue o mesmo conceito desenvolvido para a produção em massa customizada, de modo que a construção de uma plataforma comum possa atender diversos tipos de software do mesmo domínio.

Para Northrop (2002), uma LPS é como um sistema de produção intensiva de *software* que compartilham características comuns e gerenciáveis para satisfazer um segmento particular do mercado. Os recursos da LPS incluem, mas não se limitam a: arquitetura; os componentes reusáveis; o modelo do domínio; documentação e especificação; planos de testes e descrição do processo. Ao final da produção, cada sistema criado através da LPS formará um sistema único para atender um cliente.

O processo de desenvolvimento de uma LPS não segue um fluxo fixo, podendo voltar ou avançar entre as etapas a qualquer momento para obter os melhores resultados. Porém, a etapa de desenvolvimento inicia-se com a análise dos requisitos que os *softwares* do mesmo domínio possuem. Essa análise pode ser realizada avaliando *softwares* do mesmo domínio ou entrevistando *stakeholders*. Após obter as primeiras características em comum desses *softwares*, deve ser planejado e implementado na plataforma comum, todas as características em comum, ou também como são chamadas: as comunalidades. Essa plataforma também deve ser baseada em componentes de forma a permitir a flexibilidade para durante a produção do produto final, poder adicionar novas customizações necessárias de cada *software*.

As flexibilidades presentes na LPS são chamadas de variabilidades, ou seja, são os pontos pré-definidos que podem receber customizações durante o desenvolvimento do produto final. No framework, a implementação dos pontos que podem ocorrer uma variabilidade, pode ser dada através de estruturas de decisões. Estes, por sua vez, podem ser chamados de pontos de variação. Os pontos de variação são os locais ou momentos nos artefatos onde uma estrutura de decisão pode ser realizada, enquanto as variantes são as alternativas dessas escolhas (Lobo,

Brito, & Rubira, 2007). As variações podem ocorrer na documentação, na arquitetura, código-fonte ou durante a execução do código (Silva, Neto, Garcia, & Muniz, 2011).

A representação das comunalidades e variabilidades nas LPS costumam ser expressas através de *feature model*, que é apresentado originalmente através do modelo *Feature Oriented Domain Analysis* (F.O.D.A.), desenvolvido pelo *Software Engineering Institute* e se tornando o método mais popular de representação de requisitos na década de 90 (Silva et al, 2011). As características presentes em um domínio, sejam comunalidades ou variabilidades, também podem ser representadas como *features*.

O *feature model* é representado por características do tipo:

Obrigatória/Mandatória – As características presentes em todos os produtos da LPS;

Opcional – As características que podem ou não estar presentes nos produtos da LPS;

Alternativa – Conjunto de características quem podem ser escolhidas uma (XOR) ou mais (OR) para preencher um determinado componente.

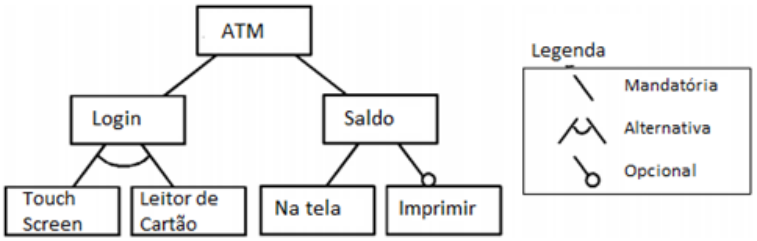

Figura 1 - *Feature Model* de uma ATM

Na Figura 1 é possível observar um simples exemplo da representação de um ATM (Caixa Eletrônico) através do *feature model*. O sistema ATM acima possui como característica alternativa a identificação do usuário por um mecanismo *Touch Screen* ou por leitor de cartão. Também é possível ver como característica opcional, a impressão do saldo do cliente.

A empresa que resolver realizar a construção de uma LPS poderá observar nos primeiros desenvolvimentos um custo investido elevado ao se comparar com o desenvolvimento isolado de um *software*. O custo está relacionado ao fato de que no primeiro momento, além de desenvolver o software, será preciso pensar e planejar o desenvolvimento da LPS que será utilizada na produção dos softwares. Porém, ao continuar utilizando a LPS desenvolvida, já será possível observar uma redução no investimento dos softwares aproximadamente a partir do terceiro produto desenvolvido (McGrego, 2002).

As vantagens encontradas no uso de uma LPS podem ser observadas através (Pohl et al, 2005): redução dos custos no desenvolvimento; aumento de qualidade do produto final; redução do tempo de produção.

Durante um estudo realizado na empresa *Celsius Tech*, foi possível observar que a implantação da LPS resultou em uma redução de mão de obra de 200 funcionários para apenas 50 funcionários; o tempo de entrega passou de anos para meses; 70 a 80% do *software* era implementado por componentes reusáveis e houve um aumento na qualidade e satisfação dos clientes em relação ao produto final (Durscki, Spindola, Burnett, & Reinehr, 2004).

A facilidade encontrada no desenvolvimento de *software* através do uso de uma LPS ocorre devido ao fato do processo ser dividido em duas etapas:

- Engenharia de Domínio – processo que estabelece as comunalidades e variabilidades do domínio. Também é nessa etapa onde é construída a plataforma comum. Esse processo é dividido em quatro subprocessos: Engenharia de Requisitos do Domínio; Projeto do Domínio; Implementação do Domínio; Teste do Domínio.

• Engenharia de Aplicação – Na engenharia da aplicação é possível encontrar os processos da construção do produto final utilizando os artefatos e a plataforma construídos no processo de engenharia do domínio. Esse processo é dividido em quatro subprocessos: Engenharia de Requisitos da Aplicação; Projeto da Aplicação; Implementação da Aplicação; Teste da aplicação final.

Os processos de definição dos requisitos, projeto, implementação e testes ocorrem de forma tanto na Engenharia do Domínio, quanto na Engenharia da Aplicação, de modo que no Domínio é desenvolvido um artefato genérico que possa ser utilizado por diferentes *softwares* do mesmo domínio, enquanto na Engenharia da Aplicação, esses artefatos são utilizados para gerar a aplicação final.

Seguindo o modelo de Pohl et al (Pohl et al, 2005), na Engenharia de Requisitos do Domínio são definidas as comunalidades e variabilidades que os softwares deste domínio podem possuir. Na Engenharia de Requisitos da Aplicação, os requisitos identificados no domínio são selecionados e novos requisitos exclusivos deste aplicativo são adicionados.

No Projeto do Domínio é definida uma arquitetura através de diagramas de componentes com o uso de componentes e interfaces para representar como os requisitos podem atender o desenvolvimento de um *software*. No Projeto da Aplicação, os componentes e interfaces presentes na aplicação em desenvolvimento são selecionados.

Na Implementação do Domínio é desenvolvido um *framework* ou arcabouço utilizando como base o diagrama de componentes do Projeto do Domínio. Na Implementação da Aplicação, é utilizado o *framework* desenvolvido para realizar o desenvolvimento da aplicação final.

Por fim, durante no Teste do Domínio são realizados testes que visam verificar se a LPS está atendendo a flexibilidade desejada entre os artefatos, como também se as comunalidades estão de fato sendo utilizadas pelos *softwares*. No Teste da Aplicação, são realizados testes para verificar se o *software* desenvolvido está atendendo as necessidades dos *stakeholders* identificados nos requisitos da aplicação.

2.4 Implementação da Variabilidade

Para realizar a flexibilidade durante a implementação do *framework* da LPS, algumas técnicas podem ser utilizadas, permitindo que o desenvolvedor da aplicação final possa escolher as características que irá utilizar na aplicação final com facilidade.

Existem várias formas de realizar a implementação da variabilidade, e nesta seção serão apresentadas sete técnicas.

A compilação condicional é uma técnica amplamente utilizada nos projetos, principalmente baseados em C, e se trata de um mecanismo comum na implementação de LPS. A compilação condicional por ser um mecanismo de pré-processadores faz o uso de anotações nos códigos através de *tags* como #ifdef [nome da variante] e #endif, para determinar que uma determinada parte do código esteja relacionada a uma *feature* (característica) da LPS. Assim, nas Linhas de Produtos de Software, ao criar um produto final, o desenvolvedor deve selecionar quais *features* desejadas utilizar na aplicação final através do uso da compilação condicional. Ao gerar a aplicação final, o código compilado retornará apenas os trechos dos códigos e arquivos relacionados às *features* selecionadas (Kästner, Giarruso, Rendel, Erdweg, Ostermann, & Berger, 2011). De tal modo, é possível representar cada variante do código como uma *feature* isolada pelas anotações da compilação condicional, que apenas irá ser adicionada ao produto final, caso a variante tenha sido selecionada. Caso a LPS tenha sido desenvolvida de modo que não haja a necessidade do programador trabalhar na aplicação final, bastará então apenas

selecionar as *features* desejadas e a compilação condicional se encarregará de gerar a aplicação com todos os requisitos selecionados.

Overloading é uma técnica baseada na reutilização dos nomes de métodos já existentes. Porém, a forma como o método é chamado na aplicação, resultará na execução de tarefas diferentes. Portanto, é possível existir dois métodos diferentes com o mesmo nome, porém que recebam parâmetros, ações e retornos diferentes. A técnica de *overloading* é bastante utilizada na programação utilizando tipos ou quantidade de parâmetros diferentes na chamada do método. Um exemplo deste uso poderia ser um método chamado "Soma" que recebe dois parâmetros para realizar a soma dos dois números informados, enquanto outro método com o mesmo nome "Soma", recebe apenas um parâmetro, retornando a soma do valor informado, com os valores informados anteriormente. O uso de *overloading* promove o reuso de código de forma que na LPS um ponto de variação poderia ser representado pelo uso de métodos com técnica de *overloading*, enquanto a escolha da variação ocorreria através dos parâmetros recebidos pelos métodos. Todavia, está técnica pode não ser muito recomendada devido ao fato dos programadores que a utilizem, podem não saber quando deve usar a forma correta da técnica (Gacek & Anastasopoules, 2001).

A técnica de delegação separa dos códigos todas as variabilidades das classes que implementam as comunalidades. Deste modo, para cada variabilidade será criada uma nova classe que irá implementar essas funcionalidades. Assim, quando um código precisar executar uma variação seja ela opcional ou alternativa, bastará a classe que recebeu a função de implementar a variação ser chamada (Júnior, 2008).

A Programação Orientada a Aspecto (POA) é uma metodologia estendida da Programação Orientada a Objetos, que permite separar funcionalidades secundárias dos módulos principais do código, transformando essas funcionalidades em aspectos. A POA é uma técnica considerada por muitos autores como efetiva para dar suporte à variabilidade e estabilizar a arquitetura de uma Linha de Produto de Software. Na LPS, a POA tem por objetivo trabalhar o encapsulamento de *features* transversais em novas unidades modulares, sendo os aspectos. A intenção é fazer com que a variação de *features* transversais se torne mais modulares e evoluídas quando comparadas com mecanismos de variabilidade convencionais, onde o código principal e os pontos de variação ficam unidos (Figueiredo et al, 2008).

A Programação Parametrizada possui o conceito básico de maximizar o reuso no programa através de armazenamento dos códigos na forma mais genérica possível (Goguen, 1984). Esse tipo de técnica é bastante utilizada em classes de acesso a banco de dados através da metodologia de *Object/Relational Mapping* (ORM), que consiste em representar uma classe como uma tabela no banco de dados. Sendo assim, para não ser preciso criar uma classe que gerencie o acesso ao banco para cada classe representada no banco de dados, é criada uma única classe genérica utilizando técnica de parametrização. A parametrização pode melhorar a usabilidade em uma linha de produtos e também facilitar a rastreabilidade para projetar decisões. No entanto, centralizar código acessado por parametrização, que são usados muitas vezes, é uma tarefa muito complexa, se não impossível em alguns casos (Gacek & Anastasopoules, 2001).

A herança é um mecanismo bastante conhecido e utilizado em linguagens com o uso de Orientação a Objeto. Na herança uma classe mãe contem as características comuns, enquanto as classes filhas ficam responsáveis apenas pelos comportamentos especializados de um determinado grupo. Com isso, o uso da herança na LPS ocorre através da implementação das comunalidades nas classes mães, enquanto as classes filhas ficariam responsáveis pelas variações da LPS (Júnior, 2008). A vantagem desta técnica está no fato de ser um mecanismo bastante conhecido pelos desenvolvedores.

Por fim, temos os Padrões de Projetos que podem ser explorados em um contexto Linha de Produtos, uma vez que muitas das técnicas dos Padrões de Projetos servem para identificar os aspectos do sistema que podem variar e fornecer soluções para o gerenciamento da variação (Gacek & Anastasopoules, 2001). Nos padrões de projetos podemos ver técnicas como *Factory, Builder, Adapter, Composite, Observer, Strategy* entre outros.

2.5 Ferramentas de suporte ao desenvolvimento de LPS

Para facilitar o trabalho de desenvolvimento de LPS, existem alguns programas que se propõem a dar suporte a algumas etapas da construção, em especial a implementação da variabilidade. O número de ferramentas que se propõem a esta atividade ainda é limitado, porém nesta seção serão apresentadas três ferramentas que se destacam na proposta.

Colored Integrated Development Environment, também conhecido apenas como CIDE, é um *plugin* do programa Eclipse, que dá suporte ao desenvolvimento de LPS, baseado na técnica de Compilação Condicional. O principal destaque na ferramenta está na facilidade de implementação da técnica de compilação condicional, por não se basear nas anotações tradicionais com o uso de *tags*. O CIDE apresenta um editor[1] especializado com a função de definir trechos de códigos na compilação condicional através do uso de cores ao invés das *tags*.

Outra ferramenta que se destaca no desenvolvimento de LPS é a FeatureIDE. FeatureIDE é um *framework* open-source baseado no Eclipse que dá suporte ao *Feature-Oriented Software Development* (FOSD). O destaque desta ferramenta está na proposta de cobrir não apenas a implementação do *framework* como todo o processo de desenvolvimento e a possibilidade em incorporar ferramentas que auxiliem a implementação de LPSs em um ambiente de desenvolvimento integrado (Thüm et al, 2014). Os recursos do programa FeatureIDE vão desde desenvolver o *feature model* até usar recursos como: Programação Orientada a Aspecto; Compilação Condicional; Programação Orientada a Features; e a Delta.

Uma terceira ferramenta que pode auxiliar o desenvolvimento de LPS é conhecida como Colligens. O Colligens é uma ferramenta baseada também no ambiente Eclipse para o desenvolvimento de LPS na linguagem C, integrado com a funcionalidade de pre-processadores. Esta ferramenta se assemelha ao CIDE, porém dando ênfase a verificação de ausências de produtos inválidos, visando à redução de falhas relacionadas aos erros de sintaxes devidos a anotações incompletas no código (Medeiros et al, 2016).

3 Trabalhos Relacionados

Nesta seção serão apresentados três LPS. A escolha da seleção destas LPS está no fato de serem focadas no desenvolvimento de softwares educativos e jogos. Como a LPS proposta neste trabalho visa jogos educativos, os trabalhos correlatos, entre outros serviram como base para o desenvolvimento e avaliação do trabalho proposto.

[1] Uma figura completa do editor do CIDE, pode ser visto no link a seguir:
http://wwwiti.cs.uni-magdeburg.de/iti_db/research/cide/

A LPS desenvolvida por Dalmon e Brandão (2013) tem como intenção solucionar os problemas encontrados no desenvolvimento de *softwares* educativos conhecidos como Módulos de Aprendizagem Interativa (iMA). Os iMA são *softwares* educacionais que tratam de recursos interativos, cujas principais características presentes são: execução em navegadores web, apresentar ferramentas de autoria, promover atribuições de interatividade e proporcionar a comunicação e integração com os Sistemas de Gestão de Aprendizagem ou Sistemas de Gerenciamento de Cursos como por exemplo o Moodle.

O processo de desenvolvimento da LPS foi iniciado usando como base o estudo de cinco *softwares* da família iMA desenvolvidos anteriormente pelo grupo de pesquisa da Universidade de São Paulo. Os *softwares* analisados foram: iGraf, um sistema para estudo de funções matemáticas e gráficos; iCG, um sistema que emula o computador; iComb, um sistema de ensino de combinatória; iGeom, um sistema de Geometria Interativa; e iVProg, um sistema de programação visual (Dalmon, Brandão, Brandão, & Isotani, 2012). O processo de avaliação da LPS de iMA foi realizado através da refatoração de *softwares* da família iMA, utilizando a LPS desenvolvida e entrevistas realizadas com os desenvolvedores.

A segunda LPS apresentada nessa seção foi desenvolvida por Nascimento, Almeida & Meira (2008) e possui o foco no desenvolvimento de jogos para *mobile*. O desenvolvimento de jogos para *mobile* apresentam complexos desafios relacionados à variedade de aparelhos, fabricantes, restrições de plataformas e diferentes configurações de *hardwares*. Devido a esses fatores, o desenvolvedor de um jogo para *mobile* pode optar por realizar o desenvolvimento através de uma LPS.

O desenvolvimento da LPS de jogos para *mobile* foi baseada em jogos existentes, observando as restrições citadas anteriormente. Na implementação da plataforma comum que foi utilizada no desenvolvimento dos jogos, usou-se técnica de compilação condicional para a seleção dos requisitos de cada jogo.

Na análise, os pesquisadores buscaram realizar um estudo de caso envolvendo o desenvolvimento de três jogos de aventura presentes no mercado de jogos para *mobile* e o desenvolvimento de um quarto jogo explorando as características comuns entre eles. Na avaliação foram definidas métricas mínimas envolvendo a complexidade do código, restrições envolvendo o domínio e rastreabilidade dos artefatos que deveriam ser atendidas pelos quatro jogos desenvolvidos.

O terceiro trabalho correlato é a LPS *Arcade Game Maker* (AGM), que foi desenvolvida pelo *Software Engineering Institute* (SEI) com o intuito de criar um exemplo de Linha de Produto para dar suporte ao processo de aprendizagem e experiência no uso de Linha de Produtos de Software (Software Engineering Institue, 2009). A Linha de Produto de Software AGM conta com a presença de três jogos *arcades*. Ao contrário da maioria dos jogos, os produtos do *Arcade Game Maker* não possuem foco na interface gráfica e nem em jogos cooperativos (multiplayer), pois o real objetivo desta LPS é ser um exemplo abrangente, de forma a ser um exemplo simples, porém completo. O material disponibilizado pela SEI é focado em duas partes distintas. A primeira relacionada aos recursos e produtos da LPS como arquitetura, requerimentos, planos de testes e os códigos dos jogos. A segunda parte é voltada para o setor pedagógico com sugestões de exercícios usando os conteúdos da Linha de Produto.

Diante dos três trabalhos, pode-se observar que o primeiro (iMA) se propõe a criar *softwares* educativos para ambientes como o Moodle; o segundo busca o desenvolvimento de jogos para aplicativos *mobile* sem a preocupação com o lado pedagógico; e o terceiro (AGM) apesar de produzir jogos, possui uma maior preocupação em ser um exemplo prático para professores ensinarem como é o processo de desenvolvimento de uma LPS.

A LPS proposta nesse trabalho levou como base o processo de desenvolvimento destas e outras LPS, buscando o diferencial de desenvolver jogos com foco nos princípios de aprendizagem construcionistas (Kafai, 2016), como pode ser visto na Seção 2.1, por possibilitarem um aprendizado mais aprofundando através da construção de artefatos.

4 Metodologia

Neste capítulo iremos apresentar uma LPS produzida com a finalidade de proporcionar um meio mais fácil para o desenvolvimento de jogos educativos construcionistas. A abordagem da LPS apresentada nesta seção será sobre o desenvolvimento da Engenharia do Domínio.

Os problemas que são trabalhados numa LPS são identificados como os requisitos que os *softwares* necessitam. Esses requisitos podem ser tratados como *features* ou as comunalidades e variabilidades da LPS.

Durante a primeira etapa de desenvolvimento da LPS, na Engenharia de Requisitos do Domínio, são identificadas as *features* que serão trabalhadas. A identificação costuma ser realizada através de entrevistas com *stakeholders* para colher as suas reais necessidades. Entretanto, devido a LPS não estar sendo desenvolvida para um cliente específico, mas sim para um público geral de desenvolvedores de jogos educativos, a seleção das *features* seguiram o mesmo padrão de seleção de outras LPS, como é o caso da iMA, selecionando *softwares* do mesmo ramo para coletar seus requisitos (Dalmon & Brandão, 2013).

Nesta etapa foram selecionados 10 jogos educativos que apresentavam como foco a construção de artefatos de forma discreta ou indiscreta. Os jogos selecionados foram: Sim Investigador (Domínguez, Filho, Paraguaçu, & Oliveira, 2015), Linguagem Logo (Fein et al, 2013), *RoboCode* (Robocode, 2017), *SimCity* (MAXIS, 2016), *Civilization* (Firaxis Game, 2016), *Business Tycoon Online* (Dovogames, 2016), *Software Development Manager* (Kohwalter et al, 2014), Gene2 (Morelato et al, 2008), *Code Hunter* (Microsoft Research, 2015) e Geogebra (Barros & Stivam, 2012).

Tais jogos foram selecionados devido a já possuírem contribuições acadêmicas no ensino ou por serem jogos comerciais com grande público e com contribuição ao ensino. O desenvolvimento de um artefato neste tipo de jogo dá a liberdade ao jogador de realizar as ações como bem entenderem, de forma que não exista uma única solução real. As possíveis combinações levaram a resultados diferentes, que deverão ser interpretados pelo jogador.

A seleção dos jogos usados para a construção da LPS tiveram como base os jogos escolhidos através de suas contribuições em artigos ou devido a sua grande utilização comercialmente e ainda proporcionarem um processo de aprendizagem pelo construcionismo.

O processo de seleção dos requisitos de cada jogo foi dividido em duas etapas, na qual a primeira etapa buscava os requisitos necessários para o funcionamento do jogo, enquanto a segunda etapa buscava os requisitos presentes no artefato sendo construído pelo jogador.

Na seleção dos requisitos dos jogos foram identificados 21 requisitos como sistema de *login*, multi-idiomas, controlador de pontuação, menus de navegação e gerador de mapas ou desenhos 2D. Nos requisitos do artefato, foram identificados 16 requisitos, entre eles o nome e descrição do artefato que está sendo construído e o uso de subcomponentes para formar o artefato finalizado.

Após a identificação dos requisitos, seguiu-se o modelo proposto por Pohl et al (2005), utilizando a Matriz de Requisitos. A Matriz de Requisitos tem como finalidade identificar quais requisitos são comunalidades, variabilidades ou requisitos exclusivos do *software*. Nesta matriz,

são listados os requisitos na vertical e os *softwares* na horizontal e entre eles é assinado um "X" para indicar se um jogo possui ou não um determinado requisito.

Os requisitos exclusivos são aqueles presentes apenas em um único *software* ou em um grupo pequeno, portanto são descartados dos requisitos trabalhados na LPS. As comunalidades são os requisitos presentes em todos ou na maioria dos *softwares* do domínio, ou que podem tornar características obrigatórias dentro de certo período. As comunalidades também não podem entrar em conflitos com outros requisitos, uma vez que estarão presentes em todos os *softwares*. As variabilidades são os requisitos que apresentam grande relevância para um determinado grupo de *softwares*, mas não para outros, ou que são relevantes em vários *softwares*, mas que entram em conflito com outros requisitos também presentes.

Na LPS desenvolvida foram identificados como requisitos obrigatórios (comunalidades) dos jogos conteúdos como: uso de personagens; menu de navegação; e definição de objetivo para o jogador. No artefato entraram como requisitos obrigatórios um identificador do artefato, nome, texto e uso de componentes. Como variabilidade do jogo, foram identificados componentes que entravam em conflitos como: o sistema *multiplayer* ou *single player*, ou alternativas como tipo de *login* por rede social ou por *email*; tipos de comunicação por e-*mail* ou *chats;* e tipo de interface gráfica. No artefato dos jogos foi possível identificar a variabilidade em requisitos opcionais como o tipo ou classificação do artefato e seu status.

Após filtrar e classificar os requisitos, estes foram organizados através do *Feature Model*, que permite através de uma representação visual mostrar as comunalidades, requisitos mandatórios, e a variabilidades em forma de escolha opcional ou alternativa (Figura 2) (Feigenspan, Kästner, Frisch, Dachsel, & Apel, 2010).

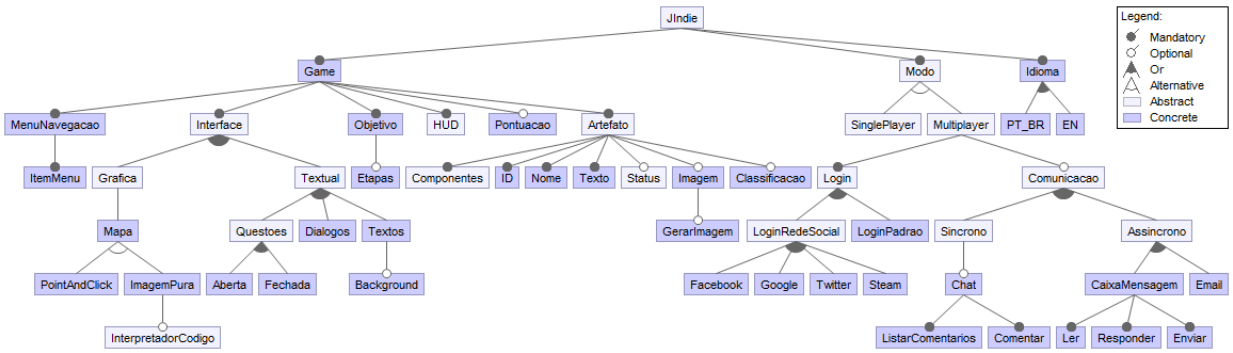

Figura 2 - O *feature model* da LPS proposta

Para melhor representar e compreender, cada ponto de variabilidade foi trabalhado em algum diagrama como é o caso do Diagrama de Caso de Uso no exemplo da Figura 3. Nesta figura é possível que o jogador possa interagir com o jogo simplesmente clicando sobre o objeto, como ocorre em jogos como *SimCity, Civilization* ou *Business Tycoon Online*. A segunda forma de interação com um mapa em um jogo é semelhante ao que ocorre nos jogos *Logo* e *RoboCode*, onde o usuário deve inserir um comando como "paraFrente()" e o personagem no mapa executa a ação.

A etapa de Projeto do Domínio tem como finalidade solucionar os problemas identificados com as features da Engenharia do Requisito do Domínio. Para solucionar os problemas é desenvolvida uma arquitetura. Pressman (2010) define uma arquitetura como um conjunto de componentes de uma construção de forma a coexistir como um todo. Na concepção de software ou sistemas, a arquitetura se trata da estrutura que abrange os componentes de softwares, propriedades externas visíveis e a relação entre eles.

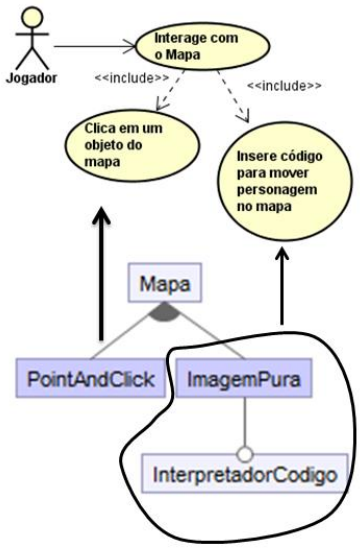

Figura 3 - Documentando a variabilidade através de Diagramas

Diante desta concepção, foram criadas quatro camadas para representar a arquitetura da LPS:

• **Funções do *Framework*** – Na camada de Funções do *Framework* ficam todas as funcionalidades responsáveis pelas configurações e regras do funcionamento de como a aplicação deve fluir.

• **Funções do Jogo** – Na camada das Funções do Jogo ficam todas as funcionalidades relacionadas ao funcionamento do jogo. Enquanto na camada do *Framework* ficavam os *scripts* relacionados à execução de códigos, regras de navegação e comportamento da aplicação, na camada de Funções do Jogo ficarão os *scripts* que estão relacionados com o domínio do jogo.

• **Funções Globais** – Na camada de Funções Globais ficarão os *scripts* mais simples que não fazem parte do funcionamento geral da aplicação e nem do domínio do jogo, mas que ainda apresentam papel importante como dar suporte às duas camadas citadas anteriormente. Nessa camada será possível encontrar *utils, helpers* e bibliotecas.

• **Controles Básicos** – Na camada de Controles Básicos estarão presentes as funcionalidades ofertadas pela própria linguagem de programação, independente da aplicação.

Além das camadas apresentadas acima, foram definidos pacotes presentes nestas camadas, relacionando os pontos de variação com os pacotes.

Para o desenvolvimento da arquitetura que serviria como base para a implementação, foi utilizada o Diagrama de Componentes, por ser o diagrama mais adequado para desenvolvimento de *software* visando a reusabilidade de código. O uso de componentes e artefatos permite a construção de uma arquitetura flexível e ideal para a construção de LPS, devido ao fato que a remoção ou a troca de um componente não causará conflito nos demais componentes. O uso de interfaces também possibilita que novos componentes possam ser adicionados com facilidade ao projeto.

No exemplo da Figura 4, podemos ver através do Diagrama de Componentes, como o artefato desenvolvido pelos jogadores poderia facilmente se comunicar com diferentes componentes de acordo com os jogos criados na LPS.

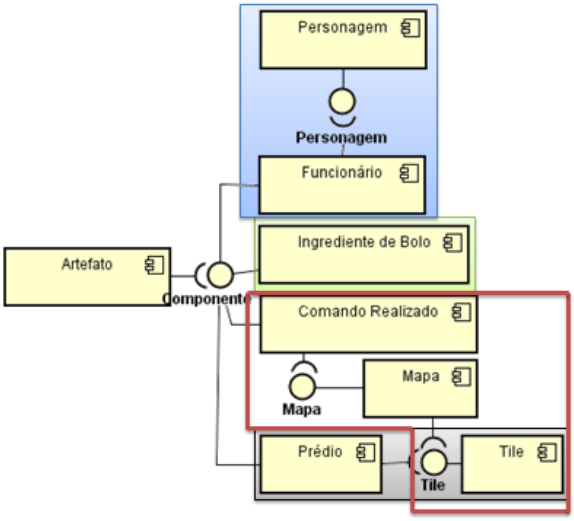

Figura 4 - Diagrama de componente do Artefato

Neste exemplo temos quatro jogos usando a mesma arquitetura do Artefato para diferentes componentes. Como citado anteriormente, o desenvolvimento de um artefato nestes jogos ocorrerá adicionando componentes. No bloco azul, o artefato seria uma empresa, enquanto os componentes adicionados representariam os funcionários. No bloco verde, o artefato assumiria o papel de um bolo, enquanto os componentes seriam os ingredientes usados na preparação do bolo. No bloco vermelho, o artefato seriam desenhos construídos com a adição de comandos, semelhando ao jogo Logo. Por fim, no último exemplo do bloco cinza, temos a construção de cidades como artefato, e os componentes seriam os prédios adicionados a essa cidade.

A terceira etapa do desenvolvimento é baseada no desenvolvimento do *framework* utilizando a arquitetura planejada. No desenvolvimento da LPS proposta, foram disponibilizados dois *frameworks* baseados na arquitetura do Projeto do Domínio.

O primeiro *framework* desenvolvido utilizou a linguagem de programação PHP. A escolha da linguagem está relacionada à praticidade de implementar um conteúdo *Multiplayer* e devido a alguns jogos rodarem diretamente pelo *browser*. A escolha desta linguagem também foi influenciada para a produção de um dos jogos do Estudo de Caso, o Sim Investigador. O Sim Investigador, além de ser um dos jogos propostos para o desenvolvimento, também contou com a participação do desenvolvedor do jogo original na avaliação, possibilitando que na análise fossem observados opiniões do desenvolvedor e comparações com o código original.

Nesta primeira versão foram utilizadas técnicas de implementação da variabilidade como agregação e delegação, parametrização, herança e técnicas de padrões de projetos como o *Factory*. A técnica *Factory* permite ao programa poder instanciar uma classe através de uma interface em particular ou protocolo sem necessariamente saber com precisão a classe que está sendo obtida (Ellis, Stylos, & Myers, 2007).

A LPS desenvolvida conta com recursos pré-implementados para conexão com redes sociais, tendo como variabilidade o *login* através do Google, Facebook, Twitter e Steam. No Diagrama de Classe (Figura 5) é possível ver como uma classe ControllerLogin, que herda Controller, pode solicitar um *login* por uma rede social. A técnica utilizada para implementar essa variabilidade foi baseada no factory.

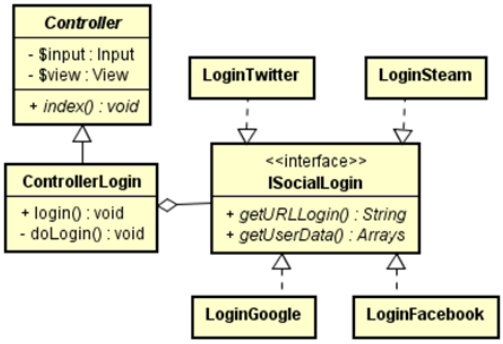

Figura 5 - Diagrama de Classe de login por rede social

O segundo *framework* desenvolvido além de conter as técnicas citadas anteriormente, também utiliza a técnica de compilação condicional utilizando a ferramenta de suporte a desenvolvimento de LPS, CIDE. Esta é uma ferramenta baseada em Eclipse para desenvolvimento de LPS utilizando a linguagem Java. Por este motivo, a segunda implementação foi realizada utilizando a linguagem Java e possuía como finalidade avaliar as vantagens do uso da técnica de compilação condicional e uma ferramenta de suporte no desenvolvimento.

Na quarta etapa, no Teste do Domínio, foram realizados testes com a finalidade de verificar se a LPS possui condições de produzir diferentes tipos de jogos educativos e se não haveria problemas entre as variabilidades e comunalidades. Para avaliar as comunalidades e variabilidades, foi utilizada a *Commonality and Reuse Strategy* durante o desenvolvimento da LPS. Essa estratégia consiste em desenvolver um pequeno protótipo funcional para verificar todas as comunalidades presentes no *software*. Para avaliar todas as variabilidades, no mesmo protótipo desenvolvido são trocados trechos de códigos onde ocorrem os pontos de variação para testar se está acontecendo algum conflito ou falha na escolha da variação (Pohl et al, 2005).

Após a conclusão da LPS, foi utilizada a *Sample Application Strategy*, que é uma estratégia voltada ao desenvolvimento de algumas aplicações de forma a verificar se a LPS é capaz de criar softwares completos atendendo a flexibilidade.

Na próxima subseção serão vistos os jogos desenvolvidos durante o teste realizado na LPS.

4.1 Avaliando a LPS desenvolvida

Para avaliar a LPS proposta, foi trabalhado um Estudo de Caso. Um Estudo de Caso seguindo o modelo de Runeson e Höst (Runeson & Höst, 2009) passa pelas etapas de planejamento, realização e coleta dos resultados.

Para a definição dos dados que serão coletados no Estudo de Caso, optou-se pelo uso do paradigma *Goal Question Metric* (GQM). Nesse método, primeiro se define os objetivos que se pretende alcançar, seguidos das perguntas que surgem dos objetivos, e por fim as métricas que servem para responder as perguntas (Runeson & Höst, 2009).

Diante deste método, esse Estudo de Caso teve como Meta:

1. Verificar se a LPS desenvolvida é capaz de construir diferentes jogos mantendo a qualidade e reduzindo custos, tempo, complexidade e esforço do desenvolvedor;
2. Avaliar a influência do uso da compilação condicional nos jogos construídos com a LPS;
3. Avaliar a influência da ferramenta CIDE.

Após definidas as metas, foi possível observar as seguintes questões:

1. Qual o tempo gasto na produção dos jogos?
2. Quantas Linhas de Código o desenvolvedor do jogo precisou criar?
3. Qual a Complexidade Ciclomática dos códigos criados?
4. Um jogo construído com a LPS apresentou desempenho melhor do que a sua versão original?
5. A construção de um jogo com compilação condicional usando o CIDE apresentou facilidade no desenvolvimento comparado ao jogo construído sem a compilação condicional?

Por último foram definidas as métricas:

1. Quantidade de tempo gasto na produção dos jogos;
2. Quantidade de Linhas de Código que o desenvolvedor criou;
3. Complexidade Ciclomática dos códigos criados pelo desenvolvedor;
4. Nível de satisfação do desenvolvedor dos jogos com a LPS;
5. Comparação das métricas anteriores em relação aos jogos com o uso e sem o uso da compilação condicional.

Para realizar esse Estudo de Caso, foram desenvolvidos quatro jogos, sendo três deles na primeira versão do *framework* sem a compilação condicional, e o quarto jogo na versão do *framework* com a compilação condicional e o CIDE.

Os jogos desenvolvidos foram baseados em três jogos construcionistas que já foram utilizados em sala de aula no processo de ensino. Os jogos selecionados foram Sim Investigador, Linguagem Logo e RoboCode. Os três jogos selecionados foram desenvolvidos utilizando o *framework* em PHP, sendo que o *RoboCode* foi desenvolvido novamente utilizando a segunda versão do *framework* da LPS, envolvendo a compilação condicional e a ferramenta CIDE para novas análises. O desenvolvimento do jogo Sim Investigador na primeira versão em PHP, possibilitou realizar uma comparação entre a versão desenvolvida com a LPS e a versão original também produzido nesta linguagem de programação, podendo ser avaliado a complexidade do código, esforço de desenvolvimento e opinião do desenvolvedor. Além destas análises, os jogos produzidos passaram por métricas baseadas em trabalhos correlatos.

Os três jogos selecionados também apresentam artefatos a serem construídos, que são compostos por diferentes componentes (Figura 6).

As métricas adotadas para a avaliação dos jogos desenvolvidos através da LPS foram: Número de linhas de código implementadas pelo desenvolvedor da aplicação final; Duração do tempo de desenvolvimento de cada jogo; Dificuldades e facilidades encontradas durante a produção; e a Complexidade Ciclomática.

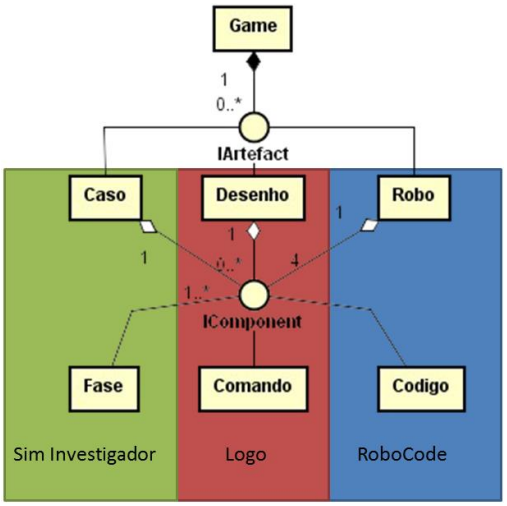

Figura 6 - Artefatos e componentes construídos nos jogos

Complexidade Ciclomática (CC), desenvolvida por McCabe, é uma métrica para avaliação da complexidade de *softwares* baseado nas linhas de códigos (Watson, McCabe, & Wallace, 1996). A CC avalia a complexidade do código baseada na quantidade de opções da estrutura lógica de decisão e repetição em um determinado módulo do *software*. Para realizar esta análise, é criado um diagrama do fluxo do código, de modo que cada linha de código seja representada através de um nó no fluxograma, enquanto cada possível direção de uma linha para outra é representada por uma aresta.

A representação da CC pode ser expressa através da formula abaixo (Equação 1):

$$CC = A- N+2 \qquad (1)$$

De modo que "A" representa o número de arestas, ou seja, linhas de código, e "N" o número de nós, os possíveis caminhos. De acordo com McCabe (1976), quanto maior a CC, maior será a chance de o sistema apresentar erros durante as manutenções no *software*. Embora muitos programas possuam módulos com complexidade entre 40 e 50 pontos, um estudo realizado demonstra que CC acima de 20 apresenta um grande risco e deve ser diminuída (Clark, Salesky, Urmson, & Brenneman, 2008).

Além das métricas citadas anteriormente, a análise realizada buscou comparar a versão do jogo *RoboCode* construído através da LPS em PHP e a versão do jogo construída através da LPS em Java com a compilação condicional e a ferramenta CIDE. Essa comparação tinha como finalidade avaliar as facilidades e melhorias que poderiam ser obtidas no desenvolvimento do jogo e da LPS através do uso da ferramenta CIDE na implementação da variabilidade com a compilação condicional.

O primeiro jogo desenvolvido foi baseado no jogo Sim Investigador (Figura 7), que trata de um jogo no qual o jogador assumirá o papel de um detetive com a finalidade de solucionar casos e prender ladrões, utilizando os conhecimentos aprendidos através de disciplinas como matemática ou português, respondendo perguntas através de um questionário que o aproximarão da conclusão do caso. O jogo também permite aos jogadores criarem suas próprias histórias narrativas com opções de adicionar questões de múltipla-escolha (Dominguez et al, 2015). Este jogo foi escolhido devido ao fato de se ter acesso ao código e ao programador do jogo original.

Figura 7 - Tela do jogo Sim Investigador desenvolvido com a LPS

Esta avaliação contou com a participação do desenvolvedor do código original, que desenvolveu uma nova versão do jogo com a LPS, possibilitando que fosse possível realizar comparação de tempo de desenvolvimento, número de linhas de códigos e arquivos implementados, complexidade do código e tempo de resposta das páginas.

O próximo jogo desenvolvido durante a avaliação da LPS foi baseado no famoso jogo utilizado em colégios na disciplina de matemática para o ensino de formas geométricas, Logo (Figura 8). Dentro do universo do jogo Logo, o estudante terá o controle de uma tartaruga em um ambiente de cor sólida. A cada movimento realizado pela tartaruga, será deixando uma linha indicando o caminho realizado. Para realizar os movimentos o jogador irá inserir linhas de comandos como “parafrente” ou “paratras(30)”, que indicará o movimento que será realizado e quantos passos serão executados. O conceito do jogo Logo é que o aluno possa criar formas geométricas através do rastro deixado no caminho da tartaruga, levando o estudante a aprender noções de distância, direção e dos formatos geométricos (Fein et al, 2013). O jogo desenvolvido com a LPS, além de contar com os comandos básicos, também permite a criação de comandos personalizados pelo jogador.

Figura 8 - Tela do jogo Logo desenvolvido com a LPS

O terceiro e quarto jogo desenvolvido com o uso da LPS foram baseados no jogo de programação *RoboCode* (Figura 9). O jogo *RoboCode* é um jogo voltado ao aprendizado de programação, permitindo os jogadores criarem seus próprios robôs através das linguagens Java ou C#, para batalharem entre si (Robocode, 2017). Embora o jogo original permita batalha entre jogadores diferentes, é necessário que o código desenvolvido por ambos estejam na mesma máquina, impossibilitando batalhas *multiplayers* em ambientes distintos. Os jogos

desenvolvidos com a LPS para jogos educativos proposta nesse trabalho possibilita essa batalha de forma remota, sem a necessidade dos jogadores estarem no mesmo recinto. Para facilitar esta batalha de robôs entre amigos, os jogos contam com um *login* realizado através da rede social Facebook usando o código fornecido pela LPS, de modo que se dois amigos estiverem conectados no jogo e na rede social, poderão realizar as batalhas de suas próprias casas.

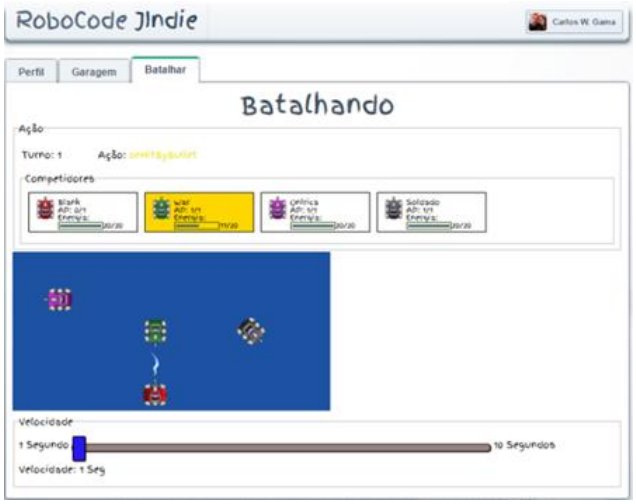

Figura 9 - Tela do jogo RoboCode desenvolvido com a LPS

5 Resultados e Discussão

Após planejar e realizar o Estudo de Caso, iniciou o processo de avaliação dos jogos e da LPS usando as métricas citadas anteriormente.

O desenvolvimento do primeiro jogo, Sim Investigador, contou com a comparação do desenvolvimento do código original e a versão da LPS. Durante a implementação do jogo original para a versão com a LPS, não existiram impedimentos ou dificuldades durante a realização, uma vez que o padrão adotado na LPS segue o modelo *Model-View-Control* (MVC) utilizado por muitos *frameworks*. Na integração do domínio do jogo a arquitetura desenvolvida na LPS também não apresentou dificuldades devido ao uso dos componentes e interfaces. No jogo, o artefato construído ficou sendo um caso ou narração, enquanto os componentes ficaram sendo as fases ou partes da história.

As vantagens no desenvolvimento puderam ser vistas no fato da LPS entregar vários recursos secundários como acesso a banco de dados, filtros de segurança e outros recursos que permitiram o foco no desenvolvimento do domínio do jogo. A LPS também proporcionou um fácil acesso e manuseio no artefato e sistema de pontuação utilizada no jogo.

A versão original do jogo Sim Investigador contem 6.937 linhas de código que foram implementadas pelo desenvolvedor, enquanto a versão com a LPS apresenta ao todo 20.210 linhas. Todavia, apenas 1.849 deste total foram de fato implementadas pelo desenvolvedor, repre-sentando 73,3% menos linhas implementadas pelo desenvolvedor comparado com a produção do jogo original. O grande valor de número de linhas a mais no código do jogo com a LPS está relacionado ao fato de que a arquitetura desenvolvida na LPS tende a ser mais genérica e flexível possível para atender diversos tipos de jogos educativos. Além dos recursos de outros pontos de variação disponibilizados e não utilizados no jogo, outro fator que resulta no aumento do número de linhas está nos recursos secundários de apoio que não haviam na versão original do jogo como uso de *logs*, filtros de segurança e *parse* de URL amigáveis. Neste ponto, o jogo

original também ganhou em relação ao uso da LPS em número de arquivos, por motivos semelhantes ao número de linhas.

Na análise realizada quanto a Complexidade Ciclomática (CC) comparando os dois códigos, foram escolhidas as principais funções do jogo. Como resultado, a versão utilizando a LPS apresentou melhorias significativas tornando o código mais limpo e menos complexo. Um dos principais destaques ficou no cadastro de usuário, que na versão original apresenta CC de 14 pontos, enquanto a versão utilizando a LPS conseguiu reduzir a complexidade para apenas 1 ponto, devido a seus recursos vinculados a validação no sistema de *login* do jogo. Na Tabela 1 é possível observar as 11 principais funcionalidades do jogo que foram usadas para comparar a CC do jogo original com o jogo produzido através da LPS.

Tabela 1 - Comparação da Complexidade Ciclomática dos jogos Sim Investigador

Código	Original	Com a LPS
Cadastro de jogador	14	1
Jogar um caso	3	2
Responder uma pergunta do caso	7	5
Criar um caso	11	2
Adicionar uma pergunta ao caso em desenvolvimento	14	14
Investigar jogadores	8	1
Listar Rank com pontuação dos jogadores	9	4
Enviar mensagem	7	3
Ler mensagem	3	4
Aprovar um caso	3	3
Reprovar um caso	5	4

Durante o desenvolvimento algumas falhas de implementação puderam ser encontradas, porém todas foram corrigidas evitando que os novos jogos cometessem os mesmos erros, garantindo uma qualidade maior para próximos jogos.

No geral, a implementação do jogo Sim Investigador com a LPS gerou uma redução de 45% do tempo de desenvolvimento (55 horas) comparado com a versão original e disponibilizou uma lista de requisitos simplificando e direcionando o desenvolvimento.

A produção do segundo jogo foi baseada na Linguagem Logo. Seu artefato construído foram os desenhos formados pela tartaruga, enquanto os comandos correspondiam aos componentes do artefato.

Para realizar a implementação do jogo Logo, foram implementadas pelo desenvolvedor apenas 409 linhas de código, sendo as demais funcionalidades trabalhadas nos códigos disponibilizados pela própria LPS. A parte mais avançada no desenvolvimento deste jogo está em gerar o desenho e interpretar os comandos inseridos pelo usuário. Todavia, a LPS já disponibiliza dois componentes que facilitam esse processo: gerador de imagem e um interpretador de comandos. Deste modo, o desenvolvedor apenas precisaria se preocupar em definir as regras e comandos disponíveis para o jogador usar no jogo.

A CC dos comandos desenvolvido pelo desenvolvedor do jogo estiveram todas abaixo de 5 pontos, demonstrando um código limpo e de fácil manutenção. Além da baixa complexidade de desenvolvido, foi possível finalizar a produção do jogo em apenas 16h.

Entretanto, ao contrário da versão do Sim Investigador, no jogo Logo houve alguns problemas no uso de um dos recursos do jogo. O gerador de imagem utilizado na

implementação do desenho formado pela tartaruga, usa como base blocos de desenhos em uma matriz com posições x e y. Desta forma, a rotação da tartaruga é permitida apenas em 8 direções, que são os blocos vizinhos ao atual. Para evitar que desenhos fiquem muito quadrados, há a necessidade de reduzir o tamanho padrão das imagens dos blocos de 32 pixels para 8 pixels, possibilitando desenhos de figuras geométricas mais arredondadas.

O desenvolvimento do terceiro jogo, *RoboCode*, com o uso da primeira versão do *framework* em PHP, resultou em uma aplicação com 19.641 linhas, sendo que apenas 1.194 foram desenvolvidas pelo programador. Ao comparar com o mesmo jogo desenvolvido com a versão da compilação condicional, resultou em uma aplicação com 3.803 linhas, um valor bem abaixo, uma vez que os recursos não usados na aplicação não estão presentes na aplicação final e ao fato que na versão em Java a contagem de linhas de algumas bibliotecas por estarem em jar, não entraram na contabilização. Todavia, embora o número de linhas seja menor, o trabalho de desenvolvimento do programador foi de 1.457 linhas, sendo um pouco maior por diferenças de linguagem de programação.

Diante dos resultados obtidos em relação ao número de linhas de código foi possível observar que a compilação condicional pode reduzir consideravelmente o número de linhas da aplicação final. Porém, essa redução apenas ocorre no código fornecido pela LPS e não no código desenvolvido pelo programador, de modo que o esforço realizado pelo desenvolvedor não mudará muito ao usar a técnica de compilação condicional. Entretanto, ao analisar a redução da CC do código da LPS é possível ver uma redução de módulos de CC de 19 pontos para 5 pontos, como é o caso do recurso de validação de formulários.

Também foi possível observar que usar a Compilação Condicional no desenvolvimento da LPS tornou o desenvolvimento da aplicação final mais fácil. Entretanto, o processo de manutenção da LPS se tornou mais complicado, uma vez que no mesmo script existirão diversas linhas de códigos para situações e produtos finais diferentes, podendo tornar mais confuso o código da LPS para manutenção.

Ao avaliar todos os jogos desenvolvidos com a LPS, foi possível observar que nenhum ultrapassou o limite de CC sugerido na literatura. Na Tabela 2 é possível ver um maior número de ocorrências de CC com valores baixos.

Tabela 2 - Complexidade Ciclomática dos jogos desenvolvidos com a LPS

CC	Ocorrência
1-5	19
6-10	4
11-15	2
16-20	1
>20	0
Média 4,96	

Segundo estudo realizado pelo Instituto de Engenharia de Software (Clark et al, 2008), um problema pode ser classificado como de alto ou baixo risco, através da sua Complexidade Ciclomática conforme a Tabela 3. Os módulos implementados pelo desenvolvedor utilizando a LPS, conseguiram apresentar um baixo risco.

Tabela 3 - Classificação da Complexidade Ciclomática

CC	Avaliação do Risco
1-10	Um programa simples, sem muito risco.
11-20	Mais complexo, risco moderado.
21-50	Programa complexo, alto risco.
>50	Programa intestável (Risco muito elevado)

Em relação à avaliação do uso da ferramenta CIDE no processo de desenvolvimento da LPS, foi possível observar pontos positivos quanto ao uso desta ferramenta. O desenvolvimento com o CIDE permitiu o desenvolvimento do *feature model* em um único ambiente de desenvolvimento, como também possibilitou o uso da técnica de compilação condicional sem o programador, da LPS proposta, ter uma vasta experiência no uso desta técnica. O fácil uso da compilação condicional no CIDE está relacionado ao fato do desenvolvedor não necessitar ter o conhecimento das diferentes *tags* utilizadas por essa técnica, apresentando um conceito novo baseado na coloração de trechos de códigos e arquivos, que se deseje utilizar na técnica.

Entretanto o uso de cores para a compilação condicional pela ferramenta CIDE, pode apresentar certas dificuldades quando há muitos pontos de variação na LPS. Ao ter muitos pontos de variações, haverá o uso de cores com pequenas variações de tonalidade, como um verde mais claro e outro mais escuro, podendo com isso, confundir o desenvolvedor da LPS.

A ferramenta CIDE também apresentou alguns problemas no momento de gerar a arquitetura que seria usada na aplicação final. Basicamente, houve dois problemas que necessitaram ser corrigidos de forma manual ao gerar a arquitetura selecionando os requisitos que seriam usados no jogo. O primeiro problema identificado consistiu na má remoção de alguns trechos de código, onde o código era removido, porém mantendo as chaves das estruturas de decisão. Esse problema ocorria ao solicitar que a arquitetura da aplicação final fosse utilizada sem usar alguns *features*. Em alguns trechos do código, embora minoria, a ferramenta CIDE não removeu corretamente o código, mantendo ainda as chaves.

O segundo problema encontrado ao gerar a arquitetura da aplicação final foi observado nos arquivos de configurações XML. Ao gerar a arquitetura, o CIDE adicionava um espaçamento extra dentro das tags dos arquivos XML gerando erros de má formatação nas *tags*.

Numa avaliação geral, os erros apresentados no CIDE podem ser corrigidos de forma manual e poderão ser corrigidos em versões futuras da ferramenta, de modo que os benefícios superam os problemas encontrados. Entretanto, o uso do CIDE comercialmente pode apresentar riscos relacionados à alta dependência da ferramenta e suas atualizações, uma vez que o uso do recurso de compilação condicional através da coloração de códigos, não apresenta uma interoperabilidade entre outras ferramentas. Devido a isso, a ferramenta é mais adequada para o uso acadêmico no ensino e pesquisa, além da dificuldade de trabalhar com uma grande quantidade de pontos de variabilidade identificados por cores com pequenas variações de tonalidade, o que é uma realidade em projetos reais de LPS.

5.1 Ameaças aos Resultados

O trabalho desenvolvido apresenta como ameaças aos resultados: (i) O fato do código do Sim Investigador ser reescrito pode já contribuir com o aprimoramento da segunda versão do jogo; (ii) A experiência do programador no desenvolvimento da primeira versão do jogo Sim Investigado ter sido mudada desde a produção da segunda versão; (iii) Os jogos produzidos nesse trabalho serem de tamanhos pequenos e médios; (iv) Jogos desenvolvidos para plataformas diferentes; (v) A avaliação do tempo pode não ser precisa; (vi) A redução da Complexidade Ciclomática pode ser coincidência.

Dentro do intervalo do desenvolvimento da versão original do jogo Sim Investigador e a versão utilizando a LPS foram realizados novos estudos e análises pelo desenvolvedor da aplicação. Por esse motivo, acredita-se que apenas o fato de reescrever o código já poderia resultar em melhorias na aplicação. Porém, independente das melhorias que poderiam ser realizadas pelo programador sem o uso da LPS, ainda levaria um tempo maior de investimento ao precisar criar funções e recursos disponibilizados de forma reusável para a aplicação.

Em relação à experiência do programador na produção do jogo Sim Investigador no intervalo entre as duas versões, que foi aprimorada com novos estudos, poderiam levar a redução da complexidade do código original. Todavia, ao observar os resultados obtidos nos demais jogos desenvolvidos em relação à Complexidade Ciclomática, ainda é possível observar que o uso da LPS mantém um valor baixo e aceitável na sua complexidade, facilitando a manutenção e tornando o código limpo.

A ameaça relacionada ao tamanho dos jogos está no fato de que nenhum jogo de porte grande foi desenvolvido neste trabalho, envolvendo várias pessoas na produção, para verificar os resultados obtidos em uma grande escala de tempo. O fato está relacionado à condição da produção de um jogo de porte grande poder resultar em um desenvolvimento de meses ou anos, se tornando inviável pela limitação do tempo da pesquisa. Todavia, acredita-se com base na literatura, que a LPS ainda seja capaz de obter bons resultados com redução de tempo e custos.

Alguns dos jogos usados na Engenharia de Requisitos do Domínio estão presentes em plataformas como desktop e mobile, enquanto a LPS desenvolvida visa a plataforma *web*. Apesar desta limitação, foi possível observar a migração dos jogos Logo e RoboCode da plataforma *desktop* para a *web*. Para o desenvolvimento de jogos em *smartphone*, o uso de um ambiente *web* também pode ser utilizado com técnicas de responsividade, tornando as telas dos jogos responsivas para diferentes tamanhos.

A análise do tempo investido na construção de um *software* costuma ser uma atividade muito complexa, uma vez que pequenas distrações como receber e-*mails* ou até mesmo beber água podem influenciar na contagem final.

Quando a avaliação da redução da Complexidade Ciclomática do jogo Sim Investigador, também pode representar uma ameaça uma vez que a redução destes valores pode ser uma coincidência para essa situação, sendo recomendado realizar novos testes com outros jogos para uma confirmação mais real.

6 Conclusão

Os jogos educativos buscam contribuir com a solução do problema de falta de motivação aos estudos por parte dos alunos, entretanto a produção de um jogo é cercada por diversas dificuldades. Este artigo propõe uma Linha de Produto de Software para jogos educativos. O desenvolvimento tomou como base o modelo proposto por Pohl et al (2005) e sua avaliação ocorreu através da produção de quatro jogos, buscando avaliar a viabilidade e qualidade na produção de jogos através da LPS proposta.

Este trabalho apresenta uma nova ferramenta que pode servir de auxílio no processo de desenvolvimento de jogos educativos visando diminuir as dificuldades nas definições do projeto, implementações e reduções de falhas, além do desenvolvimento de novas versões de três jogos. Também é apresentado nesse trabalho, um modelo de LPS para jogos educativos.

Para trabalhos futuros pretende-se realizar melhorias na LPS como desenvolvimento de um jogo construcionista para o ensino de lógica de programação para aplicá-lo em sala de aula e observar suas contribuições; criar um ambiente *web* para disponibilizar o código e a documentação da LPS para comunidade; realizar análise no desenvolvimento de jogos de grande porte; e analisar o uso da LPS por diferentes grupos de desenvolvimento. Também se pretende em trabalhos futuros, poder comparar o uso dessa LPS em seu estado final com outras metodologias em jogos construcionistas, para avaliar se está técnica realmente se sobrepõe as demais metodologias.

Referência

Ariffin, M. M., Oxley, A., & Sulaiman, S. (2014). Students' inclinations towards games and perceptions of Game-Based Learning (GBL). *2014 International Conference on Computer and Information Sciences, ICCOINS 2014 - A Conference of World Engineering, Science and Technology Congress, ESTCON 2014 - Proceedings*. doi: 10.1109/ICCOINS.2014.6868839 [GS Search]

Bae, J., & Kim, A. (2014). Design and Development of Unity3D Game Engine-Based Smart SNG (Social Network Game). *International Journal of Multimedia and Ubiquitous Engineering*, *9*(8), 261–266. doi: 10.14257/ijmue.2014.9.8.23 [GS Search]

Barros, A.P.R.M., Stivam, E. P. (2012). O software GeoGebra na Concepção de Micromundo. *Revista Do Instituto GeoGebra Internacional de São Paulo*, *1*(1), 184–194. [GS Search]

Cadavid, N. A., Ibarra, G. D., & Salcedo, L. S. (2014). Using 3-D Video Game Technology in Channel Modeling. *IEEE Access*, *2*, 1652–1659. doi: 10.1109/ACCESS.2014.2370758 [GS Search]

Carmen Sandiego. (2015). Where in the world is Carmen Sandiego. Retrieved from http://www.hmhco.com/parents-and-kids/the-learning-company/carmen-sandiego/history.

Catete, V., Peddycord, B., & Barnes, T. (2015). Augmenting introductory Computer Science Classes with GameMaker and Mobile Apps. *SIGCSE '15 Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 709–709. doi: 10.1145/2676723.2678307 [GS Search]

Chan, J. C. P., Leung, H., Tang, J. K. T., & Komura, T. (2011). A Virtual Reality Dance Training System Using Motion Capture Technology. *IEEE Transactions on Learning Technologies*, *4*(2), 187–195. doi: 10.1109/TLT.2010.27 [GS Search]

Cho, K. (2009). Three Reasons Why Indie Devs Don't Finish Their Games. Bunkyo Gakuin University.

Clark, M. N., Salesky, B., Urmson, C., & Brenneman, D. (2008). Measuring Software Complexity to Target Risky Modules in Autonomous Vehicle Systems. *AUVSI North America Conference*, 1–11. [GS Search]

Clua, E. W. G., & Bittencourt, J. R. (2003). Uma Nova Concepção para a Criação de Jogos Educativos (p. 36). Minicurso do Simpósio Brasileiro de Informática na Educação. [GS Search]

Dalmon, D. L., & Brandão, L. D. O. (2014). Sobre o Desenvolvimento de Software Educacional: proposta de uma Linha de Produto de Software para Módulos de Aprendizagem Interativa. *Revista Brasileira de Informática Na Educação*, *21*(3), 113–130. doi: 10.5753/RBIE.2013.21.03.113 [GS Search]

Dalmon, D. L., Brandão, L. O., Brandão, A. A. F., & Isotani, S. (2012). A domain engineering for interactive learning modules. *Journal of Research and Practice in Information Technology*, *44*(3), 309–330. [GS Search]

Dovogame. (2010). Business Tycoon Online. Retrieved from http://bto.dovogame.com/btoguide/index.php

Domínguez, A. H., Filho, C. A. C. L., Paraguaçu, F., & Oliveira, P. (2015). Um jogo digital baseado no construcionismo. *Revista Brasileira de Informática Na Educação*, *23*(2), 175. doi: 10.5753/rbie.2015.23.02.175 [GS Search]

Durscki, R. C., Spinola, M. M., Burnett, R. C., & Reinehr, S. S. (2004). Linhas de Produto de Software : riscos e vantagens de sua implantação. *VI Simpósio Internacional de Melhoria de Processos de Software*, 155–166. [GS Search]

Ellis, B., Stylos, J., & Myers, B. (2007). The factory pattern in API design: A usability evaluation. *Proceedings - International Conference on Software Engineering*, 302–311. doi: 10.1109/ICSE.2007.85 [GS Search]

Feigenspan, J., Kästner, C., Frisch, M., Dachselt, R., & Apel, S. (2010). Visual support for understanding product lines. *IEEE International Conference on Program Comprehension*, 34–35. doi: 10.1109/ICPC.2010.15 [GS Search]

Fein, G. G., Scholnick, E. K., Schwartz, S. S., & Frank, R. (n.d.). Computer Space: A conceptual and Developmental Analysis of LOGO. In Psychology Press (Ed.), Constructivism in the Computer Age. [GS Search]

Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., … Dantas, F. (2008). Evolving software product lines with aspects: an empirical study on design stability. *International Conference on Software Engineering*, 9. doi: 10.1145/1368088.1368124 [GS Search]

Firaxis Games. (2016). Civilization. Retrieved from http://www.civilization.com/en/games/.

Gacek, C., & Anastasopoules, M. (2001). Implementing product line variabilities. *Proceedings of the 2001 Symposium on Software Reusability: Putting Software Reuse in Context*, *26*(3), 109–117. doi: 10.1145/379377.375269 [GS Search]

Goguen, J. A. (1984). Parameterized Programming. *IEEE Transactions on Software Engineering*, *SE-10*(5), 528–543. doi: 10.1109/TSE.1984.5010277 [GS Search]

Júnior, P. O. A. M. (2008). *Analysis of Techniques for Implementing Software Product Lines Variabilities*. Universidade Federal de Pernambuco. [GS Search]

Kafai, Y. B. (2006). Cognitive apprenticeship. In *The Cambridge Handbook of The Learning Sciences* (pp. 35–46). doi: 10.1192/bjp.bp.106.029678 [GS Search]

Kakola, T., & Leitner, A. (2014). Introduction to Software Product Lines: Engineering, Services, and Management Minitrack. *2014 47th Hawaii International Conference on System Sciences*, 4984–4984. doi: 10.1109/HICSS.2013.327 [GS Search]

Kästner, C., Giarrusso, P. G., Rendel, T., Erdweg, S., Ostermann, K., & Berger, T. (2011). Variability-aware parsing in the presence of lexical macros and conditional compilation. *ACM SIGPLAN Notices*, *46*, 805. doi: 10.1145/2076021.2048128 [GS Search]

Kohwalter, T. C., Clua, E. W. G., & Murta, L. G. P. (2014). Reinforcing software engineering learning through provenance. *Proceedings - 28th Brazilian Symposium on Software Engineering, SBES 2014*, 131–140. doi: 10.1109/SBES.2014.16 [GS Search]

Lobo, A. E. de C. P., Brito, P. H. da S., & Rubira, C. M. F. (2007). *A Systematic Approah for Architectural Design of Component-Based Product Lines*. [GS Search]

Luo, X., Wei, X., & Zhang, J. (2010). Guided Game-Based Learning Using Fuzzy Cognitive Maps. *Learning*, *3*(4), 344–357. [GS Search]

MAXIS. (2014). SimCity. Retrieved from http://www.simcity.com/en_US/faq.

Mccabe, T. J. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering*, (4), 308–320. doi: 10.1109/TSE.1976.233837 [GS Search]

McGregor, J. D., Northrop, L. M., Jarrad, S., & Pohl, K. (2002). Initiating Software Product Lines. *IEEE Software*, *19*(August), 24–27. doi: 10.1109/MS.2002.1020282 [GS Search]

Medeiros, F., Lima, T., Dalton, F., Ribeiro, M., Gheyi, R., & Fonseca, B. (2016). Colligens. Retrieved from https://sites.google.com/a/ic.ufal.br/colligens/home .

Mello, T. S., & Rebouças, A. D. D. de S. (2015). GameMaking: Uma Metodologia para o Ensino de Informática para Alunos do Ensino Fundamental através da criação de Jogos Digitais. *Revista Brasileira de Informática Na Educação*, *23*(1), 197. doi: 10.5753/rbie.2015.23.01.197[GS Search]

Microsoft Research. (2017). Code Hunter. Retrieved from https://www.codehunt.com/about.aspx .

MIT Media Lab. (2016). Scratch. Retrieved from https://scratch.mit.edu/parents/.

Molina, L. G. (2013). Jogos digitais como espaço de atuação do historiador : o caso Avant-Garde Introdução Homo Ludens Ludens, 64–77. [GS Search]

Morelato, L. D. A., Guima, R. Y., & Borges, M. A. F. (2008). Gene2 : jogo via internet de apoio ao aprendizado de genética. In *Simpósio Brasileiro De Informática Na Educação* (Vol. 19, pp. 1–4). Fortaleza. [GS Search]

Nascimento, L. M., Almeida, E. S. De, & Meira, S. R. D. L. (2008). A Case Study in Software Product Lines - The Case of the Mobile Game Domain. *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, 43–50. doi: 10.1109/SEAA.2008.14 [GS Search]

Northrop, L. M. (2002). SEI's Software Product Line Tenets. *IEEE Software*, *19*(4), 32–40. doi: 10.1109/MS.2002.1020285 [GS Search]

Online Universitie. (2012). 20 Educational Games That Were Ahead of Their Time. Retrieved from http://www.onlineuniversities.com/blog/2012/09/20-educational-games-that-were-ahead-their-time/

Papert, S., & Harel, I. (1991). *Constructionism*. Ablex Publishing Corporation. [GS Search]

Prensky, M. (2001). *Digital game-based-learning*. New York, Editora McGraw-Hill. [GS Search]

Pohl, K., Böckle, G., & Linden, F. Van Der. (2005). *Software product line engineering*. *Springer*. Retrieved from http://link.springer.com/content/pdf/10.1007/3-540-28901-1.pdf [GS Search]

Robocode. (2017). Robocode. Retrieved from http://robocode.sourceforge.net/

Rocha, R. V., Bittencourt, Ig, I., & Isotani, S. (2015). Análise , Projeto , Desenvolvimento e Avaliação de Jogos Sérios e Afins : uma revisão de desafios e oportunidades. *IV Congresso Brasileiro de Informática Na Educação X Conferência Latino-Americana de Objetos E Tecnologias de Aprendizagem*, (October), 11. doi: 10.5753/cbie.sbie.2015.692 [GS Search]

Rogers, S. (2010). *Level Up!: The Guide to Great Video Game Design*. *_Imp_001_N*. Retrieved from http://books.google.com/books?hl=en&lr=&id=2LNaAwAAQBAJ&oi=fnd&pg=PR5&dq=Level+Up!:+The+Guide+to+Great+Video+Game+Design&ots=iOigkHpYQG&sig=GaU5W42X74D2DmrgQaiaoiRwqXM

Ruben, B. D. (1999). Simulations, Games, and Experience-Based Learning: The Quest for a New Paradigm for Teaching and Learning. *Simulation & Gaming*, *30*(4), 498–505. doi: 10.1177/104687819903000409 [GS Search]

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. doi: 10.1007/s10664-008-9102-8 [GS Search]

Santos, W. O. dos, Neto, S. R. da S., Junior, C. G. da S., & Bittencourt, I. I. (2015). Avaliação de Jogos Educativos: Uma Abordagem no Ensino de Matemática. *Simpósio Brasileiro de Informática Na Educação*, (26), 657. doi: 10.5753/cbie.sbie.2015.657 [GS Search]

Savi, R., & Ulbricht, V. R. (2008). Jogos Digitais Educacionais: Benefícios e Desafios. *Novas Tecnologias Na Educação*, *6*, 1–10. [GS Search]

Silva, A., Anselmo, P., Garcia, V. C., & Muniz, P. (2011). Linhas de Produtos de Software: Uma tendência da indústria. *Escola Regional de Informática Ceará, Maranhão, Piauí: Livro Texto Dos Minicursos, 07 E 08 de Novembro de 2011. [Livro Eletrônico]*, 7–31.[GS Search]

Software Engineering Institute. (2009). Arcade Game Maker: Pedagogical Product Line. Retrieved from http://www.sei.cmu.edu/productlines/ppl

Sudarmilah, E., Ferdiana, R., Nugroho, L. E., Susanto, A., & Ramdhani, N. (2013). Tech review: Game platform for upgrading counting ability on preschool children. *Proceedings - 2013 International Conference on Information Technology and Electrical Engineering: "Intelligent and Green Technologies for Sustainable Development", ICITEE 2013*, (October), 226–231. doi: 10.1109/ICITEED.2013.6676243 [GS Search]

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., & Leich, T. (2014). FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming*, *79*, 70–85. doi: 10.1016/j.scico.2012.06.002 [GS Search]

Unity Technologies. (2016). Unity. Retrieved from http://unity3d.com/pt/unity.

Von Wangenheim, C. G., Nunes, V. R., & Santos, G. D. Dos. (2014). Ensino de Computação com SCRATCH no Ensino Fundamental – Um Estudo de Caso. *Revista Brasileira de Informática Na Educação*, *22*(3), 115. doi: 10.5753/rbie.2014.22.03.115 [GS Search]

Watson, a. H., McCabe, T. J., & Wallace, D. R. (1996). Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. *NIST Special Publication*, *500*(235), 1–114. [GS Search]

Wolf, M. J. P. (2008). *A History from Pong to Playstation and Beyond. Editora: Greenwood Press. Londres.*

Wood, T. (1992). Artigo Fordismo , Toyotismo E Volvismo : Os Caminhos Da Industria. *RAE - Revista de Administração de Empresas*, *32*(4), 6–18. [GS Search]

Yoon, D. M., & Kim, K. J. (2015). Challenges and Opportunities in Game Artificial Intelligence Education Using Angry Birds. *IEEE Access*, *3*, 793–804. doi: 10.1109/ACCESS.2015.2442680 [GS Search]

YoYoGames. (2016). GameMaker. Retirado de: https://www.yoyogames.com/gamemaker