Prevention of Architectural Problems in Product Line Architecture Design Optimization

Tiago Tadeu Madrigar[1], Thelma Elita Colanzi[1], Willian Oizumi[2], and Alessandro Garcia[2]

[1] UEM - Universidade Estadual do Maringá - Maringá, PR, Brasil
tiago@madrigar.com.br, thelma@din.uem.br
[2] PUC-Rio - Rio de Janeiro, RJ, Brasil
woizumi@inf.puc-rio.br, afgarcia@inf.puc-rio.br

Resumo O uso de algoritmos de busca tem sido explorado com sucesso na otimização de projeto de Arquitetura de Linha de Produto de Software (PLA) na abordagem seminal chamada *Multi-Objective Approach for Product-Line Architecture Design* (MOA4PLA). Tal abordagem produz um conjunto de alternativas de projeto de PLA que melhora os diferentes fatores otimizados. Além da otimização desses fatores, os projetos de PLA obtidos deveriam idealmente ser livres de anomalias arquiteturais. Uma anomalia arquitetural pode impactar negativamente tanto a variabilidade e a extensibilidade da PLA como outros atributos não funcionais. No entanto, abordagens baseadas em busca, tais como a MOA4PLA, adversamente introduzem anomalias arquiteturais nas soluções geradas automaticamente. Portanto, no presente trabalho, apresentamos a ferramenta OPLA-Tool-ASP, que implementa diretrizes para prevenir as anomalias *Unused Interface, Unused Brick* e *Concern Overload* no contexto da MOA4PLA. Além disso, investigamos a prevenção da anomalia *Link Overload*. Um estudo envolvendo dois experimentos foi conduzido a fim de analisar a eficiência dessas diretrizes. Os resultados deste estudo mostram que a prevenção das anomalias arquiteturais estudadas é viável e que existem pontos de melhoria com relação à diretriz de prevenção da anomalia *Link Overload*.

Keywords: Software Product Line, Architectural Smell, Search-based Software Engineering

1 Introdução

A aplicação de algoritmos de busca tem alcançado resultados satisfatórios para problemas complexos da Engenharia de Software no campo de pesquisa denominado *Search Based Software Engineering* (SBSE) [8]. Dentre os problemas beneficiados estão alguns relacionados à abordagem de Linha de Produto de Software (LPS). Tal abordagem tem como alvo o desenvolvimento de uma família de produtos de software para um determinado domínio, reutilizando os artefatos gerados [9]. Um dos principais artefatos de uma LPS é a Arquitetura de Linha de Produto (PLA) que contém o projeto comum a todos os produtos

deriváveis da LPS [9]. O projeto de PLA é uma atividade que exige grande esforço do arquiteto e para o qual não há uma única solução possível devido aos fatores envolvidos, que vão desde a variabilidade e extensibilidade da PLA até a modularização de características comuns e variáveis.

Nesse contexto, Colanzi et al. [2] propuseram uma abordagem que usa algoritmos de busca para avaliar e melhorar projetos de PLA denominada MOA4PLA (*Multi-Objective Approach for Product-Line Architecture Design)* a fim de facilitar o trabalho do arquiteto. A aplicação desta abordagem foi automatizada por meio da ferramenta OPLA-Tool [4] que gera novas soluções a partir de uma solução inicial dada (o projeto original da PLA). Entretanto, a OPLA-Tool não é capaz de identificar e remover anomalias arquiteturais nos projetos gerados. Anomalias arquiteturais (do inglês *Architectural Smells* ou *Architectural Bad Smells*) são problemas decorrentes de decisões de projeto de arquitetura, intencionais ou não, que afetam negativamente a qualidade do sistema [7]. Em um estudo prévio [10], várias soluções arquiteturais geradas pela OPLA-Tool foram estudadas a fim de identificar as anomalias arquiteturais mais frequentes nas soluções. Uma das principais contribuições desse estudo foi a proposta de diretrizes para detecção e prevenção das anomalias arquiteturais mais frequentes. Porém, tais diretrizes ainda não foram implementadas nem validadas.

A identificação de anomalias arquiteturais em tempo de projeto da PLA: (i) evitaria que os problemas congênitos fossem propagados para a implementação da LPS e (ii) maximizaria atributos não-funcionais de qualidade desde a concepção da PLA. Em particular, certos atributos de qualidade – e.g., reusabilidade e manutenibilidade – são de extrema importância para a engenharia de qualquer LPS [9]. Porém, ainda não existem abordagens para a identificação de anomalias arquiteturais nem para a prevenção da introdução de anomalias em projetos de PLA otimizados por técnicas de SBSE [10].

Diante desse contexto, nosso principal objetivo é contribuir para a evolução da otimização de projeto de PLA por meio de uma ferramenta capaz de detectar e prevenir a introdução de anomalias arquiteturais durante a otimização de projeto de PLA. Para isso, no presente trabalho foi desenvolvida uma evolução da OPLA-Tool, denominada OPLA-Tool-ASP, a qual incorpora as diretrizes propostas por Perissato et al. [10] a fim de prevenir a presença de anomalias arquiteturais nos projetos de PLA gerados automaticamente (Seção 3). A OPLA-Tool-ASP tem o objetivo de prevenir as 3 anomalias mais frequentes no estudo prévio: *Unused Interface* [6], *Unused Brick* [6] e *Concern Overload* [6]. Um estudo experimental foi realizado utilizando a OPLA-Tool-ASP e a versão original da OPLA-Tool, a fim de responder a seguinte questão de pesquisa: ***QP1*** *- Qual é a efetividade da OPLA-Tool-ASP na detecção e prevenção das anomalias Unused Interface, Unused Brick e Concern Overload?* (Seções 4 e 5).

Segundo Perissato et al. [10], *Link Overload* [6] é a quarta anomalia mais frequente nos projetos analisados em seu estudo. Porém, eles perceberam que o cálculo do *threshold* precisa ser adaptado para o contexto de LPS antes de aplicar a diretriz proposta, pois foram identificados vários falsos positivos. Esse fato nos motivou a investigar seguinte questão de pesquisa antes da implemen-

tação da diretriz de prevenção de *Link Overload* na OPLA-Tool-ASP: ***QP2*** - *A estratégia utilizada em [10] para identificar a anomalia Link Overload é viável no contexto de LPS?* Nesse sentido, os resultados do nosso estudo experimental foram utilizados para analisar essa questão (Seção 5.3).

Dentre as principais contribuições deste trabalho está um incremento no estado da arte ao oferecer uma ferramenta para detectar e prevenir a geração de alternativas de projeto de PLA que contenham as anomalias arquiteturais *Unused Interface*, *Unused Brick* e *Concern Overload*. Outra contribuição é a adaptação do cálculo do *threshold* para anomalia *Link Overload* visto que o cálculo utilizado atualmente pelo estado da arte não é específico para LPS. Os principais conceitos envolvidos no presente trabalho são apresentados a seguir.

2 Background

In this section, we present the main concepts related to this work.

2.1 Search-based Software Engineering

In the Search-based Software Engineering (SBSE) field [8], Software Engineering problems are formulated as optimization problems, with the goal of minimizing or maximizing a function or a group of factors through search techniques. As a result, it is expected to reach (quasi-)optimal solutions. In a SBSE approach, there are usually two main aspects: a search space that contains all possible solutions for the problem; and a *fitness* function, which is responsible for evaluating the quality of the solutions [8].

Genetic algorithms are among the most used SBSE techniques. A Genetic Algorithm (GA) [1] is a metaheuristic inspired on the natural selection and genetic evolution theory. Starting from a initial population – which in our case are alternatives for the PLA design – search operators are applied to evolve the population throughout multiple generations. The selection operator is responsible for selecting solutions that present the best *fitness* values to survive as parents for the next generation. The crossover operator combines parts of two parent solutions to create a new one. Finally, the mutation operator randomly changes a solution. The succession population created from the selection, crossover, and mutation replaces the parent population.

Some GAs are adapted for optimizing multi-objective problems. Such GAs are called Multi-Objective Evolutionary Algorithms (MOEAs). The most popular and largely applied MOEA is the *Non-dominated Sorting Genetic Algorithm* (NSGA-II) [3]. A multi-objective problem depends on multiple factors (objectives) that may be conflicting and, therefore, there is not a single possible solution. Therefore, there may be multiple (quasi-)optimal solutions that represent the trade-off between the different objectives. Such solutions are called non-dominated solutions and form the Pareto frontier.

2.2 Otimização de Projeto de PLA

Colanzi et al. [2] propuseram a abordagem MOA4PLA com a finalidade de automatizar a busca por melhores projetos de PLA utilizando MOEAs. A MOA4PLA

recebe como entrada um projeto de PLA modelado em um diagrama de classes UML contendo todos os elementos arquiteturais comuns e variáveis. Estereótipos são utilizados para associar cada elemento às características que eles realizam. O arquiteto de LPS deve então escolher qual MOEA deve ser utilizado no processo de otimização assim como os parâmetros de execução. O projeto recebido como entrada é otimizado por meio dos operadores de busca definidos pela abordagem. Esses operadores realizam movimentos básicos dos AGs sendo: operadores de cruzamento de indivíduos e operadores de mutação dos novos indivíduos gerados. A abordagem inclui operadores de mutação específicos para projeto de PLA. Eles são: *Move Method, Move Attribute, Add Class, Move Operation, Add Component* e *Feature-Driven Operator* [2]. Este último visa melhorar a modularização de características entrelaçadas e espalhadas nos elementos arquiteturais.

A MOA4PLA usa um modelo de avaliação proposto para projeto de PLA [12]. Esse modelo fornece um conjunto de funções objetivo baseado em métricas de software, relacionadas a várias propriedades arquiteturais, dentre elas modularização de características, extensibilidade de LPS, variabilidade, coesão e acoplamento [12]. O arquiteto deve selecionar o subconjunto de funções objetivo que ele deseja otimizar. O valor das funções objetivo define o *fitness* de cada alternativa de projeto obtida durante o processo de otimização. A qualidade de cada alternativa de projeto é avaliada conforme seu *fitness*, que está diretamente relacionado aos objetivos selecionados pelo arquiteto a partir do modelo de avaliação. Após o processo de busca, a abordagem retorna um conjunto com as alternativas de projeto de melhor *trade-off* entre os objetivos otimizados. O arquiteto deve escolher uma das soluções obtidas de acordo com suas prioridades para adotar como PLA da LPS em desenvolvimento.

A ferramenta OPLA-Tool [4] automatiza a aplicação da MOA4PLA, permitindo a escolha de funções objetivo, MOEA e operadores de busca a serem utilizados, bem como a visualização das soluções (alternativas de projeto). Dentre os MOEAs disponíveis na OPLA-Tool está o NSGA-II. Como o principal objetivo da MOA4PLA é a otimização de projetos de PLA visando maximizar atributos de qualidade como reusabilidade, extensibilidade e manutenibilidade, é desejável que as soluções de projeto geradas pela abordagem sejam livres de problemas arquiteturais, os quais podem afetar diretamente os atributos mencionados. Porém, a versão atual da ferramenta não previne anomalias arquiteturais.

2.3 Identificação de Anomalias Arquiteturais em projeto de PLA

Anomalias arquiteturais são problemas decorrentes de decisões de projeto de arquitetura, intencionais ou não, que afetam negativamente a qualidade do sistema [7]. A existência de anomalias arquiteturais pode impactar negativamente nos aspectos que afetam as propriedades e estrutura interna do sistema como a capacidade de compreensão, capacidade de teste, extensibilidade e capacidade de reutilização [5] [6]. Após realizarmos um mapeamento sistemático da literatura identificamos que não existem ferramentas para a prevenção de anomalias arquiteturais em projetos de PLA otimizados por técnicas de SBSE.

As quatro anomalias arquiteturais alvo de estudo no presente trabalho estão descritas na Tabela 1. Neste trabalho, foram consideradas as anomalias *Unused*

Interface, Unused Brick, Concern Overload e *Link Overload* porque elas foram identificadas como as mais frequentes nas 24 instâncias de projeto de PLA geradas pela OPLA-Tool analisadas em [10]. Na Tabela 1 também são apresentadas as estratégias de identificação utilizadas no estudo prévio bem como as diretrizes propostas para a prevenção de tais anomalias nas soluções obtidas pela OPLA-Tool. No entanto, nenhuma das diretrizes propostas foi implementada.

No estudo prévio [10] a proposta para a prevenção de *Unused Interface* e *Unused Brick* foi descartar as soluções que contivessem tais anomalias de modo a prevenir sua propagação entre as soluções geradas durante o processo de otimização. Para a anomalia *Concern Overload* foi proposta a aplicação do operador de mutação *Feature-Driven Operator* da MOA4PLA, que propicia a modularização de alguma das características associadas à classe/interface anômala, com o intuito de resolver ou amenizar a manifestação da anomalia.

Vários falsos positivos foram identificados ao aplicar o cálculo de *threshold* proposto em [6] porque relacionamentos de herança, normalmente utilizados para resolver variabilidades da LPS, são computados impactando na identificação da anomalia *Link Overload* [10]. Sugeriu-se alterar a estratégia de identificação para não contar este tipo de relacionamento e apontaram a aplicação de uma penalidade ao *fitness* da solução como diretriz para prevenção desta anomalia [10].

3 OPLA-Tool-ASP

Além das funcionalidades da versão original da OPLA-Tool em termos de otimização de projeto de PLA, a primeira versão da OPLA-Tool-ASP implementa as diretrizes para a detecção e a prevenção das anomalias *Unused Interface*, *Unused Brick* e *Concern Overload* propostas em [10]. As próximas subseções apresentam os aspectos de implementação de tais diretrizes na OPLA-Tool-ASP [3].

3.1 Prevenção das anomalias *Unused Interface* e *Unused Brick*

Considerando que a anomalia *Unused Brick* é gerada devido à existência de *Unused Interface*, resolvendo a primeira anomalia, a segunda é resolvida automaticamente. Seguindo a estratégia de identificação de ambas as anomalias (Tabela 1), foi adicionada uma restrição na ferramenta para considerar inválida qualquer solução que contenha interfaces ou classes sem qualquer relacionamento com outros elementos arquiteturais. Para isso foi criado um método chamado *isValidSolution* que é executado sempre que uma nova solução é criada e retorna false quando uma interface/classe isolada for identificada.

O método *isValidSolution* é chamado no laço de repetição em que os operadores de cruzamento e de mutação são aplicados, a fim de gerar populações descendentes como mostrado nas linhas 8 e 11 do Algoritmo 1. Este método também é chamado durante a formação da população inicial. Assim, nenhuma das populações geradas durante o processo de busca conterá soluções com ocorrência das anomalias *Unused Interface* e *Unused Brick*, pois soluções inválidas não são adicionadas às novas populações.

[3] O pacote experimental está disponível em https://github.com/otimizes/opla-tool-asp.

Algorithm 1: Geração da população filho no NSGA-II

```
1  for int i = 0 ; i < (populationSize / 2); i++ do
2      if (evaluations < maxEvaluations) then
3          parents[0]=(Solution)selectionOperator.execute (population);
4          parents[1]=(Solution)selectionOperator.execute (population);
5          Object execute=crossoverOperator.execute (parents);
6          if (execute instanceof Solution) then
7              Solution offSpring=(Solution) crossoverOperator.execute(parents);
8              if (isValidSolution((Architecture) offSpring.getDecisionVariables()[0]))
                 then
9                  problem_.evaluateConstraints(offSpring);
10                 mutationOperator.execute(offSpring);
11                 if (isValidSolution((Architecture) offSpring.getDecisionVariables()[0]))
                     then
12                     problem_.evaluateConstraints(offSpring);
13                     problem_.evaluate(offSpring);
14                     offspringPopulation.add(offSpring);
```

3.2 Prevenção da anomalia *Concern Overload*

Para a identificação da anomalia *Concern Overload* foi implementado um método chamado *detectCO* (Algoritmo 2). Conforme sugerido na estratégia de identificação (Tabela 1), todas as características associadas às classes e interfaces da PLA são consideradas como *concerns* e são computadas.

Tabela 1: Catálogo de anomalias arquiteturais identificadas na MO4PLA

Tipo	**Definição**
Unused Interface	**Descrição**: Uma interface de componente é dita não utilizada quando não está ligada a nenhum componente, adicionando, portanto, uma complexidade desnecessária ao sistema e dificultando sua manutenção [6]. **Identificação**: foram consideradas para esta anomalia as interfaces e classes isoladas, ou seja, que não possuíam nenhum relacionamento. **Diretriz**: Considerar inválida a solução que contenha essas anomalias e então descartá-la.
Unused Brick	**Descrição**: Ocorre quando todas as interfaces de um componente apresentam a anomalia Unused Interface, ou seja, quando nenhuma de suas interfaces possui ligação a outro componente [6]. **Identificação**: foram considerados os pacotes em que nenhuma interface ou classe possuía relacionamentos. **Diretriz**: Considerar inválida a solução que contenha essas anomalias e então descartá-la.
Concern Overload	**Descrição**: Um componente que realiza um número excessivo de características, quebrando o princípio da responsabilidade única [6]. **Identificação**: Considerar as características de cada LPS como sendo os *concerns* do projeto de PLA. Utilizar o algoritmo proposto por Garcia [6] para calcular o *threshold* (valor limite) e contar as características associadas às classes e interfaces do projeto. **Diretriz**: Aplicar o operador de mutação *Feature-Driven Operator* para modularizar alguma das características associadas à classe/interface anômala.
Link Overload	**Descrição**: Um componente que possui excessiva dependência de outros componentes, prejudicando assim a separação das funcionalidades e o isolamento das mudanças. Essas dependências podem se manifestar como *links* de entrada, de saída, ou ambos [6]. **Identificação**: o algoritmo proposto em [6] foi utilizado para definir o valor limite para cada tipo de *link*. Relacionamentos de dependência, realização, herança, etc. foram considerados *links*, bem como uma classe possuir um atributo que é do tipo de outra classe. O *threshold* foi calculado para cada direcionalidade: entrada, saída e bidirecional, arredondando valores fracionários. Depois, fez-se a contagem em todas classes e interfaces. **Diretriz**: Penalizar o *fitness* da solução, de modo que ela seja pior avaliada no processo de seleção dos melhores projetos para permanecerem no processo de otimização.

Algorithm 2: Detecção da anomalia Concern Overload

```
1  public boolean detectCO(Solution solution) throws JMException {
2  final Architecture arch = ((Architecture) solution.getDecisionVariables()[0]);
3  final List < Package > allPackage = new ArrayList < Package > (arch.getAllPackages());
4  if (!allPackage.isEmpty()) then
5      for (Package selectedPackage: allPackage) do
6          List < Class > lstClass = new ArrayList < >(selectedPackage.getAllClasses());
7          for (Class selectedClass: lstClass) do
8              List < Concern > lstConcern = new ArrayList <
                >(selectedClass.getOwnConcerns());
9              if (lstConcern.size() > threshold) then
10                 return true;
11             end
12         end
13         List < Interface > lstInterface = new ArrayList <
             >(selectedPackage.getAllInterfaces());
14         for (Interface selectedInterface: lstInterface) do
15             List < Concern > lstConcern = new ArrayList <
                 >(selectedInterface.getOwnConcerns());
16             if (lstConcern.size() > threshold) then
17                 return true;
```

O método *detectCO* também calcula o *threshold* que é utilizado para identificar se a classe/interface é anômala. O *threshold* é a média do número de *concerns* somada ao desvio padrão [6]. Este trecho foi omitido por falta de espaço.

Nas linhas 6-11 do algoritmo detecta-se a presença de *Concern Overload* nas classes do projeto. Para isso, nesse *loop* verifica-se se o número de características associadas a uma determinada classe é maior do que o *threshold*, retornando *true* quando a condição é verdadeira. O mesmo procedimento é realizado para as interfaces entre as linhas 12 e 17.

O método *detectCO* foi implementado na classe *PLAFeatureMutation*, responsável por implementar os operadores de mutação da MOA4PLA, incluindo o operador *Feature-driven Operator*, que propicia a modularização de características e como consequência, pode diminuir a ocorrência de *Concern Overload*.

Na OPLA-Tool-ASP, conforme sugerido em [10], o método *detectCO* é executado antes de selecionar qual operador de mutação será aplicado. Caso seja detectada a presença da anomalia *Concern Overload*, o operador *Feature-driven Operator* é selecionado para aplicação, caso contrário, o código retorna a seu fluxo normal, sorteando aleatoriamente qualquer operador de mutação.

4 Projeto do Estudo Experimental

Neste trabalho foram conduzidos dois experimentos com intuito de avaliar a efetividade da OPLA-Tool-ASP em prevenir as anomalias *Unused Interface*, *Unused Brick* e *Concern Overload*, além de analisar a estratégia de identificação da anomalia *Link Overload* proposta em [10] no contexto de LPS. O experimento denominado OPLA-Tool foi utilizado como controle, pois trata-se da versão original desta ferramenta. O experimento OPLA-Tool-ASP se refere à versão da ferramenta que contém as diretrizes para detecção e prevenção das anomalias.

Os projetos de PLA utilizados nos dois experimentos são *Arcade Game Maker* (AGM) e *Mobile Media* (MM). A AGM [11] é uma LPS que engloba um conjunto

de jogos arcade (Brickles, Bowling e Pong). A LPS MM [13] oferece recursos que lidam com música, vídeos e fotos para dispositivos portáteis.

Para os dois experimentos (OPLA-Tool e OPLA-Tool-ASP) foram utilizadas as seguintes configurações: todos os operadores de mutação da MOA4PLA com probabilidade de mutação igual a 0.9; população com 100 indivíduos; número de avaliações de *fitness* igual a 30.000 (300 gerações); funções objetivo: *Relational Cohesion* (COE), *Class Coupling* (ACLASS) e *Feature Modularization* (FM). As funções COE, ACLASS e FM referem-se à coesão relacional, acoplamento de classes e modularização de características [12]. Em cada experimento foram realizadas 30 execuções independentes para cada PLA. Essas mesmas configurações foram utilizadas em estudos anteriores [2] [12].

Dentre todas as soluções geradas nos dois experimentos, foram coletadas as 36 soluções não-dominadas geradas pelas duas ferramentas após as 30 execuções. A identificação das anomalias foi realizada manualmente por um dos autores e revisada por outro autor. Para cada anomalia arquitetural avaliada neste trabalho, utilizou-se a estratégia de identificação apresentada na Tabela 1. Os valores do *threshold* de *Concern Overload* são AGM = 4 e MM = 5. Devido à formula resultar em valores fracionários, os valores foram arredondados para cima.

5 Resultados

Nesta seção são apresentados os resultados encontrados no estudo experimental realizado. A OPLA-Tool-ASP encontrou 7 soluções para a AGM e 7 soluções para a MM enquanto a versão original da OPLA-Tool encontrou 13 soluções para a AGM e 9 soluções para a MM.

A Figura 1 apresenta as soluções encontradas para AGM dispersas no espaço de soluções. Nos dois gráficos, a curva azul apresenta as soluções encontradas pela OPLA-Tool. A curva em vermelho apresenta as soluções obtidas pela OPLA-Tool-ASP. E a solução original é apresentada em verde. Os valores de *fitness* da função objetivo FM estão apresentados no eixo Y dos dois gráficos e a função COE é apresentada no eixo X do gráfico da esquerda enquanto a função ACLASS está no eixo X do gráfico da direita.

A Figura 2 apresenta a dispersão das soluções encontradas para MM. Nos dois gráficos, o projeto original e as soluções encontradas pela OPLA-Tool e pela OPLA-Tool-ASP são apresentadas em verde, azul e vermelho, respectivamente.

Por meio dos gráficos apresentados, pode-se comparar os valores de *fitness* das soluções encontradas pelos dois experimentos com o *fitness* da PLA original. Considerando que quanto menor o valor das três funções, melhor é o resultado, é possível perceber que ambos experimentos conseguiram melhorar os valores de *fitness* originais das funções objetivo FM e ACLASS tanto para a AGM como para a MM. No caso da AGM, todas as soluções obtidas pela OPLA-Tool-ASP têm melhor valor da função objetivo COE e pior valor da função ACLASS do que a PLA original. Em todos os outros casos o valor da coesão relacional (função objetivo COE) foi pior do que o valor original, mostrando que ao melhorar o acoplamento (ACLASS) , a coesão tende a piorar.

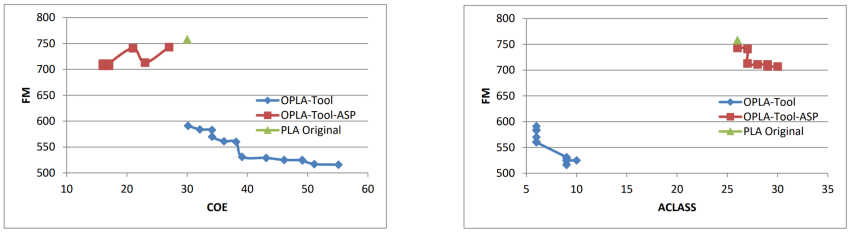


Figura 1: Soluções encontradas para a AGM

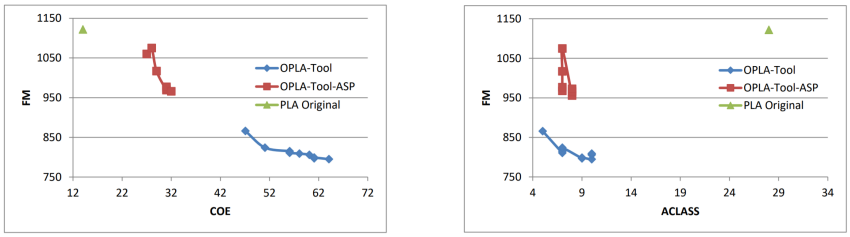


Figura 2: Soluções encontradas para a MM

Tabela 2: Total de anomalias detectadas nas soluções geradas

Anomalia	Original	OPLA-Tool													OPLA-Tool-ASP						
		S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S1	S2	S3	S4	S5	S6	S7
Arcade Game Maker																					
Unused Interface	0	5	4	4	4	5	1	3	2	4	1	6	3	3	0	0	0	0	0	0	0
Unused Brick	0	11	8	8	8	6	9	6	8	9	9	6	6	8	0	0	0	0	0	0	0
Concern Overload	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Mobile Media																					
Unused Interface	0	9	8	6	5	15	7	6	8	15					0	0	0	0	0	0	0
Unused Brick	0	4	3	1	2	2	2	1	2	3					0	0	0	0	0	0	0
Concern Overload	2	1	1	2	1	1	2	2	2	1					1	3	3	1	2	2	1

Comparando as soluções obtidas pelos 2 experimentos (OPLA-Tool-ASP e OPLA-Tool), percebe-se que as soluções obtidas pela OPLA-Tool têm melhores valores da função objetivo FM para as 2 PLAs, o que pode indicar que a soluções obtidas pela OPLA-Tool-ASP têm pior modularização das características. Por outro lado, OPLA-Tool-ASP conseguiu soluções mais coesas (função COE) do que a OPLA-Tool para as 2 PLAs.

Tão importante quanto o *fitness* das soluções é a existência de anomalias arquiteturais nas soluções de projetos obtidas automaticamente. Nesse sentido, a Tabela 2 apresenta o número de ocorrências de cada tipo de anomalia arquitetural em cada solução obtida nos dois experimentos para as PLAs AGM e MM. As soluções obtidas pelos experimentos são identificadas como S[n], onde n representa o número da solução. A quantidade de anomalias observadas nas PLAs originais é apresentada na coluna rotulada como Original.

5.1 *Unused Interface* e *Unused Brick*

As anomalias *Unused Interface* e *Unused Brick* não estão presentes nos projetos originais. Observando a Tabela 2 é possível perceber que nos experimentos

conduzidos com a OPLA-Tool, estas anomalias aparecem em todas as soluções obtidas. Já as soluções obtidas com a ferramenta OPLA-Tool-ASP não apresentaram a existência destas anomalias. Devido ao método *isValidSolution* (Seção 3), qualquer solução contendo classes ou interfaces que não possuíam ligação com os demais elementos arquiteturais foi descartava, por ser considerada inválida. Desse modo, todas as populações de soluções geradas durante o processo de busca estava livre das anomalias *Unused Interface* e *Unused Brick*. A implementação da diretriz de prevenção dessas anomalias se mostrou eficaz.

No caso das soluções geradas pela OPLA-Tool onde há ocorrência destas anomalias, percebeu-se que os elementos anômalos são, em geral, novos pacotes que foram criados por operadores de mutação, os quais contêm interfaces que não estão conectadas a nenhum outro elemento arquitetural. Esses novos pacotes geralmente contêm elementos arquiteturais para realizar uma única característica o que impacta beneficamente nos valores de *fitness* da função objetivo FM, por outro lado manifestam ao menos uma das anomalias *Unused Interface* e *Unused Brick*. Por outro lado, as soluções obtidas pela OPLA-Tool-ASP não têm valores de *fitness* da função objetivo FM tão baixos como as soluções da OPLA-Tool, mas não apresentam as anomalias supracitadas. Portanto, por meio dos resultados obtidos pela OPLA-Tool-ASP, observou-se que é pertinente otimizar menos e obter soluções sem tais anomalias arquiteturais.

> ***QP1:*** *A OPLA-Tool-ASP é efetiva para detectar e prevenir as anomalias Unused Interface e Unused Brick.*

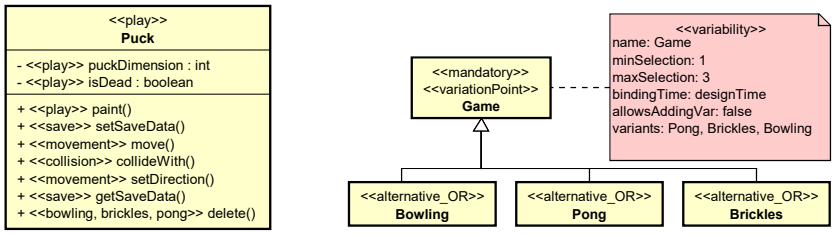
5.2 *Concern Overload*

A Tabela 2 mostra que todas as soluções obtidas pela OPLA-Tool-ASP e pela OPLA-Tool contêm a anomalia *Concern Overload*. Nota-se também a existência desta anomalia nas PLAs originais.

No caso da AGM, percebe-se que o número de ocorrências desta anomalia permaneceu constante para todas as soluções obtidas nos dois experimentos. A AGM tem classes associadas a várias características o que dificulta a modularização das características durante o processo de busca. Apesar de os experimentos terem obtido soluções com valor de *fitness* da função objetivo FM melhor do que o valor original (FM=758.0), o número de ocorrências da anomalia não abaixou. As duas classes da AGM que se mantiveram anômalas em todas as soluções geradas são as classes *Puck* (mostrada na Figura 3a) e *Sprite*. Nota-se pelos estereótipos da classe *Puck* que ela tem atributos e métodos que auxiliam na realização de 7 diferentes características («play», «save», «movement», «collision», «bowling», «brickles» e «pong»).

A Tabela 3 apresenta os resultados do experimento OPLA-Tool-ASP para a anomalia *Concern Overload* para alguns elementos arquiteturais (classes ou interfaces) associados a várias características. O número de características associadas ao elemento no projeto original é mostrado nas colunas rotuladas como **O**. Pode-se observar que em alguns elementos o número de características sofreu uma leve redução, como por exemplo a classe *GameBoard* da AGM que na

solução original estava associada a 3 *concerns* e foi reduzido para 2 na solução S7 e a interface *ICheckScore* que na solução original estava associada a 2 *concerns*, mas está associada a 1 *concern* nas soluções S4 e S7. O *fitness* da PLA original é (COE=30, ACLASS=26, FM=758), enquanto os *fitness* de S4 e S7 são S4:(COE=17, ACLASS=28, FM=711) e S7:(COE=21, ACLASS=27, FM=741). Logo, nota-se que a modularização das características impacta no valor de FM e que esta função é um bom indicador para o controle de *Concern Overload*.



(a) Classe Puck (b) Uso de generalização em ponto de variação

Figura 3: Exemplos de Classes da AGM

Tabela 3: Anomalia *Concern Overload* nas soluções da OPLA-Tool-ASP

Arcade Game Maker									**Mobile Media**								
Classe/Interface	**O**	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**Classe/Interface**	**O**	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**
GameBoard	3	3	3	3	3	3	3	2	IMediaMgt	8	7	6	6	6	7	6	6
ICheckScore	2	2	2	2	1	2	2	1	MediaMgr	8	4	7	6	4	5	4	4
Score	2	2	2	2	1	2	2	1	MediaCtrl	7	5	6	6	3	6	6	5

No caso da MM, houve comportamento semelhante. A classe *MediaMgr*, que na solução original realizava 8 *concerns*, está associada a 5 e 4 *concerns* em várias soluções, deixando de ser anômala, pois o *threshold* foi definido como 5. As alterações no número de características realizadas pelos elementos arquiteturais se devem à aplicação do operador de mutação *Feature-driven Operator*, que tem como objetivo modularizar características que estejam entrelaçadas com outras nos componentes do projeto. Na maioria das soluções, foram criados novos componentes para modularizar características impactando diretamente no número de elementos anômalos.

Tanto para a MM como para a AGM, o número de componentes criados para modularizar características foi maior nas soluções da OPLA-Tool-ASP. Isto se deve ao fato de que na versão original da OPLA-Tool, os operadores de mutação são selecionados aleatoriamente enquanto na OPLA-Tool-ASP o *Feature-driven Operator* sempre é aplicado quando há ocorrência de *Concern Overload*.

Conclui-se que como o número de *concerns* diminuiu em certos elementos (classes/interfaces) e alguns deixaram de ser anômalos, a OPLA-Tool-ASP apresentou resultados satisfatórios em relação a *Concern Overload*. Esses resultados nos permitem supor que o algoritmo de busca precisaria de mais gerações para otimizar ainda mais o valor da função objetivo FM e, assim, obter a soluções sem a ocorrência desta anomalia.

QP1: *A diretriz implementada na OPLA-Tool-ASP contribui para prevenir a anomalia Concern Overload.*

5.3 *Link Overload*

Em relação a anomalia *Link Overload*, no estudo realizado por Perissato et al. [10], a estratégia de identificação utilizada seguiu o algoritmo definido em [6], no qual todos os relacionamentos são computados para o cálculo do *threshold*. O cálculo consiste na média do número de relacionamentos somada ao desvio padrão. Deve-se calcular o *threshold* para os 3 tipos de direcionalidades dos relacionamentos: entrada, saída e ambos[6]. Seguindo esse cálculo, os *thresholds* de *Link Overload* seriam AGM = (entrada: 2, saída: 2, ambos: 1) e MM = (entrada: 2, saída: 3, ambos: 0).

No entanto, apesar de tal cálculo ser apropriado para projeto arquitetural, ele não é específico para o contexto de LPS gerando falsos positivos. Desse modo, umas das propostas de Perissato et al. foi adaptar o cálculo de *threshold* desconsiderando os relacionamentos de herança, a fim de aumentar a acurácia da estratégia de identificação. No contexto de LPS é habitual utilizar generalização para definir os pontos de variação e seus variantes como ilustrado na Figura 3b, na qual a superclasse *Game* é o ponto de variação e as classes *BowlingGame, PongGame* e *BricklesGame* são os variantes.

A Figura 4 ilustra como seria a avaliação do *threshold* para as LPSs AGM e MM com o *threshold* original (cálculo definido em [6]) e o cálculo do *threshold* proposto (sem considerar as generalizações). Percebe-se que o *threshold* proposto detectaria um número menor de ocorrências dessa anomalia, já que o limiar seria menor, indicando um menor número de prováveis falsos positivos.

A título de ilustração, se não contarmos as generalizações, a classe *Game* (Figura 3b) possui nenhum relacionamento de entrada e de saída e 1 relacionamento bidirecional, logo os valores não excedem o *threshold*. Se as generalizações forem contadas, *Game* seria considerada uma classe anômala,o que constitui um falso positivo. Comportamentos similares ocorreriam com outras classes das duas LPS utilizadas no nosso estudo, como por exemplo, *StationarySprite* da AGM e *Game* da MM. Portanto, conclui-se que a adaptação do cálculo do *threshold* de *Link Overload* é apropriada para o contexto de LPS e será utilizada na implementação da diretriz desta anomalia na OPLA-Tool-ASP.

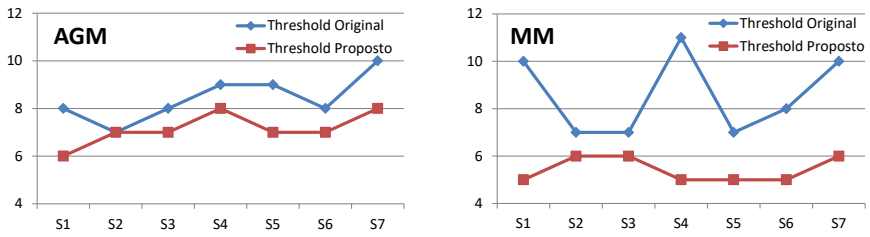


Figura 4: Total de ocorrências de *Link Overload* nas soluções da OPLA-Tool-ASP

Outra análise realizada diz respeito ao número de ocorrências desta anomalia nas soluções obtidas nos experimentos. Nas PLAs originais da AGM e da MM o número de elementos que continham *Link Overload* era 3. Porém, na Figura 4 os números são maiores que 3, independentemente do cálculo de *threshold* utilizado. Além do impacto causado pelos operadores de busca que movimentam elementos arquiteturais no projeto aumentando as ocorrências de *Link Overload* [10], nós percebemos uma outra necessidade de adaptação no cômputo do *threshold* dessa anomalia no que se refere às novas classes/interfaces criadas.

Considerando todas as soluções obtidas para a MM, 10 classes/interfaces foram criadas e seus relacionamentos são bidirecionais. Como o *threshold* para a direcionalidade ambos é 0 para a MM, as novas classes/interfaces, criadas pelo operador *Feature-driven Operator* para modularizar características entrelaçadas, passam a ser consideradas anômalas ainda que tenham somente 1 relacionamento, o que se constitui um falso positivo, já que pressupõe-se ao menos um relacionamento para garantir a visibilidade entre os elementos arquiteturais.

> ***QP2:*** *Para o contexto de LPS é apropriado não contar os generalizações no cálculo do threshold da anomalia Link Overload e classes/interfaces com somente 1 relacionamento não devem ser consideradas anômalas independentemente do valor do threshold.*

5.4 Ameaças à validade

A validade interna e externa estão relacionadas ao conjunto de projetos de PLA utilizados neste estudo. Apesar das soluções analisadas terem sido obtidas a partir de LPS acadêmicas, MM e AGM, foi possível identificar a existência das anomalias arquiteturais investigadas no presente trabalho.

A validade de construção está relacionada à configuração do estudo. Com relação às estratégias de identificação de anomalias utilizadas, buscou-se sempre que possível utilizar critérios objetivos baseados na literatura. Um dos pesquisadores envolvidos nesta pesquisa realizou uma análise por inspeção visual nos 36 projetos de PLA gerados pelas ferramentas. Para mitigar esta ameaça, um especialista realizou uma validação das anomalias identificadas.

A principal ameaça à validade das conclusões é o número de LPSs (2) e projetos de PLA (36) avaliados. Apesar de se tratar de um estudo cujo resultado não pode ser generalizado, foi possível identificar diferenças em relação aos projetos originais que geraram soluções diferentes entre si e que permitiram tanto avaliar o efeito da OPLA-Tool-ASP na prevenção de anomalias no contexto da AGM e da MM, como identificar pontos de adaptação para o cálculo do *threshold* da anomalia *Link Overload*.

6 Conclusão

Neste trabalho apresentamos a ferramenta OPLA-Tool-ASP, que tem como alvo gerar automaticamente projetos de PLA que otimizam os objetivos selecionados

pelo engenheiro de software, além de prevenir anomalias arquiteturais nas soluções de projeto geradas. A primeira versão da ferramenta inclui diretrizes para prevenir as anomalias *Unused Interface, Unused Brick* e *Concern Overload*.

As principais contribuições deste trabalho envolvem: (a) o avanço do estado da arte na otimização de projeto de PLA prevenindo a ocorrência de algumas anomalias arquiteturais; (b) os resultados experimentais que indicam a efetividade da OPLA-Tool-ASP para prevenir as anomalias supracitadas e (c) descobertas a respeito da necessidade de adaptação do cálculo do *threshold* para a identificação da anomalia *Link Overload* no contexto de LPS.

Atualmente estamos implementando a prevenção de *Link Overload* na ferramenta. Pretendemos executar novos estudos com maior número de gerações para reavaliar a ocorrência de *Concern Overload* e realizar uma avaliação qualitativa das soluções geradas com especialistas. Na sequência, outras anomalias estudadas em [10] devem ser incluídas na OPLA-Tool-ASP e devidamente validadas por meio de estudos experimentais.

Referências

1. Coello, C.A.C., Lamont, G.B., Van Veldhuizen, D.A., et al.: Evolutionary algorithms for solving multi-objective problems, vol. 5. Springer (2007)
2. Colanzi, T.E., Vergilio, S.R., Gimenes, I., Oizumi, W.N.: A search-based approach for software product line design. In: Proc. of the 18th International Software Product Line Conference-Volume 1. pp. 237–241. ACM (2014)
3. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans. Evolutionary Comp. **6**(2), 182–197 (2002)
4. Féderle, É.L., do Nascimento Ferreira, T., Colanzi, T.E., Vergilio, S.R.: Opla-tool: a support tool for s9earch-based product line architecture design. In: Proc. of the 19th International Conference on Software Product Line. pp. 370–373. ACM (2015)
5. Fowler, M.: Patterns of enterprise application architecture. Addison-Wesley Longman Publishing Co., Inc. (2002)
6. Garcia, J.: A unified framework for studying architectural decay of software systems. University of Southern California (2014)
7. Garcia, J., Popescu, D., Edwards, G., Medvidovic, N.: Identifying architectural bad smells. In: 2009 13th European Conf. on Software Maintenance and Reengineering. pp. 255–258. IEEE (2009)
8. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys **45**(1), 11 (2012)
9. Linden, F.J., Schmid, K., Rommes, E.: Software product lines in action: the best industrial practice in product line engineering. Springer Science & Bus.Media (2007)
10. Perissato, E.G., Neto, J.C., Colanzi, T.E., Oizumi, W., Garcia, A.: On identifying architectural smells in search-based product line designs. In: VII Brazilian Symposium on Software Components, Architectures, and Reuse. pp. 13–22 (2018)
11. Software Engineering Institute: Arcade game maker: Pedagogical product line `http://www.sei.cmu.edu/productlines/ppl`
12. Verdecia, Y.D., Colanzi, T.E., Vergilio, S.R., Santos, M.C.B.: An enhanced evaluation model for search-based product line architecture design. In: XX Ibero-American Conf. on Software Engineering (CIbSE2017). Buenos Aires (2017)
13. Young, T.J.: Using aspectj to build a software product line for mobile devices. Ph.D. thesis, University of British Columbia (2005)