UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA
Autorizada pelo Decreto Federal nº 77.496 de 27/04/76
Recredenciamento pelo Decreto n°17.228 de 25/11/2016
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
COORDENAÇÃO DE INICIAÇÃO CIENTÍFICA

XXV SEMINÁRIO DE INICIAÇÃO CIENTÍFICA DA UEFS
SEMANA NACIONAL DE CIÊNCIA E TECNOLOGIA - 2021

CORRELAÇÃO ENTRE ANOMALIAS DE CÓDIGO E PADRÕES DE PROJETO

Gabriel Yago de O. Moreira[1]; José A. M. Santos [2]

1. Bolsista PIBIC/CNPq, Graduando em Engenharia de Computação, Universidade Estadual de Feira de Santana, e-mail: gyagom@gmail.com

2. Orientador, Departamento de Tecnologia, Universidade Estadual de Feira de Santana, e-mail: zeamancio@uefs.br

PALAVRAS-CHAVE: Anomalias de código; Padrões de projeto; Desenvolvimento de software.

INTRODUÇÃO

A pesquisa tem como objetivo propor uma estratégia de classificação de smells (anomalias). Para isso, vamos selecionar alguns smells de livros de autores já conceituados como Brown et al. (1998), Fowler (1999), e os trabalhos de Marinescu (Lanza, Marinescu, 2005). A partir dos smells discutidos por esses autores, selecionamos as principais características que os definem. A partir da extração desse conteúdo, os smells foram classificados com relação à: i) igualdade, ii) semelhanças; e iii) diferença entre eles. Para isso foi lido de cada um dos autores mencionados e como eles descrevem os smells. As principais características foram registradas em um documento de texto, separado por cada autor. Esse registo forma um resumo das características dos smells, onde é possível comparar as semelhanças entre os mesmos. Além disso, registramos a forma como o autor explica a característica marcante de um determinado smell e o que o diferencia do outro, apesar de admitir as semelhanças entre eles. Com essas informações foi possível montar em uma tabela como referência comparando suas igualdades, semelhanças e a diferença, que o faz ter uma classificação diferente, tornando-o único.

Através dessa pesquisa, buscamos contribuir com as discussões apontadas por diversos autores como os desafios para a área de desenvolvimento de software. Um deles trata da falta de consistência nas nomenclaturas adotadas para definição de smells. Discutimos se é realmente necessária uma classificação diferente ou se é possível ter uma classificação única para um grupo de smell semelhantes, com poucas ou apenas uma única característica diferente. O seguinte questionamento tenta ser respondido como: 1. É realmente necessário definir de forma tão específica cada smell, ou eles podem ser agrupados pelas suas características principais?

MATERIAL E METODOLOGIA

Foram selecionados ao todo 30 (trinta) smells para análise de suas características e posteriormente suas classificações. Desses, foram selecionados os smells que estão sendo mais investigados por estudos. Além disso, decidimos discutir apenas os smells que se referem a classes. Desconsideramos os smells referentes a métodos para simplificar e viabilizar o estudo. Dessa forma, os seguintes smells foram discutidos: Brain Class, Good Class (Lanza, Marinescu, 2005); Large Class (Fowler, 1999) e

(Lanza, Marinescu, 2005); The Blob, Swiss Army Knife (Brown et al., 1998). Estes serão a fonte e matéria de estudo desta pesquisa.

RESULTADOS E DISCUSSÃO

Nosso estudo destaca a necessidade de investigar as correlações entre os smells que são encontrados no desenvolvimento de software quando aplicados os princípios básicos de orientação a objeto (OO) e a qualidade do projeto. Ele amplia o número de investigações sobre os smells nos projetos de software. Estamos ampliando e escrevendo um artigo com a colaboração bolsista-orientador, para a posterior submissão. Além disso, essa pesquisa proporcionou o aumento no volume de dados sobre o tema de smells no código fonte. Foi possível a ampliação do grau de compreensão sobre os conceitos de anomalias de código em processos de desenvolvimento de software. Essa compreensão permite direcionar pesquisas na área que desconsideram aspectos práticos no desenvolvimento de software, bem como a produção de ferramentas de suporte.

O primeiro smell analisado foi o Large Class (Lanza, Marinescu, 2005), que é caracterizado por ser uma classe que tenta fazer muito trabalho, além de ter muitas variáveis de instância e um provável código duplicado. Fowler (1999) ainda acrescenta que, por ser uma classe com muita redundância de código, os métodos podem apresentar muito código em comum. Seus principais sintomas são: um grande número de variáveis de instância, bem como um grande número de métodos e um grande número de linhas.

O segundo smell foi o Brain Class (Lanza, Marinescu, 2005), que é caracterizado por serem classes complexas que tendem a acumular uma quantidade excessiva de inteligência. São classes complexas que não acessam dados de classes "satélite" de forma abusiva, ou são um pouco mais coesas. Têm a tendência de centralizar a inteligência do sistema. Além disso, apresenta uma agregação de diferentes abstrações e mal-uso de outras classes (muitas vezes, mero portador de dados) para executar sua funcionalidade. Na maioria das vezes, eles são contra os princípios básicos do projeto orientado a objetos, que diz que uma classe deve ter uma responsabilidade. Seus principais sintomas são: possuem pelo menos alguns métodos afetados pelo Brain Method, no caso de ser muito grande em LOC (em termos de linhas de código do inglês lines of code - LOC), apresentar baixa coesão e ser muito complexo. Outro sintoma é a classe ser um "monstro" em termos de tamanho (LOC) e complexidade funcional (Contagem de método ponderado do inglês Weighted Method Count - WMC), então a classe é considerada uma Brain Class, mesmo que tenha apenas um Brain Method. De forma mais detalhada para a detecção temos de observar se a classe contém mais de um Brain Method e o método é muito grande.

O terceiro smell foi o God Class (Lanza, Marinescu, 2005), que é caracterizado por serem classes que tendem a centralizar a inteligência do sistema. Uma Good Class realiza muito trabalho por conta própria, delegando apenas pequenos detalhes a um conjunto de classes triviais e usando os dados de outras classes. Este problema pode ser comparável ao smell Large Class citado por Fowler (1999). Seus principais sintomas são: a característica forte de acessar os dados de outras classes mais simples, diretamente ou usando métodos de acesso. A classe usa diretamente mais do que alguns atributos de outras classes. E quanto mais atributos estrangeiros são usados pela classe, maior é a probabilidade de que uma classe seja (ou esteja prestes a se tornar) uma God Class.

O quarto smell foi o The Blob (Brown et al., 1998), que é caracterizado por um o design de estilo procedimental levando a um objeto com a maior parte das responsabilidades, enquanto a maioria dos outros objetos apenas mantém dados ou executa processos

simples. Consume arquiteturas orientadas a objetos inteiras. A classe tende a monopolizar o processamento enquanto as outras classes têm a tendência de encapsular os dados. Ou seja, a maioria das responsabilidades é atribuída a uma única classe. Geralmente é um projeto procedural, embora possa ser representado usando notações de objeto e implementado em linguagens orientadas a objetos. Tendendo a separar suas responsabilidades dos seus dados; em outras palavras, eles são de estilo procedural em vez de arquiteturas orientadas a objetos. Além de ser frequentemente acompanhado por código desnecessário, tornando difícil diferenciar entre a funcionalidade útil da Classe Blob e o código não mais usado. Seus principais sintomas são: Classe única com um grande número de atributos, operações ou ambos. Uma classe com 60 ou mais atributos e operações geralmente indica a presença do Blob (Akroyd, 1996). Apresenta ainda uma discrepante coleção de atributos e operações encapsuladas não relacionadas em uma única classe. Ou seja, uma falta geral de coesão dos atributos e operações.

O quinto smell foi o Swiss Army Knife (Brown et al., 1998), que apesar de ser uma interface tem todos os seus métodos implementados pela classe, tornando a classe que implementa a sua interface um smell. Esse smell é caracterizado por ser uma interface de classe excessivamente complexa. O projeto tenta fornecer todos os usos possíveis da classe. Na tentativa, ele adiciona um grande número de assinaturas de interface em uma tentativa inútil de atender a todas as necessidades possíveis. Seus principais sintomas são: incluir de dezenas a milhares de assinaturas de método para uma única classe. O projeto pode não ter uma abstração ou propósito claro para a classe, o que é representado pela falta de foco na interface.

Existe um número significativo de características iguais entres os smells, os que se destacam são:

1. Executa muito trabalho por conta própria, acumular uma quantidade excessiva de inteligência ou é um objeto com a maior parte das responsabilidades.
2. Muito complexa para reutilização e teste. Por ser complicada, é difícil para outros programadores entenderem e observar como a classe deve ser usada.
3. Uma agregação de diferentes abstrações e (mal) uso de outras classes (frequentemente mero portador de dados) para executar sua funcionalidade. Ou seja, uma classe única com um grande número de atributos, operações ou ambos.
4. Têm a tendência de centralizar a inteligência do sistema. O principal problema aqui é que a maioria das responsabilidades é atribuída a uma única classe.
5. A classe é muito complexa e não coesa. Uma coleção discrepante de atributos e operações não relacionados encapsulados em uma única classe. Uma falta geral de coesão dos atributos e operações.

As diferenças citadas pelos autores como suas principais diferenças que o separam de serem classificados como outro smell. Podemos destacar as seguintes características:

1. Uns apresenta com maior destaque o acesso a dados de outras classes que o outro smell.
2. Níveis de coesão mais baixo em relação a outro smell.
3. Apresentou uma falta de arquitetura orientada a objetos maior que outro smell.

Visto que a maioria das características que foi apresentada pelos autores são iguais para descrever o mesmo smell (veja a Tabela 1), levando a nos confundir facilmente na hora de classificar ou de dizer o nome de um determinado smell. Além disso, conforme vemos na Tabela 2. Suas semelhanças não contribuem fortemente para uma separação dos smells, visto ser características parecidas que podem influenciar em uma determinada classificação, mas não necessariamente uma característica marcante para

defini-la. Mas quando vemos a Tabela 3. Percebemos a real razão da separação de cada classificação, porém algumas características são um pouco subjetivas, por exemplo ambas apresentam coesão baixa, mas God Class apresenta uma coesão mais baixa que as outras. Outra característica seria acessar diretamente muitos atributos de outras classes, o qual todos têm essas características, porém o Brain Class apresenta pouco essa característica. E finalmente a outra característica a ser destacada como uma diferença entre os smells é a falta de uma arquitetura orientada a objetos o qual é mais destacada no smells The Blob, muitas vezes explicada pelo código ter sua origem de uma programação procedural. Entretanto, a falta de encapsulamento, e um objeto tendo a maior parte das responsabilidades já o torna contra os princípios da programação orientada a objetos. Por isso ficaria uma questão se é realmente necessário ter tantos termos para separar smells que apresenta tantas igualdades como semelhanças separadas por poucos detalhes.

CONSIDERAÇÕES FINAIS

Após executamos as etapas para a pesquisa do conteúdo, informações de como proceder para fazer a discussão dos smells com muitas características parecidas, seleção dos smells mais estudados, leitura das características, seleção das características. Foi feita a discussão de igualdades, semelhanças e diferenças. E os resultados apontaram para poucas características que podem ser consideradas distintas e essas características podem ser unificadas como mostrado na parte final da sessão RESULTADOS E DISCUSSÃO, como: coesão baixa, falta de encapsulamento e um objeto tendo a maior parte das responsabilidades resultando em falta de aplicar os princípios de programação orientada a objetos. Tal tema é pouco explorado em Engenharia de Software, o que pode direcionar estudos para caminhos inadequados.

REFERÊNCIAS

Brown, W.H., Malveau, R.C., McCormick, H.W.S., Mowbray, T.J., 1998. AntiPatterns: Refactoring

Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., Gamma, E., 1999. Refactoring: improving the design of existing code. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Lanza, M., Marinescu, R., 2005. Object-Oriented Metrics in Practice. SpringerVerlag New York, Inc., Secaucus, NJ, USA.

Riel, A.J., Object−Oriented Design Heuristics, Reading, MA: Addison−Wesley, 1996.

Software, Architectures, and Projects in Crisis. 1st ed., John Wiley & Sons, Inc., USA.

Wake, W. C., Venables, R., Fuller, J., 2003. Refactoring Workbook. Addison-Wesley Professional.

Akroyd, Michael, “AntiPatterns Session Notes,” Object World West, San Francisco, 1996.

N. Dale, J. Lewis, Computer Science Illuminated, Jones and Bartlett Publishers, Inc., USA, 2016.

E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements
of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing Co., Inc., USA, 1995.

J. A. M. Santos, J. B. Rocha-Junior, L. C. L. Prates, R. S. do Nascimento, M. F. Freitas, M. G. de Mendon_ca, A systematic review on the
code smell e_ect, Journal of Systems and Software 144 (2018) 450-477.