Técnicas de Inspeção para Diagramas de Classes UML: Uma Revisão Sistemática

Débora dos Santos e Siqueira[1], **Matheus Montanha Paulon**[1], **Gilleanes Thorwald Araujo Guedes**[1]

[1]Universidade Federal do Pampa - Campus Alegrete (UNIPAMPA)
Av. Tiaraju, 810 - Ibirapuitã, Alegrete – RS – Brasil

{deborasiq99, matheusmontanhapaulon, gtaguedes}@gmail.com

Abstract. *In this paper we analyse the inspection techniques for UML class diagrams - a modeling language for use in software projects. This study aims to investigate the state-of-the-art of the inspection techniques that verify class diagrams, establishing how these diagrams are usually inspected, what types of errors and inconsistencies it seeks to discover, and the cost benefit of its application. To achieve this goal, we performed a systematic literature review, retrieving all the relevant articles and presenting each of the techniques discovered, highlighting their strengths and weaknesses and their possible gaps.*

Resumo. *Neste artigo analisamos as técnicas de inspeção para diagramas de classes UML - uma linguagem de modelagem para uso em projetos de software. Este estudo tem como objetivo investigar o estado da arte das técnicas de inspeção que verificam diagramas de classes, estabelecendo como estes diagramas costumam ser inspecionados, que tipos de erros e inconsistências procura-se descobrir e qual o custo-benefício de sua aplicação. Para atingir este objetivo, realizamos uma revisão sistemática da literatura, recuperando todos os artigos considerados relevantes e apresentando cada uma das técnicas descobertas, destacando seus pontos fortes e fracos e suas possíveis lacunas.*

1. Introdução

O desenvolvimento de software é um processo amplo, que não apenas se preocupa em criar um produto para atingir uma meta para a qual ele foi desenvolvido, mas também em garantir a criação de produtos com cada vez mais qualidade. Uma atividade importante no desenvolvimento do software como um todo é a verificação e validação, que visa analisar os artefatos (como por exemplo, modelos UML) produzidos durante o ciclo de desenvolvimento. Uma forma de verificar estes artefatos, é por meio de inspeções a fim de auxiliar na garantia da qualidade [Fagan 1976].

As etapas de engenharia de requisitos e de projeto produzem muitos artefatos, tais como modelos UML - uma linguagem de modelagem conhecida e usada internacionalmente no desenvolvimento de software. O objetivo desta linguagem é permitir a criação de modelos gráficos que auxiliem engenheiros de software a entender o produto que será desenvolvido. Um dos modelos mais utilizados da UML é o diagrama de classes, que proporciona uma visão de como as classes estarão organizadas, como elas se relacionam e quais seus atributos e métodos [Guedes 2018].

O diagrama de classes serve como base para a criação de diversos outros modelos UML. Considerando essa dependência, se os modelos de classe forem criados com defeitos ou inconsistências, isso afetará outros diagramas e, por conseguinte, o próprio código. Portanto, para garantir que os diagramas de classe apresentem uma qualidade de alto nível, é necessário a utilização de técnicas que inspecionem esses diagramas com o intuito de encontrar possíveis erros e anomalias. Sendo assim, existem técnicas de inspeção que visam verificar os defeitos e inconsistências contidos nos diagramas de classe, bem como ajudar a reduzir custos, uma vez que os possíveis erros que passariam para os próximos passos seriam identificados e corrigidos nos estágios iniciais do desenvolvimento [Myers et al. 2011].

A fim de estabelecer o estado-da-arte dessas técnicas de inspeção, foi realizada uma revisão sistemática da literatura, reunindo os trabalhos considerados relevantes para alcançar o objetivo deste artigo. Nesta revisão, analisamos quais técnicas inspecionam modelos de classes e se os trabalhos apresentam o custo-benefício da sua aplicação. Este trabalho está estruturado da seguinte forma: na seção 2 é apresentada a fundamentação teórica; na seção 3 o planejamento desta revisão sistemática; na seção 4 os resultados obtidos e por fim na seção 5 a conclusão e os trabalhos futuros deste estudo.

2. Fundamentação Teórica

2.1. Inspeção

Fagan, sendo um dos primeiros a estabelecer o conceito de inspeção em [Fagan 1976], declara que inspeções melhoram a produtividade e a qualidade do produto, além de tornar a descoberta de erros muito mais barata do que os custos referentes à manutenção. Já [Felizardo 2004] menciona que inspeção de software é um método que tem sido utilizado na Engenharia de Software para obter mais eficiência e eficácia em produtos de software.

2.2. UML

UML (Unified Modeling Language) é uma linguagem para modelagem de software que pode ser usada na engenharia de requisitos e no projeto de software e foi concebida para auxiliar engenheiros de software a definir características do sistema, como seus requisitos, estrutura lógica, o comportamento de seus processos etc. Essa linguagem fornece vários tipos de diagramas. Cada diagrama tem um objetivo diferente e é composto por vários elementos gráficos relacionados [Guedes 2018].

3. Protocolo da Revisão Sistemática

Este estudo foi realizado por meio de uma revisão sistemática. Para [Kitchenham 2004] caracteriza-se pela identificação, avaliação e interpretação de um conjunto de trabalhos relevantes para uma questão de pesquisa, a área temática ou o mecanismo de pesquisa para a confirmação de uma certa conclusão.

Para realizá-la, na primeira etapa do processo, foi criado o planejamento do estudo. Esse planejamento incluiu o objetivo da revisão, critérios de inclusão e exclusão de trabalhos, questões de pesquisa e qualidade, bem como a string de pesquisa para a seleção de trabalhos. Nas subseções a seguir, descrevemos os componentes do protocolo de pesquisa desenvolvidos para permitir a replicação desta revisão no futuro.

3.1. Questões de Pesquisa

O planejamento desta pesquisa está relacionado aos trabalhos que apresentam técnicas de inspeção para diagramas de classes UML. Foram identificadas três questões de pesquisa: (Q1) Quais as técnicas de inspeção existentes que verificam diagramas de classes UML? (Q2) Que tipos de erros e inconsistências essas técnicas buscam encontrar? e (Q3) As técnicas apresentam qual o custo-benefício de sua utilização?

3.2. Processo de Busca

Para encontrar os trabalhos, realizamos um processo de busca em três base de dados digitais (IEEE Xplore, ACM Digital Library e Scopus). Além disso, utilizamos a técnica de backward snowballing [Wohlin 2014], ou seja, verificamos as citações nos artigos selecionados para tentar identificar mais trabalhos relevantes. Como as obras foram pesquisadas em três bases de dados diferentes, uma string genérica foi criada. Contudo, é importante ressaltar que, para cada uma das bases de dados pesquisadas, a string sofreu modificações para que assim atenda as particularidades de cada mecanismo de busca dessas bases. A string genérica é descrita a seguir: (Techniques OR Technics OR method AND Inspection OR Verification OR Analysis OR survey OR Checking OR Consistency OR Inspections OR Reading AND Class Models OR Class Model OR Class Diagrams AND Unified Modeling Language OR UML).

3.3. Critérios de Inclusão e Exclusão

A fim de obter trabalhos relevantes para a revisão sistemática, os seguintes critérios de inclusão e exclusão foram estabelecidos no início do processo, como mostra a Tabela 1.

Tabela 1. Critérios de inclusão e exclusão

Tipo	Critério
Inclusão	Descrever uma ou mais técnicas de inspeção para diagramas de classes UML.
Inclusão	O estudo deve ter sido escrito em inglês ou português.
Exclusão	Contenham apenas listas de inspeções sem descrever como as técnicas funcionam.
Exclusão	Trabalhos que não abordam inspeção em Diagrama de Classes.
Exclusão	Trabalhos que não sejam completos (resumos).
Exclusão	Trabalhos com menos de 5 páginas.
Exclusão	Trabalhos repetidos (trabalhos não originais).

4. Resultados

A Figura 1 ilustra as etapas do processo de pesquisa, onde cada base de dados retornou um número específico de estudos. A Scopus retornou 2117 artigos, para isso foram selecionadas três subáreas, sendo elas: ciência da computação, matemática e engenharia. Já nas demais bases não foi utilizada nenhum filtro, retornando 101 estudos na ACM Digital Library e 117 na IEEE Xplore. Vale ressaltar que diversos trabalhos encontrados na base de dados Scopus tem origem de outras bases, como a IEEE, ACM, Science Direct, Springer Link, o que justifica o alto número de trabalhos retornados.

Pode-se perceber que houve um retorno de vários artigos em cada base de dados. Destes, apenas 31 foram selecionados e, após a aplicação dos critérios de exclusão e

inclusão, o número de estudos selecionados foi reduzido, restando apenas 19, que foram lidos e analisados. Logo após, foi aplicada a técnica de backward snowballing nos artigos incluídos, onde foram adicionados mais 5 estudos.

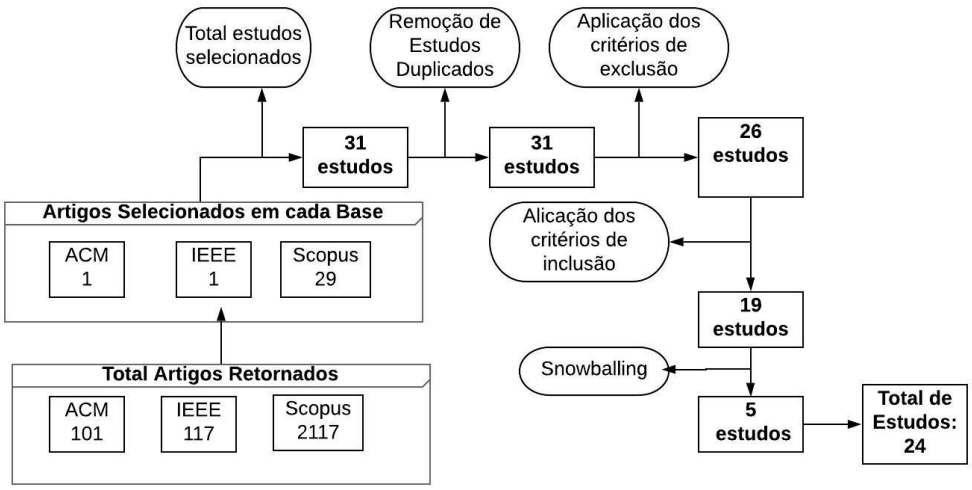

Figura 1. Ilustração do processo de seleção dos trabalhos.

É importante ressaltar que identificamos uma revisão sistemática com o objetivo próximo ao deste trabalho. O trabalho de [Mumtaz et al. 2019] concentra-se em identificar técnicas que inspecionem diagramas UML em busca de maus cheiros. A nossa pesquisa, por outro lado, visa identificar os tipos de erros e inconsistências que as técnicas propostas buscam encontrar em diagramas de classes especificamente e se os estudos apresentam qual o custo-benefício de sua utilização.

4.1. Respondendo as Questões de Pesquisa

Após o processo de extração de dados dos trabalhos, identificou-se qual técnica era apresentada em cada um deles. Desse modo, no que diz respeito à primeira questão de pesquisa **"Quais as técnicas de inspeção existentes que verificam diagramas de classes UML?"**, foram identificadas as seguintes técnicas, como mostra a Tabela 2 abaixo.

Tabela 2. Técnicas de inspeção identificadas.

Referência	Técnica
[George and Samuel 2018] [Ekanayake and Kodituwakku 2015]	XMI Parser/Mathematical.
[Lopes et al. 2018]	Comunication-based.
[Cheng and Zhang 2011]	Natural Language Processin.
[Shaikh et al. 2010]	Slicing Technique/Formalization.
[Sabaliauskaite et al. 2003]	Checklist(CBR)/Pespective(PBR).

[Clariso et al. 2019] [Sulaiman et al. 2019] [Hilken and Gogolla 2016] [Khan. et al. 2013] [Chavez and Shen 2009] [Evans] [Wang and Shen 2007] [Szlenk 2006] [Kaneiwa and Satoh 2010] [Cheng and Zhang 2011]	Mathematical Formalization.
[Neto et al. 2019]	Model Scoping/Expected Model Elements.
[Bettin et al. 2018]	Checklist(CBR).
[Cabot et al. 2014]	Constraint Satisfaction Problem (CSP)/OCL constraints.
[Travassos et al. 1999]	Defects(DBR)/Pespective(PBR)/Use(UBR).
[Geraldi 2017]	Checklist(CBR).
[Travassos et al. 2002]	OORTs.
[Shaikh et al. 2018]	Slicing Technique/Feedback Technique.

A abordagem mais utilizada é a de formalismo matemático, seguida de transformações em XMI e Checklist, entre as demais identificadas. O formalismo tem como conceito, segundo [Mondini 2008][Mondini 2008], provar que uma determinada ideia está isenta de contradições, sendo este o motivo pelo qual esta abordagem vem sendo mais utilizada pelos pesquisadores. Esta é uma atividade com custo elevado se aplicada em apenas um projeto, contudo se utilizada em diversos projetos o custo é compensador a longo prazo [Bibi et al. 2014].

As abordagens baseadas em transformações em linguagem XMI, utilizam ferramentas que extraem os atributos e métodos do modelo, buscando por inconsistências ou erros na estrutura. Vale ressaltar que as ferramentas dão suporte a inspeção a outros modelos proporcionados pela UML não se limitando apenas a diagramas de classe.

A terceira abordagem mais utilizada, aplica inspeções baseadas em checklist. Esta é uma técnica que verifica diversos aspectos em um diagrama UML. Nesta técnica, um indivíduo (inspetor) realiza a inspeção com base nos aspectos anteriormente escolhidos. Esta abordagem é uma das formas mais acessíveis, pois o indivíduo que realiza a inspeção, não necessariamente precisa ter uma base grande de experiência, segundo [Sabaliauskaite et al. 2003].

As demais abordagens identificadas, não menos importantes, apresentaram suas peculiaridades. Porém, há diversos aspectos que podem ser melhorados, como na questão do custo-benefício em que nenhum trabalho apresentou dados concretos visando esse aspecto tão importante. Percebe-se assim a necessidade de aplicar essas técnicas em cenários reais, como o da indústria, de maneira a determinar se o custo de sua aplicação é economicamente viável, bem como verificar se as técnicas realmente funcionam.

Em relação à segunda questão de pesquisa **"Que tipos de erros e inconsistências essas técnicas buscam encontrar?"**, nota-se que as técnicas propostas nos estudos de [Lopes et al. 2018], [Sabaliauskaite et al. 2003], [Geraldi 2017], [Travassos et al. 1999] e [Travassos et al. 2002] visam identificar os mesmos defeitos, que consistem em: omissão,

inconsistência, ambiguidade, fato incorreto e informação estranha. Já os demais trabalhos não deixam explícito o tipo de erro que buscam encontrar.

No que tange à terceira questão de pesquisa **"As técnicas apresentam o custo-benefício de sua utilização?"**, apenas dois dos trabalhos selecionados abordam parcialmente a questão do custo benefício: [George and Samuel 2018] e [Sabaliauskaite et al. 2003]. Porém esses trabalhos apenas sugerem um possível custo-benefício, sem demonstrar dados concretos. Em [Sabaliauskaite et al. 2003] é apresentado um experimento realizado com estudantes em que foram divididas técnicas de inspeções entre eles, de forma a obter dados sobre os custos de tempo que cada técnica exigia. Já [George and Samuel 2018], só sugerem que a técnica proposta pode reduzir o custo do desenvolvimento e manutenção de software, mas não apresentam comprovação.

5. Conclusão

Um aspecto importante do desenvolvimento de software sempre foi a qualidade. Isto implica, além de cumprir os requisitos estabelecidos, garantir a excelência da qualidade do produto. Este aspecto permanece sob constante estudo, de modo a obter as técnicas e medidas que possam ajudar a descobrir e prevenir os defeitos de um software. Dentro deste contexto, considerando a ampla difusão da UML e seu impacto no desenvolvimento de software, é claramente necessário inspecionar os modelos produzidos por esta linguagem para evitar que possíveis erros e defeitos possam ser repassados para os desenvolvedores, acarretando altos custos para o projeto.

Como apresentado anteriormente, as técnicas aplicadas nos trabalhos selecionados mostraram uma preferência em utilizar formalismo matemático para realizar as inspeções, procurando obter maior precisão em seus resultados. Entretanto, a falta de profissionais capacitados nessa área é um desafio para o uso desta abordagem, considerando tanto o tempo necessário para sua aplicação quanto o custo que gera.

Além disso, outra aspecto é que nenhum dos estudos abordou a questão do custo-beneficio ao aplicar essas técnicas. Pode-se dizer que essa lacuna é a maior fraqueza que essa pesquisa encontrou, pois é necessário aplicar essas técnicas em ambientes reais para demonstrar que elas são economicamente viáveis e que realmente funcionam. Também com a aplicação das técnicas em projetos reais, pode-se verificar outros possíveis defeitos que podem ser usados para realizar melhorias das mesmas.

Apesar de alguns aspectos que foram encontrados como possíveis lacunas, o uso de técnicas de inspeção em diagramas de classe deve ser uma decisão recorrente quando se planeja desenvolver software para garantir qualidade. Além disso, é preciso avaliar as técnicas em situações reais para determinar sua factibilidade para eventualmente virem a ser adotadas pela indústria.

Considerando as lacunas encontradas durante esta revisão, pretendemos, como trabalho futuro, propor uma nova técnica. Particularmente, estamos interessados em produzir uma técnica fortemente focada na prevenção de maus cheiros de código em modelos UML, bem como na verificação das consistências entre os vários modelos de UML que compõem projetos de sistemas. Pretendemos aplicar essa nova técnica em ambientes industriais reais para determinar o custo-benefício e a viabilidade de sua aplicação, que é um aspecto pouco coberto pelos trabalhos anteriores.

Referências

Bettin, G. C. S., Geraldi, R. T., and OliveiraJr, E. (2018). Experimental evaluation of the SMartyCheck techinique for inspecting defects in UML component diagrams. In *Proceedings of the 17th Brazilian Symposium on Software Quality - SBQS*. ACM Press.

Bibi, S., Mazhar, S., Minhas, N. M., and Ahmed, I. (2014). Formal methods for commercial applications issues vs. solutions. *Journal of Software Engineering and Applications*, 07(08):679–685.

Cabot, J., Clarisó, R., and Riera, D. (2014). On the verification of UML/OCL class diagrams using constraint programming. *Journal of Systems and Software*, 93:1–23.

Chavez, H. M. and Shen, W. (2009). Finding inconsistency for UML-based composition at program level. In *2009 ICSE Workshop on Modeling in Software Engineering*. IEEE.

Cheng, L. and Zhang, Y. (2011). Model checking security policy model using both UML static and dynamic diagrams. In *Proceedings of the 4th international conference on Security of information and networks - SIN 11*. ACM Press.

Clariso, R., Gonzalez, C. A., and Cabot, J. (2019). Smart bound selection for the verification of UML/OCL class diagrams. *IEEE Transactions on Software Engineering*, 45(4):412–426.

Ekanayake, E. M. N. K. and Kodituwakku, S. R. (2015). Consistency checking of UML class and sequence diagrams. In *2015 8th International Conference on Ubi-Media Computing (UMEDIA)*. IEEE.

Evans, A. Reasoning with UML class diagrams. In *Proceedings. 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*. IEEE Comput. Soc.

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211.

Felizardo, K. R. (2004). Apoio computacional para inspeção de software. *INFOCOMP*, 3(2):14–18.

George, R. and Samuel, P. (2018). Fixing class design inconsistencies using self regulating particle swarm optimization. *Information and Software Technology*, 99:81–92.

Geraldi, R. T. (2017). Towards initial evidence of SMartyCheck for defect detection on product-line use case and class diagrams. *Journal of Software*, 12(5):379–392.

Guedes, G. T. A. (2018). *UML 2 - Uma Abordagem Prática*. Novatec Editora.

Hilken, F. and Gogolla, M. (2016). Verifying linear temporal logic properties in UML/OCL class diagrams using filmstripping. In *2016 Euromicro Conference on Digital System Design (DSD)*. IEEE.

Kaneiwa, K. and Satoh, K. (2010). On the complexities of consistency checking for restricted UML class diagrams. *Theoretical Computer Science*, 411(2):301–323.

Khan., A. H., Rauf., I., and Porres., I. (2013). Consistency of uml class and statechart diagrams with state invariants. In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, pages 14–24. INSTICC, SciTePress.

Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26.

Lopes, A., Campos, U., Conte, T., and de Souza, C. (2018). ComD2: Family of techniques for inspecting defects in models that affect team communication. In *Proceedings of the 30th International Conference on Software Engineering and Knowledge Engineering*. KSI Research Inc. and Knowledge Systems Institute Graduate School.

Mondini, F. (2008). O logicismo, o formalismo e o intuicionismo e seus diferentes modos de pensar a matemática. *EBRAPEM, 12*, pages 1–10.

Mumtaz, H., Alshayeb, M., Mahmood, S., and Niazi, M. (2019). A survey on UML model smells detection techniques for software refactoring. *Journal of Software: Evolution and Process*, 31(3):e2154.

Myers, G. J., Sandler, C., and Badgett, T. (2011). *The Art of Software Testing*. Wiley.

Neto, C., Neto, A., Kalinowski, M., de Oliveira, D. M., Sabou, M., Winkler, D., and Biffl, S. (2019). Using model scoping with expected model elements to support software model inspections: Results of a controlled experiment. In *Proceedings of the 21st International Conference on Enterprise Information Systems*. SCITEPRESS - Science and Technology Publications.

Sabaliauskaite, G., Matsukawa, F., Kusumoto, S., and Inoue, K. (2003). Further investigations of reading techniques for object-oriented design inspection. *Information and Software Technology*, 45(9):571–585.

Shaikh, A., Clarisó, R., Wiil, U. K., and Memon, N. (2010). Verification-driven slicing of UML/OCL models. In *Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE 10*. ACM Press.

Shaikh, A., Mahoto, N. A., Saddar, S., and Shaikh, M. (2018). A technique for detection of violating property among UML/OCL class diagram. In *2018 5th International Multi-Topic ICT Conference (IMTIC)*. IEEE.

Sulaiman, N., Ahmad, S. S. S., and Ahmad, S. (2019). Logical approach: Consistency rules between activity diagram and class diagram. *International Journal on Advanced Science, Engineering and Information Technology*, 9(2):552.

Szlenk, M. (2006). Formal semantics and reasoning about UML class diagram. In *2006 International Conference on Dependability of Computer Systems*. IEEE.

Travassos, G., Shull, F., Fredericks, M., and Basili, V. R. (1999). Detecting defects in object-oriented designs. In *Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications - OOPSLA 99*. ACM Press.

Travassos, G. H., Shull, F., Carver, J. C., and Basili, V. R. (2002). Reading techniques for oo design inspections.

Wang, K. and Shen, W. (2007). Runtime checking of UML association-related constraints. In *Fifth International Workshop on Dynamic Analysis (WODA 07)*. IEEE.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE 14*. ACM Press.