UNIVERSIDADE FEEVALE

MATHEUS FELIPE BALZ BASTIAN

GERADOR DE ARQUIVOS E CÓDIGO FONTE ABAP A PARTIR DE DIAGRAMA DE CLASSES UML DESENVOLVIDO EM PLANTUML

(Título Provisório)

Anteprojeto de Trabalho de Conclusão

Novo Hamburgo
2022

MATHEUS FELIPE BALZ BASTIAN

GERADOR DE ARQUIVOS E CÓDIGO FONTE ABAP A PARTIR DE DIAGRAMA DE CLASSES UML DESENVOLVIDO EM PLANTUML

(Título Provisório)

Anteprojeto de Trabalho de Conclusão de Curso, apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale

Orientador: Adriana Neves dos Reis

Novo Hamburgo
2022

RESUMO

Descrever a arquitetura de um código orientado a objetos de maneira clara e padronizada se torna cada vez mais necessário na área de desenvolvimento de software. Equipes de trabalho maiores e espalhadas pelo mundo, muitas vezes de forma remota devido a constantes mudanças na forma de trabalho, fazem com que o código, antes de ser escrito, tenha de ser bem elaborado e pensado previamente, o que justifica a padronização na modelação de diagramas que expressam os requisitos. Daí surge, por exemplo, a linguagem UML (*Unified Modeling Language*) para criação de diagramas de classe, sequência, atividade entre outros, que facilitam a descrição de uma arquitetura para os desenvolvedores que irão implementá-la. Uma vez criados os diagramas, o próximo passo é a criação dos arquivos fontes e código a ser escrito. Este passo, inicialmente, consiste na mera declaração de classes, interfaces e enumeradores contendo seus atributos e métodos, sem conter comportamento que implemente os requisitos. Criar arquivos de código fonte em uma plataforma *SAP NetWeaver* (plataforma que serve como fundação para diversas aplicações), assim como escrever o código dos objetos, demanda tempo e esta tarefa pode ser atrasada por eventuais instabilidades do servidor ou conexão com ele. Sendo assim, este trabalho tem como objetivo a criação de uma ferramenta automatizada que gera o código fonte em ABAP (linguagem proprietária de alto nível da SAP utilizada na plataforma NetWeaver) contendo a declaração dos objetos e as classes de teste unitário a partir de um diagrama de classes no padrão UML. Sendo que a ferramenta utilizada para criação dos diagramas será a PlantUML, espera-se contribuir para a área de desenvolvimento de software com uma nova forma de gerar código fonte tendo como entrada um diagrama de classes: a partir da conversão direta de código PlantUML para ABAP ao invés do usual padrão XMI, uma vez que este não é suportado pela ferramenta, e investigar sua efetividade em cumprir com o objetivo de reduzir o tempo gasto com a criação dos arquivos fonte e com a escrita do código destes objetos.

Palavras-chave: UML. Arquitetura. PlantUML. Geração de código. ABAP.

SUMÁRIO

MOTIVAÇÃO

O tema da geração automática de código a partir de modelos já é bem conhecido na comunidade acadêmica. Seu principal objetivo é a diminuição de tempo gasto em tarefas que podem ser concluídas com auxílio parcial ou total de uma ferramenta CASE (*Computer-aided Software Engineering*). Porém, para chegar na geração de código a partir de um diagrama, por exemplo, é preciso entender por que eles são feitos, por qual motivo se investe tempo no desenvolvimento de novas ferramentas que facilitam a geração de diagramas de classe, sequência, atividade etc. Depois, observar quais os padrões de modelagem de diagramas mais difundidos e as ferramentas mais utilizadas para geração destes para que, ao final, seja possível demonstrar o valor e a contribuição deste projeto.

A busca por qualidade tem como objetivo cumprir com os requisitos do software sendo desenvolvido ao mesmo tempo que se mantém a legibilidade, clareza, manutenibilidade do código e sua capacidade de ser facilmente estendido para cumprir com novos requisitos de forma rápida e ágil, sem gerar novo *bugs*. Obtém-se qualidade de software a partir da adoção de boas práticas de desenvolvimento de software, como por exemplo as que estão expostas no aclamado livro de Martin (2009): Clean code: A Handbook of Agile Software Craftsmanship. Neste livro, as técnicas e estratégias demonstradas buscam produzir os efeitos mencionados anteriormente, como legibilidade, clareza, manutenibilidade do código e capacidade de extensão para cumprir com novos requisitos. O motivo pelo qual se escolhe produzir código com mais qualidade é devido aos resultados que advém do esforço empregado. Um código que possui boa legibilidade, nomes de variáveis significativos e objetivos, classes e interfaces menores e com responsabilidades únicas e coesas além nomes de métodos claros traz maior manutenibilidade, reduz o número de *bugs* e torna mais fácil a implementação de novos requisitos, como afirmam Fowler (2022), notório autor da área, e estudos como o de Koller (2016).

Sabendo disso, a implementação de um pequeno requisito pode se tornar mais trabalhosa e demorada já que a quantidade de objetos criados será maior, devido a aplicação das boas práticas de desenvolvimento. Portanto, o detalhamento na descrição dos objetos antes da escrita do código consequentemente fica maior, e isso exige uma maneira de expor essa descrição a partir de um modelo com linguagem bem definida e padronizada. Na área de desenvolvimento de software, uma das linguagens mais conhecidas para esse propósito é a UML (*Unified Modeling Language*).

UML é uma linguagem de modelagem que permite representar um sistema de forma padronizada. É utilizada para a modelagem de sistemas, sendo uma linguagem muito expressiva, abrangendo as visões necessárias ao desenvolvimento e implantação desses sistemas (BOOCH et al., 2006). Como resultado, podem ser criados vários diagramas que têm objetivos diferentes. Para descrição estrutural de um conjunto de objetos, o diagrama a ser produzido deve ser o de classes, pois contém as informações cruciais como nome do objeto, nomes e assinaturas de métodos, bem como sua visibilidade, além de relações entre classes como herança e implementação, no caso de interfaces. Já para descrever comportamento, e principalmente o fluxo de chamadas entre objetos, o diagrama a ser produzido deve ser o de sequência. Caso exista um diagrama de classes, os nomes das classes e nomes de métodos devem estar consistentes com o diagrama de sequência.

Existem diversos websites e ferramentas que proveem uma interface gráfica para geração destes diagramas seguindo o padrão UML como *draw.io*, *lucidchart*, *miro*, os quais permitem também a exportação dos diagramas em formatos como PDF, JPG etc. Porém, existe uma ferramenta que utiliza uma descrição de texto simples e legível chamada PlantUML, que é usada para desenhar diagramas UML (https://plantuml.com/faq#4228ce659e6db48d). A principal diferença do PlantUML para as outras ferramentas é o fato de ser 100% texto. É possível fazer revisões dos diagramas, armazená-los em um repositório *git* e tratá-los como qualquer outro código fonte, além de também permitir a exportação para muitos formatos.

Ao trabalhar com a linguagem ABAP, os desenvolvedores devem ser conectar a um servidor de desenvolvimento para poder editar os arquivos contendo o código fonte. Conforme explicado anteriormente, seguir as boas práticas de desenvolvimento pontuadas em livros como o Clean Code leva à criação de mais classes, interfaces e enumeradores, que devem ser escritas em arquivos de código fonte separados, para evitar problemas de bloqueio de edição, uma vez que não é possível a edição concomitante de um mesmo arquivo fonte ABAP. A criação destes arquivos em no servidor não é trivial, pois deve ser informado o nome do arquivo, o pacote onde será salvo, o seu título e uma descrição, com este processo envolvendo múltiplas telas de diálogo. Imaginando um cenário em que um arquiteto de software ou desenvolvedor já criou o diagrama de classes necessário para a correta implementação de um requisito, é necessário então criar um arquivo para cada objeto que se encontra no diagrama de classes e escrever o código que define os atributos e métodos com suas corretas visibilidades em cada arquivo. Isso demanda tempo que pode ser poupado com o

uso de uma ferramenta que gera tanto o código desses objetos como já os separa em arquivos já que, até este momento, nenhuma linha de código interfere em como o requisito será implementado, pois essa responsabilidade é dos desenvolvedores. Assim, conforme Herrington (2003, p. 28), se trata de uma ferramenta geradora de código passiva. Além disso, a ferramenta pode também executar os mesmos passos e criar classes de testes unitários, o que pode acelerar o processo de desenvolvimento independentemente do fato de o desenvolvedor escolher fazer os testes antes ou depois de escrever o código das classes. Ferramenta similar foi desenvolvida por Tannús e Neto (2021), porém gerando código na linguagem C#, e por Pando e Castillo (2022) gerando código para a linguagem PHP.

O uso desta ferramenta pode diminuir o tempo necessário para o início do desenvolvimento de um novo requisito em times de desenvolvimento que trabalham com ABAP, possibilitando a entrega mais rápida e até mesmo testes manuais com mais duração e mais minuciosos por parte dos colegas envolvidos. De acordo com os estudos de Necco, Tsai e Chang (1994), Limayem, Khalifa e Chin (2004) e Morales et al. (2015), os desenvolvedores que utilizam ferramentas CASE para adiantar o trabalho ficam livres para assumir outras tarefas, tornando o time mais eficiente e ágil, sem perder a qualidade do código, garantindo boa manutenibilidade no futuro. Assim, este trabalho busca responder a seguinte questão de pesquisa: Como uma ferramenta CASE pode auxiliar no desenvolvimento de requisitos e quais as vantagens e desvantagens do seu uso em um time ágil.

OBJETIVOS

O objetivo geral deste trabalho é construir e experimentar uma ferramenta para geração de código e arquivos ABAP a partir de diagramas elaborados em PlantUML, de modo a identificar os fatores que podem oferecer mais velocidade no desenvolvimento de requisitos por uma equipe ágil.

Os objetivos específicos são:

1. Construir uma dinâmica de uso para a ferramenta.
2. Elaborar um processo que aumente os benefícios da ferramenta e que possa ser integrado a times ágeis.
3. Analisar as vantagens e desvantagens do uso de uma ferramenta CASE logo após a criação dos diagramas de classe em comparação com a forma manual.
4. Analisar se as vantagens e/ou desvantagens são as mesmas indiferente do tamanho do requisito a ser implementado, ou se observa-se uma curva de ganho.
5. Descobrir se a ferramenta precisa de complementação, em se provando não ser suficiente para reduzir o tempo necessário para o desenvolvimento.

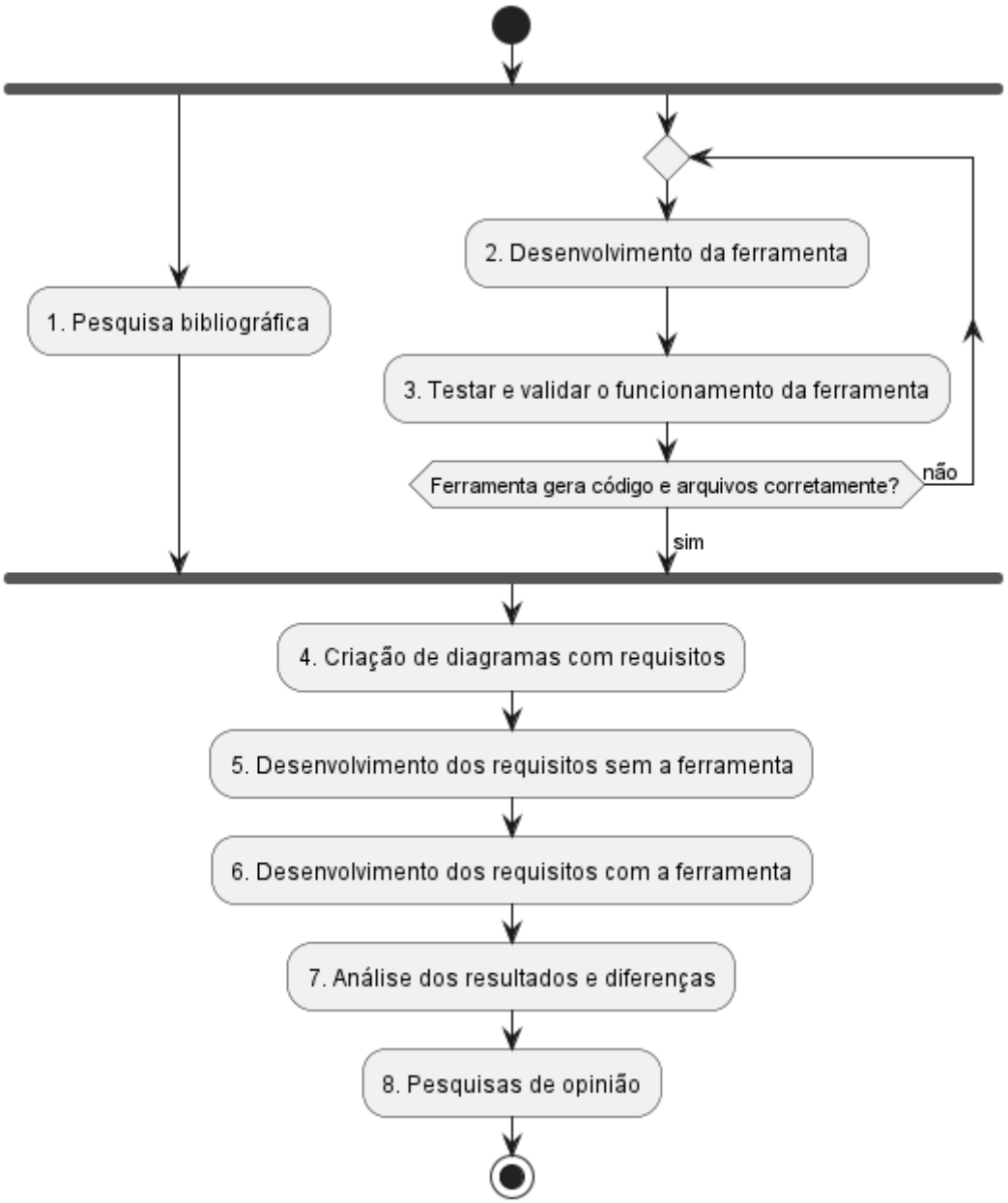
METODOLOGIA

O estudo é classificado como pesquisa aplicada, pois gerará conhecimento a ser aplicado em um problema definido, originando um processo ao final (PRODANOV; FREITAS, 2013). Quanto aos objetivos, tem caráter exploratório, buscando evidências sobre a questão de pesquisa durante o estudo. Para isso, primeiramente será feita uma pesquisa bibliográfica, de modo a encontrar trabalhos semelhantes e onde foram aplicados, quais resultados surtiram, como a ferramenta CASE contribuiu para o time de desenvolvimento envolvido e quais são as vantagens e desvantagens.

Para cumprir os objetivos propostos, será então desenvolvida uma ferramenta CASE. A maneira de avaliação destes objetivos compreenderá uma pesquisa exploratória, sendo feitos testes com a ferramenta e simulações de um início de desenvolvimento de um requisito sem o uso desta, para poder comparar as duas alternativas e concluir se há vantagem ou não, levando em consideração diversos cenários como diagramas de classes com números altos ou baixos de classes. Ou seja, a variável que representa o número de classes no diagrama será manipulada de modo a observar como ela afeta os resultados.

Será feita também uma pesquisa de campo com os desenvolvedores que testarão a ferramenta produzida neste trabalho, para responder as perguntas que foram delineadas no capítulo dos objetivos específicos, a partir de formulários na internet. Na Figura 1 são apresentados os passos para condução da pesquisa.

Figura 1 - Passos da pesquisa



Fonte: elaborada pelo autor

1. Pesquisa bibliográfica para compreender como ferramentas CASE contribuíram para o desenvolvimento de requisitos.
2. Desenvolver a ferramenta CASE geradora de código ABAP a partir de diagrama de classes UML.
3. Testar e validar o funcionamento da ferramenta.
4. Desenhar três diagramas de classes com variações no número de objetos.
5. Criar manualmente os arquivos fonte e escrever o código para implementação dos objetos e medir o tempo que foi gasto.

6. Utilizar a ferramenta CASE usando os diagramas de classes criados e medir o tempo que foi gasto.
7. Medir a diferença de tempo entre a criação manual dos objetos e a utilização da ferramenta CASE, considerando os três diagramas e analisar as diferenças, vantagens e desvantagens entre os dois experimentos.
8. Fazer pesquisa de opinião entre desenvolvedores ABAP que utilizaram a ferramenta para entender as principais vantagens e desvantagens e, dadas possíveis desvantagens, se é necessário complementar a ferramenta para suportar outros diagramas.

CRONOGRAMA

Trabalho de Conclusão I

Etapa	Meses			
	Ago	Set	Out	Nov
Escrita do anteprojeto				
Revisão do anteprojeto				
Pesquisa bibliográfica				
Desenvolvimento da ferramenta				
Testes da ferramenta				
Escrita TCC 1				
Revisão TCC 1				
Entrega TCC 1				

Trabalho de Conclusão II

Etapa	Meses			
	Mar	Abr	Mai	Jun
Desenvolvimento da ferramenta				
Testes da ferramenta				
Criação dos requisitos				
Implementação dos requisitos sem ferramenta				
Implementação dos requisitos com ferramenta				
Experimentação e uso da ferramenta com coleta de dados sobre tempo				
Análise dos resultados e diferenças				
Pesquisa de opinião				
Escrita TCC II				
Revisão TCC II				
Entrega TCC II				

BIBLIOGRAFIA

ABAP. Disponível em https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm. Acesso em 27 ago. 2022.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML:** guia do usuário. Rio de Janeiro: Elsevier, 2006.

DRAW.IO. **About diagrams.net**. Disponível em https://www.diagrams.net/about. Acesso em: 24 ago. 2022.

FOWLER, Martin. **Is High Quality Software Worth the Cost?** Disponível em https://martinfowler.com/articles/is-quality-worth-cost.html. Acesso em: 24 ago. 2022.

HERRINGTON, Jack. **Code generation in action**. Greenwich: Manning, 2003.

KOLLER, Henning Grimeland. **Effects of Clean Code on Understandability**. Department of Informatics, University of Oslo, 2016.

LIMAYEM, M.; KHALIFA, M.; CHIN, W. W. **CASE Tools Usage and Impact on System Development Performance.** Journal of Organizational Computing and Electronic Commerce, *14(3), 153–174,* 2004.

LUCIDCHART. Disponível em https://www.lucidchart.com/. Acesso em: 24 ago. 2022.

MARTIN, Robert Cecil. **Clean code**: A Handbook of Agile Software Craftsmanship. Upper Saddle River, NJ: Prentice Hall, 2009.

MIRO. Disponível em https://miro.com/. Acesso em: 24 ago. 2022.

MORALES, V. Y. Rosales; HERNÁNDEZ, G. Alor; ALCARÁZ, J. L. García; CABADA, R. Zatarain; ESTRADA, M. L. Barrón. **An analysis of tools for automatic software development and automatic code generation**. Revista Facultad de Ingeniería Universidad de Antioquia, (77), 2015.

NECCO, Charles; TSAI, Nancy; CHANG, Shyh-Jen. **The impacts and benefits of using CASE tools in the system development life cycle**. Journal of International Information Management: Vol. 3 : Iss. 1 ,Article 7, 1994.

PANDO, Brian; CASTILLO, Jose. **PlantUMLGen**: A tool for teaching Model Driven Development. 17th Iberian Conference on Information Systems and Technologies (CISTI), 2022.

PLANTUML. **PlantUML in a nutshell**. Disponível em https://plantuml.com/. Acesso em: 24 ago. 2022.

PRODANOV, Cleber Cristiano; FREITAS, Ernani Cesar de. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico**. 2. ed. Novo Hamburgo, RS: Feevale, 2013.

SAP NETWEAVER. Disponível em https://www.sap.com/products/technology-platform/netweaver.html. Acesso em 27 ago. 2022.

TANNÚS, Pedro Henrique Albernaz Machado A.; NETO, Milton Miranda. **Desenvolvimento e uso de Ferramenta CASE de geração de código automatizado baseada em *templates***. Uberlândia, MG: UNITRI, 2021.

XMI. Disponível em: https://www.omg.org/spec/XMI/2.5.1/PDF. Acesso em 02 set. 2022.