Análise de Ferramentas CASE quanto às Boas Práticas de Modelagem de Software com UML

Warteruzannan Soyer Cunha[1], Heitor Costa [2], Paulo Afonso Parreira Júnior [2]

[1] Instituto de Ciências Exatas - Universidade Federal de Goiás – Regional Jataí – Caixa Postal 03 – 75801-615 – Jataí-GO – Brasil

[2] Departamento de Ciência da Computação - Universidade Federal de Lavras Caixa Postal 3.037 – 37.200-000 – Lavras – MG – Brasil

warteruzannan@gmail.com, {heitor, pauloa.junior}@dcc.ufla.br

Resumo. *Diagramas UML que são construídos utilizando-se um conjunto de diretrizes para modelagem de software podem resultar em um software de maior qualidade e que atenda às reais necessidades do cliente. Dessa forma, o objetivo deste trabalho é comparar e analisar um conjunto de ferramentas CASE, com o intuito de descobrir quais são as diretrizes para confecção de diagramas UML que estas ferramentas atendem. Foram analisadas 5 (cinco) ferramentas CASE, a saber, Rational Software, MagicDraw, UModel, Visual Paradigma e Enterprise Architect. Nesta análise, a ferramenta que atendeu ao maior número de diretrizes foi a Rational Software. Além disso, foi realizado um experimento no qual verificou que os diagramas confeccionados utilizando a ferramenta que atendeu o maior número de diretrizes contêm menos erros do que os confeccionados em outras ferramentas.*

Abstract. *UML diagrams constructed using a set of guidelines for software modeling may result in higher quality software that meets the real customer needs. Thus, the goal of this paper is to compare and analyze a set of CASE tools, in order to find out what are the guidelines addressed by these tools. Five CASE tools were analyzed: Rational Software, MagicDraw, UModel, Visual Paradigm and Enterprise Architect. In this analysis, the tool that addresses the highest number of guidelines was the Rational Software. In addition, an experimental study was conducted and the results indicated that the diagrams built in this tool presented fewer errors than those built in other CASE tools.*

1. Introdução

Segundo Booch *et al*. (2005), durante o processo de desenvolvimento de um software é importante que o engenheiro de software tenha entendimento adequado do domínio do problema para o qual o software está sendo desenvolvido. Segundo os autores, dessa forma, maiores serão as chances de o software ser desenvolvido com qualidade.

Sommerville (2011) afirma que modelos de software podem ser utilizados para se obter uma maior compreensão do domínio do problema. Ainda segundo o autor, modelos de software são representações simplificadas (abstrações) de algo do mundo real, enfatizando suas características principais e descartando as irrelevantes. Modelos de software podem ser expressos por representações gráficas e uma linguagem

amplamente conhecida e utilizada para representação gráfica de modelos de software, por meio de diagramas, é a UML (*Unified Modeling Language*).

Segundo Ambler (2003), um modelo de software deve ser confeccionado com base em algumas diretrizes (*guidelines*) e boas práticas para modelagem de software, com o intuito de evitar erros de modelagem, bem como tornar os modelos do software mais fáceis de serem lidos, entendidos e mantidos pelos engenheiros de software. Por exemplo, na confecção de diagramas de casos de uso, uma das diretrizes propostas por Ambler (2003) é que os nomes (identificadores) desses casos de uso iniciem com um verbo no infinitivo, como por exemplo, "cadastrar", "alterar", "excluir", entre outros, pois isso deixa claro o tipo de interação que deve haver entre o(s) ator(es) e esse caso de uso.

No contexto do desenvolvimento de software, há diversas ferramentas computacionais, comumente chamadas de ferramentas CASE (*Computer-Aided Software Engineering*), que oferecem suporte a diversas atividades do processo de desenvolvimento de software, tais como análise de requisitos, modelagem de software, depuração e teste. Khaled (2009) relata que o uso de ferramentas CASE no contexto do desenvolvimento de um software pode melhorar a comunicação entre a equipe de desenvolvimento e reduzir risos e custos na construção do software. Segundo Chen e Zhenhua (2011), existem vários trabalhos na literatura que propõem técnicas e/ou ferramentas para verificação da qualidade de digramas UML. Apesar disso, há escassez de trabalhos na literatura que fazem uma análise comparativa das ferramentas CASE já existentes, com enfoque em verificar a adequação das mesmas às diretrizes e boas práticas para modelagem de software preconizadas pela Engenharia de Software. Isso é um problema, pois segundo Khaled (2009), é importante que o engenheiro de software saiba escolher a ferramenta que usará, pois algumas limitações existentes nessas ferramentas podem atrasar o andamento do projeto.

Neste sentido, o objetivo deste trabalho, definido com base na abordagem GQM (*Goal-Question-Metrics*) (Basili e Rombach, 1994), é: *analisar* um conjunto de ferramentas CASE para construção de diagramas UML, *com o propósito* de verificar, *com respeito* à eficácia dessas ferramentas quanto à adequação dos diagramas desenvolvidos por meio delas às diretrizes para modelagem de software propostas por Ambler (2003), do *ponto de vista de* Engenheiros de Software, *no contexto de* alunos de graduação em ciência da computação. Além disso, verificou-se quais tipos de erros de modelagem podem ser capturados por estas ferramentas.

A análise realizada neste trabalho baseou-se no conjunto de diretrizes para confecção de diagramas UML proposto por Ambler (2003). Este conjunto foi utilizado, pois é baseado nas recomendações para construção de diagramas UML propostas pelo OMG (*Object Management Group*). Além disso, as diretrizes de Ambler contemplam os cinco principais tipos de diagramas para modelagem de software (diagrama de classe, diagrama de casos de uso, diagrama de sequência, diagrama de atividades e diagrama de estados), segundo Erickson (2007) *apud* Sommerville (2011).

Este trabalho está organizado como segue: a Seção 2 apresenta os conceitos básicos sobre modelagem de software. Na Seção 3 são apresentados e discutidos os trabalhos relacionados. A Seção 4 apresenta os detalhes e resultados da avaliação conduzida e, por fim, a Seção 5 descreve as considerações finais e propostas de trabalhos futuros.

2. Terminologias e conceitos básicos

2.1 Modelagem de Software

Segundo Sommerville (2011), modelos de software geralmente são representados utilizando-se alguma notação gráfica. Segundo Booch *et al*. (2005), a UML (*Unified Modelling Language*) é uma linguagem padrão para a confecção de modelos de software, a qual pode ser usada como meio de visualização, especificação, construção e documentação de sistemas de software. Ainda segundo os autores, a UML tem sido utilizada de forma efetiva em vários domínios, tais como: sistemas de informação corporativos, sistemas bancários e financeiros, telecomunicações, transportes, defesa/espaço aéreo, vendas de varejo, eletrônica médica, modelos científicos, serviços distribuídos baseados na web, entre outros.

Segundo o OMG (*Object Management Group*, 2005), a versão mais recente da UML (versão 2.X) possui 13 (treze) diagramas, sendo eles: diagrama de classes, diagrama de objetos, diagrama de componentes, diagrama de estruturas compostas, diagrama de casos de uso, diagrama de sequência, diagrama de comunicação, diagrama de estados, diagrama de atividades, diagrama de implantação, diagrama de pacote, diagrama de temporização e diagrama de visão geral de interação. Apesar disso, Erickson (2007) *apud* Sommerville (2011) destaca que a maioria dos engenheiros de software julgam que apenas 5 (cinco) diagramas são suficientes para representar a essência de um software, sendo eles: o diagrama de atividades, o diagrama de sequência, o diagrama de classes, o diagrama de estados e o diagrama de casos de uso.

2.2 Diretrizes para Modelagem de Software com UML

Ambler (2003) relata que diagramas UML que são confeccionados seguindo algumas diretrizes como boas práticas para modelagem de software são mais fáceis de serem entendidos, o que pode resultar na construção de softwares de melhor qualidade e que atendam às reais necessidades dos clientes. Segundo o autor, estas diretrizes são formas de se guiar a confecção dos diagramas, com o intuito de tornar esses diagramas mais claros e objetivos.

Ambler (2003) traz uma série de diretrizes baseadas nas recomendações para modelagem de software do OMG (*Object Management Group*). O autor apresenta algumas diretrizes que podem ser aplicadas para a maioria dos diagramas, *i.e.*, diretrizes de propósito geral e diretrizes que são específicas de alguns diagramas. Este trabalho baseia-se nas diretrizes de propósito geral e naquelas que podem ser aplicadas aos cinco diagramas citados por Erickson (2007) *apud* Sommerville (2011), a saber: diagrama de classes, diagrama de casos de uso, diagrama de sequência, diagrama de atividades e diagrama de estados. No total, Ambler (2003) apresenta 190 diretrizes, sendo 28 (vinte e oito) de propósito geral, 27 (vinte e sete) para diagramas de casos de uso, 57 (cinquenta e sete) para diagramas de classes, 26 (vinte e seis) para diagramas de sequência, 23 (vinte e três) para diagramas de estados e 29 (vinte e nove) para diagramas de atividades. A seguir são exemplificadas algumas dessas diretrizes.

Um exemplo de diretriz para a construção de diagramas de casos de uso é “começar nomes de casos de uso com um verbo no infinitivo”. Exemplos de nomes de casos de uso com verbos no infinitivo são: “registrar cadastro”, “enviar pacote”, “realizar autenticação”, entre outros. Segundo Ambler (2003), nomes que começam com um verbo no infinito deixam claro o que o caso de uso faz, ajudam no

entendimento do diagrama e na comunicação com o cliente. Outras diretrizes para confecção de diagramas de casos de uso são: (i) associar cada ator com um ou mais casos de uso; (ii) usar o estereótipo «System» para indicar que um ator é um sistema de software; (ii) não permitir que um ator interaja com outro ator; (iii) evitar mais do que dois níveis de associação em um caso de uso; (iv) evitar colocar setas direcionadas para um ator do caso de uso; entre outros.

Uma diretriz especificada para o diagramas de classes é "ser consistente com nomes de atributos". Segundo Ambler (2003), ser consistente com nomes de atributos facilita identificar a finalidade desse atributo. Por exemplo, os atributos "endereço" e "nome", quando aplicados a uma classe "Pessoa", deixam claro que representam o endereço e o nome de uma pessoa, respectivamente. Em contrapartida, nomes como: "i" e "attr" são confusos e difíceis de se entender. Outros exemplos de diretrizes para a confecção de diagramas de classes são: (i) identificar visibilidade dos métodos de uma classe; (ii) usar substantivos singulares para nome das classes; (iii) usar substantivos singulares para nome de atributos; (iv) começar nome dos métodos com um verbo no infinitivo; (v) nunca criar classes com apenas dois compartimentos; (vi) ao definir uma interface, defini-la separadamente de sua classe; (vii) sempre indicar a multiplicidade entre as classes; entre outros.

Para a condução deste trabalho, todas as diretrizes propostas por Ambler (2003) foram analisadas, com o intuito de selecionar o subconjunto daquelas que tratam de aspectos que possam ser verificados de forma automática pelas ferramentas CASE, ou seja, que tratam de aspectos sintáticos. Diretrizes que tratam de aspectos semânticos não foram abordas neste trabalho. Um exemplo de diretriz que não foi considerada neste trabalho é: "apresentar somente o que tem que ser apresentado em um diagrama". Outro exemplo é: "nomear classes de acordo com o domínio do problema". Percebe-se que esta diretriz está relacionada ao conhecimento do engenheiro de software quanto à solução do problema, tornando complexa a identificação desse tipo de erros por parte das ferramentas.

3. Trabalhos Relacionados

Segundo Chupac *et al.* (2013), existem diversas ferramentas que auxiliam na confecção de digramas UML. Além de facilitar a confecção de diagramas, estas ferramentas tornam o trabalho do engenheiro de software mais produtivo, bem como à confecção de diagramas que são mais fáceis de entender. A principal contribuição do trabalho de Chupac *et al.* (2013) foi selecionar as ferramentas que auxiliam em alguns aspectos para a confecção de diagramas UML e verificar quais critérios elas atendem, tais como "possibilidade de simulação de modelos", "versões da UML que estas ferramentas suportam", "formatos para exportação de projetos e diagramas", entre outros critérios.

Os autores escolheram um conjunto de ferramentas para análise, a partir de uma lista publicada em 2013 pelo OMG. Esta lista contém as ferramentas que são amplamente usadas e consideradas pela OMG como as principais ferramentas para a modelagem de software. Foram escolhidas 14 (quatorze) ferramentas para a análise, sendo elas: Visual Paradigm 8.3 (Enterprise Edition); MagicDraw 17.0.1 (Enterprise Edition); StarUML 5.0; UModel 2012 (Enterprise Edition); CaseComplete 2012; EnterpriseArchitect 9.2 (Corporate Edition); SketchUML; Bridgepoint 3.4.6; PowerDesigner 16.1; Artisan Studio 7.3; Objecteering 6.1 (Enterprise Edition); Rational

Software Architect 8.0; Blueprint Software Modeler (Community Edition); e Innovator 11 for software architects.

Os autores verificaram quais ferramentas atendem aos critérios estabelecidos por eles e ordenaram as ferramentas de acordo com a pontuação obtida por cada uma. Os critérios estabelecidos receberam um nível de importância e um peso. Após isso, cada ferramenta foi analisada, verificando-se quais são os critérios atendidos por ela. A pontuação final da ferramenta é a soma dos produtos do peso dos critérios contemplados por ela pelo nível de importância do critério. Alguns dos critérios escolhidos pelos autores foram: "suporte a trabalho em equipe", "suporte à simulação de diagramas", "suporte à MDA (Model-driven Architecture)", "suporte à exportação de documentos", "suporte à engenharia reversa", "suporte à geração de código fonte", entre outros. Segundo os autores, as ferramentas mais bem colocadas foram a Enterprise Architect e Visual Paradigm, tendo elas contemplado a maioria dos critérios. Algumas desvantagens encontradas na Enterprise Architect são que ela não permite a geração de código automático para a linguagem Ruby, nem permite exportar diagramas no formato SVG (Scalable Vector Graphics). A desvantagem da ferramenta Visual Paradigm é que ela não permite a geração, de forma automática, de bibliotecas de artefatos.

O trabalho de Khaled (2009) reúne uma série de ferramentas para confecção de diagramas UML e define critérios de avaliação, a fim de dizer quais são as ferramentas mais indicadas para determinadas tarefas. Foram escolhidas 4 (quatro) ferramentas para serem analisadas, a saber: *ArgoUML*, *MagicDraw*, *Rational Rose* e *Enterprise Architect*. O autor compara estas ferramentas, descrevendo-as e apontando quais critérios elas atendem. Os critérios (ou funcionalidades) que as ferramentas deveriam atender, segundo o autor, são os seguintes:

1. Capacidade de gerar documentação HTML (*HyperText Markup Language*) para os diagramas desenvolvidos na ferramenta;
2. Capacidade de suportar todos os diagramas da UML 2.0;
3. Capacidade de gerar código-fonte a partir de um modelo e vice-versa;
4. Capacidade de integração com ferramentas de modelagem para banco de dados;
5. Capacidade de exportação de diagramas em formatos como: PDF, GIF, JPEG, entre outros; e
6. Capacidade de recuperação de um estado válido dos diagramas desenvolvidos caso aconteça algum imprevisto, como por exemplo, o desligamento inesperado do computador.

Segundo o autor, caso o engenheiro de software esteja procurando por uma ferramenta que forneça suporte a todos os diagramas da UML, esta ferramenta é a *Rational Rose*. Além de suportar todos os diagramas, ela pode ser integrada com outras ferramentas. Segundo ele, a melhor ferramenta encontrada para projeto da arquitetura do software é a *ArgoUML*. Caso o engenheiro de software queira uma ferramenta para controle da qualidade do software e que gere documentação para os diagramas desenvolvidos, a melhor escolha é a MagicDraw. Por último, a Enterprise Architect é a melhor escolha para a modelagem de negócio e atividades do projeto de software.

Em comparação com o trabalho proposto, o trabalho de Chupac *et al*. (2013) faz uma análise de ferramentas CASE para a confecção diagramas UML. Porém, uma diferença é que ele não verifica se essas ferramentas contemplam alguma diretriz para verificação de diagramas quanto às boas práticas de modelagem de software. Isto é importante, pois proporciona ao engenheiro de software uma informação a mais para tomada de decisão quanto à escolha da ferramenta CASE que melhor atenda seus interesses. Outra desvantagem é que os autores não relatam o porquê da escolha dos critérios utilizados por eles, de que fonte vieram estes critérios e porque eles são importantes para comparação de ferramentas CASE.

Quanto ao trabalho de Khaled (2009), mais uma vez, uma diferença encontrada é que ele não avaliou se as ferramentas CASE realizam algum tipo de verificação dos diagramas UML gerados por meio delas. Outra diferença é que ele não deixa claro quais foram os critérios utilizados para a seleção das ferramentas analisadas. Isso é importante porque pode haver ferramentas que não sejam tão utilizadas ou estejam obsoletas. Outra desvantagem é que ele também não descreve quais foram os critérios utilizados para a escolha das funcionalidades que as ferramentas deveriam atender.

4. Análise das Ferramentas CASE

4.1 Escolha das diretrizes e das ferramentas CASE para análise

O primeiro procedimento metodológico realizado para se atingir o objetivo deste trabalho foi selecionar um subconjunto das diretrizes apresentadas por Ambler (2003), destacando aquelas que são passíveis de serem verificadas de forma automática. Apesar de o autor apresentar 190 diretrizes, nem todas podem ser verificadas automaticamente. Por isso foram selecionadas apenas diretrizes que tratam de aspectos sintáticos, de forma que a verificação do diagrama não dependa da interferência do engenheiro de software. As diretrizes escolhidas são apresentadas na Tabela 1. A primeira coluna dessa tabela representa o diagrama para qual a diretriz foi confeccionada, que por sua vez é descrita na terceira coluna. Já a coluna 2 apresenta o "identificador" único para tal diretriz.

Para que tais diretrizes fossem selecionadas as seguintes etapas foram realizadas: (i) cada autor selecionou um subconjunto de diretrizes entre as 190 mencionadas; e (ii) os subconjuntos escolhidos pelos autores foram confrontados, a fim de se formar apenas um subconjunto. Desta forma, caso alguma diretriz estivesse presente no subconjunto de todos os autores, esta já seria escolhida para o subconjunto final. Caso alguma destas estivesse presente em apenas um ou dois subconjuntos, haveria uma discussão a fim de verificar a necessidade de inclusão (ou não) de tal diretriz no subconjunto final. O primeiro autor a apresentar os argumentos que levaram à inclusão de tal diretriz foi o que não inclui a diretriz em questão; ou o autor que inclui tal diretriz e os outros não. É importante salientar que as diretrizes analisadas neste trabalho foram elaboradas para a versão 2.0 da UML, logo, estas não atendem todos os aspectos da versão atual (2.5)

Uma vez escolhidas as diretrizes, o próximo passo foi selecionar as ferramentas que seriam analisadas. Para isso, foram escolhidas as 05 (cinco) ferramentas CASE mais bem pontuadas no trabalho de Chupac *et al*. (2013), a saber (em ordem crescente): *EnterpriseArchitect, Visual Paradigm, MagicDraw, Rational Software Architect e UModel.*

Tabela 1. Diretrizes utilizadas para avaliação das ferramentas CASE.

Diagrama	#	Diretriz
Diagrama de Casos de uso	01	Associar cada ator com um ou mais casos de uso.
	02	Não permitir que um ator interaja com outro ator.
	03	Usar «`System`» para indicar um ator sistema.
Diagrama de Classes	04	Não colocar nome de associação com o mesmo nome de alguma classe.
	05	Inicia nome de operações com verbo no infinitivo.
	06	Indicar a visibilidade dos métodos da classe.
	07	Criar classes que contenha um box para nome, um para atributos e outro métodos.
	08	Sempre indicar multiplicidade da relação.
Diagrama de Sequência	09	Nomear Ator com o mesmo nome de uma classe.
	10	Evitar destruição de objetos.
	11	Sempre indicar valor de retorno ao lado das mensagens.
	12	Evitar usar boxes de ativação durante a execução do diagrama.
	13	Evitar criar objetos durante a execução do diagrama.
Diagrama de Atividades	14	Sempre indicar um ponto inicial.
	15	Sempre indicar um ponto final.
	16	Garantir que um "*fork*" tenha somente uma entrada.
	17	Garantir que um "*join*" tenha somente uma saída.
	18	Evitar usar mais do que 5 "*Swim Lanes"*.
Diagrama de Estados	19	Criar um estado inicial.
	20	Indicar a ação de entrada.
	21	Colocar nomes de transição no passado.

4.2 Análise das ferramentas CASE com base nas diretrizes escolhidas

As ferramentas CASE selecionadas no passo anterior foram analisadas utilizando-se os seguintes procedimentos:

- Foi especificado um caso de teste que contradiz cada uma das 21 diretrizes selecionadas (Tabela 1). Por exemplo, a diretriz 02 (dois) descreve que um ator em um diagrama de casos de uso deve estar associado a um ou mais casos de usos. Logo, o caso de teste especifica um ator que não está associado a qualquer caso de uso; e
- Com os casos de testes especificados para todas as diretrizes, o próximo passo foi confrontar estes com todas as ferramentas escolhidas. Esse passo teve o objetivo de verificar quais ferramentas conseguem: (i) evitar que determinados diagramas incorretos sejam criados; e/ou (ii) identificar e exibir ao usuário, mensagens que o alertem sobre possíveis erros de modelagem.

A Tabela 2 apresenta quais diretrizes são verificadas utilizando as ferramentas analisadas. A coluna 1 indica a ferramenta analisada; a coluna 2 representa a lista de diretrizes que foram atendidas (verificadas) pela ferramenta CASE - a diretriz é indicada de acordo com seu identificador, que foi apresentado na Tabela 1. Por fim, a coluna 3 representa a quantidade de diretrizes contempladas por cada ferramenta.

Percebe-se que poucas são as diretrizes verificadas pelas ferramentas analisadas. A ferramenta *Rational Software* atendeu a 8 (oito) diretrizes, sendo a ferramenta com maior número de diretrizes contempladas. Entretanto, as outras ferramentas não foram tão eficazes neste quesito, uma vez que a segunda colocada atendeu somente 4 (quatro) diretrizes.

Tabela 2. Resultado da análise realizada nas ferramentas CASE com base nas diretrizes escolhidas.

Ferramenta	Diretrizes Contempladas	Total
EnterpriseArchitect	[02],[06],[15]	3
Visual Paradigm	[06],[10]	2
MagicDraw	[06],[17]	2
Rational Software	[01],[02],[06],[07],[12][13],[16],[17]	8
UModel	[02],[08],[07],[13]	4

A Figura 1 apresenta um diagrama confeccionado a partir da ferramenta *MagicDraw*. Esta representa um caso de uso no qual não há relacionamento com o único ator do sistema, a saber, "Leitor". Como foi relatado anteriormente, tal erro deve ser evitado pelas ferramentas. Apesar de a existência de mecanismos que identifiquem erros presentes nos diagramas gerados por esta, a mesma não foi capaz de alertar sobre o ocorrido. Tal mecanismo é ativado a partir do momento em que o usuário clica sobre botão "*Check Syntax*" (Figura 2).

Figura 1. Diagrama confeccionado a partir da ferramenta *MagicDraw*.

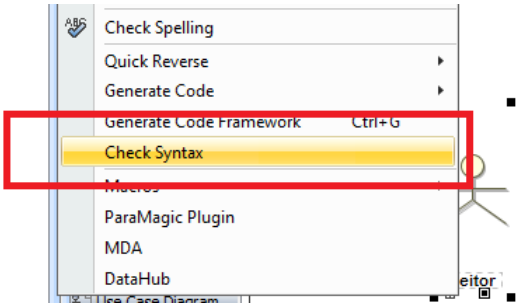

Figura 2. Mecanismo de checagem da ferramenta *MagicDraw*.

A Figura 3 apresenta um erro encontrado nos diagramas de classes gerados a partir da ferramenta *UModel*. Tal diagrama contradiz a diretriz [11] que relata que um diagrama de classes não deve conter relacionamentos nomeados com o mesmo nome de alguma classe existente.

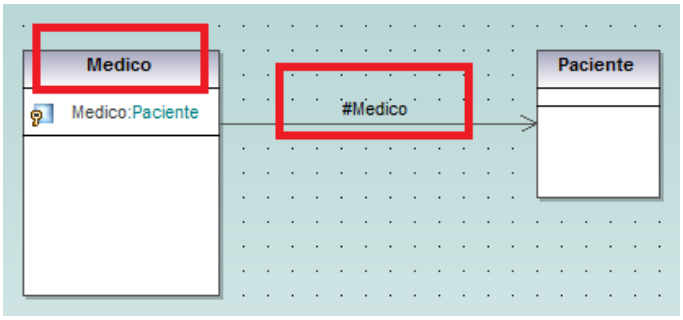

Figura 3. Diagrama confeccionado a partir da ferramenta *UModel*.

4.3 Verificação da qualidade dos modelos UML

Um estudo experimental foi conduzido, a fim de se verificar a qualidade dos diagramas UML gerados por meio das ferramentas avaliadas no passo anterior. O intuito foi descobrir se as ferramentas que cobrem mais diretrizes para verificação da adequação de diagramas UML às boas práticas de modelagem levam os participantes a gerarem diagramas mais corretos.

Sendo assim, 10 (dez) alunos de graduação do curso de Ciência da Computação da Universidade Federal de Goiás – Regional Jataí foram selecionados para serem participantes nesse experimento. É importante ressaltar que todos os participantes já haviam cursado a disciplina de Engenharia de Software e, dessa forma, já tinham conhecimento prévio sobre modelagem de software, UML e ferramentas CASE. Os participantes foram divididos em 2 (dois) grupos de cinco alunos, Grupo A e Grupo B.

Além dos grupos, duas das ferramentas analisadas anteriormente, nomeadas aqui de F1 e F2, foram selecionadas para o experimento, de forma que a ferramenta F1 contemplou a maior quantidade de diretrizes e a ferramenta F2 contemplou a menor quantidade. Como duas ferramentas empataram quanto à menor quantidade de diretrizes contempladas, a saber, *Visual Paradigm* e *MagicDraw*, foram definidos os seguintes critérios de desempate. A ferramenta que atendeu a maior quantidade de diretrizes no: (i) diagrama de classes; (ii) diagrama de casos de uso; (iii) diagrama de sequência; (iv) diagrama de atividades; e (v) diagrama de estados. Estes critérios foram definidos de acordo com a importância do diagrama para a modelagem de software. Erickson (2007) relata que o diagrama de classes é o mais utilizado para confeccionar modelos de software. Logo, definiu-se que o primeiro critério de desempate define a ferramenta que atendeu maior quantidade de diretrizes neste diagrama. Analogamente, foram definidos os critérios II, III, IV, e V. Dessa forma, a ferramenta *Rational* foi escolhida como F1 e a *Visual Paradigm*, como F2. Isso porque, *Visual Paradigm* atendeu maior número de diretrizes relacionadas ao diagrama de sequência (terceiro critério de desempate).

Assim, os resultados desse experimento servem também de validação para a análise apresentada na seção anterior, *i.e.*, a hipótese é que, se as ferramentas foram corretamente avaliadas, a ferramenta F1 deve gerar melhores resultados do que a ferramenta F2.

A Tabela 3 apresenta o *design* do experimento, o qual é descrito a seguir:

- Houve um treinamento com todos participantes sobre modelagem de software. Foram explicados aos dois grupos os princípios da modelagem de software usando UML, com base nas diretrizes propostas por Ambler (2003). Além disso, foram explicados os seguintes diagramas da UML: diagrama de classes, diagrama de casos de uso e diagrama de sequência. É importante ressaltar que foram utilizados apenas estes três diagramas, pois considerou-se que, caso o experimento fosse conduzido utilizando mais que três (diagramas), o mesmo seria cansativo e impactaria diretamente nos resultados. Este treinamento também consistiu em exemplificar a todos participantes como confeccionar cada diagrama nas duas ferramentas escolhidas anteriormente (F1 e F2);
- A primeira etapa do experimento consistiu em requisitar aos dois grupos que, utilizando as ferramentas F1 e F2, confeccionassem três diagramas

UML (diagrama de classe, casos de uso e de sequência), a partir da especificação de um software. O grupo A utilizou a ferramenta F2 para isso e recebeu a especificação de um software hipotético, denominado S1. Analogamente, o grupo B, utilizando a ferramenta F2, confeccionou os mesmos diagramas para a especificação de outro software hipotético, denominado S2; e

- A segunda fase do experimento consistiu em trocar as ferramentas que os dois grupos utilizaram para F1 e inverter a especificação de software cada grupo utilizou para confecção dos diagramas. Dessa forma, o Grupo A, utilizando a ferramenta F1, confeccionou diagramas para a especificação S2, enquanto que o Grupo B, também utilizando a ferramenta F1, confeccionou diagramas para a especificação S1.

Tabela 3. *Design* do estudo experimental

Fase	Grupos	
	A	**B**
Treinamento	Ferramentas CASE (F1 e F2) e Modelagem de Software.	Ferramentas CASE (F1 e F2) e Modelagem de Software.
Primeira Etapa	Modelagem do software S1 com ferramenta CASE F2	Modelagem do software S2 com ferramenta CASE F2
Segunda Etapa	Modelagem do software S2 com ferramenta CASE F1	Modelagem do software S1 com ferramenta CASE F1

As especificações S1 e S2 são apresentadas na Tabela 4. Procurou-se criar especificações que fossem equivalentes em termos de esforço para confecção dos diagramas e que levasse o participantes a exercitar os mesmos elementos de modelagem.

Uma vez finalizado o experimento, os diagramas UML confeccionados pelos participantes foram analisados por professores da área de Engenharia de Software e os erros cometidos por cada um foram anotados. A Tabela 5 e Tabela 6 apresentam, respectivamente, a quantidade de erros identificados para os diagramas gerados seguindo as especificações S1 e S2. Cada participante é identificado com P*i*, sendo que *i* vai de 1 a 10. Sendo assim o participante 1 foi nomeado como P1, o participante 2 como P2 e assim por diante.

Como pode ser visto nas Tabelas 5 e 6, a quantidade de erros identificados nos diagramas é sempre menor, quando foi utilizada a ferramenta F1 (*Rational Software*). Outro ponto a ser considerado é que foram identificados menos erros nos diagramas elaborados para a especificação S1 do que nos diagramas da especificação S2. Isso pode indicar que houve maior dificuldade de compreensão da especificação S2, por parte dos participantes. Uma das possíveis explicações para isso é que o domínio de biblioteca é mais familiar aos estudantes de graduação do que o domínio de uma clínica veterinária. Contudo, mesmo assim, a ferramenta F1 levou os participantes a cometerem uma menor quantidade de erros de modelagem.

Tabela 4. Especificações utilizadas no estudo experimental.

***Especificação S1**: Uma biblioteca necessita de um sistema online para reservas de livros. O sistema deve permitir ao leitor (principal usuário do sistema) consultar um livro, reservar um livro e cancelar reserva. Para reservar um livro o usuário deve "logar" no sistema usando seu login e senha (considere que os usuário já foram cadastrados). Após isso ele pode consultar a existência do livro de seu interesse e realizar a reserva. Para realizar a reserva o livro deve estar disponível para reserva. Cada*

leitor tem uma lista de livros reservados. Para cancelar uma reserva o usuário pode consultar sua lista de reservas, selecionar uma reserva e cancelar a mesma. Cada livro contém as seguintes informações: nome, data de publicação, nome do autor, nome da editora, disponibilidade e número de páginas. Além disso, cada leitor contém as seguintes informações: nome, email, data de nascimento, endereço, login e senha. E por fim, cada registro de reserva deve conter o nome do leitor, nome do o livro e data da reserva.
***Especificação S2**: Uma clínica veterinária necessita de um software para guardar informações sobre as consultas realizadas. O sistema deve permitir ao usuário (recepcionista da clínica) cadastrar uma nova consulta (essa consulta só é cadastrada após ser realizada). Para cadastrar uma consulta o usuário deve autenticar-se no sistema usando seu login e senha. Após isso, ele pode cadastrar essa nova consulta. Para que seja cadastrada uma nova consulta ela deve conter as seguintes informações: um médico, um paciente (este paciente é um animal) e data da consulta. Dessa forma, uma consulta só poderá ser cadastrada se estiver relacionada com um médico e com um paciente. Além disso, todos os pacientes têm um proprietário. Cada paciente tem as seguintes informações: nome, data de nascimento, peso e nome do proprietário. Um proprietário (considere que um proprietário NÃO pode ser um usuário) é uma pessoa que contém as seguintes informações: nome, telefone e endereço. Além disso, um médico tem: nome, data de início do contrato, telefone e endereço. Por fim, um usuário(recepcionista) contém as seguintes informações: nome, telefone, endereço, email, login e senha.*

Tabela 5. Resultados – Especificação S1.

Ferramenta	***Visual Paradigm***					***Rational Software***				
Participante	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**
Diagrama de classes	5	5	4	5	9	1	0	0	1	1
Diagrama de casos de uso	2	1	2	0	0	1	0	0	0	2
Diagrama de sequência	9	9	6	5	14	1	1	1	0	2
Erros/participante	**9**	**9**	**6**	**5**	**14**	**3**	**1**	**1**	**1**	**5**
Erros/ferramenta	**43**					**11**				
Média de erros/participante	**8,6**					**2,2**				

Tabela 6. Resultados – Especificação S2.

Ferramenta	***Visual Paradigm***					***Rational Software***				
Participante	**P6**	**P7**	**P8**	**P9**	**P10**	**P1**	**P2**	**P3**	**P4**	**P5**
Diagrama de classes	5	7	11	8	7	0	4	3	3	0
Diagrama de casos de uso	0	3	0	1	2	0	0	0	0	0
Diagrama de sequência	0	6	4	2	6	4	2	0	3	1
Erros/participante	5	16	15	11	15	4	6	3	6	1
Erros/ferramenta	**62**					**20**				
Média de erros/participante	**12,4**					**5,0**				

Outro fato interessante que pode ser notado é que, nos diagramas de classes e de sequência, foram identificados mais erros do que nos demais diagramas, independentemente da especificação e/ou da ferramenta. Isso pode indicar que estes são diagramas mais complexos de serem confeccionados e/ou que os alunos apresentam maior dificuldade de compreender.

Percebe-se que o participante P7 foi o que cometeu a maior quantidade de erros com a ferramenta *Visual Paradigm (*16 erros). Entretanto, quando o mesmo utilizou a ferramenta *Rational Software* foi identificado apenas um problema de modelagem. A diferença na quantidade de erros pode indicar que o suporte oferecido pela ferramenta *Rational* pode aumentar a qualidade do diagrama gerado, uma vez que muitos erros podem ser evitados pela própria ferramenta. Logo, este fato indica que a ferramenta utilizada pode interferir diretamente na qualidade dos diagramas confeccionados, uma vez que muitos erros são passíveis de serem identificados e evitados por tais ferramentas. Sendo assim, mesmo os dois grupos tendo conhecimento sobre as boas práticas para modelagem de software, os participantes que utilizaram a ferramenta F1

apresentaram melhores resultados do que aqueles que utilizaram a ferramenta F2. É importante ressaltar que o escopo dos softwares (S1 e S2) modelados pelos participantes foi reduzido a fim de se diminuir a complexidade envolvida no entendimento do problema.

5. Considerações Finais

Este trabalho descreveu o processo de escolha e avaliação de cinco ferramentas CASE de apoio ao processo de modelagem de software com UML, quanto a sua eficácia para verificação da adequação dos diagramas confeccionados a partir delas, quanto às boas práticas para modelagem de software. Além disso, um estudo experimental foi conduzido a fim de destacar o efeito que a não verificação dos diagramas UML pode ter na qualidade dos diagramas gerados por meio dessas ferramentas.

Como trabalhos futuros pretende-se realizar novos experimentos aumentando a quantidade de diretrizes, ferramentas e diagramas analisados. Dessa forma, ferramentas como *StarUML, Case Complete, Astah,* entre outras, poderão ser analisadas, assim aumentando a eficácia do trabalho. Além disso, pretende-se construir um *plug-in* para a ferramenta *Rational* que, baseado nas diretrizes apresentadas no decorrer deste trabalho, consiga identificar erros gerados durante a confecção dos diagramas UML. Desta forma, pode-se aumentar a quantidade de diretrizes verificadas por meio desta ferramenta e consequentemente, aprimorar a qualidade dos diagramas gerados a partir dela. Também pretende-se analisar diretrizes que tratam de aspectos semânticos, e assim, permitir que tais ferramentas detectem erros que dependem do domínio do problema a ser modelado. Além disso, pretende-se realizar novos experimentos com ferramentas *open-source*, a fim de confrontar os diagramas gerados por estas com diagramas gerados por ferramentas proprietárias. Com isso, pretende-se verificar se tais diagramas (gerados por ferramentas *open-source*) são sintaticamente tão corretos do que os diagramas gerados por ferramentas proprietárias.

Referências

Ambler, S. C. The Elements of UML Style. São Paulo: Cambridge University Press, 2003. 146p.

Basili, V.; Rombach, H. Goal question metric paradigm. In: Encyclopedia of Software Engineering, v. 2, 1994.

Booch, G.; Rumbaugh, J.; Jacobson, I. UML: Guia do usuário. 2ª Edição. Rio de Janeiro: Elsevier, 2005. 474 p.

Braga, M. A diagram is not a model: The huge difference between them.Disponível em: https://www.ibm.com/developerworks/community/blogs/invisiblethread/entry/a_diagram_is_not_a_model_the_huge_difference_between_them?lang=em. Acesso em 30 de março de 2014.

Chen, Z; Zhenhua, D. Specification and Verification of UML2.0 Sequence Diagrams using Event Deterministic Finite Automata. Fifth International Conference on Secure Software Integration and Reliability Improvement, 2011.

Chupac, L.; Mudron, I.; Kana, K.. COMPARISON OF UML TOOLS. 13th International Multidisciplinary Scientific GeoConference SGEM, 2013.

Dong, X.; Miao, H.; Philbert, N. Model Checking UML Activity Diagrams in FDR+. Eigth IEEE/ACIS International Conference on Computer and Information Science. 2009.

Hnatkowska, B. Verification of Good Design Style of UML Models, Proc. Int. Conf. Information System Implementation and Modeling, 2007.

Khaled, L. A comparison between UML tools. Second International Conference on Environmental and Computer Science, 2009.

Object Management Group. Disponível em: http://doc.omg.org/formal/2005-07-05.pdf. Acesso em 29 de março de 2015.

Soeken, M.; Wille, R.; Drechsler, R. Verifying Dynamic Aspects of UML Models. Design, Automation & Test in Europe Conference & Exhibition, 2011.

Sommerville, Ian. Engenharia de Software. 9ª edição. Pearson Prentice Hall, 2011. 529 p.