APRENDIZAGEM DE DESIGN PATTERNS UTILIZANDO MAPAS CONCEITUAIS

LEARNING DESIGN PATTERNS USING CONCEPT MAPS

Oswaldo de Souza Dutra[48]
Sergio Akio Tanaka, Simone Sawasaki Tanaka[49]

RESUMO

Este trabalho teve como objetivo destacar a importância do conhecimento de design patterns entre desenvolvedores e como a aplicação dos mapas conceituais pode auxiliar no seu ensino e aprendizagem. Para tal finalidade, um estudo sobre o design patterns Factory Method foi utilizado para demonstrar como a aplicação da técnica é importante em diferentes situações durante o desenvolvimento de software.
PALAVRAS-CHAVE: Design Patterns. Mapas Conceituais.

ABSTRACT

This work's goal is to show the importance of the knowledge in design patterns between developers and how the concept maps can be used to help teaching and learning them. For this purpose, a study about the design pattern Factory Method was used to demonstrate how important the application of this technique is during the software development.
KEYWORDS: Design Patterns. Concept Maps.

1. INTRODUÇÃO

Aproximadamente três décadas já se passaram desde o surgimento dos métodos orientados a objetos para a criação de software, e ainda hoje este paradigma que procura traduzir conceitos do mundo real em objetos que podem ser compreendidos pelos computadores vem sendo amplamente utilizado na criação de sistemas de informação. Diante de tal realidade, é esperado dos profissionais que trabalham com o desenvolvimento de sistemas comerciais o domínio de conceitos e ferramentas que fazem parte da disciplina de orientação a objetos, como por exemplo, a *Unified Modeling Language* (UML) e linguagem de programação orientada a objetos.

No entanto, além dos conhecimentos de ferramentas e conceitos, o desenvolvimento orientado a objetos exige do desenvolvedor a capacidade de definir, relacionar, organizar objetos hierarquicamente e ainda manter a estrutura do software genérica o suficiente para possibilitar futuras mudanças e reaproveitamento de código em projetos que utilizam o mesmo contexto. A necessidade dessas habilidades torna o desenvolvimento orientado a objetos complexo e exige experiência dos profissionais da área. Com base nesta realidade, é de fundamental importância que o desenvolvedor de software tenha conhecimento dos *design patterns* e princípios aplicados na estruturação de sistemas.

Os *design patterns* documentam um conjunto de boas práticas aplicadas no desenvolvimento de software para a solução de problemas recorrentes da área, são soluções elaboradas por desenvolvedores experientes e que tiveram sua eficiência comprovada em softwares já desenvolvidos. Assim, seu estudo permite ao desenvolvedor iniciante a exposição a situações ocorridas e solucionadas em experiências anteriores da comunidade.

Embora seja fundamental, existem dificuldades encontradas na aprendizagem e compreensão dos *design patterns*, este trabalho procurou encontrar uma forma de facilitar

48 Pós-Graduado

49 Centro Universitário Filadélfia de Londrina - UniFil

o aprendizado utilizando a técnica de *workflow* em conjunto com os mapas conceituais para proporcionar um melhor entendimento do assunto. Mostra também, como a aplicação de *design patterns* e um desenvolvimento guiado por princípios aplicados no desenvolvimento de *software* podem melhorar a qualidade do código fonte de um sistema.

Como base para este trabalho foi escolhido um dos principais *design patterns* utilizados no desenvolvimento de *software*, que são sugeridos por Gamma et al, o *design pattern* utilizado foi o *Factory Method*.

2. DESIGN PATTERNS

A técnica de *design pattern* foi inicialmente aplicada por Christopher Alexander na década de 70, para ser utilizada na área de arquitetura. Alguns anos depois a técnica foi introduzida na área de desenvolvimento de *software*. A técnica consiste em encontrar situações problemáticas que ocorrem com frequência e documentar soluções para elas, de forma que possam ser reaproveitadas no futuro por outros profissionais.

A utilização de *design patterns* no desenvolvimento de software foi popularizada após o lançamento do livro ***Design Patterns***: *Elements of Reusable Object-Oriented Software*, escrito por quatro grandes nomes da Engenharia de *Software* (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) também referenciados como GoF (Gang of Four), o livro descreve os *design patterns* em forma de catálogo, contendo os padrões que os autores reconheciam até o momento.

Os *design patterns* não são específicos para uma determinada linguagem de programação, eles são descritos de forma abstrata, de forma que toda linguagem orientada a objetos pode ser utilizada na sua aplicação. Embora em alguns casos a aplicação de *design patterns* exija um pouco mais de trabalho, esse esforço é recompensado com o ganho em flexibilidade e reutilização de código (GAMMA et al., 1995).

Os padrões de projetos são divididos em três categorias, *Creational* (Criação), *Structural* (Estrutura) e *Behavioral* (Comportamento), cada padrão de projeto catalogado contém as seguintes informações:

- **nome e classificação:** o nome transmite de forma sucinta a essência do padrão, enquanto sua classificação indica a qual categoria ele pertence.
- **intenção:** em poucas palavras explica o que o padrão faz e quais problemas ele resolve.
- **motivação:** uma situação em que o padrão pode ser aplicado e como a estrutura que compõe o padrão resolve o problema.
- **aplicabilidade:** em quais situações o padrão pode ser aplicado, como reconhecer essas situações e quais erros ele evita de serem cometidos.
- **estrutura:** representação gráfica dos objetos que compõem o padrão.
- **participantes:** classes e outros objetos que compõem o padrão e suas responsabilidades.
- **colaborações:** como as partes envolvidas colaboram umas com as outras.
- **consequências:** quais os prós e contras da utilização do padrão.
- **implementação:** o que é preciso conhecer para implementar o padrão em termos de técnicas e recursos de linguagem de programação.

- **código de exemplo:** código fonte em linguagem de programação que exemplifica a aplicação do padrão.
- **usos conhecidos:** situações reais nas quais o padrão já foi aplicado com sucesso.
- **padrões relacionados:** quais padrões são parecidos e qual a diferença entre eles e com quais outros padrões pode ser combinado.

A Figura 1 mostra os principais *design patterns* separados em categorias:

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method (83)	Adapter (108)	Interpreter (191) Template Method (254)
	Object	Abstract Factory (68) Builder (75) Prototype (91) Singleton (99)	Adapter (108) Bridge (118) Composite (126) Decorator (135) Facade (143) Proxy (161)	Chain of Responsibility (173) Command (182) Iterator (201) Mediator (213) Memento (221) Flyweight (151) Observer (229) State (238) Strategy (246) Visitor (259)

Figura 1 – *Design Patterns*

Fonte: (GAMMA et al., 1995)

Além de auxiliar na estruturação de sistemas e produção de código de qualidade, os *design patterns* possibilitam uma comunicação eficiente entre desenvolvedores de *software*, de forma que esses possam discutir o desenvolvimento em uma linguagem comum.

3. MAPAS CONCEITUAIS

Os mapas conceituais também conhecidos como mapas de conceitos, são diagramas que indicam relações entre conceitos, ou entre palavras utilizadas para representar conceitos (MOREIRA, 1997).

Uma das características dos mapas conceituais é a representação de conceitos de forma hierárquica, onde os conceitos mais gerais ficam no topo e os mais específicos, são organizados hierarquicamente abaixo (NOVAK, 2008).

Os mapas conceituais foram desenvolvidos em 1972 durante um programa de pesquisa de Novak em Cornell, onde teria sido solicitado a ele entender as mudanças no conhecimento de ciências das crianças. Durante os estudos, pesquisadores entrevistaram crianças, e eles encontraram dificuldades para identificar mudanças específicas no

entendimento de conceitos relacionados a ciências, examinando apenas as entrevistas realizadas. Esse programa foi baseado na psicologia da aprendizagem de David Ausubel. A ideia fundamental na psicologia de Ausubel é que a aprendizagem ocorre com a assimilação de novos conceitos, proposições em conceitos existentes e estruturas de raciocínio criadas pelo aluno. Tal estrutura de conhecimento é conhecida também como estrutura cognitiva individual. Diante da necessidade de encontrar a melhor maneira de representar o entendimento conceitual das crianças, os pesquisadores tiveram ideia de representar o conhecimento delas na forma de um mapa conceitual (NOVAK, 2008).

Para o desenvolvimento dos mapas conceituais, podem ser utilizadas figuras geométricas para identificar os conceitos, porém não existe uma regra específica sobre quais formas utilizar. Entretanto, cada autor pode escolher suas próprias regras no momento da elaboração de um mapa conceitual, como, por exemplo, definir que conceitos mais gerais, mais abrangentes, devem estar dentro de elipses e conceitos bem específicos dentro de retângulos. Embora as formas utilizadas no diagrama não tenham um valor significativo, o fato de dois conceitos estarem unidos por uma linha é importante, isso significa que existe uma relação entre esses conceitos, de acordo com o entendimento do autor do mapa conceitual (MOREIRA, 1997).

A Figura 2 mostra um exemplo de mapa conceitual:

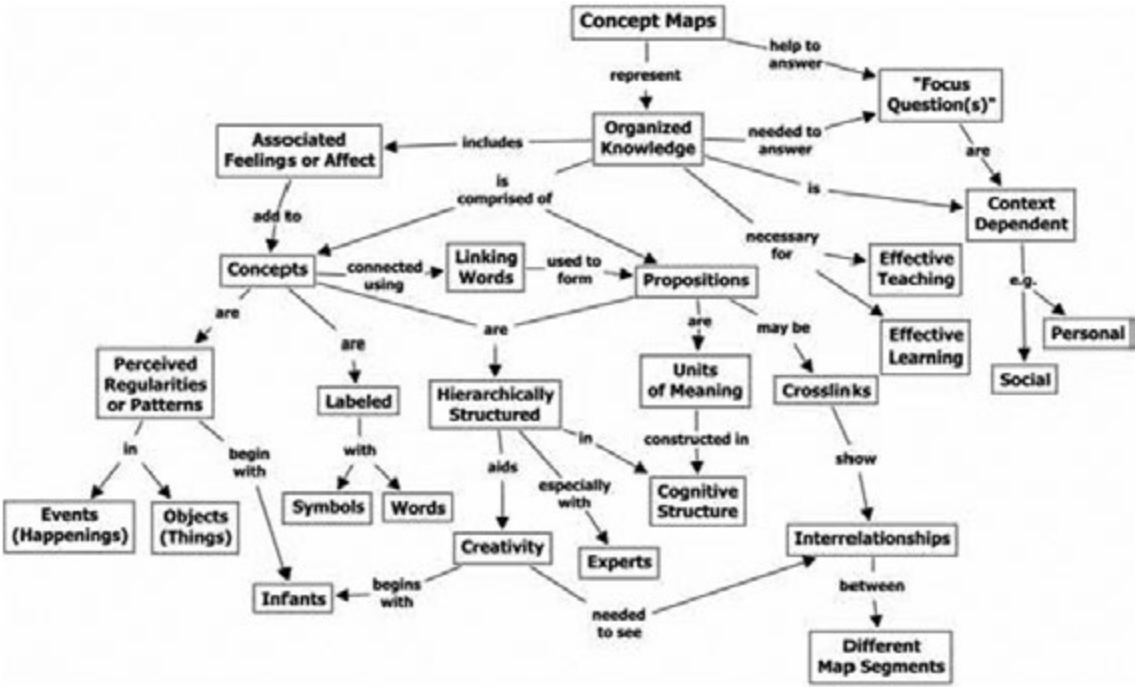

Figura 2 – Exemplo de mapa conceitual

Fonte: <http://cmap.ihmc.us/Publications/ResearchPapers/TheoryUnderlyingConceptMaps.pdf>

O mapa conceitual é uma ferramenta capaz de auxiliar tanto no ensino quanto na aprendizagem. Uma vez que possibilita a evidência de significados, unindo conceitos de forma a criar um conjunto interligado de conhecimentos isolados, formando uma teia a partir das relações entre eles, possibilitando o desenvolvimento cognitivo de quem o utiliza (TANAKA, 2011).

4. UNIFIED MODELING LANGUAGE - UML

A UML é uma linguagem de modelagem, não deve ser confundida com um

método. Uma vez que um método deve consistir, pelo menos em princípio, tanto em uma linguagem quanto em um processo de utilização. A linguagem de modelagem é apenas a notação gráfica, e deve ser utilizada em conjunto com um método. Um método utiliza processos para orientar sobre quais medidas tomar ao fazer um projeto (FOWLER, 2000).

A UML começou a ser definida por Rumbaugh e Grady Booch que tentaram combinar dois métodos populares utilizados na modelagem de aplicações orientadas a objeto, são eles: Booch e *Object Modeling Language* (OMT). Algum tempo depois, Ivar Jacobson, o criador do método *Objectory*, uniu-se aos dois (TANAKA, 2011).

Atualmente a UML possui 14 diagramas que estão divididos em duas categorias, Estruturais e Comportamentais, os diagramas estruturais dizem respeito aos objetos utilizados na construção de um sistema, enquanto os diagramas de comportamento ilustram a interação entre as partes componentes do sistema.

A Figura 3 mostra os diagramas da UML organizados em suas categorias:

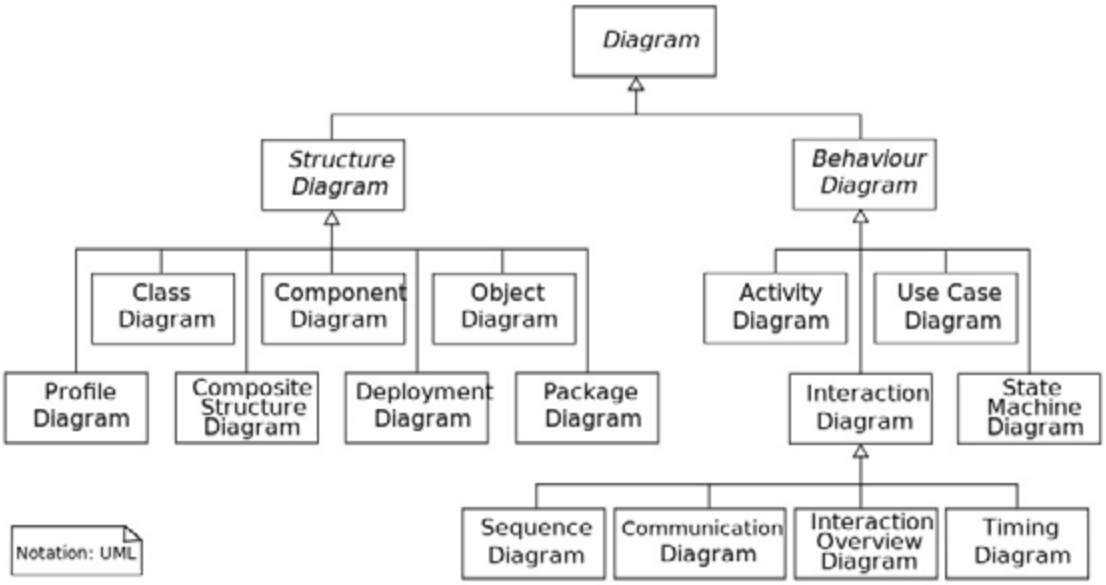

Figura 3 – Diagramas da UML

Fonte: < http://en.wikipedia.org/wiki/Unified_Modeling_Language>

Segundo o *Object Management Group* (OMG), atualmente a UML é sua especificação mais utilizada, e também a linguagem mais adotada para a modelagem de estrutura, comportamento e arquitetura de aplicações, além disso, pode ser utilizada para modelar processos de negócios e estrutura de dados (http://www.uml.org).

5. IMPLEMENTAÇÃO DO MAPA CONCEITUAl E WORFLOW

Levando em consideração a dificuldade de aprendizado dos *design patterns* entre alunos e também a dificuldade no ensino desse conteúdo, foram elaborados *workflows* e mapas conceituais dos principais *design patterns* utilizados no desenvolvimento de *software*.

Através dessas duas técnicas foi criada uma estrutura que tem como objetivo fornecer uma visualização da lógica aplicada para determinar a aplicação do *design pattern* selecionado, e também as técnicas de orientação a objetos e princípios de *software* relacionados. Para a criação do *workflow*, foi utilizada a ferramenta BizAgi Process Modeler e para criação dos mapas conceituais, foi utilizada a ferramenta IHMC

CmapTools.

A Figura 4 mostra o mapa conceitual de conceitos relacionados aos *design patterns*.

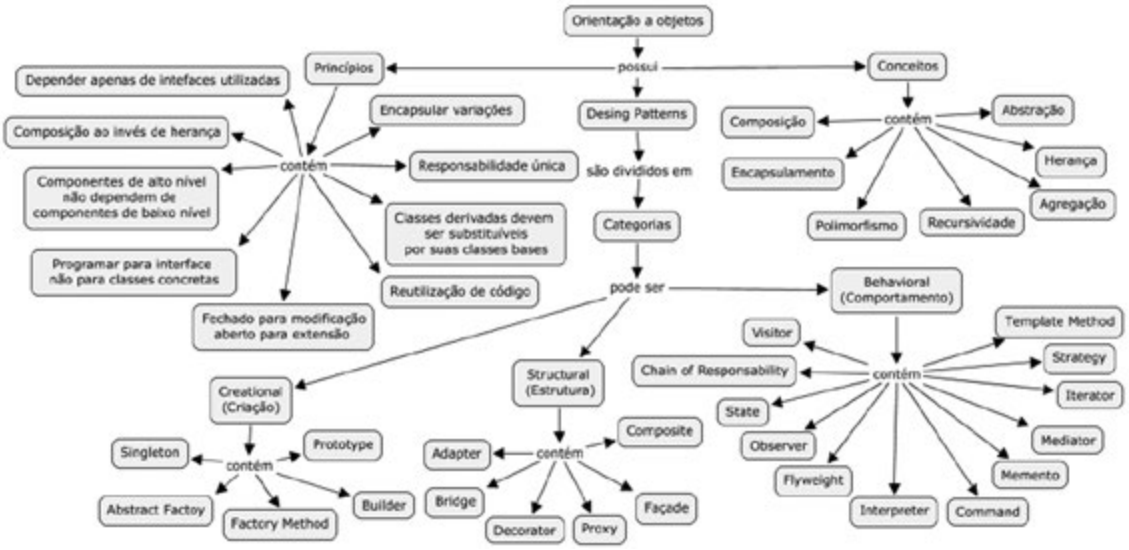

Figura 4 – Mapa conceitual dos conceitos de design pattern

Na seção seguinte, será apresentado um mapa conceitual que relaciona o padrão de projeto abordado neste trabalho aos seus princípios de desenvolvimento de *software* e conceitos de orientação a objetos.

5.1 MAPA CONCEITUAL E WORKFLOW DO DESIGN PATTERN FACTORY METHOD

O *Factory Method* é um padrão de projeto utilizado para controlar a dependência entre classes de um sistema. Devido ao fato de que um sistema é suscetível a mudanças, é necessário adotar estratégias para controlar o efeito que cada alteração pode provocar no sistema como um todo. A dependência entre classes é uma fonte de problemas de manutenção quando não tratada corretamente. Identificar áreas do sistema sujeitas a mudanças não é uma tarefa simples, por isso, para auxiliar o processo de identificação o princípio a seguir pode ser utilizado como diretriz:

- **Programar para uma interface não para uma classe concreta**: Segundo (Gamma et al), quando a técnica de herança é aplicada de forma correta, todas as classes derivadas de uma classe abstrata ou que implementam uma interface, são consideradas membros de uma família de objetos que possuem interfaces idênticas, de forma que qualquer dos objetos pode responder a qualquer operação presente na interface. Dessa forma, classes que utilizam as operações executadas por uma determinada família de objetos, não fazem referência direta a um objeto, mas sim a interface comum entre eles.

O padrão de projeto *Factory Method* consiste na criação de um método responsável pelo processo de instanciação de objetos utilizados por uma determinada classe do sistema. Sua responsabilidade é manter a classe ignorante em relação ao objeto concreto, permitindo que a classe dirija à execução de operações ao objeto instanciado por meio de uma interface que a classe conhece. Assim, temos ainda o benefício de proteger a classe cliente das alterações ocorridas nas classes auxiliares, evitando a propagação de futuras

alterações ocorridas nessas classes, pode-se dizer então que a classe cliente está protegida contra alterações originadas em classes com as quais tem dependência.

A Figura 5 mostra o *workflow* para a tomada de decisão da aplicação do padrão *Factory Method.*

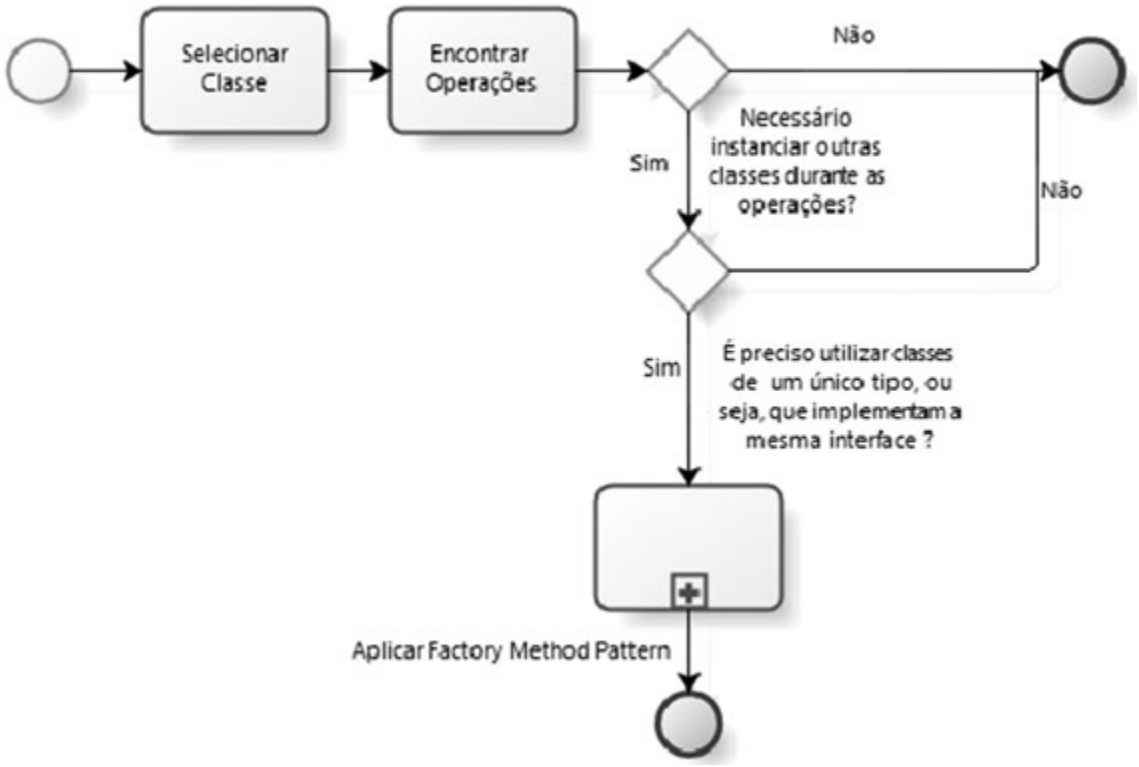

Figura 5 – *Workflow* de tomada de decisão do padrão *Factory Method*

Como pode ser observado na Figura 5 existem duas condições para que seja necessária a aplicação do *Factory Method,* a primeira é que seja necessária a utilização de outros objetos para auxiliar uma classe a executar suas operações e a segunda é que os objetos utilizados pertençam a uma única família.

A Figura 6 exibe o diagrama UML simplificado do padrão *Factory Method.*

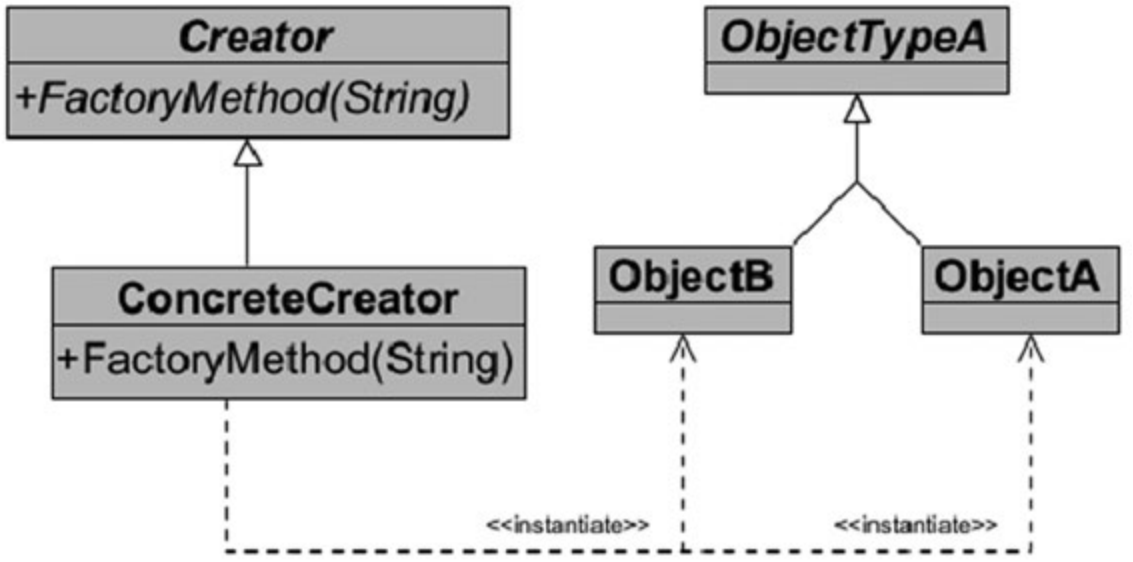

Figura 6 – Diagrama UML do padrão *Factory Method*

A Figura 7 mostra o *workflow* para a aplicação do padrão *Factory Method.*

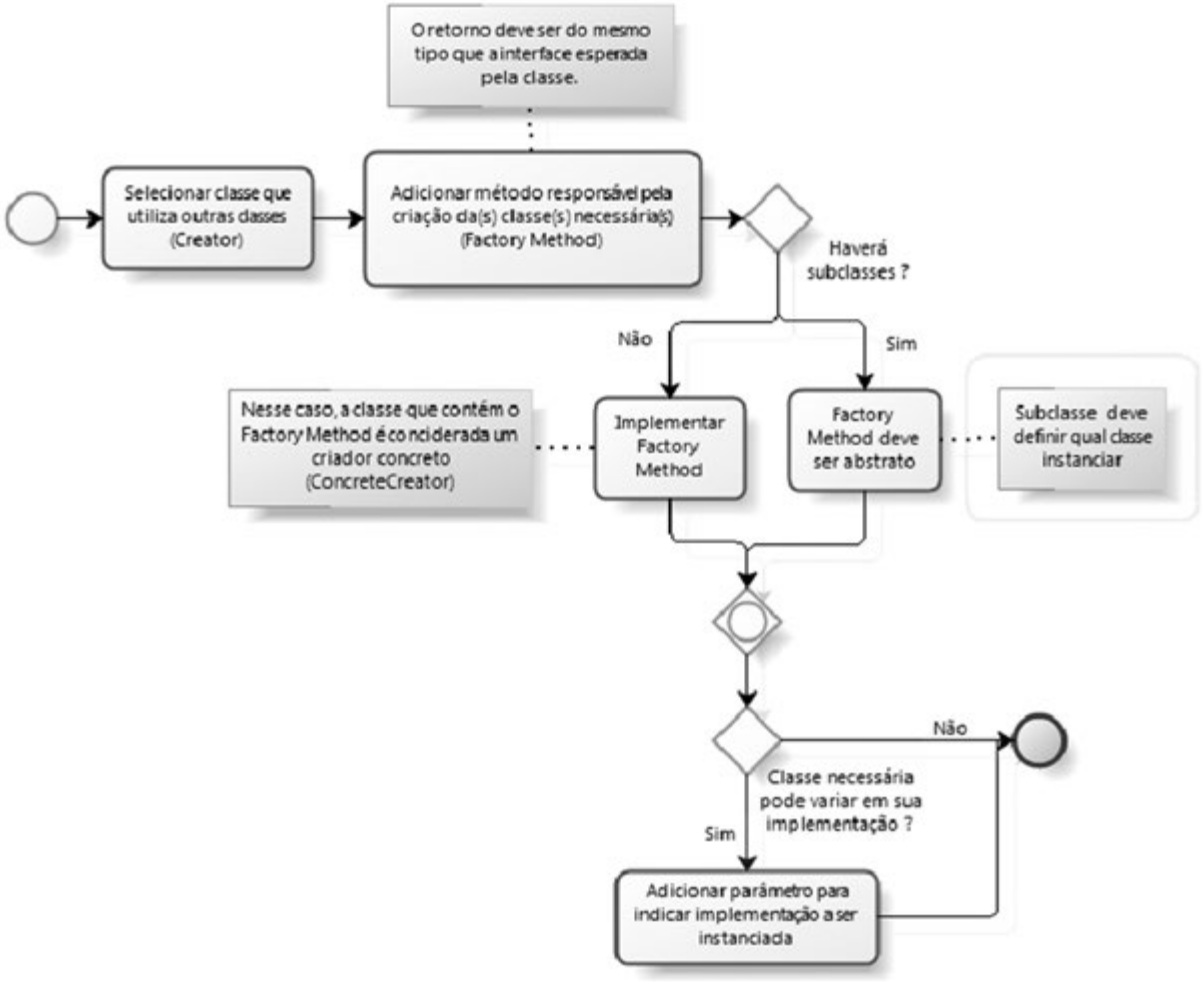

Figura 7 – *Workflow* de aplicação do padrão *Factory Method*

Como exibido na Figura 7, o primeiro passo é identificar a classe (*C1*) que irá receber o *Factory Method (FM)*, ou seja, qual classe necessita executar operações de um objeto externo (*O1*).

O segundo passo é criar o método *(FM)* responsável por retornar uma instância do objeto (*O1*) necessário, entretanto o tipo do retorno desse método *(FM)* deve ser igual ao da interface (*I1*) implementada pelo objeto (*O1*) externo. Dessa forma, sempre que a classe (*C1*) precisar acionar uma operação do objeto (*O1*) externo, o *Factory Method (FM)* deve ser acionado e seu retorno armazenado em uma variável do tipo (*I1*) retornado, ou seja, do tipo da interface (*I1*) implementada pelo objeto (*01*) externo.

Existe a possibilidade de que a classe que contém o *Factory Method* seja uma classe abstrata, nesse caso o *Factory Method,* pode ser marcado como abstrato, para que as subclasses forneçam sua implementação. É possível também que diferentes objetos forneçam implementações diferentes para a mesma interface utilizada pela classe, e que podem ser necessários, neste caso o *Factory Method* deve possuir um parâmetro de entrada especificando qual dos objetos possíveis deve ser retornado.

A Figura 8 mostra o mapa conceitual do padrão de projeto *Factory Method,* onde é possível observar os conceitos e técnicas de orientação a objetos relacionados ao padrão.

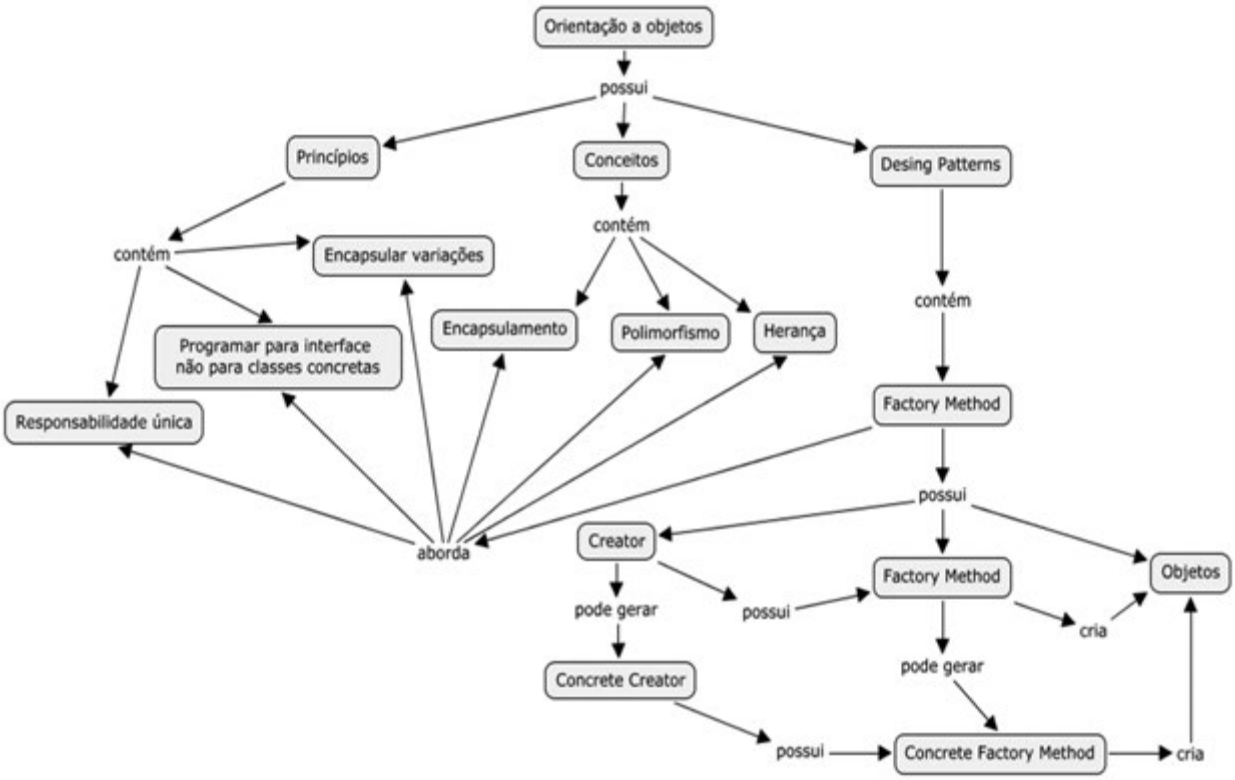

Figura 8 – Mapa conceitual do padrão *Factory Method*

6. CONSIDERAÇÕES FINAIS

Este trabalho apresentou como é possível utilizar mapas conceituais e *workflows* no ensino e aprendizagem de *design patterns*. Utilizando as técnicas em conjunto, foi possível criar uma linha de raciocínio, relacionar os conceitos de orientação a objetos e princípios que guiam de desenvolvimento de software envolvidos na aplicação do *design pattern* abordado. Esta pesquisa serve como base para que novos estudos sejam feitos em relação à simplificação do aprendizado em áreas do desenvolvimento de *software* que necessitam da compreensão de vários conceitos relacionados para se chegar a uma única técnica, como por exemplo, o gerenciamento de *software*.

Embora muitas ferramentas que facilitam o desenvolvimento orientado a objetos estejam disponíveis, como *frameworks* e bibliotecas, é preciso que o desenvolvedor conheça e saiba aplicar os *design patterns* e princípios de *software* durante a estruturação de sistemas. As tecnologias de *workflow* com os mapas conceituais vêm de encontro com essa necessidade de conhecimento. Tal conhecimento permite uma maior compreensão de como cada parte de um sistema é planejada e como se relaciona com as demais, uma habilidade fundamental para compreender a arquitetura de sistemas já desenvolvidos e também para criação de novos.

REFERÊNCIAS

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES. John. **Design Patterns**: Elements of Reusable Object Oriented Software. Canada - Toronto: Addison-Wesley, 1995.

FREEMAN, Eric; FREEMAN, Elisabeth; BATES, Bert; SIERRA, Kathy. **Head First Design Patterns**. USA – Sebastopol: O'Reilly, 2004.

MARTIN, Robert Cecil; MARTIN, Micah. **Agile**: Principles, Patterns, and Practices in C#.

USA – New Jersey: Prentice Hall, 2006.

HAMILTON, Kim; Miles, Russell. **Learning UML 2.0.** USA – Sebastopol: O'Reilly, 2006.

FOWLER, Martin; KENDALL, Scott. **UML Distilled**. Second Edition. Canada - Toronto: Addison-Wesley, 2000.

TANAKA, Simone Sawasaki. **O poder da tecnologia de workflow e dos mapas conceituais no processo de ensino e aprendizagem da UML**. 2011. 116 fls. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2011.

MOREIRA, Marco Antonio. **Mapas Conceituais e Aprendizagem Significativa**. 1997. Disponível em: <http://www.if.ufrgs.br/~moreira/mapasport.pdf>, Acesso em Novembro 2013.

NOVAK, Joseph D. **The Theory Underlying Concept Maps and How to Construct and Use Them**. Technical Report IHMC CmapTools 2006-01 Rev 01-2008. Disponível em: <http://cmap.ihmc.us/Publications/ResearchPapers/TheoryUnderlyingConceptMaps.pdf>, Acesso em Novembro 2013.

OBJETCT MANAGEMENT GROUP. **Getting Started With UML**. Disponível em: <http://www.uml.org>, Acesso em Outubro 2013.