Uma abordagem baseada em problemas para o ensino de Padrões GRASP

David Moises Barreto dos Santos

Departamento de Ciências Exatas – Universidade Estadual de Feira de Santana (UEFS)
Km 03, BR-116 Campus Universitário – 44.031-460 – Feira de Santana – BA – Brasil

davidmbs@uefs.br

Abstract. *In short, the main advantage of patterns is the reuse of good ideas. Therefore, they had become so popular in Software Engineering area. In this context, GRASP patterns are very important because they describe basic principles of oriented-object design. In this way, it is extremely important the efficacious education of this curricular component. In this direction, this paper presents an experience of teaching of such component of a Computation Engineering course, using the problem/project-based learning as didactic-pedagogical approach. Besides the methodology promotes the learning sediment on practice, it develops techniques and no techniques abilities.*

Resumo. *Em suma, a principal vantagem de padrões é o reuso de boas idéias, por isso, se tornaram tão populares na área de Engenharia de Software. Neste contexto, padrões GRASP tem grande importância por descrever princípios básicos de projeto orientado a objetos. Deste modo, é extremamente importante o ensino eficaz deste componente curricular. Neste sentido, este artigo apresenta uma experiência de ensino de tal componente em um curso de Engenharia de Computação, usando o aprendizado baseado em problemas como abordagem didático-pedagógica. A metodologia promove aos estudantes, além do aprendizado sedimentado na prática, o desenvolvimento de habilidades técnicas e não técnicas.*

1. Introdução

Padrões têm como objetivo fornecer um conjunto de pares problema-solução para a criação de software, de forma que estes sejam genéricos suficiente para resolverem problemas recorrentes em diferentes domínios, como pode ser constatado em [Gamma *et al*, 1995][Fowler, 2002][Larman, 2004].

Mais especificamente, existem os Padrões GRASP (*General Responsibility Assignment Software Patterns*) que descrevem princípios fundamentais de projeto baseado em objetos e atribuição de responsabilidades aos mesmos [Larman, 2004]. Assim, é de suma importância que desenvolvedores de software iniciantes sejam capazes de compreender e aplicar tais padrões, pois esses princípios são básicos para um projeto de software. Consequentemente, é essencial ter uma metodologia adequada para o ensino deste componente.

Neste sentido, a principal contribuição deste trabalho é apresentar uma metodologia baseada no aprendizado baseado em problemas (PBL – *Problem-Based*

Learning) [Delisle, 1997] para introduzir padrões GRASP, assim como relatar a experiência de sua aplicação na disciplina Padrões e Frameworks do curso de Engenharia de Computação na Universidade Estadual de Feira de Santana (UEFS), em Feira de Santana – BA.

A metodologia PBL difere daquelas tradicionais, que baseiam-se essencialmente em aulas expositivas e provas. Em contrapartida, na filosofia PBL, um problema é apresentado aos alunos com o intuito de os mesmos explorarem o domínio da solução através de sua capacidade auto-didática. Para direcionar o aprendizado, são indicados recursos de aprendizagem (livros, artigos, vídeos, entre outros) apropriados à atividade designada. Todavia, os estudantes também são estimulados a buscar outras fontes de conhecimento.

A aplicação da metodologia na disciplina é apresentada na Seção 2. Na Seção 3, é relatado todo o estudo de caso, enquanto que, na Seção 4, é apresentada a conclusão, assim como trabalhos futuros.

2. Aprendizado Baseado em Problemas

Nesta abordagem, a carga horária da disciplina era dividida em aulas teóricas (aulas expositivas) e práticas (seções tutoriais). Neste contexto, as aulas teóricas podem ter diferentes características: (1) palestra, que visa a dar uma visão geral de um determinado assunto, esteja ele relacionado ou não com os problemas trabalhados; (2) aprofundamento, que objetiva apresentar minuciosamente um tema mais complexo, de mais difícil compreensão por parte dos alunos; (3) consultoria, cujo propósito é dirimir dúvidas a respeito de tópicos relacionados ao problema. Pode acontecer ainda um mesclado destes tipos de aulas, por exemplo, uma aula pode-se caracterizar tanto palestra quanto aprofundamento, a partir do momento em que é exposta uma visão geral de um determinando assunto, mas em certo momento é necessário aprofundar um de seus sub-tópicos.

Em síntese, nas seções tutoriais, ocorre a discussão entre os alunos a respeito do problema apresentado. Neste momento, o professor assume a função de tutor e, durante a discussão, suas principais atividades são: (1) promover a uniformidade da discussão entre os alunos, estimulando o tímido, bem como "podando" aquele que fala demais; (2) formular questões apropriadas para que os alunos enriqueçam suas discussões, quando apropriado; (3) favorecer o bom relacionamento de todos os envolvidos, ajudando a construir um ambiente de respeito mútuo.

Como a turma tinha em torno de 12 alunos, a mesma foi dividida em dois grupos de seis alunos para uma melhor discussão – segundo Woods (1996), um grupo deve conter até dez estudantes, no máximo, para um melhor aproveitamento. Portanto, nesta turma, adotou-se o modelo de tutor flutuante [Duch *et al*, 2001], onde o tutor dedica um período do tempo em cada grupo formado.

Uma vez dividido os grupos, era então dado início a seção tutorial de fato, que, baseado em [Delisle, 1997], resumidamente, respeita os seguintes passos:

1. Ponto de Partida: o problema é apresentado aos alunos, e então lido e interpretado;

2. *Brainstorming*: os estudantes associam idéias ao problema. Nesta etapa, é importante que não haja críticas em relação as idéias associadas, pois, desta forma, pode-se perder boas contribuições e/ou desestimular um aluno menos participativo;
3. Sistematização: aqui, as idéias mais relevantes são selecionadas e agrupadas. Neste momento, é importante discutir as idéias criticamente, argumentado o porquê de cada opinião;
4. Apresentação: após a sistematização, cada grupo expõe suas idéias para toda a turma no intuito de compartilhar o conhecimento. Em seguida, há um momento para uma discussão inter-grupos do que foi apresentado;
5. Metas de Aprendizagem: de volta aos seus grupos, os alunos estabelecem metas, buscando desenvolver um plano de ação para descobrir a solução do problema. Até a definição das metas, pode-se expor também informações relevantes (fatos) ou levantar questionamentos, tudo em prol da busca pela resolução do problema.

Com o plano de ação definido, os alunos devem buscar novos conhecimentos para que conceitos sejam esclarecidos e novas idéias sejam criadas no intuito de conceber a solução do problema. Neste sentido, na seção tutorial seguinte, a discussão é retomada e todo este ciclo é repetido. Esta iteração acontece até a última seção prevista para o problema apresentado ou até a solução ser de fato construída.

Periodicamente, ainda há uma avaliação de todo este ciclo a fim de analisar se o mesmo está tendo um bom andamento ou não. Caso não esteja, cabe aos membros do grupo identificar as melhorias e buscar corrigi-las para um melhor aproveitamento de toda a equipe.

O formato de um problema apresentado aos alunos pode ser visualizado na Figura 1, que descreve justamente este primeiro problema. Inicialmente, é indicado o tema bem como objetivos de aprendizagem, isto é, o que se espera que os estudantes aprendam com o problema.

Em seguida, é dada a descrição do problema propriamente dito. A seção produto aponta o que deve ser entregue como resultado da solução do problema, que neste caso consta de um relatório mais um código. Por fim, são recomendados alguns recursos através dos quais os estudantes poderão adquirir conhecimento a respeito do tema trabalhado. Questões importantes sobre como elaborar bem um problema podem ser encontradas em [Deslile, 1997][Duch *et al*, 2001].

Vale destacar que esta abordagem é aplicada ao longo de todo o curso de Engenharia de Computação, em diversas disciplinas, desde o primeiro semestre até os últimos. Portanto, neste caso, não houve dificuldades de adaptação por parte dos alunos ou do professor.

3. Estudo de caso

Padrões GRASP é um dos temas que compõem o conteúdo programático da disciplina Padrões e Frameworks, um componente optativo do currículo do curso de Engenharia da Computação da Universidade Estadual de Feira de Santana (UEFS). Em suma, a disciplina, como o próprio nome sugere, é dedicada ao ensino de padrões. Mais

especificamente, além dos padrões GRASP, também fazem parte do conteúdo programático da disciplina os padrões de projeto [Gamma *et al*, 1995], padrões arquiteturais [Fowler, 2002], padrões de refatoração de código [Fowler, 2000] e frameworks [Landin & Niklasson, 1995].

Tema: Padrões GRASP

Objetivos de aprendizagem: Iniciar-se no uso de padrões para desenvolvimento de software; Compreender padrões para atribuição de responsabilidade (GRASP); Analisar qual padrão é mais adequado dada uma situação; Aplicar padrões GRASP.

Problema
Existe uma comunidade indígena que é composta de várias tribos. Cada tribo tem um nome, um cacique e fica localizada em uma determinada região. Um índio pertence a uma e somente uma tribo . Cacique também é considerado índio, e só passa a ser cacique quando assume a coordenação de uma tribo. Cada tribo pode ter diversos animais das mais variadas espécies. Frequentemente, acontece uma feira onde animais são trocados entre as tribos pelos seus respectivos caciques. Caso haja um cacique interessado em alguma troca, mas não há outro disposto a realizá-la, logicamente, não poderá ocorrer a troca. Um animal nunca é vendido, pois a comunidade não trabalha com dinheiro.
Infelizmente, devido ao grande crescimento da comunidade nos últimos anos, a demanda de troca está muito alta, provocando assim, uma confusão na organização que administra estas trocas. Para isto, foi contratada uma empresa de software para desenvolver um sistema que facilitasse isto. O desenvolvedor responsável pela criação do sistema começou a fazê-lo, porém foi demitido pouco tempo depois por estar fazendo um trabalho muito ruim. O projeto estava sendo mal elaborado, ferindo padrões essenciais de orientação a objetos.
O dono da empresa ficou sabendo que você é uma especialista em orientação a objetivos, principalmente, no que diz respeitos a padrões, então ele resolver contratá-lo para resolver tal problema.

Produto
Você deve aperfeiçoar projeto do software, através de alteração no código, usando seu conhecimento já adquirido de orientação a objetos e do uso de padrões. Em seguida, você deve escrever um relatório apontando e justificando, com argumentos consistentes, as melhorias realizadas, tanto aquelas que aplicou-se padrões quanto aquelas que não os usaram.

Deve ser entregue um arquivo compactado contendo o relatório, no formato pdf, e o código modificado. O arquivo deve ser postado até o dia 23/09 no Moodle (http://www2.uefs.br/moodle/mod/assignment/view.php?id=1680).

Recursos para aprendizagem
LARMAN, C. Utilizando UML e padrões: uma introdução a analise e projeto orientados a objetos. Porto Alegre: Bookman, 2004.
SAUVE, Jacques P. Notas de aulas: Padrões para atribuição de responsabilidade. Disponível em <http://jacques.dsc.ufcg.edu.br/cursos/map/html/map2.htm>

Figura 1. Formato do problema apresentado

Tal componente curricular tem uma carga horária de 60 horas e foi oferecido uma única vez, no período letivo 2006.1. O pré-requisito foi o Estudo Integrado temático Programação [Bittencourt & Figueiredo, 2003], cujo conteúdo programático abarca, basicamente, Estrutura de Dados e Orientação a Objetos. Vale destacar ainda que os alunos que cursaram a disciplina ou já tinham cursado Engenharia de Software ou estavam cursando no referido período letivo.

A seguir, é feita uma discussão a respeito do problema proposto, subseção 3.1, e a cerca da dinâmica dos alunos para resolvê-lo, subseção 3.2. Já o sistema de avaliação aplicado para avaliação do aprendizado pelos alunos é descrito na subseção 3.3. Por fim, na subseção 3.4, são discutidos os resultados alcançados com a aplicação do problema na disciplina.

3.1. Problema Proposto

O problema mostrado na Figura 1 foi o primeiro aplicado na disciplina, por os padrões GRASP dispor de princípios básicos necessários ao projeto de software, que, mais tarde, seria abordado de forma mais avançada. Para a discussão da solução, foram dedicadas duas seções tutoriais.

Para este problema, além da ficha apresentada, também foi disponibilizado o código para ser alterado. Em suma, a atividade a ser realizada consistia em aperfeiçoar o código passado, e elaborar um relatório apontando as alterações realizadas, além de indicar onde, por que e quais padrões foram aplicados na solução. O código disponibilizado pode ser visualizado através do apêndice deste artigo.

Com este problema, espera-se que o aprendizado dos estudantes seja motivado, principalmente, pelas seguintes discussões:

- Método Tribo.getNumPatas(): este método apresenta uma dependência muito grande para obter a quantidade de patas de animais de uma tribo. É preciso (1) analisar a espécie do animal para então (2) definir quantas patas ele tem, e, por fim, (3) fazer a soma. E é justamente nestes dois primeiros passos que está o problema. Sempre que surgir outra espécie de animal, será necessário colocar mais uma condição na estrutura de seleção presente. Por exemplo, se a Tribo passar a ter cavalos, será preciso mudar a implementação do método para acrescentá-lo na contagem de patas. Esta lógica obriga que Tribo conheça todas as espécies de animais. Desta forma, é mais coerente passar esta responsabilidade para o especialista da informação, segundo o padrão Expert;
- Método Animal.estouNestaTribo(Tribo tribo): este método informa se o objeto animal está contido ou não na coleção de animais da tribo, passada como parâmetro. O problema aqui é que a classe Animal é responsável por varrer uma coleção da classe Tribo. Ora se a coleção pertence a Tribo, então a mesma é que deveria fazer esta operação, como rege o padrão Especialista;
- Método Cacique.adicionaAnimalNaTribo(Tribo tribo, String nome, String especie): este método é responsável por criar instâncias da classe Animal e associá-las ao objeto Tribo passado como parâmetro. Desta forma, a criação de objetos não é realizada, por exemplo, por uma classe que agrega ou contém objetos da classe que está sendo instanciada. Fica claro então a necessidade da aplicação do padrão Creator.

Observe ainda que ao aplicar os padrões Expert e Creator nos respectivos casos, os padrões Baixo Acoplamento e Alta Coesão podem ser aplicados indiretamente também.

3.2. Resolução do problema

Na primeira seção tutorial dedicada ao problema, inicialmente, os alunos não conseguiam identificar problema algum no código. Com o decorrer da discussão, apesar de eles neste primeiro momento ainda não conhecerem padrões GRASP, foram observadas algumas sugestões coerentes por ambos os grupos.

Diante deste primeiro contato, o que conclui-se é que foi notável a falta de metodologia, por parte dos alunos, que não conseguiam apontar problemas no código de forma clara.

Já na segunda aula, depois de estudar os devidos padrões, os alunos vieram muito mais preparados, proporcionando uma melhor discussão, e assim, sugestões de melhor qualidade foram observadas. Mais especificamente, os padrões foram usados fazendo com as discussões fossem mais objetivas.

Diante do que foi apresentado, o problema foi proveitoso. O conteúdo dos relatórios entregues pelos alunos giravam em torno daqueles problemas existentes no código, apresentados na subseção anterior. A prática foi exercida também através das modificações do código, que, em sua grande maioria, estavam sintonizadas com o texto discorrido no relatório. Desta forma, os relatórios, bem como os códigos modificados, de uma forma geral, apresentaram uma boa qualidade.

3.3. Sistema de avaliação

A avaliação do problema foi dada através de uma nota composta pelo desempenho do aluno em cada seção tutorial (20%) e do produto final (solução) entregue (80%). Na primeira, o tutor avaliou o aluno tendo como critérios participação, contribuição, cumprimento das metas, respeito mútuo, entre outros.

Na segunda forma de avaliação, o tutor avaliou o produto gerado de acordo com o que foi especificado. Para a entrega da avaliação realizada, foi dedicada uma aula para discussão da solução com cada aluno com vistas a aperfeiçoar o produto entregue.

Vale destacar ainda que os alunos poderiam re-entregar o produto, tendo assim, a chance de fazer correções, bem como aprender com elas. Desta forma, na verdade, a nota do produto então foi uma ponderação da primeira entrega com a segunda.

3.3. Discussão

Será discutida nesta seção a experiência com esta abordagem, bem como os benefícios e dificuldades identificados ao longo de sua aplicação.

O aprendizado adquirido, através do método PBL, foi notável, principalmente devido ao seu caráter prático. Isto ficou evidente com os depoimentos dos alunos ao final da disciplina, por exemplo, foi relatado que toda vez que tem-se que elaborar o projeto de um software, padrões são a primeira coisa que vem em mente, principalmente, padrões de projetos. Inclusive, aqueles estudantes que estavam cursando Engenharia de Software expuseram que o conteúdo assimilado em Padrões e Frameworks foi bastante útil, principalmente, no desenvolvimento de software daquela disciplina.

Mais especificamente, quanto ao problema discutido neste artigo, o sucesso ficou evidente quando os alunos afirmaram que no início realmente eles não conseguiam "enxergar" problema algum no código exposto, e, depois do estudo, os problemas "começaram" a aparecer.

Todavia, também houve dificuldades, dentre as quais, a de maior grau relatada, pelos alunos, neste problema foi a difícil compreensão e aplicação dos padrões Alta Coesão e Baixo Acoplamento. Isto é compreensível por estes padrões serem mais abstratos do que Expert e Creator. Neste caso, foi necessário um esclarecimento em sala de aula para uma melhor compreensão.

Paralelamente as estas habilidades técnicas, com o método PBL, também foram trabalhadas outras habilidades não técnicas dos estudantes, essenciais em sua carreira, tais como:

- Trabalho em equipe: o novo conhecimento, necessário a resolução do problema, foi construído colaborativamente. Para tanto, foi preciso exercer uma boa forma de comunicação entre os membros para que opiniões fossem dadas e ouvidas. E nos casos de adversidade, um consenso foi estabelecido;
- Comunicação oral: em conseqüência deste trabalho em equipe, os estudantes puderam exercitar sua comunicação oral, ao argumentar a respeitos de suas opiniões nas discussões;
- Comunicação escrita: os alunos puderam aprimorar a redação através do relatório gerado. É importante ressaltar ainda que, com a avaliação contínua, re-entregando os relatórios após uma primeira correção, eles tiveram a oportunidade de aprender com os equívocos cometidos;
- Autodidatismo: o método PBL exige que: (1) o aluno busque o novo conhecimento, selecionando quais recursos de aprendizagem são úteis para a elaboração da solução; (2) uma vez com estes recursos em mãos, é preciso pensar criticamente para que a solução comece a ser construída; e (3) aplicar o que foi estudado e refletido no problema proposto. Deste modo, os estudantes aprendem a aprender, isto é, a ser autodidatas.

4. Considerações Finais

Padrões GRASP descrevem princípios básicos do projeto orientado a objetos. Tendo em vista a importância deste tema, faz-se necessário um ensino eficaz de tal componente nos cursos de graduação de computação.

Neste sentido, este artigo apresentou um trabalho que busca a melhoria contínua do ensino de padrões, através da disciplina Padrões e Frameworks oferecida em um curso de Engenharia de Computação. Para tanto, foi adotada um modelo diferente dos tradicionais: a aprendizagem baseada em problemas e/ou projetos. Além de proporcionar um aprendizado embasado na prática, que vem de encontro a este estilo de disciplina, a aplicação de PBL ajuda no desenvolvimento tanto de habilidades técnicas como não técnicas.

Entretanto, há alguns pontos que podem ser aperfeiçoados, principalmente, por ter sido a primeira vez que a disciplina foi oferecida no curso. Um deles é criar um projeto de software que tenha problemas relativos ao padrão Controlador.

Além disso, outros aspectos de codificação também podem ser abordados como variáveis mal nomeadas, falta de padronização de codificação, etc. Analisando um outro código, os estudantes podem perceber quão importante são estes detalhes.

Finalmente, serão aplicados questionários para coletar as opiniões dos alunos de forma objetiva, e, desta forma, poder-se-á apontar deficiências que podem ser suprimidas.

Referências Bibliográficas

Delisle, Robert. How to use problem-based learning in the classroom. ASCD: Alexandria, Virgina, EUA. 1997.

Duch, Bárbara J.; Groh, Susan E.; Allen, Deborah E. The power of problem-based learning: a practical how to for teaching undergraduate course in any discipline. Stylus Publishing: Sterling, Virgina, EUA. 2001.

Fowler, M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

Fowler, Martin. Refatoração: aperfeiçoando o projeto de código existente. Porto Alegre: Artmed Editora, 2000.

Gamma E. et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. Nova York: Addison-Wesley, 1995.

Landin, N.; Niklasson, A.: "Development of Object-Oriented Frameworks". Dissertação de Mestado. Department of Communication Systems, Lund University, 1995.

Larman, C. Utilizando UML e Padrões: Uma Introdução à Análise e ao Projeto Orientados a Objetos e ao Processo Unificado. 2 ed. Porto Alegre: Bookman, 2004.

Woods, Donald R. Problem-based Learning: resources to gain the most from PBL". Waterdown, ON, 1996.

Apêndice

```
public class Animal {
   private String nome;
   private String especie;

   public Animal(String aEspecie, String aNome)
{
especie = aEspecie;
nome = aNome;
   }

   public String getEspecie() {
return especie;
   }

   public boolean estouNestaTribo(Tribo tribo){
ArrayList<Animal> animais = tribo.getAnimais();
boolean achou=false;
Iterator<Animal> it = animais.iterator();
while ((it.hasNext()) && (!achou)){
   if (it.next().equals(this)){
achou = true;
   }
}

return achou;

   }

   public boolean equals(Object obj){
Animal a = (Animal) obj;
```

```
if ((this.nome.equalsIgnoreCase(a.nome)) &&
(this.especie.equalsIgnoreCase(a.especie))){
   return true;
}

return false;
   }

   public String getNome() {
      return nome;
   }

   public void setNome(String nome) {
      this.nome = nome;
   }
}

public class Indio {
   private String nome;

   public Indio(String nome){
this.nome = nome;
   }

   public String getNome() {
return nome;
   }

   public void setNome(String nome) {
this.nome = nome;
   }
}

public class Tribo {
   private ArrayList<Animal> animais;
   private ArrayList<Indio> indios;
   private Cacique cacique;

   public Tribo() {
animais = new ArrayList<Animal>();
   }

   public void adicionaAnimal(Animal animal) {
animais.add(animal);
   }

   public int getNumPatas() {
int result = 0;
for (Animal a : animais) {
   if (a.getEspecie().equals("Pato")) {
result += 2;
   } else if (a.getEspecie().equals("Cachorro")) {
result += 4;
   } else {
// ?
   }
}
return result;
   }

   public ArrayList<Animal> getAnimais() {
return animais;
   }

   public void setAnimais(ArrayList<Animal>
animais) {
this.animais = animais;
   }

   public void removeAnimal(Animal animal) {
this.animais.remove(animal);
   }

   public void addIndio(Indio indio) {
this.indios.add(indio);
   }

   public ArrayList<Indio> getIndios() {
return indios;
   }

   public void setCacique(Cacique cacique) {
this.cacique = cacique;
   }

   public Cacique getCacique() {
return this.cacique;
   }
}

public class Troca {
   public static void troca(Tribo tribo1, Animal
animal1, Tribo tribo2, Animal animal2){
tribo1.removeAnimal(animal1);
tribo1.adicionaAnimal(animal2);
tribo2.removeAnimal(animal2);
tribo2.adicionaAnimal(animal1);
   }
}


public class Cacique extends Indio {

   public Cacique(String nome) {
super(nome);
   }

   public void adicionaAnimalNaTribo(Tribo tribo,
String nome, String especie){
if
(tribo.getCacique().getNome().equals(this.getNo
me())){
   tribo.adicionaAnimal(new
Animal(especie,nome));
}
   }
}

public class Animal {
   private String nome;
   private String especie;

   public Animal(String aEspecie, String aNome)
{
especie = aEspecie;
nome = aNome;
   }
```

```
   public String getEspecie() {
return especie;
   }

   public boolean estouNestaTribo(Tribo tribo){
ArrayList<Animal> animais = tribo.getAnimais();

boolean achou=false;
Iterator<Animal> it = animais.iterator();
while ((it.hasNext()) && (!achou)){
   if (it.next().equals(this)){
achou = true;
   }
}

return achou;

   }
```

```
   public boolean equals(Object obj){
Animal a = (Animal) obj;

if ((this.nome.equalsIgnoreCase(a.nome)) &&
(this.especie.equalsIgnoreCase(a.especie))){
   return true;
}

return false;
   }

   public String getNome() {
      return nome;
   }

   public void setNome(String nome) {
      this.nome = nome;
   }
}
```