Apoio à Gerência de Fatores Conflitantes Identificados no Desenvolvimento de um Sistema de Software

HEITOR AUGUSTUS XAVIER COSTA[1,3]
heitor@ufla.br

ROGÉRIA CRISTIANE GRATÃO DE SOUZA[2]
rogeria@dcce.ibilce.unesp.br

SELMA SHIN SHIMIZU MELNIKOFF[3]
selma.melnikoff@poli.usp.br

[1]UFLA – Universidade Federal de Lavras
DCC – Departamento de Ciência da Computação
Caixa Postal 37 – CEP 37200-000 – Lavras – MG

[2]UNESP – Universidade Estadual Paulista
IBILCE – Instituto de Biociência, Letras e Ciências Exatas
DCCE – Departamento de Ciência da Computação e Estatística
Rua Cristóvão Colombo, 2265 – CEP 15054-000 – São José do Rio Preto – SP

[3]EPUSP – Escola Politécnica da Universidade de São Paulo
PCS – Departamento de Engenharia Elétrica
Av. Prof. Luciano Gualberto trav. 3 – n. 158 – CEP 05508-900 – São Paulo – SP

RESUMO

O desenvolvimento de sistemas de software, em geral, envolve a interação entre pessoas com diferentes formações e níveis de conhecimento sobre o sistema de software a ser desenvolvido. O gerente de projetos deve administrar adequadamente estas diferenças a fim de não refletir negativamente no planejamento do projeto e na qualidade do produto. Além disso, outro fator que pode comprometer significativamente o planejamento é o tempo necessário para solucionar as dificuldades que surgem durante a realização das atividades. Sendo assim, o objetivo deste trabalho é apresentar sugestões, obtidas através do desenvolvimento de um sistema de software, que apóiem as atividades de gerência do projeto diante da ocorrência de fatores conflitantes.

Palavras-chave: Engenharia de Software, Processo de Software, Responsabilidade do Desenvolvedor

1. Introdução

No decorrer do processo de desenvolvimento de sistemas de software, podem surgir dificuldades de ordem pessoal e/ou metodológica que devem ser contornadas de maneira a não influenciar no bom andamento do projeto. Sendo assim, o objetivo deste artigo é relatar a experiência adquirida com a construção de um sistema de software, apresentando fatores conflitantes identificados durante o seu desenvolvimento e sugestões para apoiar a sua gerência de maneira a viabilizar o cumprimento dos prazos previstos e facilitar o processo de tomada de decisão.

Quando o desenvolvimento de sistemas de software de médio e grande porte é considerado, a formação de uma equipe de desenvolvimento é natural. Esta equipe pode ser composta por pessoas que possuem as mais diversas experiências e formações para a construção de sistemas de software. Por um lado, estas características podem acelerar de maneira positiva o andamento do projeto, visto que o conhecimento dos membros da equipe sobre questões envolvidas no projeto facilita o processo de tomada de decisão e, conseqüentemente, otimiza o ritmo do desenvolvimento. Por outro lado, estas características podem impactar significativamente e negativamente o processo de desenvolvimento, incorrendo em atrasos nos prazos previstos e no mau relacionamento intrapessoal. Sendo assim, o gerente de projetos deve ter experiência e, principalmente, sensibilidade suficientes no momento de admitir os membros para compor a sua equipe de desenvolvimento. Além disso, também é atribuída a ele a responsabilidade de coordenar e gerenciar as atividades do processo de desenvolvimento ([12]).

A metodologia adotada deve estar bem assimilada pelos integrantes da equipe de desenvolvimento, bem

como o conhecimento da especificação do sistema de software, a fim de evitar discussões no âmbito conceitual e não proporcionar atrasos no cronograma do projeto. Neste sentido, é conveniente que o gerente de projetos estabeleça, junto ao cronograma inicial, um tempo para nivelar o conhecimento entre os membros da equipe.

Neste contexto, a Seção 2 identifica fatores conflitantes que podem ocorrer durante o desenvolvimento de um sistema de software e apresenta sugestões que auxiliam na tomada de decisão. A Seção 3 descreve algumas conclusões obtidas.

2. Análise dos Fatores Conflitantes Identificados

Nesta seção são apresentados os conflitos que podem ocorrer durante o processo de desenvolvimento de um sistema de software, impactando de maneira negativa o prazo necessário para efetivação do projeto.

Para contextualizar tais conflitos ao longo do processo de desenvolvimento, foram utilizadas as fases definidas no Processo Unificado – *Unified Process*. Sendo assim, é importante salientar que são observados apenas os fatores conflitantes identificados durante o desenvolvimento de um sistema de software: SISTEMA DE ENSINO (ver Apêndice A). Para maior detalhamento das atividades que compõem as fases do Processo Unificado pode ser encontrado em [8].

A notação UML – *Unified Modeling Language* ([2]), foi utilizada para a construção dos diagramas elaborados durante as fases de Análise e de Projeto, visto que seus elementos de modelagem possuem representação semântica bastante satisfatória.

As subseções a seguir foram divididas em três conjuntos distintos, de acordo com as fases de desenvolvimento às quais pertencem os fatores conflitantes identificados: Concepção, Elaboração e Construção.

2.1 Fase de Concepção

As atividades de planejamento do sistema de software sendo desenvolvido são elaboradas durante a fase de concepção. Três fatores de ordem pessoal, identificados nesta fase, são tratados com a finalidade de auxiliar o gerente de projetos na formação de uma equipe com um nível de conhecimento mais homogêneo.

2.1.1 Membros da Equipe

Durante a construção do sistema de software, verificou-se que é desejável que os membros da equipe responsável pela sua modelagem tenham um nível semelhante de conhecimento da metodologia adotada, do paradigma de desenvolvimento selecionado e da linguagem de programação a ser utilizada na implementação.

Caso isso não seja possível, é recomendada a realização de treinamentos específicos para nivelar o conhecimento dos membros da equipe. Este nivelamento é importante, pois possibilita que os recursos disponíveis nas ferramentas adotadas sejam utilizados de maneira correta e eficiente. Além disso, diminui o risco de discussões no âmbito conceitual que ocasionam perda de tempo no processo de desenvolvimento e acabam contribuindo para que o cronograma previsto não seja cumprido.

2.1.2 Especificação do Sistema de Software

Considera-se importante que o sistema de software a ser desenvolvido seja estudado e bem entendido pelos membros da equipe, bem como seus requisitos essenciais e funcionais, para que durante a sua manutenção ou no desenvolvimento de um outro sistema de software similar exista conhecimento suficiente, capaz de facilitar a visualização rápida do problema abordado. Quando apenas parte da equipe já possui um conhecimento mais avançado do sistema de software e de como foi modelado/implementado um outro similar, estas pessoas, em geral, tendem a manter a mesma linha de raciocínio. Isso acontece devido à primordialmente dois aspectos:

- Como já conhecem um problema parecido e a solução adotada obteve êxito, a tendência é (re)utilizar a mesma solução, adequando-a ao problema atual;
- Pressão do mercado em ter rapidamente o sistema de software, não permitindo vislumbrar uma outra solução possivelmente melhor.

Tais aspectos podem prejudicar o andamento do projeto, visto que os demais participantes podem não concordar com determinadas decisões sem antes entendê-las completamente. Com isso, pode-se gerar atritos entre os membros da equipe em razão daqueles que já conhecem a solução anterior terem que explicar o que já dominam para os demais integrantes, causando a impressão de atrasos no projeto. Além disso, pode haver certa resistência por parte daqueles que já dominam o sistema de software a ser desenvolvido para receber as sugestões de mudanças na forma como o sistema de software foi implementado anteriormente.

Assim, considera-se relevante nivelar o conhecimento dos membros da equipe envolvida no processo de manutenção ou desenvolvimento em relação à especificação do sistema de software antes de dar início às atividades, evitando a ocorrência de conflitos que certamente irão prejudicar o andamento do projeto.

2.1.3 Ferramentas de Apoio ao Desenvolvimento

A decisão de utilizar ferramentas automatizadas e de definir os ambientes de desenvolvimento a serem utilizados na construção de sistemas de software tem sua importância e deve ser considerada pelo gerente de projetos. Para adoção destes recursos em um ambiente de trabalho é preciso considerar a necessidade de treinamentos dos potenciais usuários, juntamente com um processo de conscientização em relação aos benefícios acarretados pelas mudanças previstas.

Atualmente existem no mercado várias ferramentas CASE (*Computer Aided Software Engineering*) que visam auxiliar na modelagem e, até mesmo, na geração primária de código dos sistemas de software sendo desenvolvidos ([3], [6], [9], [10], [11], [13], [14], [16] e [19]).

A manipulação destas ferramentas deve ser feita de maneira adequada para que os recursos disponibilizados sejam utilizados de maneira eficiente. Com isso, treinamentos adequados devem ser agendados pelo gerente de projetos, buscando garantir maior satisfação e confiança aos membros da equipe de desenvolvimento diante dos desafios provenientes da aquisição de novas tecnologias e sua incorporação no ambiente de trabalho.

Quando estas ferramentas são utilizadas sem treinamento, a tendência é manipular os recursos disponíveis intuitivamente, ou seja, na seqüência que acreditam ser a mais correta. Porém, isso pode causar problemas de inconsistências entre os artefatos de software gerados, pois nem sempre as alterações feitas, em um determinado artefato, são refletidas automaticamente nos demais como é esperado. Por exemplo, tem-se a ferramenta CASE *Rational Rose 98 Enterprise Edition* ([15]), onde as alterações realizadas no Diagrama de Classe são refletidas no Diagrama de Seqüência, atualizando automaticamente o nome dos métodos alterados, bem como a inserção e remoção de métodos. Todavia, caso um novo método seja inserido no Diagrama de Seqüência ou seu nome e/ou parâmetros sejam alterados, estas informações não são atualizadas automaticamente no Diagrama de Classes, causando inconsistência entre os diagramas.

O investimento em ferramentas CASE não deve ser limitado à aquisição do produto, mas estendido para o treinamento das pessoas responsáveis por sua manipulação para que o retorno esperado com a utilização da ferramenta seja alcançado ([12]).

2.2 Fase de Elaboração

Os modelos relacionados a um sistema de software têm a sua construção finalizada na fase de elaboração. A seguir, são tratados dois fatores de ordem metodológica, buscando auxiliar a equipe de desenvolvimento na tomada de decisão em relação à construção e representação do Diagrama de Objetos. Os demais diagramas não estão sendo considerados, visto que a sua relevância não foi questionada durante o desenvolvimento do SISTEMA DE ENSINO.

2.2.1 Construção do Diagrama de Objetos

O Diagrama de Objetos representa a organização do sistema de software em um certo instante da sua execução, mostrando as instâncias das classes e seus relacionamentos que foram definidos no Diagrama de Classes ([2]).

As informações presentes no Diagrama de Objetos podem auxiliar no desenvolvimento do Diagrama de Seqüência, fornecendo uma visão clara dos objetos que precisam se comunicar para que determinada funcionalidade seja alcançada.

Durante o desenvolvimento do SISTEMA DE ENSINO, entretanto, verificou-se que alguns membros da equipe de desenvolvimento preferem utilizar apenas o Diagrama de Classes para auxiliar no desenvolvimento do Diagrama de Seqüência, visto que é possível obter as informações necessárias através da sua análise.

Desta forma, é necessário que a equipe de desenvolvimento realize uma análise conclusiva para determinar se é viável investir tempo da equipe na construção do Diagrama de Objetos. Neste estudo deve-se considerar alguns aspectos, como por exemplo, a complexidade envolvida no desenvolvimento do Diagrama de Objetos, visto que a sua visibilidade se torna prejudicada à medida que aumenta a quantidade de classes.

É importante considerar que o resultado desta análise pode ser diferente para cada equipe de desenvolvimento, visto que irá depender do ponto de vista e da capacidade de abstração que cada membro apresenta.

2.2.2 Representação do Diagrama de Objetos

Caso a equipe de desenvolvimento decida construir o Diagrama de Objetos, é necessário estabelecer como a sua representação deve ser feita de maneira que a sua análise seja facilitada.

No SISTEMA DE ENSINO, a equipe optou, primeiramente, em construir apenas um Diagrama de Objetos representando o sistema de software como um todo. Entretanto, tal decisão fez com que o diagrama ficasse bastante complexo e a sua visibilidade ficou prejudicada devido à quantidade de classes existentes no Diagrama de Classes.

Como as referências presentes na literatura que abordam a construção do Diagrama de Objetos não apresentam uma sugestão para contornar esta situação, optou-se por desenvolver vários Diagramas de Objetos, cada um representando um caso de uso. Isso foi feito por considerar os casos de uso como partes pequenas

do sistema de software capazes de representar a funcionalidade esperada. Com isso, os Diagramas de Objetos ficaram mais claros e significativos para serem analisados.

2.3 Fase de Construção

Durante a codificação dos sistemas de software, realizada na fase de construção, decisões importantes deverão ser tomadas pelo programador a fim de refletir adequadamente o contexto apresentado na modelagem. Para tanto, o programador deverá analisar cuidadosamente as possíveis soluções existentes e os respectivos impactos causados no sistema de software com a sua adoção, visando selecionar a estratégia mais viável. A seguir são apresentados dois fatores relevantes que devem ser considerados durante a implementação de sistemas de software utilizando linguagem de programação puramente orientada a objetos.

2.3.1 Estrutura de Herança

Uma questão importante a ser considerada pelo programador ocorre quando, em uma hierarquia de herança, a superclasse e a(s) subclasse(s) são concretas, ou seja, as classes envolvidas na hierarquia são instanciáveis. É muito comum encontrar na literatura exemplos de Diagramas de Classes que possuem hierarquia de herança, contudo apresentam apenas as classes no nível mais inferior como sendo classes concretas. As classes que aparecem nos níveis mais acima são apresentadas como classes abstratas (não instanciáveis).

Entretanto, o SISTEMA DE ENSINO apresenta classes organizadas em uma hierarquia de herança onde, embora no primeiro nível exista uma classe abstrata (Pessoa), no segundo e no terceiro níveis existem classes instanciáveis (Estudante, Monitor e Professor). Isso não significa que a modelagem, no caso o Diagrama de Classes, esteja incorreta, muito pelo contrário, o sistema de software pode estar modelado corretamente seguindo as restrições e atendendo aos requisitos do domínio da aplicação.

A equipe de desenvolvimento identificou duas alternativas, sob a ótica do programador, para resolver tal situação:

- A primeira alternativa consiste em utilizar o recurso de classe monolítica[1]. Este recurso consiste em colocar todas as funcionalidades e atributos das classes envolvidas na classe localizada no nível mais alto da hierarquia. Com isso, esta classe se torna uma classe concreta e as demais se tornam papéis que esta classe pode desempenhar. Segundo [2], papel é o comportamento de uma entidade participando em um determinado contexto;
- A segunda alternativa consiste em atribuir as características de classe concreta às classes, que se apresentam no modelo como instanciáveis, bem como atribuir características de classe abstrata às classes que se apresentam no modelo como não instanciáveis.

Através da análise realizada com o SISTEMA DE ENSINO, observou-se que ambas abordagens possuem aspectos positivos e aspectos negativos. Ao se adotar o uso de uma classe monolítica, a sua manutenção aumenta de complexidade, pois está tudo em um único local, dificultando a localização de maneira rápida e precisa de um atributo e/ou de um método específico. Também há quebra do paradigma de orientação a objetos com respeito ao relacionamento de herança, pois deixa de existir a relação de herança entre as classes, desrespeitando o diagrama. Já a segunda alternativa torna mais fácil a visibilidade das características das classes por elas estarem presentes na classe a qual são pertinentes e por manterem consistência com a modelagem que foi apresentada. Contudo, esta última alternativa exige um controle maior sobre os atributos determinantes dos objetos. Um atributo é considerado determinante quando ele determina, como o próprio nome expressa, um único objeto, isto é, ele possui valor único para cada objeto ([1], [4], [5] [7], [17] e [18]). Por outro lado, o uso do recurso de classe monolítica torna trivial o controle de tais atributos.

Com isso, destaca-se que o programador será responsável pela tomada de decisão, sendo que ele poderá considerar para tanto, as alternativas descritas. Entretanto, é importante ressaltar que, quando a decisão tomada implicar em mudanças nos artefatos de software, as alterações necessárias devem ser realizadas para que a documentação reflita adequadamente as soluções adotadas durante a implementação e facilite a rastreabilidade entre o código e os diagramas elaborados durante as fases de Análise e Projeto. Deve ser observado que tais alterações devem ser restritas ao Modelo de Projeto, pois os requisitos do usuário, representados no Modelo de Análise, não devem ser comprometidos pelas soluções adotadas durante a implementação.

2.3.2 Abstração dos Artefatos de Software no Processo de Desenvolvimento

A decisão do programador não deve estar baseada em apenas uma parte do modelo, mas no modelo como um todo, nas características intrínsecas do domínio da aplicação e do sistema de software e nos aspectos específicos da linguagem de programação adotada.

Entretanto, muitas vezes, o sistema de software em desenvolvimento se torna abstrato para o programador, inviabilizando uma tomada de decisão. Isso não deve

[1] Monolítica é uma estrutura em que há uma só classe abrangendo todos os atributos e métodos pertencentes às classes envolvidas na hierarquia de herança.

ser visto como demérito, uma vez que ele pode não ter participado dos processos anteriores para a elaboração dos diagramas. Sendo assim, tal decisão é responsabilidade e deve ser indicada nos diagramas pelos projetistas com a ajuda dos analistas de requisitos. Dessa forma, o processo de codificação é agilizado, visto que o programador não interrompe seu trabalho para refletir e tomar decisões que, porventura, podem não ser correta por não estar interado o suficiente com o domínio da aplicação.

Uma estratégia, que pode ser adotada pelos projetistas e analistas responsáveis pelos artefatos de software oriundos das fases de Concepção e Elaboração do ciclo de desenvolvimento do Processo Unificado, é o uso do recurso de *Stereotypes* ([2]). Este recurso consiste em estender o vocabulário da notação UML, permitindo a criação de novos tipos de elementos de modelagem para os diagramas e/ou aumentar a semântica dos elementos de modelagem já existentes, mas que são muito específicos para atender um determinado problema. Em outras palavras, *Stereotypes* permite definir novos elementos de modelagem, estendendo os existentes, de forma a atender a uma determinada semântica que os elementos presentes na notação UML não conseguem representar.

Este recurso permite caracterizar uma solução no sentido de diminuir o nível de abstração existente entre o Modelo de Projeto e a sua implementação. A solução obtida pode se tornar um padrão à medida que esta solução é utilizada recorrentemente, ou seja, reutilizada em outros projetos com características similares. Sendo assim, o resultado alcançado na implementação pode contribuir para a definição de um padrão de apresentação da estrutura do código.

3. Conclusão

O processo de desenvolvimento de sistemas de software deve ser bem gerenciado para reduzir as dificuldades enfrentadas pela equipe de desenvolvimento e o grau de insatisfação dos usuários, evitando assim, atrasos no cronograma previsto e o comprometimento na qualidade do produto final. Neste contexto, este artigo apresentou fatores conflitantes que devem ser adequadamente tratados com o objetivo de eliminar eventuais dificuldades encontradas durante o processo de desenvolvimento.

O gerente de projetos representa um papel decisivo na garantia de um bom andamento do processo de desenvolvimento, uma vez que ele é o responsável pela seleção da equipe de desenvolvimento, pela boa formação de cada um de seus membros que é garantida pela realização de treinamentos especializados e pela gerência do andamento das atividades.

O projetista é responsável por tomar decisões de projeto e garantir que o nível de abstração dos artefatos de software oriundos da etapa de Projeto esteja adequado para que o programador não tenha de gerar informações extras para realizar seu trabalho.

O programador, por sua vez, é responsável pela implementação adequada do sistema de software, de maneira que a construção do código reflita corretamente as características representadas nos diagramas.

Este artigo foi realizado com base em um estudo de caso definido pelos autores, modelado usando a notação UML e seguindo as fases definidas pelo Processo Unificado. É importante observar a necessidade de desenvolver novos sistemas de software com o intuito de identificar novos requisitos a serem tratados durante o processo de desenvolvimento para auxiliar, ainda mais, no andamento do projeto.

4. REFERÊNCIAS BIBLIOGRÁFICAS

[1] AMBLER, S. W. Análise e Projeto Orientados a Objetos. IBPI Press, 1998.

[2] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. The Unified Modeling Language User guide. Addison-Wesley, 1999.

[3] COSTA, H. A. X. Modelagem Conceitual de Sistemas – Protótipo de Ambiente Baseado em Aplicativos Comerciais. Rio de Janeiro: PUC-Rio, 1997. Dissertação de Mestrado.

[4] DATE, C. J. An Introduction to Database Systems. Addison-Wesley, 1999.

[5] ELMASRI, R.; NAVATHE, S. B. Fundamentals of Database Systems. Addison-Wesley, 1999.

[6] HENNINGER, S. Building an Organization-Specific Infrastructure to Support CASE Tools. Automated Software Engineering, pp 239-259, 1996.

[7] HEUSER, C. A. Projeto de Banco de Dados. Série Livros Didáticos, Instituto de Informática da Universidade Federal do Rio Grande do Sul, Sangra Luzzato, 2000.

[8] JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. The Unified Software Development Process. Addison-Wesley, 1999.

[9] LIMA, A. de A. B.; OLIVEIRA, G. C. M. A.; NASCIMENTO, L. M. Ferramentas de Gerência de Informação: Dicionário de Dados, CASE e Repositórios. Rio de Janeiro: UFRJ, 1995. Projeto Final de Curso.

[10] LUCENA, C. J. P. A Anatomia de um Ambiente de Desenvolvimento de Software. Rio de Janeiro: PUC-Rio, 1991. Monografia do Departamento de Informática.

[11] MOURA, L. M. V. Taxonomia de Ambientes de Desenvolvimento de Software. Rio de Janeiro: UFRJ, 1992. Dissertação de Mestrado.

[12] PAGE-JONES, M. Gerenciamento de Projetos. McGraw-Hill, 1990.

[13] PAIVA, L. Uma Análise do Mercado de Ambientes CASE. Rio de Janeiro: UFRJ, 1994. Projeto Final de

Curso.

[14] PRESSMAN, R. S. Software Engineering: A Practitioner's Approach. McGraw-Hill, 2000.

[15] ROSE, http://www.rational.com/products/rose/index.jsp; consultada em 23/01/2001.

[16] SANTOS, C. J. Um Ambiente de Apoio Automatizado para Desenvolvimento de Software Básico. Rio de Janeiro: UFRJ, 1987. Dissertação de Mestrado.

[17] SETZER, V. W. Banco de Dados. Série Ciência da Computação, Edgard Blücher Ltda, 1998.

[18] SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. Sistema de Banco de Dados. Makron Books, 1999.

[19] SOMMERVILLE, I. Software Engineering. Addison-Wesley, 2000.

APÊNDICE A – ESTUDO DE CASO: SISTEMA DE ENSINO

Este sistema tem como objetivo automatizar os processos que compõem as tarefas administrativas do setor acadêmico das universidades, auxiliando a Pró-Reitoria Acadêmica na efetivação de suas atribuições. No organograma de uma universidade é muito comum encontrar departamentos compondo sua estrutura. Sendo que em cada departamento há um elenco de matérias sob sua responsabilidade, as quais podem ser oferecidas no decorrer do período letivo. Contudo, cada matéria deve ser oferecida apenas uma vez a cada período, acarretando a formação de uma turma neste período com capacidade máxima de 150 estudantes. Além disso, cada turma criada pode ter no máximo 3 monitores e, obrigatoriamente, um professor responsável. No entanto, nada impede que uma matéria seja oferecida em mais de um período. Os estudantes são matriculados em turmas. O tempo máximo de permanência de um estudante na universidade é de 30 períodos, sendo que o ano letivo é dividido em 3 períodos. Um monitor sempre é um estudante da universidade, contudo, sob o aspecto legal da relação entre patrão e trabalhador, ele possui uma espécie de contrato com a mesma determinando a agregação de uma nova função: monitor. Um monitor não pode assumir a turma na qual ele está matriculado e deve ser responsável por somente uma turma. Um professor, trabalhador da universidade, pode lecionar em, no máximo, duas turmas por período. Com base nestas informações, foi possível elaborar o Diagrama de Classes correspondente ao problema descrito, o qual é apresentado na Figura 1.

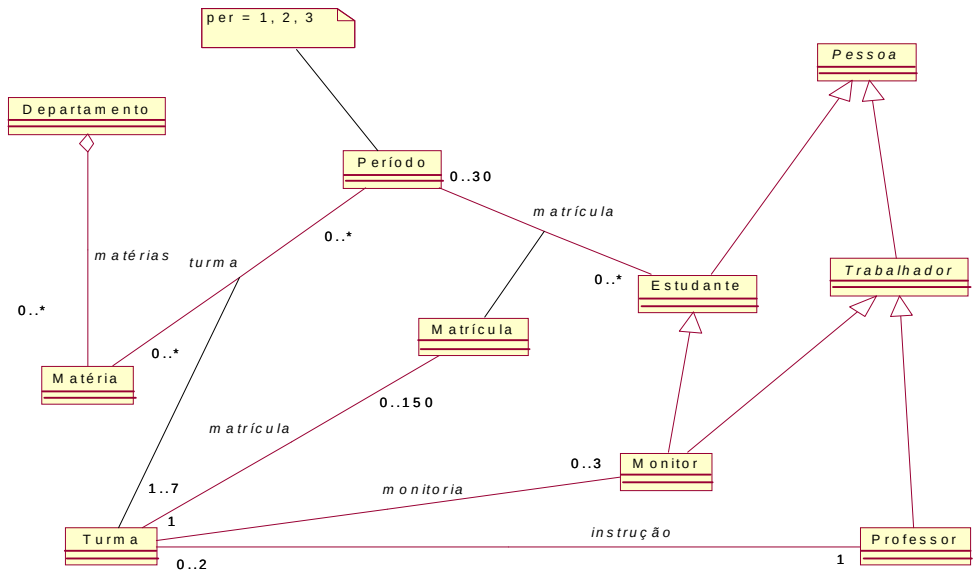

Figura 1 - Diagrama de Classes do Estudo de Caso