TRABALHO PARA A DISCIPLINA DE

TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO DA UTFPR: CUBE ADVENTURE

Gustavo Adame Domingues, Mateus Szepeilewicz Camillo

gustavoadame@alunos.utfpr.edu.br,  mateuscamillo@alunos.utfpr.edu.br

Disciplina: **Técnicas de Programação – CSE20** / S71 – Prof. Dr. Jean M. Simão

**Departamento Acadêmico de Informática – DAINF** - Campus de Curitiba

Curso Bacharelado em: Engenharia da Computação / Sistemas de Informação

**Universidade Tecnológica Federal do Paraná - UTFPR**

Avenida Sete de Setembro, 3165 - Curitiba/PR, Brasil - CEP 80230-901

**Resumo** – A disciplina de Técnicas de Programação exige o desenvolvimento de um software de plataforma, no formato de um jogo, para fins de aprendizado de técnicas de engenharia de software, particularmente de programação orientada a objetos em C++. Para tal, neste trabalho, escolheu-se o jogo Cube Adventure, no qual o jogador, representado por um cubo caricato, enfrenta inimigos em um dado cenário. O jogo possui três fases que se diferenciam por meio do estilo do mapa e também por diferentes dificuldades. Para o desenvolvimento do jogo foi elaborado um diagrama de classes utilizando a Linguagem de Modelagem Unificada (Unified Modeling Language - UML) usando como base um diagrama genérico e prévio proposto. Logo, em linguagem de programação C++, realizou-se o desenvolvimento que contemplou os conceitos usuais de Orientação a Objetos como Classe, Objeto e Relacionamentos, bem como alguns conceitos avançados como Classe Abstrata, Polimorfismo, Gabaritos Templates, Sobrecarga de Operadores e Biblioteca Padrão de Gabaritos (Standard Template Library - STL). Depois da implementação, os testes e uso do jogo foram feitos pelos próprios desenvolvedores, demonstrando sua funcionalidade conforme os requisitos propostos. Por fim, salienta-se que o desenvolvimento em questão permitiu cumprir o objetivo de aprendizado visado.

**Palavras-chave ou Expressões-chave**: Trabalho de técnicas de programação, Trabalho Acadêmico Voltado a linguagem C++, Programação Orientada a Objeto, Cube adventure.

**Abstract** – The discipline of Programming Techniques requires the development of a platform software, on a game format, for learning purposes, more specifically the C++ object-oriented language. On this project the game that was chosen is Cube Adventure, a game which the player, represented by a cartoon cube, face many types of enemies on a certain scenario. The game is composed by tree phases, with different kinds of enemies and difficulties. For the development, a class diagram was created using Unified Modeling Language – (UML), based on a generic diagram previously proposed. With C++ language, the development has Object-Oriented concepts such as Abstract classes, Polymorphism, Templates, operator overload and Standard Template libraries-STL. After the implementation, tests were made by the developers in order to show functionality. Finally, developing the game improved knowledge on the developers.

**Key-words or Key-expressions:** Programming techniques project, Academic work related to C++ language, Object oriented programming, Cube Adventure.

# INTRODUÇÃO

No curso de Engenharia da Computação da UTFPR, mais precisamente no segundo período do curso, os alunos são apresentados à disciplina de Técnicas de Programação, lecionada pelo professor Jean M. Simão. Nesta disciplina, a média é composta por uma prova escrita e um projeto, o qual tem como objetivo praticar e consolidar os conteúdos da linguagem abordados na disciplina, assim como preparar os alunos para futuros ambientes profissionais e acostuma-los com sua complexidade.

O trabalho consiste em um jogo de plataforma 2D desenvolvido de modo orientado a objetos, a partir do levantamento dos requisitos previamente abordados pelo professor, criação de um diagrama de classes por meio da UML, e por fim a aplicação do planejamento prévio em código na linguagem C++, assim como seu versionamento.

Para a melhor realização do projeto em si, os desenvolvedores utilizaram de programas como, Discord para a comunicação, compreensão e modelagem em conjunto, GitHub e Git para o versionamento e troca do código, diagrama de classes, e também anotações em documentos “.txt”. Por fim, para o desenvolvimento do código foi utilizada a plataforma Visual Studio 2019 Community e, para a criação do diagrama de classes utilizou-se do programa Star UML.

Concluindo, nas seções subsequentes será explicado o funcionamento do jogo, sua lógica, desenvolvimento e, conceitos da linguagem aplicados. Comparações entre a linguagem procedimental e a orientada a objetos, reflexões sobre o projeto, como ele foi dividido entre os desenvolvedores, agradecimento dos mesmos e por fim as referências.

# EXPLICAÇÃO DO JOGO EM SI

Nesta seção se deve discorrer a explicação do jogo em si. **Portanto, salienta-se que esta seção NÃO é (em absoluto) para explicar a modelagem (análise e projeto) ou a implementação do jogo.**

Isto dito, esta seção assim como as demais, devem seguir as regras de formatação dadas as quais foram seguidas para compor este próprio presente modelo. Justamente, quanto à formatação, o texto do trabalho deve seguir as seguintes regras:

* O trabalho deve ser totalmente digitado em fonte Times New Roman. Esta diretriz inclui, portanto, o título do trabalho, autores, filiação e endereços, títulos de seções e legendas de figuras e tabelas, além do texto normal do trabalho. O texto deve ser digitado com alinhamento ‘justificado’ ou ajustado.
* O trabalho completo, incluindo figuras e tabelas, deve ser limitado a doze (12) páginas (no máximo) em papel de tamanho padrão A4 (21 cm x 29,7 cm). Não reduzir figuras e tabelas a tamanhos que sacrifiquem o entendimento dos seus conteúdos.
* Cada página, no tamanho A4, deve ser formatada de modo a apresentar 2,5 cm de margem em todos os lados do documento. Dentro desta área o texto deve ser formatado em uma única coluna, **sem** incluir moldura no texto**.**
* O título deve ser digitado em **destaque**, em letras maiúsculas, centralizado e em tamanho 14 pt, não excedendo três linhas, seguido de uma linha em branco (12 pt) e pelas linhas que conterão o(s) nome(s) do(s) autor(es), em tamanho 12 pt. Em seguida, deverá vir a filiação e o(s) endereço(s) para correspondência do(s) autor(es) (tamanho 10 pt) separada por uma linha em branco. Deve-se deixar 3 linhas de espaço antes do resumo e uma linha entre os itens subsequentes (palavras-chave, *abstract* e *Key-words*).
* Digitar o título **Resumo** em **destaque**, alinhado à esquerda, tamanho 10 pt, seguido de um traço. Sem trocar de linha, digitar o resumo, em tamanho 10 pt com alinhamento justificado. Pular uma linha e digite o título **Palavras-chave:** em **destaque**, alinhado à esquerda, tamanho 10 pt. Digitar então no máximo quatro (4) palavras-chave, separadas por vírgulas, com somente a primeira letra de cada palavra-chave em maiúscula. Na sequência devem vir o ***Abstract*** e ***Key-words*:** em inglês, seguindo o mesmo padrão de formatação do resumo e das palavras-chave. Tanto o resumo quanto o *abstract* devem conter no máximo 200 palavras.
* A seguir, separado por 2 linhas (12 pt), o texto deve ser iniciado pela **Introdução**. Os títulos das seções (Introdução etc.) devem ser escritos em **destaque**, sem numeração,

em maiúsculo e alinhados à esquerda, sendo que o conteúdo, propriamente dito, deve ser iniciado após espaçamento de uma linha e tabulação (1 cm).

* Ao final de cada seção deve-se deixar uma linha vazia. Todo o texto deverá ser escrito em espaço simples. Para as subseções, somente a primeira letra do subtítulo deve ser maiúscula, sendo todas em **destaque**, sem numeração, com o título alinhado à esquerda. Uma vez escrito o subtítulo, pular uma linha. Após esta linha (vazia), iniciar o texto da subseção.
* As ilustrações e gráficos podem ser em escalas cinza ou coloridos, mas sempre centralizados. As notas de rodapé1 devem ser colocadas na parte inferior da página correspondente separadas por um traço conforme modelo. Usar o tamanho de 8 pt.
* As referências bibliográficas devem ser listadas no fim do artigo, na ordem de citação, conforme formato da Associação Brasileira de Normas Técnicas (ABNT). No texto, as citações devem ser referenciadas por seu número colocado entre colchetes, por exemplo, [1] e [2].

Em tempo, para melhor explicar o Jogo em Si, aconselha-se utilizar de recursos como gráficos, telas e figuras do próprio jogo. **Na verdade, telas e figuras do jogo DEVEM ser usadas, inclusive para a explicação do jogo em si com as devidas imagens do jogo executando. Isto facilita por demais a compreensão do jogo contemplado no trabalho.**

A propósito, as figuras, tabelas etc., devidamente referenciadas (citadas explicitamente) no texto, podem ser colocadas da maneira mais conveniente para o autor em uma ou duas colunas, desde que o texto permaneça em apenas uma coluna. Antes e após os elementos não textuais e suas respectivas legendas, deve-se deixar uma linha de espaçamento.

Os autores não devem se esquecer da colocação de legendas nas figuras, tabelas e outros elementos gráficos. As figuras devem ser numeradas sequencialmente com algarismos arábicos conforme o exemplo da figura 1. Outrossim, as figuras devem ser apropriadamente intercaladas pelo texto.

Y

X

Figura 1. Centralizada na coluna e com legenda abaixo da figura.

Aproveitando o ensejo, talvez os autores façam uso de equações em alguma parte do texto. Neste âmbito, todas as equações deverão ser tabuladas a 1 cm da margem esquerda e numeradas sequencialmente, com os números entre parênteses, conforme o exemplo abaixo

# DESENVOLVIMENTO DO JOGO NA VERSÃO ORIENTADA A OBJETOS

Nesta seção se deve discorrer a explicação do desenvolvimento do jogo/*software* utilizando orientação a objeto**, começando pelos requisitos e avançando para a modelagem, salientado o projeto em diagrama(s) de classes em UML,** e culminando na programação em C++. A explicação deve ser feita de maneira tal a **não** ser um relatório técnico repleto de detalhes, mas que seja capaz de sintetizar e valorizar os recursos técnicos utilizados (i.e., sucinto e suficiente).

Nesta explicação, deve-se primeiramente ter texto introdutório que leva a listar textualmente em tabela os **requisitos funcionais** definidos para o jogo/*software* em questão, os quais são justamente os requisitos definidos na Tabela 1 neste presente documento. Estes requisitos não podem ser alterados, mas eventualmente interpretados em reuniões com o professor, **reuniões estas obrigatórias**. Isto disto, os requisitos devem estar enquadrados em uma tabela de duas colunas na qual a primeira coluna traz os requisitos e a segunda coluna a sua situação (*status*) que pode ser ‘realizado’, ‘semi-realizado’, ‘abandonado’ etc.

Ademais, quando o requisito estiver como ‘realizado’, ‘semi-realizado’ ou similares, faz-se absolutamente necessário indicar sucintamente quais classes ou conjuntos de classes que realizaram cada requisito no preenchimento tabela, no tocante ao campo ‘Implementação’ da Tabela 1. Eventualmente, pode-se também nomear objetos que se julguem pertinentes, se não for suficiente apenas nomear suas classes. **A Tabela 1 exemplifica o exposto definindo, ademais, os requisitos mínimos que cada jogo deve ter**.

Tabela 1. Lista de Requisitos do Jogo e suas Situações.

|  |  |  |  |
| --- | --- | --- | --- |
| **N.** | **Requisitos Funcionais** | **Situação** | **Implementação** |
| 1 | Apresentar graficamente menu de opções aos usuários do Jogo, no qual pode se escolher fases, ver colocação (*ranking*) de jogadores e demais opções pertinentes. | Requisito previsto inicialmente, e realizado | Requisito cumprido via classe Menu, suas heranças, e seus respectivos objetos, com suporte da SFML. |
| 2 | Permitir um ou dois jogadores com representação gráfica aos usuários do Jogo, sendo que no último caso seria para que os dois joguem de maneira concomitante. | Requisito previsto inicialmente, e realizado | Requisito cumprido via classes CuboCowboy e CuboExplorador. |
| 3 | Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas, via menu, nas quais jogadores tentam neutralizar inimigos por meio de algum artifício e vice-versa | Requisito previsto inicialmente, e realizado | Requisito cumprido via menus, FaseGrama, FaseCaverna e FaseCavernaChefao. |
| 4 | Ter pelo menos três tipos distintos de inimigos, cada qual com sua representação gráfica, sendo que ao menos um dos inimigos deve ser capaz de lançar projétil contra os jogadores e um dos inimigos dever ser um ‘Chefão’. | Requisito previsto inicialmente, e realizado | Requisito cumprido via classes Inseto, Aranha, Alien e Projetil. |
| 5 | Ter a cada fase ao menos dois tipos de inimigos com número aleatório de instâncias, podendo ser várias instâncias e sendo pelo menos 3 instâncias por tipo. | Requisito previsto inicialmente, e realizado | Requisito cumprido via classes FaseGrama, FaseCaverna e FaseCavernaChefao. |
| 6 | Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos um causa dano em jogador se colidirem. | Requisito previsto inicialmente, e realizado | Requisito cumprido via classes Espinho, BlocoVoador, BlocoGrama e BlocoCaverna |
| 7 | Ter em cada fase ao menos dois tipos de obstáculos com número aleatório de instâncias (i.e., objetos), sendo pelo menos 3 instâncias por tipo. | Requisito previsto inicialmente, e realizado | Requisito cumprido via classes FaseGrama, FaseCaverna e FaseCavernaChefao. |
| 8 | Ter em cada fase um cenário  de jogo constituído por obstáculos, sendo que parte deles seriam plataformas ou similares, sobre as quais pode haver inimigos e podem subir jogadores. | Requisito previsto inicialmente, e realizado | Requisito cumprido via classes FaseGrama, FaseCaverna e FaseCavernaChefao. |
| 9 | Gerenciar colisões entre jogador para com inimigos e seus projeteis, bem como entre jogador para com obstáculos. | Requisito previsto inicialmente, e realizado | Requisito cumprido via classe GerenciadorColisao. |
| 10 | Permitir: (1) salvar nome do usuário, manter/salvar pontuação do jogador (incrementada via neutralização de inimigos) controlado pelo usuário e gerar lista de pontuação (*ranking*). **E** (2) Pausar e **Salvar** Jogada. | Requisito previsto inicialmente, porém parcialmente realizado | Requisito parcialmente cumprido via classes MenuRanking e CadastradorJogada, apenas em tempo de execução. |
| **Total de requisitos funcionais apropriadamente realizados.** | | | **90%** (noveta por cento). |

Isto feito, a explicação do desenvolvimento segue devendo-se:

* + Utilizar Diagrama(s) de Classes em *UML* para explicar as classes e suas relações, que DEVEM atender aos requisitos.
  + Utilizar como base o Diagrama de Classes proposto na FIGURA 2, completando-o, melhorando-o, expandindo-o etc2.
  + A luz do(s) diagrama(s), explicar o desenvolvimento de maneira sucinta e suficiente no texto. Assim, em poucos parágrafos deve se explicar as principais classes ou agrupamentos de classes (quiçá em pacotes) e como elas se inter-relacionam.
  + Em tempo, valorizar as ‘sofisticações’ que tenham sido realizadas, como comportamento mais elaborado de inimigos.
  + Valorizar a interdisciplinaridade como a aplicação de conceitos de física e matemática aprendidos em disciplinas do ensino médio e preferencialmente em disciplinas da graduação.
  + Deixar no diagrama apenas o que de fato foi implementado em C++ ~~ou ao menos~~ ~~bem indicar o que foi efetivamente implementado~~.

Para a implementação em C++ orientado a objetos (OO), que é a expressão técnica do projeto em diagrama(s) de classes em UML, deve-se usar uma biblioteca gráfica (*e.g*. *SFML*, *Allegro*, *TCL*/*TK*, *OpenGL* ou outra, mas preferencialmente uma OO), pois isto valoriza o trabalho esteticamente além de demonstrar a capacidade de ‘pesquisa’ e aprendizado nisso. Não deixe de valorizar esta capacidade de pesquisa e aprendizado nesta seção. Em tempo, no site da disciplina a exemplos pedagógicos prontos com *SFML (OO em si)* e *Allegro* 4.x (procedimental em si), este tanto em programa procedimental quanto integrado em programa orientado a objetos.

Esta seção em questão é muito importante no trabalho e será corrigida com muita atenção pelo professor. Pede-se, por fim, que todos os autores revisem cuidadosamente a versão final do trabalho (como um todo) para evitar erros de português, digitação e/ou

2 Eventualmente, pode-se utilizar outros diagramas UML caso os conheça ou os tenha estudado.

formatação. Na verdade, além disto, uma equipe poderia revisar o trabalho escrito da outra e vice-versa para fins de aprimoramento mútuo.

Figura 2. Diagrama de Classes de base em UML.



Listas

Depende

TL : Class

**ListaEntidades**

1

1

+LEs: Lista<Entidade>

Realiza

\*

gerenciar

Conhecer Proximo

0..1

1

\*

Realiza

<<Abstrata>>

***Entidade***

Utilizar

0..1

#x: int #y: int

<<Infraestrutura >>

**Gerenciador\_Grafico**

\*

1

0..1

Conhecer

0..1

Conhecer 0..1

\*

Conhecer Ultimo

*+executar():void*

+imprimir-se(): void

0..1

Fases

Obstaculos

1 \*

<<Abstratata>>

***Fase***

Personagens

**Personagem**

#num\_vidas: int

**Projetil**

**Obstaculo**

*+excutar():void* \*

+gerenciar\_colisoes(): void

Arremessar

1..2

**Jogador**

**Inimigo**

1

<<Fase>>

**Fase1**

<<Fase>>

**Fase2**

<<Obstaculo>> <<Obstaculo>>

**Obst\_TipoA Obst\_TipoB**

1

1

1

1

**Menu**

<<PRINCIPAL>>

**Jogo**

1

<<Imigo>>

**Inim\_TipoA**

<<Inimigo>>

**Inim\_TipoB**

1 1

Eis um diagrama que começa a atender os requisitos postos e deve ser extendido e finalizado para tal.

A partir desta classe, criar outras derivadas visando atender os requisitos postos

Primeiro

-pProx: Elemento<TE>\*

**Elemento**

TE : Class

-pPrimeiro: Elemento<TL>\*

-pUltimo: Elemento<TL>\*

**Lista**

# TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

Nesta seção, em relação aos conceitos aprendidos, deve-se apresentar uma tabela de conceitos utilizados e não utilizados tal qual a Tabela 2, sendo que a coluna de ‘Conceitos’ dessa tabela **NÃO** pode ser alterada, absolutamente. Deve-se também apresentar outra tabela justificando o uso ou não uso, tal qual a Tabela 3.

Oportunamente, todas as tabelas que venham a ser utilizadas deverão ser numeradas sequencialmente com algarismos arábicos, conforme o exemplo abaixo:

Tabela 2. Lista de Conceitos Utilizados e Não Utilizados no Trabalho.

|  |  |  |  |
| --- | --- | --- | --- |
| **N.** | Conceitos | Uso | Onde / O quê |
| **1** | **Elementares:** | | |
| * Classes, objetos. & * Atributos (privados), variáveis e constantes. & * Métodos (com e sem retorno). | Sim | Todos .h e .cpp |
| * Métodos (com retorno *const* e parâmetro *const*). & * Construtores (sem/com parâmetros) e destrutores | Sim | Todos .h e .cpp |
| - Classe Principal. | Sim | Main.cpp & Principal.h/.cpp |
| - Divisão em .h e .cpp. | Sim | No desenvolvimento como um todo. |
| **2** | **Relações de:** | | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | * Associação direcional. & * Associação bidirecional. |  |  |
| * Agregação via associação. & * Agregação propriamente dita. |  |  |
| * Herança elementar. & * Herança em diversos níveis. |  |  |
| - Herança múltipla. |  |  |
| **3** | **Ponteiros, generalizações e exceções** | | |
| - Operador *this* para fins de relacionamento bidirecional. |  |  |
| - Alocação de memória (*new* & *delete*). |  |  |
| - Gabaritos/*Templates* criada/adaptados pelos autores (e.g. Listas Encadeadas via *Templates*). |  |  |
| - Uso de Tratamento de Exceções (*try catch*). |  |  |
| **4** | **Sobrecarga de:** | | |
| - Construtoras e Métodos. |  |  |
| - Operadores (2 tipos de operadores pelo menos). |  |  |
| **Persistência de Objetos (via arquivo de texto ou binário)** | | |
| - Persistência de Objetos. |  |  |
| - Persistência de Relacionamento de Objetos. |  |  |
| **5** | **Virtualidade:** |  |  |
| - Métodos Virtuais. |  |  |
| - Polimorfismo |  |  |
| - Métodos Virtuais Puros / Classes Abstratas |  |  |
| - Coesão e Desacoplamento |  |  |
| **6** | **Organizadores e Estáticos** | | |
| - Espaço de Nomes (*Namespace*) criada pelos autores. |  |  |
| - Classes aninhadas (*Nested*) criada pelos autores. |  |  |
| - Atributos estáticos e métodos estáticos. |  |  |
| - Uso extensivo de constante (*const*) parâmetro, retorno, método... |  |  |
| **7** | **Standard Template Library *(STL)* e String OO** | | |
| - A classe Pré-definida *String* ou equivalente. &  *- Vector* e/ou *List* da *STL* (p/ objetos ou ponteiros de objetos de classes definidos pelos autores) |  |  |
| - Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-  Conjunto, Mapa **OU** Multi-Mapa. |  |  |
| **Programação concorrente** | | |
| *- Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **OU** Win32API ou  afins*.* |  |  |
| *- Threads* (Linhas de Execução) no âmbito da Orientação a  Objetos com uso de Mutex, Semáforos, **OU** Troca de mensagens. |  |  |
| **8** | **Biblioteca Gráfica / Visual** | | |
| * Funcionalidades Elementares. & * Funcionalidades Avançadas como:   + tratamento de colisões   + duplo *buffer* |  | *Especificar aqui quais funcionalidades.* |

|  |  |  |  |
| --- | --- | --- | --- |
|  | - Programação orientada e evento em algum ambiente gráfico.  **OU**  *- RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc). |  |  |
| **Interdisciplinaridades via utilização de Conceitos de Matemática Contínua e/ou Física.** | | |
| - Ensino Médio. |  | Especificar quais conceitos aqui. |
| - Ensino Superior*.* |  | Especificar quais conceitos aqui. |
| **9** | **Engenharia de Software** | | |
| - Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. & |  |  |
| - Diagrama de Classes em *UML*. |  |  |
| - Uso efetivo e intensivo de padrões de projeto *GOF*. |  |  |
| - Testes à luz da Tabela de Requisitos e do Diagrama de Classes. |  |  |
| **10** | **Execução de Projeto** | | |
| * Controle de versão de modelos e códigos automatizado (via SVN e/ou afins). & * Uso de alguma forma de cópia de segurança (backup). |  | Especificar qual modo. |
|  | - Reuniões com o professor para acompanhamento do andamento do projeto.  **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]** |  | Especificar quantidade e quando (mínimo 4). |
|  | - Reuniões com monitor da disciplina para acompanhamento do andamento do projeto. |  | Especificar quantidade e quando (mínimo 10). |
|  | - Revisão do trabalho escrito de outra equipe e vice-versa. |  | Especificar qual equipe |
| **Total de conceitos apropriadamente utilizados.**  (Cada grande tópico vale 10% do total de conceitos. Assim, por exemplo, caso se tenha feito metade de um tópico, então valeria 5%.) | | | **70%** (setenta por cento). |

Ressalta-se que as legendas das figuras devem ser colocadas abaixo, enquanto as legendas das tabelas devem ser colocadas acima das mesmas.

Tabela 3. Lista de Justificativas para Conceitos Utilizados.

|  |  |  |
| --- | --- | --- |
| **No.** | **Conceitos** | ***Listar apenas os utilizados* Situação** |
| 1 | **Elementares** | Classe e Objetos foram utilizado porque ... Atributos foram utilizados porque ...  . . . |
| 2 | **Relações** | Associação foi utilizado porque....  ... |
| ... | . . . | ... |
| 9 | **Engenharia de Software** | ...  Padrões de Projeto foram utilizados porque.... |
| 10 | . . . | ... |

# REFLEXÃO COMPARATIVA ENTRE DESENVOLVIMENTOS

Nesta seção se deve apresentar uma reflexão comparativa (sucinta e suficiente) entre o desenvolvimento orientado a objetos e o desenvolvimento procedimental segundo a percepção dos desenvolvedores nos trabalhos realizados no âmbito da disciplina de Fundamentos de Programação 1 e Técnicas de Programação (ou até no âmbito de outras disciplinas ou experiências prévias se for pertinente).

# DISCUSSÃO E CONCLUSÕES

Esta seção deverá apresentar uma reflexão sobre o desenvolvimento e os resultados obtidos. Certamente uma conclusão bem elaborada auxilia na avaliação do Professor. **Outro item ainda mais fundamental para a avaliação são as reuniões com o professor, sendo que o trabalho não pode ser entregue sem elas terem ocorrido, bem entendido**.

Por sua vez, a avaliação do trabalho como um todo (pelo Professor) será baseada em:

* Quantidade e qualidade dos requisitos funcionais cumpridos na elaboração do *software*, à luz do conjunto e qualidade da modelagem e códigos, resultando em número apropriado de classes e objetos, número/forma apropriada de relacionamentos e apropriada complexidade algorítmica. Naturalmente, isto tudo envolve particularmente os bons princípios de Orientação Objetos, como organização, encapsulamento e reutilização, todos baseados no princípio de coesão e desacoplamento doutrinados na disciplina.
* Quantidade e qualidade dos conceitos utilizados na elaboração do *software*, em termos de modelagem e realização de código, o que novamente envolve a correção na aplicação dos princípios da Orientação a Objetos (i.e. coesão, desacoplamento, encapsulamento, organização e reutilização), além da utilização apropriada de cada conceito em si, naturalmente.
* O conteúdo das reuniões e as evoluções a partir delas, bem como a qualidade do trabalho escrito, da apresentação, de diagramas, de códigos e afins.

Em termos gerais, pode-se considerar o primeiro item com um peso de 35%, o segundo com um peso de 35% e o terceiro com um peso de 30%, lembrando que eles estão inter- relacionados. Não obstante, esta porcentagem é relativa, pois (por exemplo) um item muito bem desenvolvido pode até compensar (um pouco) outro não tão bem desenvolvido em proporções diferentes deste referencial dado. Lembrar ainda e novamente que fazer o projeto ser acompanhado pelo professor e monitor, a medida que avança, é **fundamental**.

# CONSIDERAÇÕES PESSOAIS

Caso os estudantes desejem expressar algum sentimento relativo, por exemplo, aos aprendizados e dificuldades encontradas, isto deve ser feita nesta seção opcional. Nesta seção pode-se até utilizar primeira pessoa, entretanto seria melhor a forma impessoal.

Neste sentido, **todas as demais seções devem ser escritas de forma impessoal** (o que significa não usar primeira (e mesmo segunda) pessoa singular ou plural – em suma não usar “eu” ou “nós” no texto). Ademais, salienta-se que o trabalho deve ser redigido em linguagem correta, na forma culta e sem exageros poéticos, com textos não prolixos e bem encadeados.

# DIVISÃO DO TRABALHO

Esta seção deverá ter uma tabela salientando quem desenvolveu cada classe/módulo do *software* e realizou demais atividades como as de ‘engenharia de *software’*, a redação do trabalho escrito, a revisão da redação do trabalho e a preparação da apresentação do trabalho. A tabela 4 pode e mesmo deveria ser melhorada à luz das tabelas de requisitos e conceitos.

Tabela 4. Lista de Atividades e Responsáveis.

|  |  |
| --- | --- |
| **Atividades.** | **Responsáveis** |
| Compreensão de Requisitos | Fulano e Ciclano |
| Diagramas de Classes | Fulano e Ciclano |
| Programação em C++ | Fulano e Ciclano em geral |
| Implementação de *Template* | Fulano |
| Implementação da Persistência dos Objetos... | Ciclano |
| ... |  |
| Escrita do Trabalho | Mais Fulano que Ciclano |
| Revisão do Trabalho | Mais Ciclano que Fulano |

Aqui, após a tabela deve-se **obrigatoriamente** constar o quanto cada um trabalhou no projeto em termos de realização (e.g., modelagem e escrita de código) e colaboração (e.g., revisão de código e testes):

* Fulano trabalhou em 100% das atividades ou as realizando ou colaborando nelas efetivamente.
* Ciclano trabalhou em 20% das atividades ou as realizando ou colaborando nelas efetivamente.

# AGRADECIMENTOS

Havendo agradecimentos de ordem profissional, como ajuda de monitores, estes deverão vir antes das referências. Neste sentido, aqui se pode também salientar e agradecer caso outra equipe tenha revisado o trabalho.

# REFERÊNCIAS CITADAS NO TEXTO

1. DEITEL, H. M.; DEITEL, P. J. C++ Como Programar. 5ª Edição. Bookman. 2006.
2. STADZISZ, P. C. Projeto de Software usando UML. Apostila CEFET-PR 2002. <http://www.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos2/EngSoftware/Apostila%20UM> L%20-%20Stadzisz%202002.pdf
3. SIMÃO, J. M. Site das Disciplina de Fundamentos de Programação 2, Curitiba – PR, Brasil, Acessado em 20/06/2021, às 20:32 - [http://www.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos2/Fundamentos2.htm.](http://www.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos2/Fundamentos2.htm)

# REFERÊNCIAS UTILIZADAS NO DESENVOLVIMENTO

* 1. BEZERRA, E. Princípios de Análise e Projeto de Sistemas com UML. Editora Campus. 2003. ISBN 85-352-1032-6.
  2. HORSTMANN, C. Conceitos de Computação com o Essencial de C++, 3ª edição, Bookman, 2003, ISBN 0-471-16437-2.