CENTRO DE EDUCAÇÃO TECNOLÓGICA DO AMAZONAS- CETAM ESCOLA ESTADUAL DE TEMPO INTEGRAL GOVERNADOR MELO E PÓVOAS CURSO TÉCNICO DE NÍVEL MÉDIO EM INFORMÁTICA

DESENVOLVIMENTO DE UM SISTEMA ESCOLAR UTILIZANDO PROGRAMAÇÃO ORIENTADA A OBJETOS

Modelagem, implementação e teste de um sistema modular em c++

MANAUS-AM

2025

GUSTAVO WILLIAMS GOMES DA SILVA
MARIA LUIZA SERRAO DA SILVA
JHONATA BANDEIRA DE OLIVEIRA
NICOLE BEATRIZ CASCAIS VIDAL
NAYANNE CHRISTINE FONSECA SARAIVA
MARIA EDUARDA CASCAES VITAL

Relatório técnico de atividade prática, apresentando como requisito parcial para aprovação na unidade curricular/curso: linguagem de programação 1- Técnico de Nível Médio em Informática.

Professora Esp. Priscila Gonçalves.

MANAUS-AM

2025

Sumário

1. Introdução

A evolução das tecnologias de informação tem impulsionado instituições a adotarem soluções computacionais capazes de otimizar processos e garantir maior confiabilidade no tratamento de dados. Entre essas instituições, as bibliotecas destacam-se por demandarem sistemas eficientes para o gerenciamento de acervos, usuários e fluxos de empréstimos. A adoção de metodologias de desenvolvimento estruturadas e orientadas a objetos permite a criação de softwares mais robustos, flexíveis e alinhados às necessidades reais de operação.

Este trabalho, intitulado **"Projeto de Software Orientado a Objetos para Gestão de Bibliotecas: Aplicação de Técnicas de Modelagem UML e Implementação em C++ com Ênfase em Modularidade"**, apresenta o desenvolvimento de um sistema destinado à automação de processos biblioteconômicos, desde o cadastro de obras até o controle de empréstimos e devoluções. O uso da linguagem C++ possibilita o emprego de conceitos avançados de modularidade, encapsulamento e reutilização de código, fatores essenciais para a construção de soluções escaláveis.

A modelagem do software é fundamentada em diagramas UML, que permitem representar elementos estruturais e comportamentais do sistema, facilitando a comunicação entre desenvolvedores e garantindo uma visão clara da arquitetura proposta. Ao integrar técnicas de modelagem e implementação orientada a objetos, o projeto busca demonstrar a importância de processos bem definidos no ciclo de desenvolvimento de software.

Dessa forma, este estudo tem como finalidade apresentar uma solução técnica que atenda às necessidades de gestão de bibliotecas, ao mesmo tempo em que evidencia a aplicação prática de conceitos de Engenharia de Software, destacando especialmente a modularidade como princípio fundamental para a construção de sistemas eficientes e de fácil manutenção.

2. Objetivo Geral

Desenvolver um software orientado a objetos destinado à gestão de bibliotecas, empregando técnicas de modelagem UML e implementação em C++ com foco na modularidade. O projeto visa demonstrar a aplicação prática dos principais conceitos da Programação Orientada a Objetos, tais como encapsulamento, herança, polimorfismo e composição, bem como validar a importância da modelagem estruturada para a organização, clareza e manutenção de sistemas. Além disso, o trabalho busca oferecer uma solução didática e adaptável, capaz de servir como base para estudos acadêmicos e para a evolução de projetos mais complexos no contexto da engenharia de software.

3. Objetivos Específicos

- **Identificar e analisar os requisitos funcionais e não funcionais** necessários para o desenvolvimento de um sistema de gestão de bibliotecas.

- **Elaborar modelos UML** que representem a estrutura e o comportamento do software, incluindo diagramas de casos de uso, classes, sequência e atividades.

- **Projetar uma arquitetura modular orientada a objetos**, aplicando princípios de encapsulamento, reutilização e separação de responsabilidades.

- **Implementar o sistema em linguagem C++**, seguindo boas práticas de programação e garantindo organização por meio de módulos e classes.

- **Realizar testes funcionais** para verificar o correto funcionamento das funcionalidades propostas e avaliar a conformidade com os requisitos levantados.

- **Documentar o processo de desenvolvimento**, incluindo decisões de projeto, estrutura do código e resultados obtidos, visando assegurar clareza e reprodutibilidade.

4. Desenvolvimento (com exemplos de POO)

O desenvolvimento do sistema seguiu uma abordagem estruturada, iniciando pela identificação das principais entidades envolvidas no processo de gestão de uma biblioteca: **Livro**, **Usuário**, **Biblioteca** e **Empréstimo**. A partir dessa análise inicial, foi elaborada uma modelagem UML contemplando diagramas de classes e relacionamentos, servindo como base para a implementação do sistema em C++.

A seguir, são apresentados exemplos práticos dos principais conceitos da Programação Orientada a Objetos (POO) aplicados ao projeto.

4.1 Encapsulamento

O encapsulamento foi utilizado com o objetivo de garantir a integridade dos dados, restringindo o acesso direto aos atributos das classes. Dessa forma, cada classe possui atributos privados e métodos públicos responsáveis pelo controle seguro das informações.

Exemplo aplicado à classe **Livro**:

```
class Livro {
private:
   string titulo;
   string autor;
   bool emprestado;

public:
   Livro(string t, string a) : titulo(t), autor(a), emprestado(false) {}

   void emprestar() { emprestado = true; }
   void devolver() { emprestado = false; }

   bool estaEmprestado() const { return emprestado; }
};
```

Nesse modelo, apenas os métodos públicos têm permissão para alterar o estado do livro, evitando manipulação indevida dos atributos.

4.2 Herança

Para permitir especialização de comportamentos, utilizou-se o conceito de herança. A classe **Usuário** serve como classe base, sendo estendida por outras classes como **Aluno** e **Professor**, que podem redefinir comportamentos conforme suas necessidades específicas.

Exemplo:

```
class Usuario {
protected:
   string nome;

public:
   Usuario(string n) : nome(n) {}
   virtual int limiteEmprestimos() const { return 3; }
};

class Professor : public Usuario {
public:
   Professor(string n) : Usuario(n) {}
   int limiteEmprestimos() const override { return 5; }
};
```

A classe **Professor** herda atributos e métodos de **Usuario**, mas redefine o limite de empréstimos permitido.

4.3 Polimorfismo

O polimorfismo permite que objetos derivados sejam tratados como objetos da classe base, mas mantendo seus comportamentos específicos. Isso é possível por meio do uso de métodos virtuais.

Exemplo prático em tempo de execução:

```
Usuario *u1 = new Usuario("Carlos");
Usuario *u2 = new Professor("Ana");

cout << u1->limiteEmprestimos(); // 3
cout << u2->limiteEmprestimos(); // 5
```

Ao chamar o método limiteEmprestimos(), o sistema identifica dinamicamente qual implementação deve ser utilizada, aplicando o princípio do polimorfismo.

4.4 Composição no Controle de Empréstimos

A classe **Biblioteca** foi projetada utilizando composição, uma vez que ela gerencia objetos das classes **Livro** e **Usuario** em sua estrutura interna. Dessa forma, a biblioteca é responsável por manter os registros e controlar as operações realizadas sobre esses objetos.

Exemplo:

```
class Biblioteca {
private:
   vector<Livro> livros;
   vector<Usuario*> usuarios;

public:
   void adicionarLivro(const Livro &l) { livros.push_back(l); }
   void adicionarUsuario(Usuario *u) { usuarios.push_back(u); }

   bool emprestarLivro(int indiceLivro) {
      if (!livros[indiceLivro].estaEmprestado()) {
         livros[indiceLivro].emprestar();
         return true;
      }
      return false;
   }
```

```
};
```

5. Metodologia Utilizada

A metodologia adotada baseou-se nos seguintes pilares:

5.1 Modelagem Estruturada com UML

A etapa de modelagem foi conduzida utilizando a **Linguagem de Modelagem Unificada (UML)**, com ênfase na construção de um **diagrama de classes previamente elaborado**. Esse diagrama serviu como representação visual da estrutura lógica do sistema, permitindo identificar com clareza os principais componentes, seus atributos, métodos e interações. A definição precisa das entidades e de seus respectivos relacionamentos possibilitou uma visão abrangente da arquitetura proposta, garantindo maior organização e coerência durante o desenvolvimento.

5.2 Programação Orientada a Objetos

Na fase de implementação, adotou-se de forma consistente os **quatro fundamentos essenciais da Programação Orientada a Objetos (POO)**: abstração, encapsulamento, herança e polimorfismo. A utilização desses princípios contribuiu para a criação de um sistema mais estruturado, flexível e de fácil expansão. Além disso, foi realizada uma **modularização detalhada do código**, dividindo-o em partes menores e independentes, o que favorece tanto a manutenção quanto a escalabilidade da aplicação ao longo do tempo.

5.3 Desenvolvimento Incremental

O desenvolvimento seguiu uma abordagem **incremental**, na qual o sistema foi construído de maneira gradual, por meio de pequenas entregas sucessivas. Cada incremento adicionava novas funcionalidades ou aprimorava módulos já existentes, permitindo ajustes contínuos durante o processo. Paralelamente a isso, foram realizados **testes progressivos das funcionalidades implementadas**, possibilitando a identificação antecipada de falhas e garantindo a estabilidade do software a cada nova versão liberada.

5.4 Boas Práticas de Código

Para assegurar qualidade e legibilidade ao código-fonte, foram adotadas diversas **boas práticas de programação**. Entre elas, destaca-se a **padronização na nomeação de variáveis, classes e métodos**, garantindo consistência e fácil interpretação por outros desenvolvedores. Priorizou-se também a criação de **métodos curtos e claros**, evitando estruturas complexas e promovendo maior objetividade. Além disso, trechos relevantes do código foram **comentados adequadamente**, contribuindo para a compreensão das funções e facilitando eventuais manutenções futuras.

6. Conclusão

O desenvolvimento do sistema de gerenciamento de bibliotecas em C++ permitiu demonstrar, de forma prática e estruturada, a aplicação dos principais conceitos da Programação Orientada a Objetos, tais como encapsulamento, herança, polimorfismo e composição. A utilização da modelagem UML foi fundamental para orientar a construção das classes, garantindo uma visão clara das responsabilidades, relações e fluxos do software ao longo de todo o processo de implementação.

O projeto atingiu plenamente seus objetivos ao apresentar uma solução funcional, modular e coerente com as boas práticas de engenharia de software. Sua arquitetura organizada facilita a manutenção e a evolução do sistema, permitindo a inclusão de novos recursos, como mecanismos de reserva, controle de multas, emissão de relatórios, cadastro avançado de usuários e a futura integração com interfaces gráficas ou bancos de dados.

Além de oferecer uma base didática sólida para o estudo da POO e da modelagem UML, o sistema desenvolvido demonstra potencial para servir como ponto de partida para aplicações mais complexas em ambientes educacionais ou institucionais. Dessa forma, o trabalho reafirma a importância do uso combinado de boas práticas de modelagem e programação para a criação de soluções escaláveis, compreensíveis e tecnicamente consistentes.

7. Referências Bibliográficas

Referências

BOOCH, Grady. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 1994.

DEITEL, Paul; DEITEL, Harvey. *C++: Como Programar*. 8. ed. Pearson, 2010.

GAMMA, Erich et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

PLANTUML. *PlantUML Documentation*. Disponível em: https://plantuml.com/. Acesso em: dia mês ano.

PRESSMAN, Roger S. *Engenharia de Software: uma abordagem profissional*. McGraw-Hill, 2011.

SOMMERVILLE, Ian. *Engenharia de Software*. 10. ed. Pearson, 2011.