

Instruções para o Projeto - BPP 2025.2

Visão Geral

A primeira entrega (Unidade 1) consiste em três partes integradas:

1. **Planejamento estratégico** do projeto, aplicando conceitos de visão de produto, definição de MVP e organização de um Backlog Priorizado
 2. **Desenvolvimento do MVP inicial** aplicando princípios de código limpo e boas práticas
 3. **Análise e refatoração** do código desenvolvido, identificando code smells e aplicando técnicas de melhoria
-

Fase 1 - Planejamento

1. Visão do Produto

A **visão do produto** é a declaração estratégica que guia todo o desenvolvimento. Ela deve responder: "Por que este produto existe?" e "Qual o impacto desejado?"

Template da Visão do Produto:

```
Para [usuários-alvo]  
Que [problema/necessidade]  
O [nome do produto] é um [categoria do produto]  
Que [benefício principal/capacidade]  
Diferente de [alternativa existente]  
Nosso produto [diferencial único]
```

Exemplo Prático - Sistema de Gestão Financeira Pessoal:

```
Para jovens universitários e profissionais iniciantes  
Que têm dificuldade em controlar gastos e planejar orçamento  
O FinanceTracker é uma aplicação de controle financeiro  
Que permite registro rápido de gastos e visualização de padrões  
Diferente de aplicativos complexos como o Mobills  
Nosso produto foca na simplicidade e gamificação do controle financeiro
```

Checklist da Visão do Produto:

- ☐ Define claramente o usuário-alvo
- ☐ Identifica o problema específico a ser resolvido
- ☐ Explicita o valor único oferecido

- ☐ É inspiradora, mas realista para o escopo deste curso e tempo disponível para dedicação ao projeto
 - ☐ Pode ser desenvolvida em 3-4 meses individualmente ou em grupo de 2 a 4 pessoas
-

2. Produto Viável Mínimo (MVP)

O MVP deve ter três características essenciais: **Viável**, **Valioso** e **Validável**, mas também deve servir como base para aplicação dos conceitos de boas práticas de programação.

2.1 Framework de Definição de MVP:

Problema Central: Qual o principal problema que seu produto resolve?

Hipótese de Valor: "Acreditamos que [usuários] vão [comportamento esperado] porque [benefício percebido]"

CrITÉrios de Qualidade de Código: Como garantir que o MVP seja bem estruturado e e tenha boas características de qualidade?

Exemplo - Sistema de Biblioteca Digital:

Problema Core: Estudantes perdem tempo procurando livros disponíveis na biblioteca

MVP Funcionalidades:

- Cadastro simples de livros (aplicando nomenclatura clara)
- Busca por título (com tratamento de entrada)
- Visualização de disponibilidade (interface limpa)
- Sistema simples de empréstimo/devolução (código bem estruturado)

Requisitos de Código Limpo para o MVP:

- Nomes descritivos para variáveis, funções e classes
 - Funções pequenas com responsabilidade única
 - Comentários apenas onde necessário
 - Formatação consistente
 - Estrutura organizacional clara
-

3. Product Backlog com Critérios de Qualidade

O backlog deve incluir não apenas funcionalidades, mas também critérios de qualidade de código e refatoração.

3.1 Estrutura do Backlog:

Pri	User Story	CrITÉrios de Aceitação	CrITÉrios de Qualidade	Est	Sprint
-----	------------	------------------------	------------------------	-----	--------

Pri	User Story	Critérios de Aceitação	Critérios de Qualidade	Est	Sprint
P1	Como estudante, quero cadastrar uma nova tarefa para não esquecer de fazê-la	<div>- Campos obrigatórios validados</div> <div>- Confirmação visual</div>	<div>- Função de cadastro < 20 linhas</div> <div>- Nomenclatura descritiva</div> <div>- Tratamento de erros</div>	4h	1
P1	Como estudante, quero ver todas as minhas tarefas para ter visão geral	<div>- Lista ordenada</div> <div>- Indicador de urgência</div>	<div>- Separação clara entre lógica e apresentação</div> <div>- Função de listagem reutilizável</div>	3h	1

3.2 Itens Específicos de Qualidade no Backlog:

Refatoração e Code Smells (P2-P3):

- ☐ Identificar e eliminar code smells no módulo principal
- ☐ Refatorar funções longas (Long Method)
- ☐ Eliminar duplicação de código (Duplicate Code)
- ☐ Melhorar nomenclatura ambígua (Poor Naming)
- ☐ Aplicar princípios SOLID onde apropriado

Fase 2 - Desenvolvimento com Boas Práticas

4. Aplicação de Código Limpo

4.1 Princípios Obrigatórios no MVP:

Nomenclatura:

- Nomes intencionais e pronunciáveis
- Evitar abreviações
- Usar termos do domínio do problema

Funções:

- Pequenas (idealmente < 20 linhas)
- Fazem apenas uma coisa
- Nível único de abstração
- Mínimo de parâmetros possível

Formatação:

- Indentação consistente
- Espaçamento vertical adequado
- Linha com no máximo 120 caracteres

Comentários:

- Apenas quando necessário
- Explicam "por que", não "o que"
- Mantidos atualizados com o código

4.2 Estrutura de Projeto Recomendada:

```
projeto/
|-- src/
    |-- models/           # Classes de domínio (nomes claros)
    |-- services/         # Lógica de negócio (funções focadas)
    |-- controllers/      # Interface/entrada (responsabilidade única)
    |-- utils/            # Utilitários reutilizáveis
|-- tests/               # Testes organizados por módulo
|-- docs/                # Documentação
|-- refactoring/         # Documentação das refatorações
|-- README.md            # Visão geral e guia de qualidade
```

4.3 Detalhamento dos Arquivos de Qualidade:

Pasta `/refactoring/`:

1. `code-smells-identified.md` - Catálogo de problemas identificados:

```
# Code Smells Identificados

## 1. Long Method
- Arquivo: src/services/user_service.py
- Linha: 45-78
- Descrição: Método process_user_registration com 34 linhas
- Severidade: Alta
- Ferramenta: pylint (função muito complexa)
- Status: Corrigido em Sprint 2

## 2. Duplicate Code
- Arquivos: src/controllers/auth.py e src/controllers/profile.py
- Linhas: 23-31 e 45-53
- Descrição: Validação de email duplicada
- Severidade: Média
- Status: Pendente
```

2. `refactoring-log.md` - Registro de todas as refatorações:

```
# Log de Refatorações

## Refatoração #1: Extract Method
```

- ****Data****: 2025-09-05
- ****Code Smell****: Long Method em UserService.process_user_registration()
- ****Técnica Aplicada****: Extract Method
- ****Arquivos Afetados****: src/services/user_service.py
- ****Justificativa****: Método fazia validação + processamento + persistência (34 linhas)
- ****Resultado****:
 - validate_user_input(): 8 linhas
 - process_business_logic(): 12 linhas
 - save_user_data(): 6 linhas
- ****Impacto****: Melhor testabilidade e legibilidade
- ****Testes****: Todos os testes passando

Refatoração #2: Rename Variable

- ****Data****: 2025-09-06
- ****Code Smell****: Poor Naming em múltiplos arquivos
- ****Técnica Aplicada****: Rename Variable/Method
- ****Justificativa****: Variáveis como 'data', 'info', 'temp' eram ambíguas
- ****Mudanças****:
 - data → user_registration_data
 - info → product_details
 - temp → formatted_address

3. **before-after-examples/** - Pasta com exemplos práticos:

```
before-after-examples/
|-- long-method-refactor/
|   |-- before.py
|   |-- after.py
|   |-- explanation.md
|-- duplicate-code-elimination/
|   |-- before_auth.py
|   |-- before_profile.py
|   |-- after_shared_validator.py
|   |-- explanation.md
```

4. **quality-metrics.md** - Métricas de qualidade coletadas:

Métricas de Qualidade

Análise Inicial (Sprint 1)

- ****Complexidade Ciclomática Média****: 8.5
- ****Linhas por Método (Média)****: 18.3
- ****Duplicação de Código****: 12%
- ****Code Smells Detectados****: 15

Análise Atual (Sprint 3)

- ****Complexidade Ciclomática Média****: 5.2
- ****Linhas por Método (Média)****: 12.1

```
- **Duplicação de Código**: 3%  
- **Code Smells Detectados**: 4  
- **Code Smells Corrigidos**: 11  
  
## Ferramentas Utilizadas  
- **pylint**: Análise estática Python  
- **radon**: Métricas de complexidade
```

5. Identificação e Tratamento de Code Smells

5.1 Code Smells Prioritários para Identificação:

Nível Método/Função:

- ☐ **Long Method**: Métodos com mais de 20-30 linhas
- ☐ **Long Parameter List**: Mais de 3-4 parâmetros
- ☐ **Duplicate Code**: Código repetido em múltiplos locais
- ☐ **Dead Code**: Código não utilizado

Nível Classe:

- ☐ **Large Class**: Classes com muitas responsabilidades
- ☐ **Data Class**: Classes apenas com dados, sem comportamento
- ☐ **God Class**: Classe que faz tudo

Nível Estrutural:

- ☐ **Feature Envy**: Método mais interessado em outra classe
- ☐ **Inappropriate Intimacy**: Classes muito acopladas

5.2 Processo de Refatoração Documentado:

Para cada refatoração realizada, documente:

1. **Code Smell Identificado**: Qual problema foi encontrado
2. **Técnica de Refatoração**: Qual técnica foi aplicada (do catálogo)
3. **Antes/Depois**: Código antes e depois da refatoração
4. **Justificativa**: Por que essa melhoria era necessária
5. **Impacto**: Como a mudança melhora a qualidade

6. Planejamento

6.1 Cronograma Detalhado (Unidade 1):

Semana 1 (22/08 - Apresentação):

- Apresentação do projeto e conceitos de planejamento ágil
- Definir visão do produto e MVP inicial

Semana 2 (29/08 - Código Limpo):

- **Desenvolvimento:** Implementar primeiras funcionalidades aplicando nomenclatura e formatação
- **Entrega Sprint 1:** Código com nomes descritivos e estrutura clara

Semana 3 (05/09 - Code Smells):

- **Análise:** Identificar code smells no código atual usando ferramentas
- **Documentação:** Catalogar problemas encontrados
- **Desenvolvimento:** Continuar MVP mantendo qualidade

Semana 4 (12/09 - SOLID):

- **Refatoração:** Aplicar princípios SOLID onde apropriado
- **Melhoria:** Reorganizar código seguindo Single Responsibility
- **Desenvolvimento:** Completar funcionalidades principais do MVP

Semana 5 (19/09 - Refatoração):

- **Refatoração Final:** Aplicar técnicas do catálogo de Fowler
- **Documentação:** Registrar todas as refatorações realizadas
- **Preparação:** Finalizar documentos e vídeo de apresentação

Entrega Final (02/10):

- Todos os artefatos com análise de qualidade incluída

6.2 Definition of Done (DoD) por Sprint:**Sprint 1 (Semana 2):**

- ☐ Funcionalidade implementada
- ☐ Código segue convenções de nomenclatura
- ☐ Funções pequenas e focadas
- ☐ Formatação consistente
- ☐ README inicial criado

Sprint 2 (Semana 3):

- ☐ Tudo do Sprint 1 +
- ☐ Code smells identificados e catalogados
- ☐ Pelo menos 2 code smells corrigidos
- ☐ Documentação das correções

Sprint 3 (Semana 4):

- ☐ Tudo do Sprint 2 +
- ☐ Princípios SOLID aplicados onde apropriado
- ☐ Estrutura de classes melhorada
- ☐ Acoplamento reduzido

Sprint Final (Semana 5):

- ☐ Tudo do Sprint 3 +
- ☐ Pelo menos 5 refatorações documentadas
- ☐ Código final limpo e bem estruturado
- ☐ Relatório de qualidade completo

7. Critérios de Avaliação

Critério	Peso	Detalhamento
Clareza da Visão e MVP	15% cada	Problema bem definido, solução coerente
Qualidade do Código	30%	Aplicação de código limpo, nomenclatura, estrutura, formatação
Identificação de Code Smells	20%	Análise crítica, uso de ferramentas, catalogação
Refatorações Realizadas	20%	Técnicas aplicadas, documentação, melhoria do código

8. Documentação de Qualidade

8.1 Relatório de Análise de Código:

```
# Relatório de Qualidade – [Nome do Projeto]

## 1. Aplicação de Código Limpo
### Nomenclatura
- Exemplos de bons nomes utilizados
- Convenções adotadas

### Estrutura de Funções
- Tamanho médio das funções
- Exemplos de funções bem estruturadas

### Formatação
- Padrões de indentação
- Organização visual do código

## 2. Code Smells Identificados
| Code Smell | Localização | Severidade | Status |
|-----|-----|-----|-----|
| Long Method | arquivo.py:50 | Alta | Corrigido |
| Duplicate Code | modulo1.py, modulo2.py | Média | Pendente |

## 3. Refatorações Realizadas
### Refatoração 1: Extract Method
**Antes:**
```python
código original
```



## Depois:

```
código refatorado
```

**Justificativa:** Método muito longo, extraída responsabilidade específica

## 4. Ferramentas Utilizadas

- Análise estática: [nome da ferramenta]
- Métricas de qualidade: [resultados]

## 5. Próximos Passos

- Code smells ainda a corrigir
- Melhorias planejadas para U3

### 8.2 Documentos de Entrega:

#### Obrigatórios:

- ☐ Visão do Produto (PDF, 2-3 páginas)
- ☐ Product Backlog (PDF)
- ☐ Relatório de Qualidade de Código (PDF, 3-4 páginas)
- ☐ Link para o repositório com o código-fonte do projeto
- ☐ Vídeo de apresentação (8-10 minutos)

### 8.3 Estrutura do Vídeo:

**Minutos 1-2:** Apresentação do problema e visão **Minutos 3-4:** Demonstração do MVP funcionando

**Minutos 5-6: Exemplos de código limpo aplicado** **Minutos 7-8: Code smells identificados e refatorações** **Minutos 9-10:** Backlog, próximos passos e lições aprendidas

---

## 9. Ferramentas Recomendadas para Análise

### 9.1 Por Linguagem:

#### Python:

- pylint, flake8, black (formatação)
- radon (métricas de complexidade)

#### Java:

- Checkstyle, PMD, SpotBugs
- SonarLint (IDE integration)

#### JavaScript/TypeScript:

- ESLint, Prettier
- SonarJS

### C/C++:

- cppcheck, clang-tidy
- Valgrind (análise de memória)

## 9.2 Métricas de Qualidade:

- **Complexidade Ciclomática:** < 10 por função
  - **Linhas por Método:** < 20-30 linhas
  - **Duplicação:** < 5% do código total
  - **Cobertura de Comentários:** Comentários úteis, não excessivos
- 

## 10. Exemplos de Aplicação

### 10.1 Antes e Depois - Exemplo de Refatoração:

#### ANTES (com code smells):

```
def process_user_data(name, age, email, phone, address, city, zip_code):
 # Long Parameter List + Long Method
 if not name or len(name) < 2:
 return False
 if not email or "@" not in email:
 return False
 if not phone or len(phone) < 10:
 return False
 if age < 18 or age > 120:
 return False

 user_data = {
 "name": name.strip().title(),
 "age": age,
 "email": email.lower().strip(),
 "phone": phone.replace("-", "").replace(" ", ""),
 "address": address,
 "city": city,
 "zip": zip_code
 }

 # salvar no banco...
 return True
```

#### DEPOIS (refatorado):

```
class UserData:
 def __init__(self, name, age, email, phone, address):
 self.name = name
 self.age = age
 self.email = email
 self.phone = phone
 self.address = address

def validate_user_data(user_data):
 """Valida os dados básicos do usuário."""
 return (validate_name(user_data.name) and
 validate_email(user_data.email) and
 validate_phone(user_data.phone) and
 validate_age(user_data.age))

def validate_name(name):
 """Valida se o nome é válido."""
 return name and len(name.strip()) >= 2

def format_user_data(user_data):
 """Formata os dados do usuário para consistência."""
 return UserData(
 name=user_data.name.strip().title(),
 age=user_data.age,
 email=user_data.email.lower().strip(),
 phone=clean_phone_number(user_data.phone),
 address=user_data.address
)

def process_user_data(user_data):
 """Processa os dados do usuário com validação e formatação."""
 if not validate_user_data(user_data):
 return False

 formatted_data = format_user_data(user_data)
 return save_user_to_database(formatted_data)
```

### Refatorações Aplicadas:

1. **Extract Method:** Separação das validações em funções específicas
2. **Introduce Parameter Object:** UserData class para reduzir parâmetros
3. **Rename Method:** Nomes mais descritivos
4. **Single Responsibility:** Cada função tem uma responsabilidade

---

## 11. Recursos e Apoio

### 11.1 Catálogos de Referência:

- [Catálogo de Code Smells e Refatorações](#)
- Refactoring by Martin Fowler (técnicas clássicas)

- Clean Code by Robert C. Martin (princípios fundamentais)

### 11.2 Ferramentas Online:

- SonarCloud (análise gratuita para projetos públicos)
- CodeClimate (métricas de qualidade)
- GitHub Actions (automação de análise)

### 11.3 Apoio Disponível:

- **Atendimento:** Segundas 14h-16h (online)
  - **Discord da disciplina:** <https://discord.gg/bbMFJBQRT8>
  - **GitHub do curso:** <https://github.com/fmarquesfilho/bpp-2025-2>
- 

## 12. Checklist Final

### 12.1 Antes de Enviar:

- ☐ Todos os arquivos no formato correto
- ☐ Link para repositório com código-fonte incluído
- ☐ **Pelo menos 3 code smells identificados**
- ☐ **Pelo menos 3 refatorações documentadas**
- ☐ **Relatório de qualidade completo**
- ☐ Links de vídeo acessíveis
- ☐ Nomenclatura segue convenções de código limpo

### 12.2 Autoavaliação de Qualidade:

- ☐ Código é legível por outro desenvolvedor
  - ☐ Funções são pequenas e focadas
  - ☐ Nomes são descritivos e intencionais
  - ☐ Estrutura está bem organizada
  - ☐ Code smells foram identificados
  - ☐ Refatorações melhoraram o código
- 

**Data de Entrega:** 02/10/2025 até 23:59 **Plataforma:** SIGAA - Tarefa "Entrega U1" **Formato:** ZIP único com todos os arquivos