



DinDin

Projeto Final de Engenharia de Software

Cristian Herrera

Gabriel Portuguese Portuguese

Thales Marques Rodrigues

Vinícius Fernandes

26 de Novembro de 2025

Parte 1A: Importância das Atividades APG



APG-1: Fundamentação

A primeira atividade foi crucial para estabelecer a base teórica. Ela nos permitiu entender o cenário do problema antes de escrever código, forçando o grupo a alinhar conceitos de requisitos e escopo.






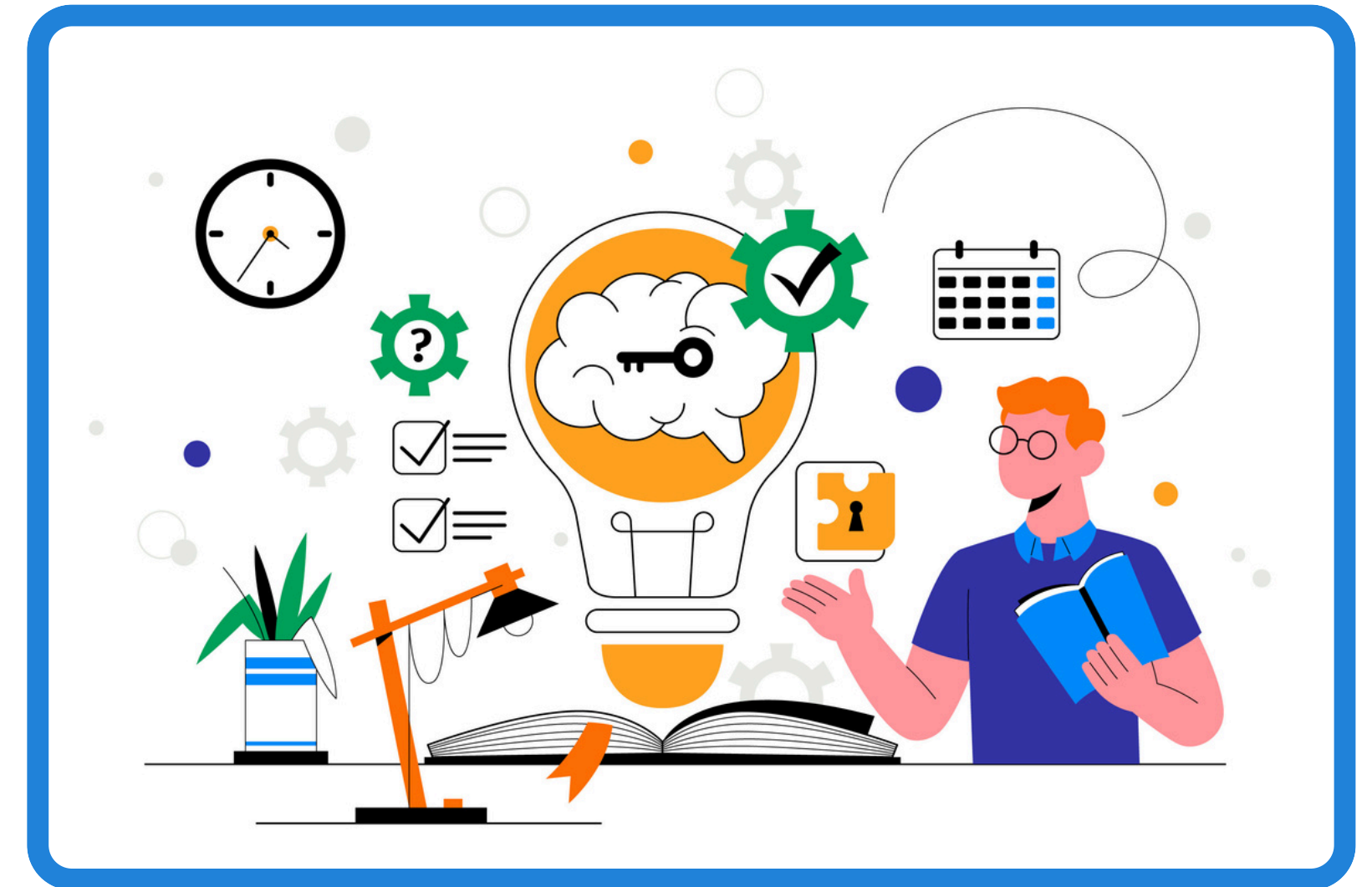
APG-2: Imersão Prática

A segunda etapa consolidou a prática. Foi o momento de transição da teoria para a aplicação das metodologias ágeis, servindo como o "setup" inicial do ambiente de desenvolvimento colaborativo.

Parte 1B: PBL & Metodologias Ágeis

A Aprendizagem Baseada em Projetos (PBL) nos tirou da passividade e nos colocou como protagonistas.

-  **Autonomia:** Busca ativa por soluções (Auth, Deploy).
-  **Rituais Scrum:** Dailies e Sprints quinzenais organizaram o desenvolvimento.
-  **Adaptação:** Mudança de planos conforme novas descobertas técnicas.



Parte 1C: Lições Aprendidas, Desafios e Soluções



Desafios Vivenciados

Desafios Durante o Desenvolvimento

- A adaptação inicial ao fluxo ágil e às ferramentas colaborativas exigiu comunicação constante e alinhamento do grupo.
- O controle de versão trouxe dificuldades no início, especialmente ao lidar com conflitos de merge e diferentes branches simultâneas.
- A integração das partes do código demandou organização para evitar retrabalho e garantir que todos seguissem o mesmo padrão.
- Ajustes recorrentes no escopo e nas regras de negócio exigiram flexibilidade e rápida tomada de decisão do time.

Parte 1C: Lições Aprendidas, Desafios e Soluções



Lições e Soluções Encontradas

Lições Aprendidas e Soluções Adotadas

- O grupo desenvolveu uma comunicação mais madura, aprendendo a discutir problemas rapidamente e compartilhar responsabilidades.
- A prática com Git tornou o time mais preparado para lidar com conflitos, revisões e versionamento profissional.
- A definição de tarefas mais claras e checkpoints semanais melhorou o ritmo e reduziu erros.
- A colaboração aumentou, permitindo que membros ajudassem uns aos outros na solução de bugs e integração de funcionalidades.
- Por fim, o ciclo completo trouxe a visão de um fluxo real de desenvolvimento: levantamento, planejamento, implementação, testes e revisão.

Parte 1D: Pontos Positivos e Negativos da Disciplina



Pontos Positivos

- Vivência prática próxima ao mercado.
- Trabalho em equipe e melhoria na comunicação.
- Uso real do GitFlow e GitHub.
- Prática com metodologias ágeis (sprints e entregas).
- Desenvolvimento técnico com PHP, MVC e testes.



Pontos Negativos

- Dificuldade inicial com comunicação e alinhamento.
- GitFlow confuso no começo (merges e conflitos).
- Carga de atividades alta em algumas sprints.
- Configuração do ambiente às vezes trabalhosa.

Parte 1E: Avaliação da Disciplina



Pontos Positivos

- Vivência prática próxima ao mercado.
- Trabalho em equipe e melhoria na comunicação.
- Uso real do GitFlow e GitHub.
- Prática com metodologias ágeis (sprints e entregas).
- Desenvolvimento técnico com PHP, MVC e testes.



Pontos Negativos

- Dificuldade inicial com comunicação e alinhamento.
- GitFlow confuso no começo (merges e conflitos).
- Carga de atividades alta em algumas sprints.
- Configuração do ambiente às vezes trabalhosa.

Apresentação do Produto:

DinDin é um aplicativo web simples e intuitivo para gerenciamento de finanças pessoais e empresariais.

O foco é ajudar o usuário a controlar receitas e despesas, organizar em categorias personalizadas e manter uma visão clara do saldo mensal.

- Com o DinDin, você pode:**
- Criar sua conta e acessar de forma segura**
- Registrar receitas e despesas de maneira rápida**
- Organizar seus lançamentos por categorias**
- Consultar histórico de transações e filtros**
- Acompanhar saldo do mês em tempo real ?**

Parte 1G: Apresentem sugestões de melhorias para a disciplina.

Algumas sugestões que contribuirão para uma experiência ainda melhor:

- Aulas práticas integradas com mini-projetos semanais. Torna o conteúdo mais aplicado e fácil de fixar.
- Material de apoio mais visual (diagramas, vídeos, fluxos). Facilita compreensão de temas complexos.
- Ambiente de colaboração com feedback contínuo
Ex.: code review guiado, atividades em grupo e painéis interativos.

- Simulações reais de fluxo de trabalho (GitFlow, Scrum, Kanban). Prepare melhor os alunos para a prática de mercado.
- Mais estudos de caso de projetos reais. Ajuda a conectar teoria e prática.



Parte 1H: Vocês têm a liberdade de incrementar a apresentação adicionando quaisquer outros pontos relevantes.

- Pontos adicionais que podem enriquecer a apresentação:
- Demonstração visual do GitFlow (diagrama)
- Exemplos reais de uso de branches no projeto DinDin
- Exemplo de Sprint real com tarefas (TS-Admin-01, US-Admin-01)
- Demonstração de fluxo de CI/CD (caso utilizem no futuro)
- Apresentação de boas práticas de versionamento e commits padronizados
- Lições aprendidas durante o desenvolvimento
- Esse conteúdo extra mostra maturidade técnica e compreensão do processo de desenvolvimento de software.

Parte 2A: Entendendo o Problema

Entendendo o Problema

Para um cenário desconhecido, utilizaríamos:

-  **Análise de mercado:** Em vez de tentar "reinventar a roda", analisamos o que os grandes do mercado já fazem bem.
-  **Imersão do Problema:** Como nós definimos o tema, o grupo assumiu o papel de "dono do produto".



Parte 2B: Priorização de Requisitos

Técnica MoSCoW

- Must Have: Indispensável (MVP).
- Should Have: Importante.
- Could Have: Desejável.
- Won't Have: Fora do escopo.

Valor de Negócio

A prioridade é definida pelo ROI (Retorno sobre Investimento) e Risco.
O que traz mais valor ao cliente com o menor esforço técnico é priorizado.

Parte 2C: Como vocês fariam para realizar a validação dos requisitos?

- A validação dos requisitos seria feita seguindo um fluxo estruturado:
- Entrevistas e reuniões com o cliente/usuário final
R: Para entender necessidades reais, expectativas e restrições.
- Construção de protótipos (wireframes ou mockups)
R: Permite ao usuário visualizar a proposta antes do desenvolvimento.
- Revisão colaborativa com o time
R: Desenvolvedores, PO e stakeholders analisam viabilidade e alinhamento.
- Registro no backlog com critérios de aceitação claros
R: Cada requisito vira uma história de usuário testável.
- Validação contínua durante as sprints
R: O usuário revisa incrementos entregues e valida funcionalidade.

Parte 2D: Como vocês fariam para definição e validação das tecnologias que seriam adotadas?






- A escolha das tecnologias aconteceria em três etapas:
- Análise técnica e comparativa
 - Avaliar performance, documentação, comunidade, estabilidade e custo.
- Prototipação rápida (PoC – Proof of Concept)
 - Criar pequenos testes para medir desempenho, integração e curva de aprendizado.
- Discussão com o time + validação com o PO
 - Confirmar se a tecnologia atende aos requisitos funcionais e não funcionais.
- Critérios utilizados na escolha:
 - Facilidade de manutenção
 - Escalabilidade
 - Segurança
 - Compatibilidade com o ambiente atual
 - Curva de aprendizado do time

Parte 2E: Importância do Planejamento das Iterações (Sprints)

O planejamento das iterações foi fundamental para o andamento do projeto. O Sprint Planning trouxe clareza sobre o que deveria ser feito, como seria feito e qual o objetivo daquela etapa.

Isso ajudou o time a ter:



-  **Resumo claro:** cada sprint tinha metas objetivas e entregáveis definidos.
-  **Organização:** permitiu dividir tarefas, priorizar o que era mais importante e evitar retrabalho.
-  **Previsibilidade:** o grupo sabia exatamente o que seria entregue ao final de cada ciclo.
-  **Melhor comunicação:** facilitou o alinhamento entre os membros e diminuiu ruídos na execução.
-  **Monitoramento constante:** a cada sprint, era possível avaliar o progresso e ajustar a rota se necessário.

Parte 2F: Importância da Definição de Valor em Cada Iteração



O que é “valor” em uma iteração?

- Valor é aquilo que realmente melhora o produto a cada sprint: funcionalidades úteis, correções críticas ou melhorias de experiência.
- Ele garante que o esforço do time impacte diretamente no usuário final.



Por que isso foi essencial no projeto?

- Ajudou na priorização: nem tudo era urgente, então focamos no que realmente agregava.
- Garantiu entregas significativas a cada sprint, fortalecendo o progresso contínuo.
- Facilitou justificar decisões e entender a importância de cada tarefa dentro do ciclo ágil.

Parte 2G: importancia de seguir padroes no projeto

Separação de Responsabilidades:

- Model cuida dos dados, View da tela, Controller da regência.
- Fim do "Código Espaguete" (HTML misturado com SQL).

Testabilidade (QA):



- Permite testar regras de negócio (Auth, Transações) isoladamente.
- Facilita o uso de Mocks e Testes Unitários.

Escalabilidade:

- O projeto cresceu (Analytics, Admin, Perfil). O padrão permite adicionar features sem quebrar o que já existe.

Trabalho em Equipe (4 Devs):

- Padronização: Todos sabem exatamente onde criar ou corrigir arquivos.
- Reduz conflitos de merge e facilita a comunicação.

importancia de seguir uma arquitetura baseada em Serviços por meio da API REST

1. **Multiplataforma:** O mesmo Backend atende Site, Mobile e integrações futuras.
2. **Conectividade:** Padrão universal (HTTP/JSON) para fácil integração com outros sistemas.
3. **Performance:** Backend focado em dados, Frontend focado em experiência.
4. **Segurança:** Ponto único de controle e validação de dados.
5. **Organização:** Separação clara de responsabilidades facilita manutenção e testes.

Parte 2K: Como vocês veem o Fluxo de Trabalho organizado pelo GitFlow?

- O GitFlow organiza o trabalho de maneira clara e eficiente:
- Separação entre desenvolvimento contínuo e versões estáveis
 - – main mantém versões prontas para produção
 - – develop recebe implementações em andamento
- Branches específicas para funcionalidades
 - – Facilita controle, testes e revisão de cada feature
 - – Ex.: feature-TS-Admin-01-US-admin02
- Padronização do fluxo
 - – Cada membro da equipe segue o mesmo processo
 - – Evita conflitos, reescrituras e bagunça no repositório
- Preparação organizada de releases
 - – Release branches permitem testes finais antes da produção
- Hotfix rápido e isolado
 - – Correções emergenciais não impactam o desenvolvimento
- Conclusão:

O GitFlow melhora a colaboração, reduz conflitos e deixa o ciclo de desenvolvimento profissional e seguro.

Parte 2L: O que vocês entendem por Release do produto?

- Uma Release é:
- Um conjunto de funcionalidades finalizadas, testadas e aprovadas para distribuição ao usuário ou cliente.
- Como entendemos Release:
 - É o marco de entrega do produto ou versão
 - – Representa um incremento completo e utilizável.
- Passa por validações, testes e revisão final
 - – Garantia de estabilidade antes de ser liberado.
- Possui documentação própria
 - – Lista de mudanças, melhorias, correções e instruções.
- Marca a evolução do software
 - – Ex.: v1.0, v1.1, v2.0

Obrigado!



<https://github.com/Thales-uft2022-2/DinDin.git>