1. Planejamento Estratégico do Cronograma

- **Decisões estratégicas iniciais críticas:**
    - **Definição clara do escopo e das exclusões:** O documento de escopo já aponta para a importância de definir claramente as entregas e exclusões, como a não inclusão de uma versão iOS e integração com redes sociais. Isso é crucial para evitar o "escopo inchado" em um projeto com prazo fixo.
    - **Priorização de funcionalidades:** Dada a restrição de tempo, é fundamental priorizar as funcionalidades essenciais para a Semana da Computação, como check-in, programação e Q&A em tempo real, em detrimento de funcionalidades secundárias.
    - **Estabelecimento de um processo formal de controle de mudanças:** O documento de escopo já prevê um comitê de mudança com análise de impacto técnico e temporal para controlar as mudanças. Isso é vital para gerenciar riscos de atrasos decorrentes de novas solicitações.
    - **Estimativas realistas e com reservas:** A equipe precisa decidir o nível de exatidão aceitável para as estimativas de duração. A inclusão de reservas para contingências para riscos identificados ("incógnitas conhecidas") é crucial para lidar com imprevistos, como bugs complexos ou atrasos em APIs.
    - **Comunicação clara e frequente com stakeholders:** Manter uma comunicação constante e clara com os stakeholders é essencial para validar requisitos e evitar desvios.
- **Impacto da metodologia de cronograma no gerenciamento de risco:**
    - A menção a conceitos ágeis no desenvolvimento do cronograma e a premissa de que a equipe possui conhecimento básico em metodologias ágeis sugerem uma abordagem mais adaptativa.
    - **Metodologia ágil/adaptativa:** Em um projeto de software onde os requisitos podem evoluir, uma metodologia ágil (como Scrum ou Kanban) permite maior flexibilidade para incorporar mudanças e gerenciar riscos. Com iterações curtas e feedback contínuo, a equipe pode identificar e mitigar riscos mais cedo, como atrasos na API ou bugs complexos. A validação constante dos requisitos com os stakeholders é uma forma de evitar o escopo inchado, um dos desafios citados.
    - **Metodologia preditiva:** Embora o desenvolvimento ágil seja sugerido, uma abordagem preditiva mais rígida pode introduzir riscos se não houver flexibilidade para adaptação. No entanto, o detalhamento das atividades e a identificação do caminho crítico, processos presentes em abordagens preditivas, são ferramentas

valiosas para monitorar e controlar o cronograma em um prazo fixo. O controle contínuo do cronograma é vital em projetos de software.

Exemplo para o aplicativo:

- **Nível de exatidão aceitável para as estimativas de duração:** A equipe poderia decidir por estimativas mais detalhadas para as primeiras iterações (por exemplo, "Cadastro e Login") e estimativas de alto nível para funcionalidades futuras, refinando-as à medida que o projeto avança.
- **Unidades de medida para os recursos do projeto:** "Horas-pessoa" ou "dias de trabalho" seriam as unidades mais adequadas para estimar as atividades de desenvolvimento de software.
- **Software de gerenciamento de projetos mais adequado:**
    - Para gerenciar o desenvolvimento e as interfaces do aplicativo, considerando uma abordagem ágil e a necessidade de colaboração, ferramentas como **Jira** ou **Trello** seriam altamente adequadas. Elas permitem o acompanhamento de tarefas, a visualização do progresso (kanban boards), e a integração com repositórios de código. **Azure DevOps** também seria uma excelente opção, oferecendo um conjunto mais completo de ferramentas para o ciclo de vida de desenvolvimento de software.

2. Detalhamento e Sequenciamento Lógico das Atividades

- **Nível de detalhe ideal:** O nível de detalhe ideal para as atividades em um projeto de software deve ser suficiente para permitir estimativas realistas, atribuição de responsabilidades e acompanhamento do progresso, mas não tão granular a ponto de se tornar um microgerenciamento oneroso. A EAP inicial já oferece uma boa granularidade em pacotes de trabalho, como "Desenvolvimento do Aplicativo" que se decompõe em "Tela de programação", "Sistema de check-in", etc.. A próxima etapa seria detalhar essas funcionalidades em atividades ainda menores e acionáveis.
- **Impacto do detalhamento excessivo ou insuficiente:**
    - **Detalhamento excessivo:** Pode levar a um overhead de planejamento, com muito tempo gasto na criação e atualização de tarefas minúsculas, resultando em perda de tempo e baixa agilidade. Também pode desmotivar a equipe ao criar uma percepção de lentidão no progresso.
    - **Detalhamento insuficiente:** Dificulta a estimativa precisa, a identificação de dependências e a atribuição de responsabilidades. Isso pode levar a atrasos, falta de controle e a identificação tardia de problemas.
- **Identificação precisa de dependências e uso de antecipações e esperas:**

- **Otimização do fluxo de trabalho:** A identificação precisa das dependências lógicas (Fim-Início, Início-Início, Fim-Fim, Início-Fim) é crucial. O uso estratégico de **antecipações (leads)** permite que uma atividade sucessora comece antes que a predecessora termine, otimizando o paralelismo e acelerando o projeto. Por exemplo, o desenvolvimento do frontend de uma funcionalidade pode começar antes que o backend esteja 100% concluído, desde que a API esteja minimamente definida.
- **Introdução de riscos:** Se as antecipações ou **esperas (lags)** forem mal aplicadas:
  - **Antecipações excessivas:** Podem levar a retrabalho se a atividade predecessora sofrer alterações significativas.
  - **Esperas excessivas:** Podem atrasar o projeto desnecessariamente, introduzindo gargalos. Por exemplo, uma espera muito longa para uma revisão de segurança que poderia ser feita em paralelo com outras atividades.
  - **Dependências não identificadas:** Levarão a atrasos inesperados e retrabalho quando uma atividade não puder começar porque sua predecessora não foi concluída.

Exemplo para o aplicativo:

Vamos escolher a funcionalidade principal "Cadastro e Login de Usuários" e detalhá-la.

- **Funcionalidade:** Cadastro e Login de Usuários
- **Lista de atividades específicas:**
  - **A1: Levantar requisitos detalhados de Cadastro e Login.**
  - **A2: Modelar banco de dados para usuários.**
  - **A3: Desenvolver API de Cadastro (Backend).**
  - **A4: Desenvolver API de Login (Backend).**
  - **A5: Desenvolver interface de Cadastro (Frontend).**
  - **A6: Desenvolver interface de Login (Frontend).**
  - **A7: Integrar Frontend de Cadastro com Backend.**
  - **A8: Integrar Frontend de Login com Backend.**
  - **A9: Testar funcionalidade completa de Cadastro e Login.**
  - **A10: Realizar revisão de segurança.**
- **Diagrama de rede do cronograma (exemplo, usando o Método do Diagrama de Precedência - MDP):**

Unset

```
A1 (Levantar Requisitos) --> A2 (Modelar BD)

A2 (Modelar BD) --> A3 (Desenvolver API Cadastro)

A2 (Modelar BD) --> A4 (Desenvolver API Login)

A3 (Desenvolver API Cadastro) --> A7 (Integrar Frontend Cadastro)

A4 (Desenvolver API Login) --> A8 (Integrar Frontend Login)

A1 (Levantar Requisitos) --> A5 (Desenvolver Interface Cadastro)
-- (com lead) --> A7 (Integrar Frontend Cadastro)

A1 (Levantar Requisitos) --> A6 (Desenvolver Interface Login) --
(com lead) --> A8 (Integrar Frontend Login)

A7 (Integrar Frontend Cadastro) --> A9 (Testar Funcionalidade)

A8 (Integrar Frontend Login) --> A9 (Testar Funcionalidade)

A9 (Testar Funcionalidade) --> A10 (Revisão de Segurança) -- (com
lag) --> Deploy (fora deste exemplo)
```

- **Aplicação de antecipações (leads) ou esperas (lags):**
    - **Antecipação (Lead):** Poderíamos aplicar uma antecipação para paralelizar o trabalho entre o desenvolvimento da interface (Frontend) e o desenvolvimento da API (Backend). Por exemplo, "Desenvolver Interface de Cadastro" (A5) pode começar assim que os requisitos (A1) estiverem claros, mesmo que a "API de Cadastro" (A3) não esteja 100% pronta. Isso significa que A5 pode começar com um "Fim-Início com Lead" de A1, e A7 ("Integrar Frontend de Cadastro") pode começar antes que A3 esteja totalmente concluída, desde que a interface da API já esteja definida.
    - **Espera (Lag):** Uma espera poderia ser aplicada em "Revisão de Segurança" (A10) antes do deploy. Por exemplo, "Revisão de Segurança" (A10) pode ter uma espera de 2 dias após "Testar Funcionalidade Completa" (A9) para garantir que haja tempo suficiente para a revisão e quaisquer correções necessárias antes do lançamento.

3. Estimativa de Recursos e Duração com Análise de Reservas

- **Maiores desafios na estimativa de duração em projetos de software com recursos limitados:**
    - **Incerteza de requisitos:** Requisitos podem evoluir, levando a retrabalho e prolongando a duração.
    - **Complexidade intrínseca do software:** Bugs complexos e problemas de integração são difíceis de prever.
    - **Produtividade variável da equipe:** A produtividade de cada membro da equipe pode variar, afetando as estimativas.
    - **Dependências externas:** Atrasos em APIs externas ou na disponibilidade de outras equipes (como a de infraestrutura para instabilidade de ambiente) podem impactar o cronograma.
    - **Estimativas otimistas:** A tendência de subestimar o tempo necessário para as tarefas.
    - **Gerenciamento de Expectativas:** A dificuldade em comunicar a incerteza das estimativas aos stakeholders e justificar prazos mais longos.
- **Justificativa para inclusão de reservas de contingência:**
    - As reservas de contingência são essenciais para lidar com "incógnitas conhecidas", ou seja, riscos identificados no planejamento do projeto.
    - **Atrasos na API:** Se o aplicativo depende de uma API externa para, por exemplo, integração com o sistema de inscrição do DECSI, pode haver atrasos na entrega ou na estabilidade dessa API. Uma reserva permitiria à equipe absorver esse atraso sem impactar o prazo final.
    - **Bugs complexos:** É quase certo que surgirão bugs inesperados durante o desenvolvimento e testes. A reserva de contingência aloca tempo para a investigação e correção desses bugs, evitando que o cronograma seja comprometido.
    - **Instabilidade de ambiente:** Problemas com o ambiente de desenvolvimento, servidores de teste ou ferramentas podem consumir tempo inesperado. A reserva oferece uma "colchão" para lidar com esses imprevistos.
    - **Mudanças de escopo:** Embora o projeto tenha um processo de controle de mudanças, pequenas alterações ou esclarecimentos de requisitos podem consumir tempo adicional.
- **Opinião especializada de desenvolvedores experientes como ferramenta valiosa:**
    - Desenvolvedores experientes possuem um histórico de projetos semelhantes e um conhecimento aprofundado das tecnologias envolvidas.

    - Eles podem fornecer estimativas mais realistas, baseadas em sua experiência com complexidades técnicas, potenciais armadilhas e a produtividade real de desenvolvimento.
    - Sua experiência permite identificar riscos técnicos específicos que podem não ser óbvios para gerentes de projeto menos técnicos.
    - Eles podem auxiliar na validação de estimativas, seja através de técnicas como "planejamento poker" em metodologias ágeis ou revisões formais de estimativas em abordagens preditivas.

4. Desenvolvimento do Cronograma e o Caminho Crítico

- **Como a análise do caminho crítico guia a equipe em um ambiente de prazo fixo:**
    - O Caminho Crítico representa a sequência de atividades mais longa e determina a duração mínima possível do projeto.
    - **Foco na execução:** A equipe sabe que qualquer atraso em uma atividade do caminho crítico impactará diretamente o prazo final do projeto. Isso permite que a equipe priorize a execução dessas atividades e aloque os melhores recursos para elas.
    - **Identificação de gargalos:** A análise do caminho crítico revela os gargalos potenciais do projeto, permitindo que a equipe tome medidas proativas para mitigar riscos associados a essas atividades.
    - **Tomada de decisões informadas:** Se houver necessidade de otimizar o cronograma, a equipe pode focar em acelerar as atividades do caminho crítico (por exemplo, através de *fast tracking* ou *crashing*).
    - **Monitoramento e controle:** O monitoramento contínuo do progresso das atividades do caminho crítico é crucial para o controle do cronograma.
- **O que acontece se uma atividade no caminho crítico atrasar:**
    - Se uma atividade no caminho crítico atrasar, a duração total do projeto será automaticamente estendida em um período igual ao atraso, a menos que ações corretivas sejam tomadas. Isso é um risco significativo em um ambiente de prazo fixo como a Semana da Computação. A equipe precisará avaliar o impacto e propor ações para compensar o atraso, como trabalhar horas extras, adicionar recursos (se possível), ou cortar funcionalidades de menor prioridade.
- **Como as técnicas de otimização de recursos (como nivelamento de recursos) podem alterar o caminho crítico e a duração do projeto:**
    - **Nivelamento de recursos:** É uma técnica utilizada para ajustar o cronograma quando há superalocação de recursos (por exemplo, a mesma pessoa designada para duas tarefas simultâneas) ou quando os recursos são limitados.

        - **Como altera o caminho crítico:** O nivelamento de recursos geralmente envolve o atraso de atividades que não estão no caminho crítico para evitar superalocação. No entanto, se uma atividade que anteriormente tinha folga for atrasada para nivelar recursos, ela pode se tornar parte do caminho crítico, ou até mesmo criar um novo caminho crítico.
        - **Como altera a duração do projeto:** O nivelamento de recursos pode, em alguns casos, prolongar a duração do projeto se as atividades precisarem ser atrasadas para esperar a disponibilidade de um recurso. Em um ambiente de prazo fixo, essa técnica precisa ser aplicada com cautela e, se prolongar o prazo, a equipe precisará reavaliar o escopo ou buscar outras formas de compressão.

5. Controle do Cronograma e Gerenciamento de Mudanças

Reflexão: Por que o controle contínuo do cronograma é vital, especialmente em projetos de software onde os requisitos podem evoluir? Como a equipe pode usar as informações sobre o desempenho do trabalho e as previsões para antecipar desvios e tomar ações corretivas?

- **Por que o controle contínuo do cronograma é vital em projetos de software com requisitos evolutivos:**
    - **Volatilidade dos requisitos:** Requisitos em projetos de software são frequentemente sujeitos a mudanças devido ao feedback dos usuários, novas tecnologias ou alterações nas prioridades de negócios. O controle contínuo permite que a equipe monitore o impacto dessas mudanças no cronograma.
    - **Identificação precoce de desvios:** Monitorar o progresso em relação à linha de base do cronograma permite identificar desvios (atrasos ou adiantamentos) o mais cedo possível. Isso é crucial para tomar ações corretivas antes que o problema se agrave.
    - **Gerenciamento de riscos:** O controle do cronograma ajuda a identificar novos riscos ou a reavaliar riscos existentes que podem impactar o prazo.
    - **Transparência e comunicação:** Fornece visibilidade do status do projeto para todos os stakeholders, permitindo comunicação proativa sobre o progresso e quaisquer desafios.
    - **Tomada de decisões informadas:** As informações de controle do cronograma são a base para decidir se são necessárias solicitações de mudança para o escopo, cronograma ou orçamento.

- **Como a equipe pode usar informações de desempenho do trabalho e previsões para antecipar desvios e tomar ações corretivas:**
    - **Informações de desempenho do trabalho:** Isso inclui a porcentagem de conclusão das atividades, a taxa de queima (burn rate) de horas, e a comparação do trabalho real com o planejado. Se uma atividade está demorando mais do que o previsto, isso é um indicador de desvio.
    - **Previsões de cronograma:** Usando dados de desempenho, a equipe pode projetar o futuro do projeto e prever a data de conclusão.
        - **Antecipar desvios:** Se as previsões indicam que o projeto será concluído após o prazo estabelecido, a equipe pode antecipar um desvio. Por exemplo, se o desenvolvimento do sistema de check-in está atrasado, as previsões mostrarão um impacto na data de conclusão geral.
        - **Tomar ações corretivas:** Com base nessas previsões, a equipe pode:
            - **Reavaliar a prioridade das atividades:** Redirecionar recursos para atividades críticas.
            - **Otimizar recursos:** Realocar membros da equipe para atividades que estão atrasadas ou aplicar nivelamento de recursos.
            - **Compressão de cronograma:** Utilizar técnicas como *crashing* (adicionar recursos para acelerar) ou *fast tracking* (executar atividades em paralelo que seriam sequenciais) nas atividades do caminho crítico.
            - **Revisar o escopo:** Se o desvio for muito grande, pode ser necessário negociar a redução do escopo para cumprir o prazo.
            - **Emitir solicitações de mudança:** Se as ações corretivas não forem suficientes, pode ser necessário formalizar uma solicitação de mudança para alterar a linha de base do cronograma, do escopo ou de ambos.

Exemplo para o aplicativo:

Cenário: O projeto do aplicativo está na metade do caminho, e surge uma nova funcionalidade prioritária ("Integração com Sistema de Inscrição do DECSI") que não estava no escopo inicial, mas o prazo para a Semana da Computação permanece o mesmo.

Discussão: Como a equipe usaria o processo de Controlar o Cronograma para avaliar o impacto dessa nova solicitação de mudança? Que ferramentas de controle

de mudanças seriam aplicadas? Eles deveriam usar técnicas de otimização de recursos ou compressão de cronograma para acomodar a nova funcionalidade, ou seria mais apropriado emitir uma solicitação de mudança para o escopo, cronograma, ou ambos?

- **Avaliação do impacto da nova solicitação de mudança (Controlar o Cronograma):**
    - **Análise de impacto:** A equipe usaria o processo de Controle do Cronograma para analisar o impacto da nova funcionalidade nas atividades existentes, nas dependências, nas estimativas de duração e nos recursos.
    - **Identificação de atividades adicionais:** Detalharia as atividades necessárias para a "Integração com Sistema de Inscrição", estimando sua duração e os recursos necessários.
    - **Atualização do cronograma:** Integraria essas novas atividades ao modelo do cronograma existente para visualizar o impacto no caminho crítico e na data de término.
    - **Análise de cenário:** Simularia diferentes cenários (ex: adicionar recursos, cortar funcionalidades existentes, aceitar o atraso) para entender as implicações.
- **Ferramentas de controle de mudanças:**
    - **Sistema de Controle Integrado de Mudanças:** O projeto já estabeleceu um processo formal de controle de mudanças. Uma ferramenta de gerenciamento de projetos como Jira ou Azure DevOps pode ser configurada para gerenciar solicitações de mudança.
    - **Formulário de Solicitação de Mudança:** Um formulário padronizado que detalha a solicitação, justificativa, impacto estimado no cronograma, custo e escopo.
    - **Comitê de Controle de Mudanças (CCM):** O documento de escopo já menciona um comitê de mudança com análise de impacto técnico e temporal. Este comitê, composto por stakeholders chave, analisaria a solicitação, o impacto e tomaria a decisão sobre a aprovação ou rejeição.
- **Decisão sobre acomodar a nova funcionalidade:**
    - **Técnicas de otimização de recursos ou compressão de cronograma:**
        - Se o impacto for pequeno e a equipe tiver folga (slacks) em atividades não-críticas, poderiam tentar **nivelamento de recursos** para otimizar a alocação.

        - Se a nova funcionalidade for de alta prioridade e o impacto for significativo, poderiam considerar **compressão de cronograma** (crashing ou fast tracking). No entanto, isso pode aumentar os custos e os riscos, exigindo horas extras ou a contratação de recursos adicionais (o que vai contra a restrição de orçamento limitado).
        - A avaliação de cada técnica (se o custo e risco valem a pena) seria feita pelo comitê.
    - **Emissão de solicitação de mudança para o escopo, cronograma, ou ambos:**

        - Dada a restrição de prazo fixo para a Semana da Computação, a emissão de uma solicitação de mudança é muito provável e, muitas vezes, a abordagem mais responsável.
        - **Solicitação de mudança no escopo:** Se a nova funcionalidade não puder ser adicionada sem comprometer o prazo, a equipe pode solicitar que a nova funcionalidade seja adicionada em troca da remoção de alguma funcionalidade de menor prioridade que já estava no escopo (redução do escopo).
        - **Solicitação de mudança no cronograma:** Se a "Integração com Sistema de Inscrição" for indispensável e não houver flexibilidade no escopo ou recursos, a equipe precisaria solicitar uma extensão do prazo. No entanto, em um evento com data fixa, isso seria extremamente problemático e poderia levar à rejeição da solicitação.
        - **Solicitação de mudança em ambos:** Uma combinação é frequentemente a mais realista: adicionar a nova funcionalidade, mas estender o prazo ligeiramente (se houver alguma flexibilidade) e/ou remover funcionalidades de menor prioridade.
    - **Priorização:** A decisão final dependeria da prioridade da nova funcionalidade em relação às funcionalidades existentes e da tolerância dos stakeholders a um atraso ou uma redução de escopo. A equipe precisaria apresentar ao comitê de mudanças uma análise clara dos impactos e das opções disponíveis.

Mês	Semana	Atividades Chave & Entregas	Referência EAP (se aplicável)	Processo PMBOK Relevante	Notas & Considerações
Mês 1: Planejamento, Design & Desenvolvimento Central					Foco em elementos fundamentais e primeiros recursos chave.
	Semana 1-2	- Refinar o plano do projeto e o plano de gerenciamento do escopo. - Formalizar os papéis da equipe e os canais de comunicação. - Configurar as ferramentas de gerenciamento de projetos (Jira/Trello/Azure DevOps). - Realizar entrevistas e workshops aprofundados com as partes interessadas. - Priorizar as funcionalidades (Check-in, Programação, Q&A). - Elaborar requisitos detalhados para as funcionalidades iniciais (Login/Cadastro, Visualização básica do Programa).	1.1 Planejamento inicial, 1.2 Coordenação da equipe, 2.1 Entrevistas com stakeholders, 2.2 Workshop de brainstorming, 2.3 Elaboração da documentação de requisitos, 3.1 Prototipagem da interface, 3.2 Modelagem de banco de dados, 3.3 Arquitetura do sistema	5.1 Planejamento do Gerenciamento do Escopo, 5.2 Coleta de Requisitos, 5.3 Definição Detalhada do Escopo	- Enfatizar uma definição clara do escopo para evitar o desvio. - Aproveitar a opinião de especialistas para estimativas realistas. - Começar com os fluxos de usuário mais críticos.

		- Desenvolver wireframes e maquetes para telas principais. - Definir a arquitetura geral do sistema e o modelo de banco de dados.			
	Semana 3-4	Sprint 1 de Desenvolvimento Central: Cadastro e Login - Desenvolver API para registro e login do usuário. - Desenvolver interfaces Frontend para registro e login. - Integrar Frontend com Backend para esses recursos. - Realizar testes unitários para os módulos desenvolvidos.	4.1 Tela de programação (inicial), 4.5 Perfil do usuário (Login/Cadastro), 5.1 Teste unitário	5.4 Estrutura Analítica do Projeto (EAP), 6.1 Desenvolver Atividades, 8.1 Controlar Qualidade	- Aplicar leads para desenvolvimento paralelo (por exemplo, Frontend e Backend). - Validação constante com as partes interessadas para evitar o desvio do escopo.
Mês 2: Desenvolvimento & Integração de Recursos					Construção de funcionalidades principais, com testes e feedback contínuos.

	Semana 5-6	Sprint 2 de Desenvolvimento de Recursos: Programação & Q&A - Desenvolver API para dados de programação. - Implementar exibição de programação com filtros e localização. - Desenvolver sistema de Q&A em tempo real (envio, votação, exibição). - Realizar testes internos e compartilhar builds preliminares com os organizadores para feedback. - Revisar o feedback e gerenciar possíveis mudanças de escopo através do comitê de controle de mudanças.	4.1 Tela de programação, 4.4 Sistema de perguntas e respostas, 5.2 Teste de usabilidade, 5.3 Validação com stakeholders, 1.3 Controle de mudanças	5.3 Definição Detalhada do Escopo, 6.2 Desenvolver Atividades, 8.2 Realizar Garantia da Qualidade, 4.4 Controlar o Escopo	- Priorizar as funcionalidades essenciais para o evento. - Usar um processo formal de controle de mudanças. - Monitorar as informações de desempenho do trabalho.

	Semana 7-8	Sprint 3 de Desenvolvimento de Recursos: Check-in & Agenda Personalizada - Implementar check-in por QR Code e registro de presença. - Desenvolver agenda personalizada (marcando interesses, lembretes). - Realizar testes internos extensivos para novos recursos. - Realizar uma sessão de validação mais formal com as principais partes interessadas. - Fornecer relatórios de progresso regulares para as partes interessadas.	4.2 Sistema de check-in, 4.3 Montagem de agenda personalizada, 5.2 Teste de usabilidade, 5.3 Validação com stakeholders, 1.4 Relatórios de progresso	6.2 Desenvolver Atividades, 8.3 Controlar Qualidade, 4.5 Controlar o Cronograma	- Abordar possíveis atrasos proativamente. - Priorizar as atividades do caminho crítico.
Mês 3: Refinamento, Testes, Implantação & Documentação					Empurrão final para entrega, garantia de qualidade e encerramento do projeto.

	Semana 9-10	Desenvolvimento Final de Recursos & Testes Abrangentes - Desenvolver recursos de perfil de usuário (nome, curso, foto). - Implementar painel do organizador. - Integrar informações adicionais (perfis de palestrantes, mapa, notícias). - Fase dedicada de correção de bugs. - Realizar testes abrangentes de integração, sistema e desempenho. - Realizar uma revisão de segurança. - Alocar tempo de buffer para problemas imprevistos (reservas de contingência).	4.5 Perfil do usuário, 4.6 Painel do organizador, 5.4 Correção de bugs, (implicitamente 5.1 Teste unitário, 5.2 Teste de usabilidade, 5.3 Validação com stakeholders)	6.2 Desenvolver Atividades, 8.3 Controlar Qualidade, 4.5 Controlar o Cronograma	- As reservas de contingência são cruciais para "incógnitas conhecidas". - Opinião de desenvolvedores experientes para estimativas realistas.

	Semana 11-12	Preparação e Documentação para Implantação - Preparar o aplicativo para envio à Google Play Store. - Concluir o processo de envio. - Fornecer treinamento aos organizadores do evento sobre como usar o painel de administração. - Ajudar na promoção do aplicativo para os participantes. - Finalizar a documentação técnica e os manuais de usuário/organizador. - Preparar para a avaliação do projeto e lições aprendidas pós-evento.	6.1 Publicação na Play Store, 6.2 Trein		