DCC App: Um Aplicativo do DCC Construído Usando Boas Práticas e Técnicas de Engenharia de Software

Helio Victor Flexa dos Santos

[1]Universidade Federal de Minas Gerais - Belo Horizonte - Minas Gerais - Brasil

[2]Departamento de Ciência da Computação - UFMG

helio.santos@dcc.ufmg.br

Resumo. *Nessa documentação, detalharemos o processo de estudo, aprendizado e implementação de um software de produção desenvolvido com base nas boas práticas e técnicas de Engenharia de Software. Nesse contexto, explicitamos todo mapeamento feito desde a idealização do produto, definição da arquitetura, modelagem,implementação, evolução e manutenção no estado atual.*

1. Introdução

Nos tempos atuais, os softwares estão presentes no dia-a-dia das pessoas, seja eles para fins de trabalho, estudo ou lazer. Com esse uso intenso no mundo atual, a qualidade das aplicações é um fator determinante para que ele tenha destaque nos demais no ambiente de desenvolvimento de software. Um bom planejamento do projeto, boas práticas de programação, o uso de frameworks estáveis, bons testes, uma boa arquitetura e outros, podem ser considerados fatores primordiais para que a excelência do software seja alcançada e com ela a popularidade por parte dos consumidores finais seja alcançada gerando retorno positivo para a empresa.

Para que essa qualidade esteja presente na sua aplicação é necessário que desde a concepção de esboços, instalação de todo ambiente, implementação, manutenção e evolução, seja elaborado seguindo as métricas e qualidades debatidas da área de Engenharia de Software, entretanto, em um cenário real por diversos fatores desde a inexperiência da equipe, contratempos e adversidades, muitas dessas práticas são deixadas de lado ou até mesmo abandonadas, ocasionando em problemas ao longo do projeto como bugs, alterações inesperadas de compatibilidade ou mudanças abruptas nos requisitos do sistema.

Sendo assim, neste projeto desenvolvemos um aplicativo, DCC News, ao qual pretende seguir a construção concreta de um aplicativo na visão do que a área prega em termos acadêmicos, sendo desde a idealização do produto, planejamento das funcionalidades, desenvolvimento da arquitetura, padrões de projeto, testabilidade, manutenibilidade e evolução do software.

É extremamente comum que empresas de desenvolvimento de software não sigam ou contemplem alguns conceitos empregados pela Engenharia de Software, seja por limitação de escopo ou tempo do projeto. Com isso, é possível perceber que durante a fase de desenvolvimento essas equipes acabam tendo alguns problemas recorrentes já descritos anteriormente como mudanças abruptas nos requisitos por falta de alinhamento com os clientes, problemas de definição de projeto que impactam na construção da arquitetura ou na extensão de alguma funcionalidade.

Em relação a isso, é importante entender o custo de acelerar o desenvolvimento e não realizar alguma das etapas consideradas chaves na Engenharia de Software como: requisitos do sistema, modelagem, princípios e padrões de projeto, arquitetura, testes podem ter o custo alto a longo prazo para a equipe, seja com atrasos, implementações que podem ser refeitas, alterações de escopo e etc. Sendo assim, nosso objetivo aqui é mostrar que percorrer o passo a passo das técnicas e boas práticas que a área propõe irá atenuar problemas que ocasionalmente poderiam ocorrer caso o cenário real comum fosse usado.

Dado o exposto, nessa primeira parte do projeto iremos descrever os resultados alcançados até esse ponto de corte, descrevendo todo percurso realizado e eventuais problemas que foram amenizados devido ao prévio planejamento. Essa documentação está organizada da seguinte forma: na Seção 2 descreveremos trabalhos relacionados ao qual nos orientamos para realizar tal projeto. Na Seção 3, iremos detalhar todo o processo de desenvolvimento do aplicativo DCC News nesta primeira parte, iniciando pelos requisitos do sistema, modelagem, padrões de projeto utilizados, arquitetura, implementação de testes e automatização. Na Seção 4 iremos apontar os assuntos que serão abordados na segunda parte do trabalho. Por fim, na Seção 5 iremos concluir o projeto realizado nessa primeira parte.

2. Trabalhos relacionados

Nessa seção abordaremos todo o conteúdo didático utilizado como orientação para a elaboração desse projeto, seguindo o proposto.

Desenvolver um projeto prático requer um certo cuidado, pois nele não estamos expondo algum tipo de investigação ou alguma pesquisa mas sim aplicando algum tipo de conceito presente na área. Sendo assim, visamos aplicar a base teórica da Engenharia de Software sobre um produto final, traçando um roteiro do passo a passo para a elaboração de um software seguindo as boas práticas e técnicas da área.

Como citado anteriormente, a popularização dos meios tecnológicos faz crescer o desenvolvimento de novas aplicações de diversos segmentos no mercado, porém, esse crescimento impacta na qualidade dos produtos disponibilizados. É possível notar que em empresas de médio e pequeno porte, a preocupação maior é com o produto final em seu estado puro e funcional, mas note que funcional não significa que esse desenvolvimento está atrelado com as definições da Engenharia de Software.

Para esse projeto, nossa direção foi de garantir que todo o desenvolvimento esteja atrelado com as boas práticas e técnicas, seja pela arquitetura, padrões de projeto, testabilidade, manutenção entre outros conceitos empregados pela área. Para isso, uma das ideias seguidas nesse projeto foi debatido por [Mira Balaban 2018]. Os autores enfatizam a falta de prática acadêmica para lidar com problemas relacionados a Engenharia de Software, isto é, durante o curso não há possibilidade dos alunos vivenciarem problemas desafiadores e que de fato são rotineiros na indústria, pois a carga prática final é controlada apenas para exercitar o conteúdo visto.

Sendo assim, em paralelo a proposta apresentada por Balaban da criação de um **SE-lab**, procuramos nos guiar pelo conceito por trás do laboratório para desenvolver o projeto onde nesse caso, aplicando as técnicas e boas práticas não teríamos a disposição um ambiente controlado e sem grandes problemas já que estamos desenvolvendo algo

para produção, que será descrito na seção 3. Assim aplicamos todos os conceitos reforçando a carga teórica, entretanto, mantendo um ambiente similar ao real com desafios ao longo do desenvolvimento.

Também, para nos guiar em relação a todo conteúdo teórico da Engenharia de Software, utilizamos o livro **Engenharia de Software Moderna** do autor [Valente 2020]. Nesse livro, temos um passo a passo para a elaboração de um software seguindo os princípios da área. Baseado integralmente nesse livro, desenvolvemos a nossa aplicação a partir de passo descrito na ordem em que os capítulos foram expostos, vide 3. Com isso, todas as fases do desenvolvimento percorreram os conceitos exercitados no livro ao qual é amplamente utilizado pela comunidade acadêmica e uma referência no que tange aos desafios e práticas da Engenharia de Software no seu caráter mais moderno utilizado pela indústria atualmente.

Neste guia elaborado por [Valente 2020], encontramos o chamado **Padrões de Projeto**, no qual são conceitos que aplicamos ao projeto do software trazendo alguns benefícios, como economia de tempo e facilidade no entendimento do código. Logo, para o nosso projeto utilizamos o padrão chamado **Mediador** que também é descrito no livro de Padrões de Projeto, [Gamma Erich 1994], e que será detalhado na seção 3.

Também, está incluído no desenvolvimento a garantia de que o software funcione, isso se dá por meio de testes. Para essa versão inicial do aplicativo, como grande parte da estrutura é composta por mais iterações com o RSS do que lógica de negócio. Vladimir Khorikov, [Khorikov 2020], em seu livro diz que a estrutura dos testes em projetos com esse aspecto de "CRUD"tem um formato losangular ao invés da conhecida pirâmide de testes na qual os testes de unidade tem uma maior porcentagem em relação aos de integração e sistema. Para Vladimir, nesse tipo de projeto, testes de integração têm a maior porcentagem comparado aos outros, já que a interação com o banco de dados/API é muito maior que a própria lógica de negócio do código, tendo então esse aspecto losangular.

Com isso, evidenciamos diversas fontes da literatura que foram utilizados para a base do desenvolvimento desse projeto bem como quais são os pontos de interesse a serem explorados para evidenciar como os conceitos da literatura e a aplicação prática podem caminhar em conjunto, de modo que seja possível seguir o que direciona a Engenharia de Software sem que haja algum tipo de perda conceitual ao longo do desenvolvimento.

3. Desenvolvimento

Essa seção contempla a maior parte do projeto, aqui detalharemos cada passo seguido a fim de oferecer o estado atual do trabalho e da aplicação.

3.1. Requisitos dos Sistema

O primeiro passo para o desenvolvimento de um software é responder a seguinte questão:

O que o software desenvolvido deve fazer?

Em primeira instância, a resposta para essa pergunta pode parecer difícil, pois temos a ideia do que será o projeto, mas ainda não temos detalhes consistentes. Porém temos um ponto de partida, o conceito do software que está sendo desenvolvido. Para o DCC News, de início, já tinha sido definida a ideia do aplicativo, voltado para divulgação

das notícias relacionadas ao departamento, bem como informações úteis para a comunidade acadêmica.Sendo assim, o primeiro passo do nosso roteiro é debater quais requisitos o sistema deve ter.

Requisitos "definem o que um sistema deve fazer e sob quais restrições. Requisitos relacionados com a primeira parte dessa definição — o que um sistema deve fazer, ou seja, suas funcionalidades — são chamados de **Requisitos Funcionais**. Já os requisitos relacionados com a segunda parte — sob que restrições — são chamados de **Requisitos Não-Funcionais**."[Valente 2020] Para que isso seja definido, temos a ideia do "cliente", uma pessoa, organização ou empresa que irá utilizar o sistema. Essa entidade, entre todos que participam do projeto, é quem melhor consegue responder a pergunta no início desta seção.

Assim, durante a nossa primeira fase de elaboração do projeto, diversas reuniões foram realizadas para que a modelagem das funcionalidades sejam debatidas e acordadas com todos os participantes do projeto. Após isso, temos uma ideia melhor do que será o software desenvolvido.

"O aplicativo DCC News a ser desenvolvido tem como cerne da sua concepção um espaço onde os alunos consigam acompanhar as principais informações do Departamento de Ciência da Computação ao longo da sua graduação, isto é, acompanhar as notícias e atualizações do departamento, acompanhar o andamento de eventos interno ou de parceria com a unidade, se situar a cerca de dúvidas previamente já computadas, uma espécie de FAQs, saber quais oportunidades de iniciação científica com bolsas e voluntárias tem disponível e etc."

3.1.1. Histórias de Usuário

Após o entendimento do que será o projeto, podemos de fato discutir quais funcionalidades ele terá. Junto com a entidade, desenvolvemos o que é conhecido como **Histórias de usuário** na Engenharia de Software.

Antes de definir esse conceito é importante ressaltar que como estamos utilizando as técnicas e boas práticas da engenharia de software moderna, o modelo de desenvolvimento ágil **Scrum** [Schwaber 1997] está sendo utilizado, logo pelo formato de desenvolvimento estamos utilizando as histórias de usuário.

Histórias de usuários consiste em uma prática utilizada pelos métodos ágeis para catalogar de forma clara e objetiva juntamente com os clientes, quais funcionalidades esperariam ter no sistema.

As histórias devem respeitar algumas características:

- ser independentes
- aberta a negociações
- agregar valor ao produto
- serem objetivas
- ser testável

Respeitar essas características é um tanto quanto desafiador em um mundo real, principalmente independentes, pois em alguns casos geram um certo desconforto durante

o desenvolvimento do produto.

Para o nosso projeto foi complicado que algumas histórias de usuários não criem dependências, pois uni-las, pode acarretar em uma história extremamente densa, o que não é uma boa prática, sendo assim em alguns casos, unir três histórias que são dependentes, ocasionaria uma avaliação de **story points** maior do que o ideal, com isso quebrá-las em três nesse caso parece ser mais vantajoso.

Assim, uma história única: "*Como usuário gostaria de ter uma aba de notícias no aplicativo*", foi dividida em três, #8 #9 #10, como exemplificado na figura 3

Figura 1. Histórias de usuário

Então, quebrá-las em histórias menores objetiva ainda mais o trabalho e facilita os testes que agora serão isolados para cada especificidade.

Ao final, temos nosso backlog do produto com as histórias de usuário que representam os **requisitos funcionais** do sistema.

3.1.2. Casos de uso

As histórias de usuários como já citadas anteriormente devem ser objetivas ao que se propõe, entretanto é necessário uma maior granularidade para entender qual será o fluxo dentro do sistema, com isso temos os chamados Casos de Uso.

O Caso de Uso são documentos textuais de especificação de requisitos. Eles incluem uma descrição mais detalhada de qual será o fluxo de execução daquela história dentro do sistema 3.1.2.

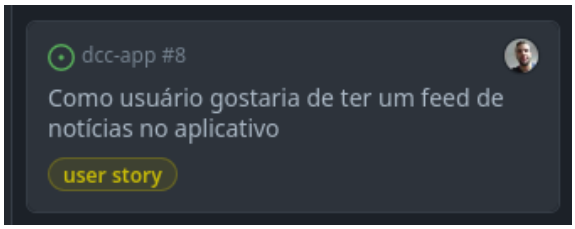

Figura 2. Caso de uso

Para essa história temos o seguinte caso de uso:

Feed de notícias
Ator: Aluno
Fluxo normal
1. Acessa o aplicativo
2. Acessa a aba de notícias
3. Desliza a tela para visualizar os cards com as notícias

Assim, conseguimos traçar qual será o fluxo de execução dessa funcionalidade ao ser implementada, com isso podemos previamente estruturar como será a lógica para desenvolver tal feature, logo iniciamos a elaboração do projeto com o primeiro passo.

3.1.3. As tecnologias utilizadas

Por fim, durante a análise de requisitos também é acordado com o time de desenvolvimento quais tecnologias serão utilizadas.

O modelo do banco de dados, relacional ou não, frameworks, linguagens de programação e todos os aspectos que irão impactar nos requisitos não funcionais descritos anteriormente são analisados cuidadosamente, pois nesse momento é possível entender as necessidades e "dores"que a entidade tem em relação ao que a aplicação irá suprir e combinar com isso, as melhores tecnologias que façam sentido dentro do contexto em que o software desenvolvido estará inserido.

Para o DCC News, como a aplicação é voltada para o contexto mobile, estamos utilizando como framework **react-native** juntamente com o banco de dados relacional offline **realm**, assim teremos um conjunto de tecnologias completa que irá atender tanto às necessidades do mundo Android quanto do mundo iOS.

3.2. Modelagem

Após a análise de requisito, tomamos o próximo passo direcionado pelo nosso roteiro guiado,[Valente 2020], a modelagem do sistema.

A princípio o sistema pode parecer difícil, complexo e cheio de nuances. A modelagem do sistema tenta atenuar isso. Nela, utilizamos técnicas visuais como o **Diagrama de Classes** para representar uma possível estrutura do nosso sistema.

Para o aplicativo, na análise de requisitos identificamos os pontos de funcionalidades que podem ser contemplados dentro da nossa modelagem, sendo assim, para um primeiro momento do projeto podemos definir a estrutura base que estará disposta no projeto de modo que as histórias do usuário planejadas para serem desenvolvidas ao longo das primeiras sprints estejam totalmente cobertas por essa modelagem.

3.2.1. Planejando o modelo do DCC News

Como é possível notar, seguindo o passo a passo do que afere as boas práticas e técnicas da Engenharia de Software, cada fase do projeto está vinculada a fase anterior, logo, com essa hierarquia de processos todo o desenvolvimento acaba sendo facilitado.

Para a primeira fase do projeto, o DCC News possui uma particularidade: o consumo dos dados é feito via RSS. Entretanto, em decorrência da análise de requisitos feita anteriormente, já temos conhecimento que além desse end-point teremos conexão com o servidor para realizar outros tipos de consultas e requisições, assim, nossa modelagem deve levar em consideração tais fatores para que haja o mínimo de "re-trabalho"possível. Aqui cabe enfatizar a diferença entre API e Servidor:

- **API**: consumimos os dados, como por exemplo o feed de notícias, através de um ponto de acesso em XML, provenientes do RSS. Nesse caso os dados são dispostos somente dessa fonte, não precisando ser estruturados.
- **Servidor**: outro ponto de consumo de dados será por meio do servidor, nesse caso, alguns dados serão armazenados no banco de dados local ou remoto e utilizaremos a comunicação com esse para requisitar os conteúdos necessários, assim podemos estruturá-los e apresentá-los aos usuários no sistema.

Podemos agora, a partir dessa estrutura conceitual, avaliar qual padrão de projeto pode ser relevante para o sistema.

3.2.2. Padrões de projeto & Mediator

Padrões de projeto descrevem objetos e classes que se relacionam para resolver um problema de projeto genérico em um contexto particular, ou seja, são conceitos que você pode aplicar ao seu projeto para resolver um determinado problema. [Gamma Erich 1994]

Quando há um conhecimento sobre tal conceito, o desenvolvedor pode se beneficiar implementando-o em seu projeto com uma solução já testada e validada ou pode ajudá-lo a entender o comportamento e a estrutura das classes que ele precisará usar.

Padrões de projeto são definidos em três classes: [Valente 2020]

- **Criacionais**: padrões que propõem soluções para criação de objetos.
- **Estruturais**: padrões que propõem soluções para composição de classes e objetos.
- **Comportamentais**: padrões que propõem soluções para interação e divisão de responsabilidades entre classes e objetos.

Como o DCC News terá suporte para dois tipos diferentes de fonte de consulta de dados, API e Servidor, e dentro de cada fonte teremos suas divisões para coleta específica do conteúdo, utilizamos no projeto o padrão de projeto chamado **Mediator**.

O Mediator tem como propósito reduzir as dependências entre os objetos, assim, ao invés deles requisitarem uns aos outros, um mediador fará esse contato entre os objetos reduzindo então a dependência entre ambos e centralizando as chamadas em um ponto específico do sistema.

Essa escolha tomou-se por conta das diversas chamadas que o software tem com as próprias requisições do tipo RSS, já que para cada tipo de dado, notícias, eventos, palestras, professores, etc, existe uma conexão diferente, além disso por conta da análise de requisito, quando as conexões com o servidor forem estabelecidas, haveríamos um fluxo muito grande de chamada para as consultas entre os métodos na qual seria necessário ter uma interface para cada uma dessas, criando um grande acoplamento.

Então, com o uso do mediador toda essa interface é abstraída, já que os métodos irão realizar apenas a chamada para a classe com o tipo de dado solicitado, ficando a cargo dele verificar qual o local que a requisição deve ser feita, sem que haja preocupação por parte dos métodos mais externos com esse tipo de conhecimento.

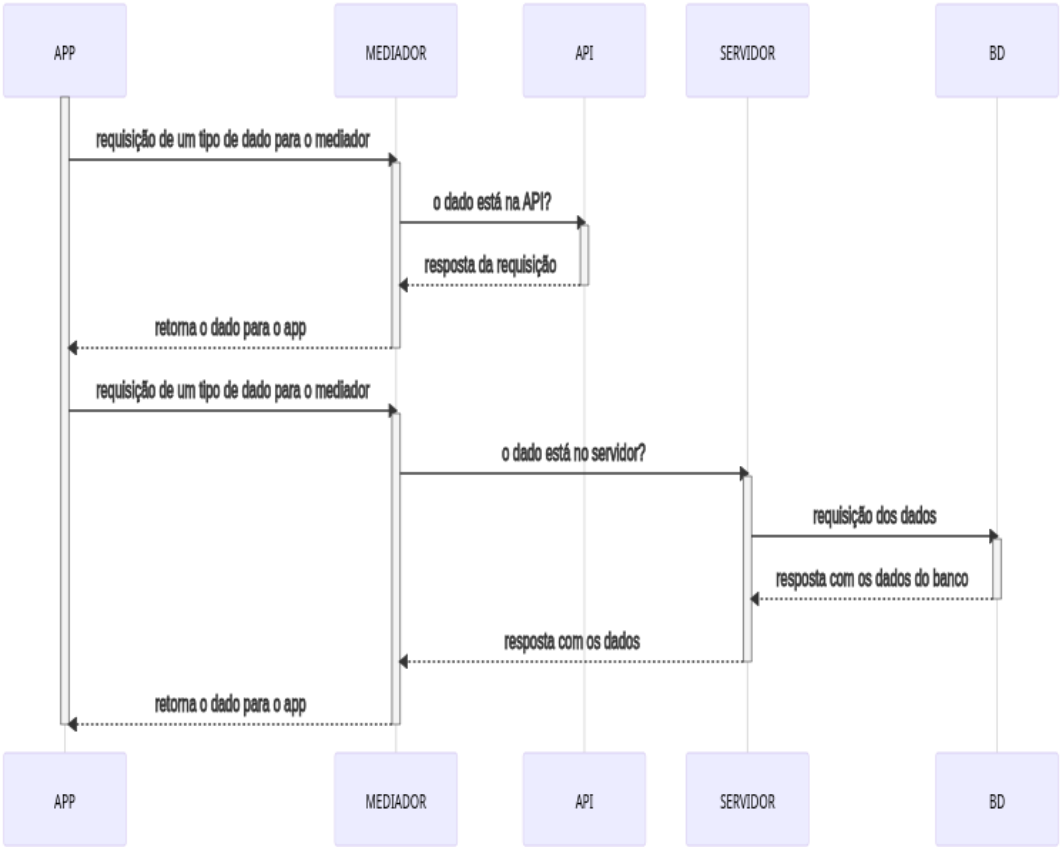

Figura 3. Modelagem do DCC News

3.3. Arquitetura

Recapitulando, já temos conhecimento sobre quais funcionalidades o aplicativo terá, também já sabemos como é a estrutura central com suas requisições e consumo de dados. Para completar esse ciclo precisamos definir a arquitetura utilizada pelo nosso projeto.

Arquitetura de software se preocupa com o projeto no mais alto nível. Ela inclui as decisões de projeto mais importantes em um sistema e que dificilmente serão revertidas

futuramente. Assim, mediante a nossa modelagem, o projeto utiliza a arquitetura do tipo **MVC**.

3.3.1. MVC (Model-View-Controller)

Esse padrão arquitetural foi proposto no final da década de 70. Ele define que o sistema deve ser organizado em três classes:

- **View**: a visão é responsável pela apresentação gráfica do sistema.
- **Controller**: o controlador é responsável por tratar os eventos gerados no sistema, por exemplo por dispositivos de entrada, teclado, mouse e assim, solicitar ao modelo ou a visão que o estado seja alterado.
- **Model**: o modelo é o responsável por armazenar os dados manipulados pela aplicação e que tem a ver com o domínio do sistema. Além disso, o modelo também não tem conhecimento ou dependência sobre a visão e o controlador.

É fácil evidenciar com todo o arcabouço citado a similaridade entre o nosso projeto e o modelo MVC. A evidência clara disso está em torno do Mediador, lembre-se que ele é responsável por fazer o intermédio entre o local que necessita dos dados e o que provê-los. Sendo assim, podemos traçar um paralelo entre ele e o Controller para definir que ambos realizam funções similares dentro do sistema.

Com a adoção desse padrão arquitetural, trazemos consigo alguns benefícios: podemos utilizar o mesmo modelo em diferentes visões, possibilitando criar especializações de trabalho no desenvolvimento, assim, para a manutenção do código podemos ter pessoas capazes de lidar somente com alguma parte do sistema, também, favorece a testabilidade do projeto. [10. 1999]

Com isso, concluímos o roteiro inicial para o desenvolvimento do projeto nos moldes da Engenharia de Software. A partir desse momento e de posse de todo esse arsenal, podemos enfim iniciar o desenvolvimento da aplicação.

3.4. Desenvolvimento

Diante de todo conhecimento disponível, a partir deste ponto iniciamos o desenvolvimento da aplicação de fato. Como já citamos anteriormente, o projeto está utilizando o desenvolvimento ágil Scrum [Schwaber 1997], logo o uso de sprints se faz necessário, juntamente com as histórias de usuários e casos de uso que já foram citados.

Nessa primeira parte, foram realizadas 3 sprints. Em cada uma delas, como segue as técnicas da Engenharia de Software, foram aplicados os conceitos referentes ao desenvolvimento de métodos ágeis, então, a cada sprint tivemos um número limitado de histórias do usuário para desenvolver, limite esse que não poderia ser maior que a quantidade de story points de cada tarefa, que nesse contexto não foi medido por essa métrica já que havia apenas uma pessoa desenvolvendo, onde no nosso contexto foi delimitado pela dificuldade das tarefas definidas em cada sprint.

3.4.1. Sprint 0 - Preparação do ambiente

É uma técnica do desenvolvimento ágil ter uma sprint que marca o ponto de início do desenvolvimento. Esse ponto é chamado de **Sprint 0**. Com uma duração menor que a convencional, ela tem o objetivo de preparar o ambiente para o desenvolvimento, instalação das tecnologias a serem utilizadas e montagem da estrutura do projeto, organização das pastas, definição de tipagens, documentação e quaisquer outros artefatos que sejam definidos como padrão para o projeto.

Logo após esse período acabar, geralmente com duração de 3 a 4 dias, temos início ao desenvolvimento.

3.4.2. Desafio das dependências

Dependências externas podem ser um grande para ao longo do desenvolvimento já que não temos o controle sobre elas.

É uma boa prática evitar o uso de tais funcionalidades, entretanto, em um cenário de produção, apesar do framework usado ser robusto, alguns detalhes específicos não são ofertados nativamente por ele, sendo assim, utilizar pacotes desenvolvimentos por outras pessoas é um tanto quanto comum nesse contexto de desenvolvimento de software. Entretanto, como não é algo implementado por você, ele estará sujeito a alterações abruptas, bugs, ou qualquer outro tipo de comportamento não controlado.

Como já descrevemos anteriormente, uma parte do consumo dos dados do DCC News parte de endpoints RSS, sendo assim, faz necessário ter algum método que converte os dados desse end point para uma forma visual ao qual o usuário poderá claramente visualizar as informações. Para realizar essa ação, foi necessário utilizar um parser dos dados XML ao qual o RSS dispõe para um formato visual acessível. Esse tipo de tratamento de dados era feito por métodos disponibilizados em um pacote, **react-native-rss-parser**, desenvolvimento pela comunidade, porém, o mesmo encontra-se descontinuado, assim seria extremamente perigoso e não atenderia as boas práticas da Engenharia de Software, disponibilizar um componente com alto poder de prejudicar a funcionalidade do sistema.

Para solucionar esse problema, criamos um fork desse componente e o atualizamos para que seja compatível com as versões mais recentes do nosso framework, a partir desse momento removemos esse controle de terceiros eliminando um obstáculo que poderia impactar no desempenho da aplicação.

Apesar de ser uma boa prática é altamente recomendável não permitir que o sistema tenha componentes ao qual você não consegue ter controle, pode ser em um cenário real, extremamente complexo manter certos tipos de pacote, seja por ter um código de difícil entendimento, legado, ou por dependerem de outros componentes que também estão descontinuados.

No projeto, como o parser do RSS é uma parte altamente importante do sistema como um todo, durante o desenvolvimento foi conveniente criar uma cópia para o seu controle, assim diminuindo preocupações que possam ser acarretadas ao longo de todo o desenvolvimento.

3.4.3. Testes e seus desafios

Software é uma construção complexa, assim é de entendimento que erros e inconsistências podem acontecer das mais variadas formas. Para isso, uma boa prática da Engenharia de Software é realizar bons testes para atenuar esses problemas.

Uma das principais orientações para uma boa suíte de testes é: *"elaborar testes que testam comportamento e não o método"*, pois assim, conseguimos validar o que está sendo desenvolvido. Para os testes, temos três áreas de abrangência: testes de unidade, testes de integração e testes de sistema. Proposto por Mike Cohn [Cohn 2009], a estrutura dos testes de um projeto é no formato piramidal, onde os testes de unidade aparecem na sua maior proporção, por serem mais rápidos, baratos e de fácil desenvolvimento.

Além de testar comportamento, os testes também devem seguir o princípio **FIRST**: serem rápidos, independentes, determinísticos, auto verificáveis, e escritos o quanto antes. Com esse conjunto de definições é possível criar uma boa suíte que seja capaz de validar as implementações e garantir que o comportamento do código esteja de acordo com o planejado.

Dado todo contexto, agora vamos de fato focar no projeto. Como citamos na seção 2, em seu livro, [Khorikov 2020] se refere que a estrutura de testes em projetos com aspecto CRUD possuem um formato losangular ao invés de piramidal [Cohn 2009]. A partir desse entendimento, iniciamos a modelagem da nossa suite de testes com o foco em testes de integração. Porém, também temos conhecimento que algumas dificuldades existem em testar os componentes criados no react-native. Isso acontece, pois uma simples estrutura utiliza diversos artifícios de implementação para funcionar corretamente, sendo assim, seria necessário utilizar técnicas de **teste double** para que os componentes sejam testados.

Testes de integração no âmbito do projeto, não envolvem chamadas com o banco de dados especificamente, já que as requisições em sua maioria nessa primeira fase são realizadas via endpoints RSS. Assim, seria necessário que além dos testes de integração para verificar essas requisições, também tivéssemos testes de renderização de componentes que para esse framework também podem ser vistos como da classe de integração.

Entretanto, como definido pelas boas práticas da Engenharia de Software, ao desenvolver uma tarefa, seu conjunto de testes também seria elaborado juntamente com elas, porém por conta do que foi descrito anteriormente, houve uma grande dificuldade de que eles sejam escritos de forma correta, já que os próprios componentes do framework possuem certas dificuldades, logo durante essa primeira parte do projeto, em alguns pontos específicos do projeto realizamos os chamados **testes manuais** que também são utilizados na área, porém em um grau menor.

Esse tipo de teste diferentemente dos automatizados, usados nas suítes, pode ser extremamente útil para ter visualizações que não estão em concordância com o definido nos requisitos, isto é, podemos validar se um componente foi visualmente definido de forma correta utilizando esse tipo de teste, já que criar testes automatizados para essa verificação não fazem sentido e são extremamente complexos. Note que em tarefas como por exemplo, verificar se o componente está com o texto alinhado, ou se as cores estão corretas, estilização de um modo geral, o uso dos testes manuais é fundamental para essa

validação, sendo assim grande parte visual das telas foi feita essa verificação para que pudesse de fato concluir a tarefa de acordo com a expectativa do cliente.

Dado o exposto, a dificuldade acerca dos testes foi algo que já estava previsto durante as fases de planejamento, logo, pudemos contornar o problema utilizando outras soluções sem que haja perda de algum quesito de boas práticas.

3.4.4. Teste Alfa

Após a descrição anterior sobre o desenvolvimento dos testes, partimos para a próxima etapa: **testes de aceitação**.

Testes de aceitação são aqueles que servem para verificar se o sistema que foi desenvolvido está em concordância com as necessidades dos clientes. Eles possuem duas características: são testes manuais feitos pelos clientes do produto e também envolve um processo de validação do sistema, logo, caso tudo esteja de acordo, o produto entra para produção.

Esses testes podem ser divididos em duas fases: a primeira é chamada de **teste alfa**, esses são realizados com um grupo restrito de usuários em um ambiente controlado e com acompanhamento dos desenvolvedores. Após a aprovação temos a segunda fase chamada de **testes beta**, nessa fase não temos mais um ambiente controlado, o grupo de usuários é maior e não há acompanhamento dos desenvolvedores. Diante disso e caso aprovado o software finalmente entra em produção.

Para o DCC News, também temos os testes de aceitação. Como proferido no cronograma desenvolvimento na fase inicial do projeto, ao final da terceira sprint teríamos a disponibilização de um MVP, mínimo produto viável [Valente 2020], que consiste em uma versão inicial do aplicativo com funções mínimas para validação dos usuários.

Assim, seguindo as técnicas da Engenharia de Software, estamos atualmente na primeira fase dos testes de aceitação, na qual disponibilizamos para o cliente e um pequeno grupo de usuários selecionados uma versão *preview* do aplicativo para que esses possam validar suas funções, fluxo de execução, interface, além de sugerir mudanças e melhorias na aplicação. Com isso desde o início garantimos que todo o sistema atenda o grau de expectativa tanto do cliente quanto da comunidade que irá utilizar.

3.4.5. Disponibilização na Play Store

Como citado acima, estamos na primeira fase dos testes de aceitação. Assim, para disponibilizar as versões chamadas de *preview* para a comunidade, a loja de aplicativos Play Store foi utilizada nesse primeiro momento. Com isso, todos os usuários que fazem parte do teste alfa podem reportar feedback do aplicativo diretamente pela loja, além das opções presentes na interface do sistema, assim logo que temos conhecimento podemos avaliar e ajustar o que for necessário.

Dado o cenário, é possível ter o ambiente controlado como a própria definição de testes alfa retrata, mantendo um contato direto com os usuários para constantemente melhorar a experiência de uso da aplicação.

3.4.6. Os benefícios das técnicas da Engenharia de Software

Como comentado anteriormente, o planejamento do projeto ocasionou em primeira instância o benefício de já termos conhecimento sobre os problemas dos testes, assim foi possível confrontar esse problema e solucioná-lo de forma rápida.

Outro ponto foi a questão dos problemas com dependências externas. Esse também foi outro ponto a ser levantado durante o planejamento e com essa ciência as medidas para solucioná-los estavam preparadas.

Temos acima dois exemplos práticos de que o uso das boas práticas e técnicas da Engenharia de Software pode favorecer o desenvolvimento do projeto, já que conseguimos contorná-los de maneira relativamente tranquila. Cabe também ressaltar que além dessas questões, o fato de termos definido a modelagem e arquitetura do sistema seguindo o que a área sugere, o desenvolvimento de cada história da sprint foi extremamente facilitada pois principalmente com a modelagem, já tínhamos conhecimento de como seria o fluxo de execução para desenvolver as tarefas, questão essa que não seria possível se não tivéssemos realizando um planejamento antes.

Além disso, por seguir as boas práticas de codificação, exemplo padrões de projeto, estender funcionalidades, como a de salvar notícias para ler depois, que já haviam sido implementadas, bem como refatorações de código foram mais simples, pois esses conceitos ajudam a melhorar legibilidade, manutenção e extensão do projeto.

Dado o exposto, podemos perceber que o uso de boas práticas e técnicas facilitou contornar problemas, manutenção, extensão, legibilidade e desenvolvimento das funcionalidades tendo um ganho de tempo de implementação considerável - em geral as sprints finalizaram em média 4 dias antes do prazo padrão de 15 dias - gerando produtividade no fluxo do projeto.

4. Trabalhos Futuros

Nessa primeira parte do projeto, conseguimos realizar todas as atividades que foram definidas no calendário, iniciando pelo planejamento, desenvolvimento e disponibilização da loja, agora vamos ressaltar aqui alguns pontos que serão explorados na segunda parte do projeto.

4.1. Disponibilização na Apple Store

Em geral as empresas que desenvolvem software já possuem ambientes preparados para o desenvolvimento da aplicação em questão, por exemplo caso seja uma fornecedora de aplicativo mobile, ambientes para Android e iOS já fazem parte do seu arsenal.

Como nesse projeto DCC News, quem está desenvolvendo não possui as mesmas condições em termos de ambiente que uma empresa, sendo assim, nessa primeira parte do projeto focamos em disponibilizar o MVP somente para usuários Android, por conta da facilidade de ambiente e baixo custo.

Já na segunda parte do projeto, iremos disponibilizar uma versão de mínimo produto viável para usuários do iOS validarem já que como é de conhecimento há alguns pormenores que devem ser ajustados relacionados ao ambiente final ao qual a aplicação vai ser disponibilizada.

4.2. Implementação do Servidor

Como citamos na seção 3.2.1, o aplicativo contará com o consumo de dados por API e pelo servidor.

Nesse primeiro momento do projeto, a parte relacionada ao servidor apesar de já estar estruturada não foi implementada de fato no projeto. Isso acontece pois durante o planejamento notou-se a necessidade de disponibilizar uma versão preview para validação dos usuários, logo implementar as requisições do servidor nessa primeira etapa não seria viável por conta do tempo, já que as três primeiras fases antes de iniciar o desenvolvimento tomaram muito tempo.

Sendo assim, para a segunda parte do projeto, iremos trabalhar juntamente com a cliente no desenvolvimento dessa plataforma ao qual será possível adicionar novas funcionalidades ao aplicativo.

4.3. Menu Oportunidades

Na Engenharia de Software usamos o termo "dores"para se referir a pontos que o que está sendo desenvolvido irá solucionar em relação a quem consumirá a aplicação.

Para o DCC News, uma das maiores dores que foi notado no planejamento é a que a comunidade acadêmica não consegue de forma fácil se cadastrar em oportunidades oferecidas pelo departamento, seja ela estágios, iniciações científicas, eventos, palestras entre outros.

Assim, uma das respostas para a pergunta mencionada na seção 3.1 foi a possibilidade do aplicativo atenuar esse problema crônico do departamento, logo para a segunda parte do projeto, um dos focos principais do planejamento é o desenvolvimento desta funcionalidade atrelado ao desenvolvimento do servidor citado anteriormente, pois a partir dele podemos construir essa e outras funcionalidades para o aplicativo.

4.4. Testes

Como vimos na seção 3.4.3, tivemos desafios significativos ao longo do desenvolvimento dessa primeira parte.

Para a segunda parte do projeto, pretendemos aprimorar a qualidade dos testes disponíveis no projeto. Além disso, também pretendemos utilizar avaliações métricas de cobertura para mensurar o quanto nosso sistema está coberto, dado nossa estrutura losangular.

Por fim, além de cobertura também é planejado utilizar técnicas como **snapshot testing**, na qual snapshots são gerados construindo a árvore html do componente e comparando-a com versões anteriores o que ajuda a identificar se houve a alteração do comportamento de algum outro módulo durante o desenvolvimento de uma tarefa.

4.5. Teste Beta

Quando as funcionalidades propostas para a segunda parte estiverem implementadas e testadas, partiremos para a segunda fase dos testes de aceitação: **testes beta**.

Com um número maior de usuários e já disponível em ambas as plataformas, podemos promover o aumento desses testadores do sistema e simular um ambiente o

mais próximo do real possível, assim será possível refinar todos os detalhes para que o aplicativo de fato seja entregue em produção aberta a toda comunidade.

4.6. Aplicativo em produção

Por fim, no período final da segunda parte do projeto, quando as funcionalidades já estiverem sendo desenvolvidas e testadas, quando os testes de aceitação se mostrarem positivos e já houver a validação por parte do cliente, podemos lançar o aplicativo a produção.

Na Engenharia de Software este é o último passo para o roteiro de boas práticas. Assim ao alcançar esse nível seguindo tudo o que foi exposto neste projeto na parte I e II, podemos evidenciar um roteiro que estará apto a servir de exemplo para quem gostaria de presenciar toda a experiência de desenvolver um software seguindo as boas práticas e técnicas da área, desde modelagem, arquitetura, requisitos, até as práticas de codificação, padrões de projeto, refatoração, testes e a disponibilização do produto final aos consumidores.

5. Conclusão

Durante toda essa documentação, expomos o passo a passo para que o projeto seja guiado seguindo as boas práticas e técnicas da Engenharia de Software no seu maior rigor possível.

Durante toda a exposição, alguns detalhes não foram mencionados como práticas de codificação, refatoração, uso de Getters e Setters, entre outras técnicas, isto porque, está intrínseco dentro do que foi exposto que as práticas sugeridas para escrever um código de qualidade - excluindo os padrões de projeto que foi mencionado nesse documento - foram seguidos.

Quaisquer outros detalhes que aqui não tenham sido abordados, pode-se levar em consideração que não coube no contexto do projeto, pois como é de conhecimento a Engenharia de Software possui uma gama vasta de conceitos que podem ou não ser aplicados durante os projetos a depender do escopo desse.

Dado todo o exposto, percebemos que utilizar as boas práticas e técnicas de Engenharia de Software é extremamente vantajoso para o desenvolvimento de um projeto, pois com todo esse roteiro traçado podemos contornar problemas que sem eles poderiam ser grandes complicadores para o bom andamento do mesmo.

Vimos também que apesar de seguir rigorosamente os conceitos, alguns pormenores ainda podem acontecer pois em cenários reais, temos um ambiente que não é controlado e assim casos expoentes surgem. Porém percebemos que com o devido planejamento e organização essas questões acabam sendo menos impactantes no projeto, logo avaliamos que para que um projeto seja bem desenvolvido, ter o conhecimento sobre todo conceito proposto pela área é fundamental para que seu produto se torne bem sucedido.

Referências

[10. 1999] (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., USA.

[Cohn 2009] Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 1st edition.

[Gamma Erich 1994] Gamma Erich, Helm Richard, J. R. V. J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-wesley professional edition.

[Khorikov 2020] Khorikov, V. (2020). *Unit Testing Principles, Practices, and Patterns*. Manning edition.

[Mira Balaban 2018] Mira Balaban, A. S. (2018). Software engineering lab – an essential component of a software engineering curriculum.

[Schwaber 1997] Schwaber, K. (1997). Scrum development process. In Sutherland, J., Casanave, C., Miller, J., Patel, P., and Hollowell, G., editors, *Business Object Design and Implementation*, pages 117–134, London. Springer London.

[Valente 2020] Valente, M. T. (2020). *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. Independente edition.