DESIGN E DOCUMENTAÇÃO ARQUITETURAL DE UM WEBAPP EDUCACIONAL COM ROADMAPS: UMA REPRESENTAÇÃO DOS MODELOS 4+1 E VISÕES E ALÉM

LÁZARO VINICIUS ALMEIDA DOS SANTOS[1]
CAMILA COSTA AMORIM[2]
WILSON VENDRAMEL[3]

RESUMO

À medida que a prática de estudar por conta própria se torna mais fluente, e que o desenvolvimento de software se torna mais amplo, passa a ser imprescindível que projetos de aplicação de software ofereçam estrutura consolidada para a execução das atividades de ensino-aprendizado. Implementando-se uma aplicação focada em arquitetura de software, a partir de seu design, a captura de requisitos arquiteturais por meio de FURPS+, casos de uso e memorandos técnicos preenchem esse entendimento, apoiados por testes arquiteturais. Este documento apresenta a abordagem de paradigmas de arquitetura de software no contexto específico do Líbero para um produto de software de ensino, unidos ao Modelo 4+1 e modelo de Visões e Além para a integração de conhecimento descrito na forma de mapas, trilhas de ensino chamadas *roadmaps* e seus assuntos relacionados.

Palavras-chave: Arquitetura de Software; Modelo 4+1; Roadmap; Visões e Além.

ABSTRACT

As the practice of self-study becomes more fluent, and software development becomes broader, it becomes essential that software application projects offer solid structures for the execution of process activities of the teaching-learning. Implementing an application focused on software architecture, from its design, the capture of architectural requirements through FURPS+, use cases and technical memos fulfill this understanding, supported by architectural tests. This document presents the approach of software architecture paradigms in the specific context of the Libero for a teaching software product, together with the 4+1 Model and Visions and Beyond model for the integration of knowledge described in the form of maps, teaching trails called roadmaps and their related subjects.

Keywords: 4+1 Model; Roadmap; Software Architecture; Views and Beyond.

[1]Graduando em Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologia da Zona Leste – FATEC Zona Leste. E-mail: lazaro.santos2@fatec.sp.gov.br
[2]Graduanda em Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologia da Zona Leste – FATEC Zona Leste. E-mail: camila.amorim@fatec.sp.gov.br
[3]Doutorando em Tecnologias da Inteligência e Design Digital pela PUC-SP. Mestre em Ciência da Computação pela UNICAMP. Professor da FATEC Zona Leste. E-mail: wilson.vendramel@fatec.sp.gov.br

INTRODUÇÃO

Ao se deparar com o cenário provocado a partir do ano de 2020 em que a sociedade foi imposta a congelar o fluxo de mercado e de educação, soube-se que pelo decorrer dessa jornada havia grandes mudanças, boas e ruins, como o desemprego e o surgimento de novas áreas de atuação (RIBEIRO; SOUSA; LIMA, 2020). A busca pelo conhecimento no ambiente cibernético torna-se uma corrida para atender as necessidades do novo normal, pois com o grande número de conteúdos, dificultou-se a idealização das metas e um bom planejamento do aprendizado e, por isso, em consonância com os trabalhos relacionados às visões arquiteturais de Avgeriou, Guelfi e Razavi (2004), a prática da arquitetura de software com abordagens industriais de Hofmeister et al. (2007), contendo a utilização de linguagens de modelagem como descrito por Prentović e Budimac (2013), estabeleceu-se como objetivo a criação do Líbero, uma aplicação de software que organiza roadmaps e assuntos.

O desenvolvimento do Líbero apoia-se em abordagens de arquitetura de software porque durante o ciclo de vida do projeto utiliza-se o Modelo 4+1 de Kruchten (1995) e Visões e Além de Clements et al. (2011), que ajuda a identificar e demonstrar quais são os atributos de qualidade e as suas melhorias, sejam estes componentes executáveis de software ou representações gráficas, findando-se na decisão mais cabível para o cenário estudado.

Para a análise do sistema utilizam-se diversos diagramas em uma linguagem de modelagem específica, muitas vezes representando mais de uma visão arquitetural, traduzindo a ideia do design dos autores de forma técnica sobre como os componentes se comportam no evento em questão e uma amostra da interface de usuário, inspirados pelo livro-texto para o ensino de projeto de arquitetura de software de Barbosa (2009), que por sua vez levantara questões sobre como um projeto com essa estrutura pode servir também de ferramenta de comunicação e garantir a integridade conceitual aos interessados.

MATERIAL E MÉTODOS

Este trabalho é de abordagem qualitativa, dado que o estudo se apoia em conceitos, modelos e técnicas de arquitetura de software. O objetivo é exploratório porque apresenta um exemplo de uso de modelos de design e documentação arquitetural para uma nova aplicação de software. O procedimento de estudo é bibliográfico dado que a teoria foi extraída de livros e artigos. Além disto, se destaca aqui a natureza empírica da pesquisa tendo em vista que esta se baseou em evidências e observações oriundas das próprias experiências de desenvolvimento do WebApp Líbero (WAZLAWICK, 2014).

O nome “Líbero”, vem do italiano “livre”, mas também utilizado para uma posição no jogo de futebol, onde a função é auxiliar o time na armação das jogadas. Levando esse termo para a esfera do projeto, o Líbero tem como função auxiliar o usuário por meio de roadmaps (trilhas a serem percorridas, mapas mentais), compostos de assuntos (tópicos, áreas de conhecimento), na sua jornada de aprendizagem, com carreiras oferecidas de acordo com o resultado de um teste vocacional, permitindo ao usuário criar “nós” de conteúdo compartilhado de variados tipos; links, vídeos, imagens, textos descritivos etc. e separados e categorizados por temas (sempre chamados no escopo do projeto de assuntos). Essa ferramenta pode ser usada de forma sequencial ou aleatória, dividida em três etapas principais: 1) efetuar um teste vocacional; 2) inserir e/ou consultar temas de ensino; 3) gerar tendências a partir disso.

Figura 1. Roadmap do Líbero – Interface Gráfica de Usuário.

Fonte: Os autores, (2022).

Um roadmap é um método visual e descritivo, similar a um mapa mental ou uma linha do tempo, que auxilia no planejamento dos estudos, gerenciando cada etapa de evolução do que está sendo aprendido ou qual projeto de estudos lógico virá na sequência. No campo educacional esse guia fornecerá aos mentores e alunos uma sequência de passos lógicos rumo ao objetivo desejado, deixando sempre os envolvidos situados sobre a progressão do conhecimento, seu registro e as atividades realizadas.

Buscou-se alicerçar o desenvolvimento do WebApp Líbero a partir do Modelo de Visões e Além de Clements et al. (2011), do Modelo 4+1 de Kruchten (1995) e documentação arquitetural de Larman (2004), que introduz cinco visões de abordagem de documentação de software; visão lógica, visão de implementação, visão de processos, visão de implantação e a visão de casos de uso (visão “mais um”). Os autores ainda garantem que a abordagem de Visões e Além não englobam somente o RUP (Rational Unified Process) como também o modelo ágil e ainda perspectivas arquiteturais, definidas por Rozanski e Woods (2005 apud CLEMENTS

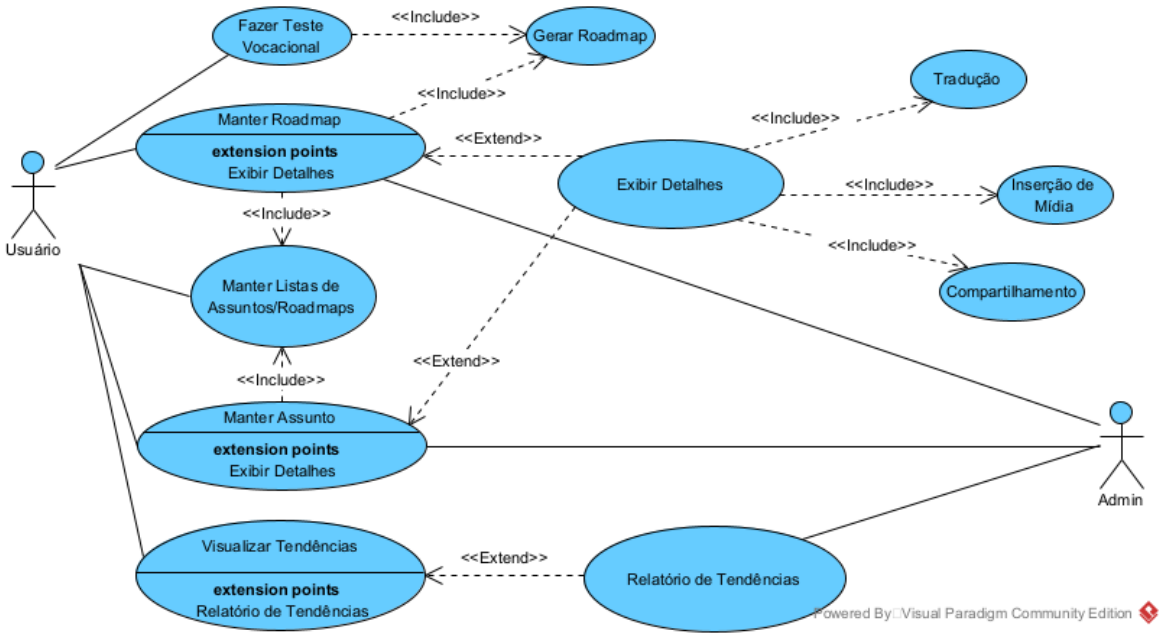
et al., 2011, p. 410) como sendo coleções de atividades, táticas e diretrizes que são usadas para garantir que o sistema exiba um determinado conjunto de propriedades de qualidade relacionadas, que requerem consideração em uma série de visões arquiteturais. Tendo isso em mente chegou-se na captura de requisitos arquiteturais; o que o WebApp deve cumprir de funcionalidade a partir de sua relevância sob fatores de qualidade.

Capturar requisitos arquiteturais não é uma tarefa trivial, e Pressman (2011) nos auxilia com uma lista de verificação que não oferece necessariamente uma medição direta, mas que também foi abordada, sistematicamente, pelo modelo de fatores de qualidade FURPS+ de Eeles (2005), que os categoriza por funcionalidade, usabilidade, recuperabilidade, portabilidade, suportabilidade e o *plus* (+) de design; sob a lente de memorandos técnicos, propostos por Larman (2004), identificou-se o que há de mais importante no contexto estrutural dos diferenciais da aplicação, quais são os fatores que influenciaram as motivações e quais são as soluções e alternativas propostas.

Casos de Uso e Memorando Técnico

Casos de uso como os representados pela Figura 2 representam comportamento, e comportamento é parte importante no entendimento holístico de uma documentação. Eles ajudam a mapear os requisitos da aplicação, que nesse caso foram resumidos ao essencial, mas que exprimem bem a ideia relacionada.

Figura 2. Diagrama de Casos de Uso.



Fonte: Os autores, (2022).

Tais conceitos se concretizam em uma adaptação de um memorando técnico (LARMAN, 2004), dispostos em um quadro para justificar a sua importância, a sua relevância arquitetural e todas as outras características de bons atributos de qualidade que foram capturados.

Quadro 1. Memorando Técnico (Design e Suportabilidade/Escalabilidade).

Fator: Serviços de roadmaps e assuntos customizados escaláveis.
Motivação: É de grande importância que os roadmaps, assuntos que compõem roadmaps e as listas dessas representações gráficas possam conter ou não, de antemão, um conteúdo previamente especificado e preenchido, de diversos tipos, e que esse conteúdo possa ser futuramente editado, incluindo alterações nos próprios objetos a nível de sistema.
Solução: Implementação do padrão de projetos de propósito de criação Builder no desenvolvimento dos componentes assunto e roadmap.
Alternativa: Uso do padrão de projetos de propósito estrutural Composite no desenvolvimento dos componentes assunto e roadmap.

Fonte: Adaptado de Larman, 2004 e Bacelli, Cardoso e Vendramel, (2021).

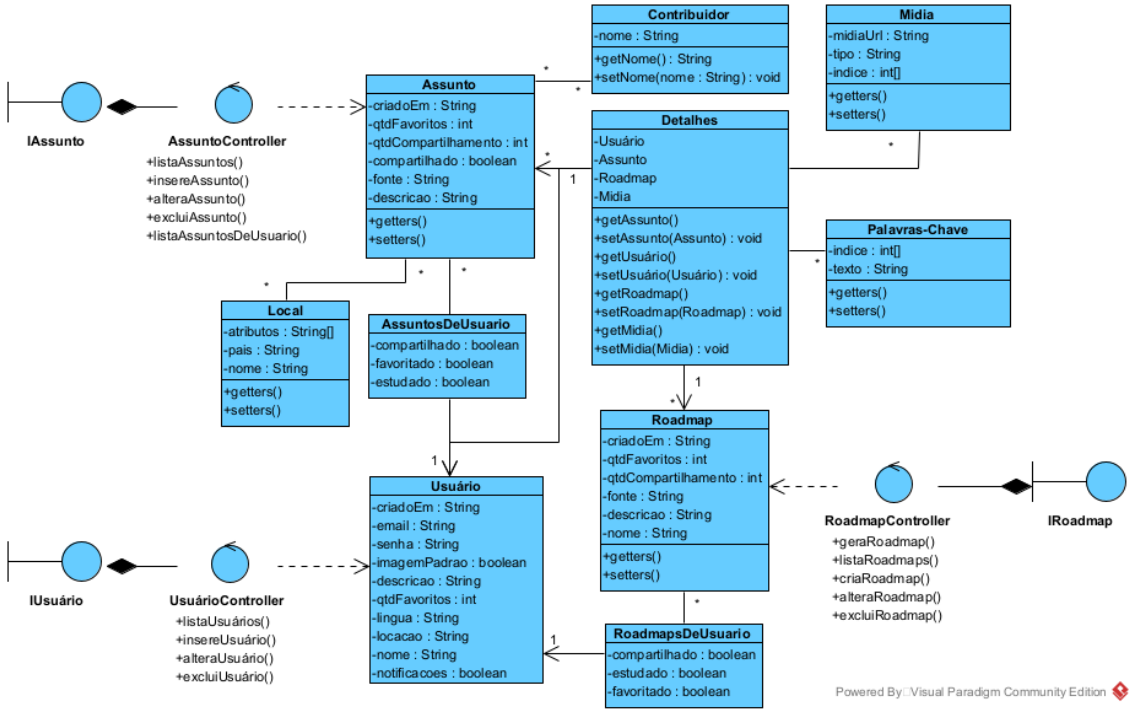
A identificação dos requisitos e das características significantes com o suporte do FURPS+ no memorando técnico (suportabilidade/escalabilidade) facilitam a validação entre decisões e requisitos arquiteturais, localizam-se de forma mais clara numa documentação e a endereçam na arquitetura. É citado o uso de padrões de projetos como solução e alternativa, mas não são o foco aqui.

O elo crítico entre o projeto e a engenharia de requisitos

Como descrito por Sommerville (2011), a arquitetura nada mais é do que “o elo crítico entre o projeto e a engenharia de requisitos.” Nesta seção abordaram-se padrões de projeto, estilos arquiteturais e como o autor os apoiam, tais como suas conexões e decomposição de componentes em subcomponentes.

Na análise de casos de uso e memorando técnico do WebApp foi identificado um “conjunto de objetos manipulados à medida que um ator interage com o sistema” (PRESSMAN, 2011, p. 142), que são as classes existentes, traduzidas em um diagrama de classes para melhor entendimento dos relacionamentos entre os objetos do projeto. Os objetos fundamentais identificados na abstração, presentes na Figura 3, diagramados em classes UML (Unified Modeling Language), apresentam tanto as trilhas de ensino, os assuntos similares e seus registros, quanto os objetos considerados menos relevantes.

Figura 3. Diagrama de Classes – Representação da estrutura do WebApp.



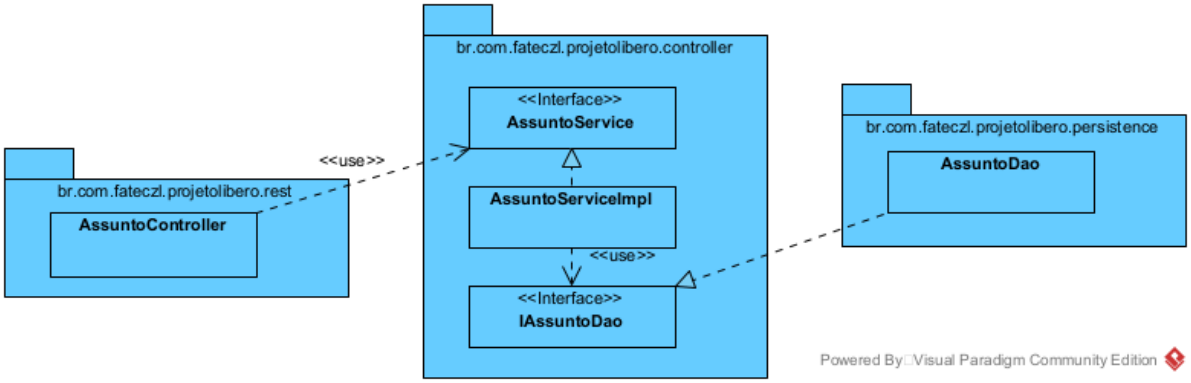
Fonte: Os autores, (2022).

Naturalmente os assuntos são ligados a um determinado conteúdo ou disciplina, e no Líbero não seria diferente. Ele vai se comportar como item colecionável em uma trilha de aprendizado roadmap. Observe que as classes Assunto e Roadmap possuem muitos atributos e comportamentos similares, pois esses são coleções de informações identificadas dos usuários, sendo o assunto um item filho de um roadmap pai, podendo herdar uma gama de características do módulo superior, superclasse, e em algum momento do desenvolvimento e da tomada de decisão foi considerada a técnica da generalização e o uso de herança para a implementação dessas funcionalidades. Em suma essa abordagem facilitaria o reaproveitamento de código, mas em vez disso, foi favorecida a composição e o uso de interfaces puras. Pôde-se então usar interfaces com composição, substituindo os benefícios de herança sem correr o risco da quebra de

encapsulamento e alto acoplamento desnecessário, ou seja, além de configurar uma ótica sob os objetos e componentes, uma ótica sob os módulos do sistema.

A camada de API é descrita pela Figura 4 (que representa sutilmente a visão de implementação do RUP, baseadas no Modelo 4+1, amarradas à visão de módulos de Visões e Além).

Figura 4. Diagrama de Pacotes por Componente Assunto – Visão de Módulos.



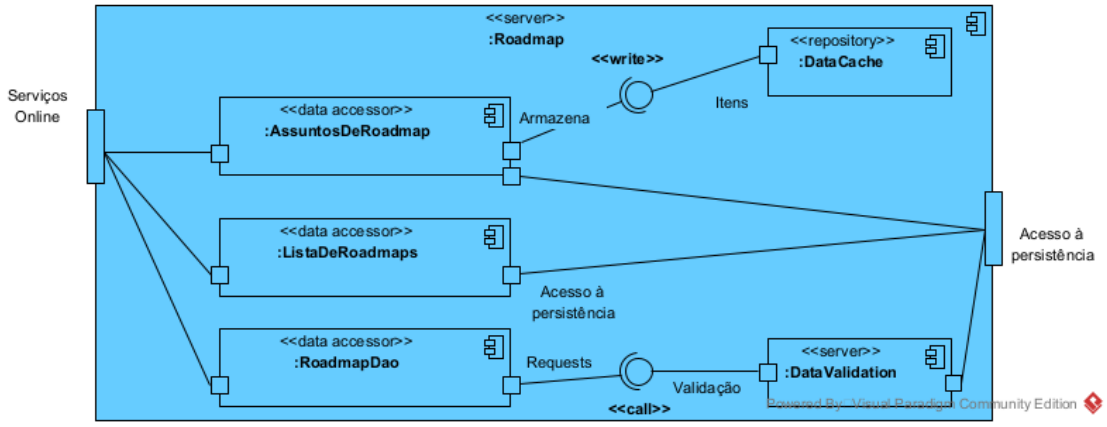
Fonte: Os autores, (2022).

Embora o Líbero seja um WebApp de escopo pequeno e o foco geral do projeto não fora desenvolver uma infraestrutura específica, foi implementada a arquitetura N-tier, divulgada especialmente pela Microsoft (2022), que vai dividir a aplicação em camadas lógicas e camadas físicas. O texto ainda aconselha que o arquiteto considere a arquitetura N-tier para o desenvolvimento de aplicações web simples, migrando um aplicativo local para um serviço específico com refatoração mínima e/ou o desenvolvimento unificado de aplicativos locais e em nuvem. Esse artigo exprime características similares da abordagem de design de arquiteturas em camadas horizontais, chamado pacote por camada, onde é feita a separação do código com base na sua função a partir de uma perspectiva mais técnica (MARTIN, 2020). Como o intuito do projeto é também estabelecer uma estrutura concreta e moderna, levando-se em conta o que o mercado tende a usufruir, as implementações

de assunto, por exemplo, similares aos roadmaps, foram apoiadas nessa arquitetura buscando sempre maior suportabilidade e customização.

A Figura 5 se ambienta no contexto das comunicações principais da aplicação com uma visão de componente e conector (C&C), que pode representar tanto a visão lógica quanto a visão de processos do Modelo 4+1, por capturar aspectos do sistema em tempo de execução (CLEMENTS et al., 2011). Aqui o usuário utiliza do serviço restful de criação de roadmaps, acessa as informações contidas em uma instância do banco de dados ou repositório já existente e até mesmo faz operações, criando, editando, excluindo e adicionando assuntos.

Figura 5. Diagrama de Componentes – Visão de Componente e Conector.

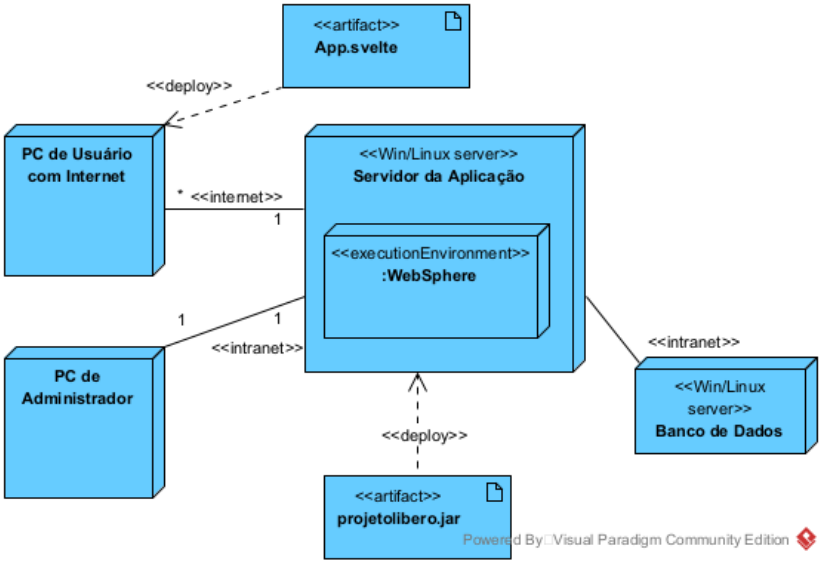


Fonte: Os autores, (2022).

Os serviços apresentados foram desenvolvidos no padrão arquitetural MVC (Model-View-Controller) que define que as classes do sistema devem ser organizadas em três grupos (modelo, visão e controle) e uma camada de serviço implementa conjuntos de fachadas finas sobre o grupo de modelo em questão (FOWLER, 2006). O grupo de controle do MVC é representado no diagrama de pacotes da Figura 4 e o grupo de modelo é representado pelo diagrama de componentes da Figura 5, já o grupo de visão pode ser melhor representado pelo diagrama de implantação da Figura 6, onde o uso de um framework específico, o

compilador de front-end Svelte, fica responsável pela apresentação da interface gráfica do usuário a partir de seu computador. Fowler (2006) associa que dessa forma se economiza esforço de desenvolvimento e tempo de resposta durante a execução, sem sacrificar a motivação primordial de garantir escalabilidade ao projeto.

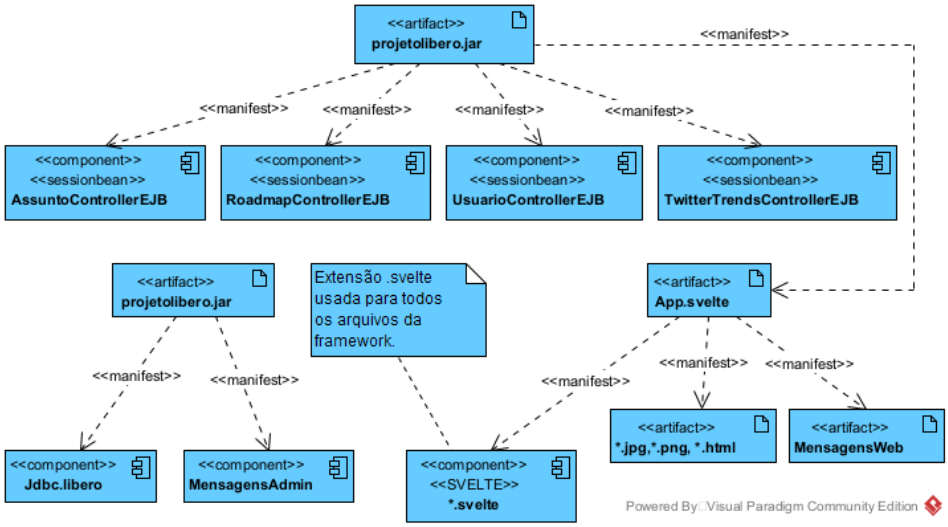
Figura 6. Diagrama de Implantação – Visão de Alocação.



Fonte: Os autores, (2022).

A visão de alocação de Visões e Além mapeia os elementos entre arquitetura de software e hardware que podem ser analisados pela performance de sistema, sendo útil para a disponibilidade, confiabilidade e segurança do projeto (CLEMENTS et al., 2011). Ainda, o diagrama de implantação mostra os nós de ambiente do sistema e por quais artefatos eles são implementados, através de suas dependências. A extensão de arquivos do Java EE (.jar) depende de um ambiente que execute tipos específicos de componentes, de maneira similar a extensão de arquivos do framework (.svelte), onde são contidos a maioria dos elementos estáticos do WebApp também tem o seu próprio contexto de execução.

Figura 7. Diagrama de Componentes – Visão de Alocação.



Fonte: Os autores, (2022).

Por sua vez a Figura 7 complementa o entendimento do diagrama de implantação e a visão física do Modelo 4+1, ajudando a organizar os arquivos a fim de manter controle maior sobre eles, que no caso de um projeto de grande porte se faz fundamental. A integridade do sistema durante os processos de construção e empacotamento do software é mantida mais facilmente e ajuda os desenvolvedores a localizar e manipular os arquivos quando necessário (CLEMENTS et al., 2011). O diagrama de implantação pode oferecer também diferentes custos de escopo, quando por exemplo um componente específico de tendências educacionais de uma grande rede social fez parte de uma versão da história dos ramos/*branches* do Líbero, mas que foi descontinuado.

RESULTADOS E DISCUSSÃO

Os resultados do estudo se mostraram claros de acordo com as figuras (principalmente a Figura 1) provenientes do desenvolvimento do design como um todo, do arquivo de aplicação gerado até seu pacote de documentação. A aplicação

Líbero foi testada através das perspectivas apresentadas, debruçando-se sobre as literaturas consultadas e melhores práticas de programação, resultando eventualmente em bons artefatos de software, um projeto de design cuidadosamente abstraído e uma documentação arquitetural ainda mais pragmática, atingindo os objetivos necessários de se construir um roadmap com assuntos relacionados. Os resultados dos testes efetuados em módulos e componentes são produtos de roadmaps e assuntos satisfatórios, devolvidos de forma coerente e eficiente. Embora em alguns momentos do texto surgissem elementos estranhos, dos quais não foram possíveis de implementação uniforme, os módulos e responsabilidades essenciais foram efetivados e testados em plataformas como o Eclipse IDE e o Postman para automação de requisições. Salienta-se que, em consonância com os trabalhos relacionados às visões arquiteturais, modelos arquiteturais e UML, a arquitetura de software garante a qualidade de um projeto e de seu pacote de documentação através de todas as etapas propostas por seus respectivos autores, desde a captura de requisitos de Eeles (2005), classificação desses requisitos em memorandos técnicos de Larman (2004), design (projeto) arquitetural com a inspiração em Barbosa (2009), implementação desses métodos com Martin (2020), chegando à documentação arquitetural instaurada nos Modelos 4+1 de Kruchten (1995) e finalmente Visões e Além de Clements et al. (2011), muito se é produzido e com seu devido valor.

CONCLUSÃO

O problema de pesquisa foi totalmente solucionado mesmo quando impostas limitações de recurso de hardware e tempo necessários para o desenvolvimento de uma aplicação de tal importância. Alguns serviços externos para o provimento de informações fundamentais no produto do projeto foram descartados ao longo do desenvolvimento por limitações de acesso e/ou falta de disponibilidade. De forma tecnológica, espera-se que a estrutura possa estar bem elaborada e que as ferramentas abordadas no WebApp possam ser intuitivas a ponto de que um usuário

iniciante consiga progredir em seus estudos de forma totalmente autônoma. O objetivo de estruturar bem um produto de software realizou-se com sucesso. Estudos futuros envolvendo metodologias ativas de ensino-aprendizado e o uso de ferramentas tecnológicas, em especial aplicações focadas em arquitetura de software, poderiam ser feitos.

Metodologias da engenharia de software principalmente sob uma análise profunda de arquitetura, com literaturas tão enriquecedoras quanto proveitosas apoiaram esse estudo. De fato, entendimentos densos como esses têm seu tempo de decantação no consciente, talvez compensatoriamente proporcionais ao intelecto, não sendo diferente nesse caso – a curiosidade por linguagens diversas se encontra nesse estudo com muitas outras linguagens necessárias para o desenvolvimento do design etc. que levaram tempo, dedicação, esforço, mas que se fizeram felizes.

REFERÊNCIAS BIBLIOGRÁFICAS

N-tier architecture style. MICROSOFT, 2022. Disponível em: <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier>. Acesso em: 12 de set. de 2022.

AVGERIOU, P.; GUELFI, N.; RAZAVI, R. **Patterns for documenting software architectures.** (Luxemburgo, Luxemburgo): Proceedings of the 9th European Pattern Languages of Programming (EuroPLOP) conference. 2004.

BARBOSA, G. M. G. **Um livro-texto para o ensino de projeto de arquitetura de software.** (Campina Grande, Paraíba, Brasil): Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2009.

CARDOSO, B. F.; BACELLI, E. W. P.; VENDRAMEL, W.; FREITAS, L. C. (org) **Coleção Desafios das Engenharias:** Engenharia de Computação 3. (Ponta Grossa, Paraná, Brasil): Atena Editora, 2021.

CLEMENTS, P. et al. **Documenting Software Architectures:** Views and Beyond. 2. ed. (Westford, Massachusetts, Estados Unidos): Pearson Education, Inc., 2011.

EELES, P. **Capturing Architectural Requirements.** IBM, 2005. Disponível em: <http://www.ibm.com/developerworks/rational/library/4706.html>. Acesso em: 26 de ago. de 2021.

FOWLER, M. **UML Distilled:** a brief guide to the standard object modeling language. (Boston: Addison-Wesley): 2003.

FOWLER, M. **Padrões de Arquitetura de Aplicações Corporativas.** (Porto Alegre, Rio Grande do Sul, Brasil): Bookman, 2006.

HOFMEISTER, Christine et al. **A general model of software architecture design derived from five industrial approaches.** Journal of Systems and Software, 2007. Disponível em: < https://www.sciencedirect.com/science/article/abs/pii/S0164121206001634>. Acesso em: 12 de nov. de 2022.

KRUCHTEN, P. **Architectural Blueprints—**The "4+1" View Model of Software Architecture. (Los Alamitos, California, Estados Unidos): IEEE Software, 1995.

LARMAN, C. **Applying UML and Patterns:** An Introduction to Object-Oriented Analysis and Design and
Iterative Development. 3. ed. (Westford, Massachusetts, Estados Unidos): Pearson Education, Inc., 2004.

MARTIN, R. C. **Design principles and design patterns.** 2000. Disponível em: <http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf>.

MARTIN, R. C. **Arquitetura Limpa:** O Guia do Artesão para Estrutura e Design de Software. (Rio de Janeiro, Rio de Janeiro, Brasil): Alta Books, 2020.

PRENTOVIĆ, F.; BUDIMAC, Z. **Documenting Software Architectures using UML.** (Novi Sad, Sérvia): Second Workshop on Software Quality Analysis, Monitoring, Improvement and Applications SQAMIA 2013, 2013.

PRESSMAN, R. S. **Engenharia de software:** uma
abordagem profissional. 7. ed. (Porto Alegre, Rio Grande do Sul, Brasil): AMGH, 2011.

RIBEIRO, M. S. S.; SOUSA, C. M. M.; LIMA, E. S. **Educação em tempos de pandemia:** registros polissêmicos do visível e invisível. 1. ed. (Petrolina, Pernambuco, Brasil): UNIVASF, 2020.

SOMMERVILLE, I. **Engenharia de Software.** 9. ed. (São Paulo, São Paulo, Brasil): Pearson Education do Brasil, 2011.

WAZLAWICK, R. **Metodologia de Pesquisa para Ciência da Computação.** 2.ed. (Rio de Janeiro, Rio de Janeiro, Brasil): Elsevier, 2014.