INSTITUTO DE COMPUTAÇÃO – ICOMP

UNIVERSIDADE FEDERAL DO AMAZONAS – UFAM

PROGRAMA PÓS-GRADUAÇÃO EM INFORMÁTICA – PPGI

Documentação de Arquitetura de Software em Contextos Ágeis de Desenvolvimento

Leonardo Augusto Picanço Barreto

Manaus – AM

2024

Leonardo Augusto Picanço Barreto

Documentação de Arquitetura de Software em Contextos Ágeis de Desenvolvimento

Dissertação de mestrado submetida à avaliação, para a obtenção do título de Mestre em Informática no Programa de Pós-Graduação em Informática, Instituto de Computação.

Orientador(a)

Tayana Uchoa Conte, Dr.

Co-orientador(a)

Anna Beatriz Marques, Dr.

Instituto de Computação – IComp

Universidade Federal do Amazonas – UFAM

Manaus – AM

2024

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Barreto, Leonardo Augusto Picanço
B273d Documentação de Arquitetura de Software em Contextos Ágeis de Desenvolvimento / Leonardo Augusto Picanço Barreto . 2024
176 f.: il. color; 31 cm.

Orientadora: Tayana Uchoa Conte
Coorientadora: Anna Beatriz Marques
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Desenvolvimento ágil de software. 2. Documentação de software. 3. Arquitetura de software. 4. Arquitetura ágil. I. Conte, Tayana Uchoa. II. Universidade Federal do Amazonas III. Título

Ministério da Educação
Universidade Federal do Amazonas
Coordenação do Programa de Pós-Graduação em Informática

FOLHA DE APROVAÇÃO

"DOCUMENTAÇÃO DE ARQUITETURA DE SOFTWARE EM CONTEXTOS ÁGEIS DE DESENVOLVIMENTO"

LEONARDO AUGUSTO PICANÇO BARRETO

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Profa. Dra. Tayana Uchoa Conte - PRESIDENTE

Profa. Dra. Patricia Gomes Fernandes Matsubara - MEMBRO EXTERNO

Prof. Dr. Ivan do Carmo Machado - MEMBRO EXTERNO

Manaus, 14 de março de 2024.

Documento assinado eletronicamente por **Tayana Uchoa Conte**, **Professor do Magistério Superior**, em 18/04/2024, às 20:57, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Patrícia Gomes Fernandes Matsubara**, **Usuário Externo**, em 24/04/2024, às 08:34, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Ivan do Carmo Machado**, **Usuário Externo**, em 02/05/2024, às 16:28, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Maria do Perpétuo Socorro Vasconcelos Palheta**, **Secretária em exercício**, em 08/07/2024, às 14:25, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufam.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1958977** e o código CRC **6679841D**.

Avenida General Rodrigo Octávio, 6200 - Bairro Coroado I Campus Universitário Senador Arthur Virgílio Filho, Setor Norte - Telefone: (92) 3305-1181 / Ramal 1193
CEP 69080-900, Manaus/AM, coordenadorppgi@icomp.ufam.edu.br

Referência: Processo nº 23105.011175/2024-55 SEI nº 1958977

AGRADECIMENTOS

Agradeço, em primeiro lugar, a Deus pela minha saúde e pela minha vida.

À minha família, na figura da minha noiva, Vilany, dos meus pais, Elizandra e Everaldo e dos meus sogros, Genilce e Disney, que me dão apoio para trabalhar na minha pesquisa. Sem o suporte e o incentivo deles, dificilmente eu entregaria os resultados aqui descritos.

À minha orientadora, prof. Tayana Conte, que me deu a oportunidade de fazer pesquisa na minha área de interesse. Agradeço por ter acreditado em mim e de confiar na minha capacidade de realizar um bom trabalho.

Às professoras Ana Oran e Anna Beatriz, pela contribuição e pelo apoio durante a realização do estudo em ciclo de vida e do experimento controlado. Os seus conselhos e comentários foram fundamentais para a execução dos estudos e a escrita dos resultados.

Aos membros do grupo de pesquisa USES que me acolheram e ajudaram a caminhar nesta estrada. Agradeço, em especial, às colegas Marcela Pessoa, Nayane Maia e Marcia Sampaio, que me estenderam a mão no momento mais difícil da minha pesquisa. Serei sempre grato a vocês.

Por fim, agradeço aos membros da banca examinadora, Prof. Patricia

Matsubara, Prof. Ivan Machado, Prof. Bruno Gadelha e Felipe Curty por aceitarem o convite em participar desta defesa de dissertação.

Resumo

O contexto de desenvolvimento de software atual contém alta competitividade e impõe às empresas a necessidade crescente de entregar valor rapidamente e com custo baixo. Nesse caso, os recursos são alocados no desenvolvimento acelerado em detrimento de atividades de planejamento e de documentação de arquitetura de software. Dessa forma, informações relevantes sobre o sistema como interações do usuário, infraestrutura e plataforma de desenvolvimento, por exemplo, não são definidas ou documentadas da melhor forma, levando à maior complexidade de desenvolvimento e manutenção dos sistemas. Nesse contexto, a questão de pesquisa deste trabalho é **quais abordagens de documentação da arquitetura de sistemas são viáveis em contextos ágeis de desenvolvimento de software e como utilizá-las?** Para respondê-la, uma revisão da literatura e uma *Feature Analysis* foram realizadas, para avaliar as abordagens 4+1, S4V, ADD, BAPO/CAFCR, C3A e C4. A análise indicou as abordagens 4+1 e C4 com potencial de aplicação no contexto estudado e, para validá-las, três estudos experimentais foram realizados. Foi possível mostrar que não há diferença significativa na corretude da documentação ao utilizar as duas abordagens. Entretanto, algumas visões, como a de contexto (C4) e física (4+1), por exemplo, são mais importantes e mais fáceis de utilizar nas fases iniciais de desenvolvimento e outras, como a de componentes (C4) e lógica (4+1), por exemplo, só serão úteis em estágios futuros. Por fim, verificou-se também que, para não impactar o processo de desenvolvimento, uma solução é definir uma pessoa da equipe para documentar a arquitetura do sistema, desde que tenha conhecimento amplo sobre o produto e, se não o tiver, permita a contribuição por parte dos outros membros da equipe.

Palavras-chave: Desenvolvimento Ágil de Software, Documentação de software, Arquitetura de software, Arquitetura Ágil.

Abstract

Today's software development environment is highly competitive and imposes on companies the growing need to deliver value quickly and spend fewer resources. In this case, these companies allocate resources to accelerated development to the detriment of planning activities and software architecture documentation. As a result, they do not accurately plan or document relevant information about the system such as user interactions, infrastructure, and development platform, leading to greater complexity in systems development and maintenance. In this context, our research question is **which system architecture documentation approaches are suitable in agile software development contexts and how to use them?** We executed a literature review and a feature analysis to evaluate the 4+1, S4V, ADD, BAPO/CAFCR, C3A, and C4 approaches. The analysis indicated that the 4+1 and C4 approaches had potential for application in the context studied, and we carried out three more experimental studies to validate it. There is no significant difference in the correctness of the documentation when using the two approaches. However, some views, such as context (C4) and physical (4+1), for example, are more important and easier to use in the early stages of development, and others, such as component (C4) and logical (4+1), will only be useful in the future. Finally, we also found that, in order not to impact the development process, one solution is to define a person from the team to document the system's architecture, as long as they have extensive knowledge of the product and, if they do not, allow other team members to contribute.

Keywords: Agile software development, Software documentation, Software Architecture, Agile Architecture.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1

INTRODUÇÃO

Neste capítulo, é descrita a motivação que levou à realização desta pesquisa de mestrado, além do contexto estudado e do problema que se buscou atacar. Nas seções seguintes, a questão de pesquisa, os objetivos, a metodologia geral e a organização do trabalho também são descritos com detalhes.

1.1 Motivação

Antes do início do mestrado, eu trabalhei com desenvolvimento de software, principalmente, mantendo sistemas legados. Durante esse processo, a minha maior dificuldade sempre foi entender sobre o sistema que eu deveria manter: a sua organização, como os componentes se relacionavam e o que cada parte do sistema fazia.

Entretanto, a quantidade de informações registradas sobre o sistema era mínima e defasada. A única fonte escrita de conhecimento era o código-fonte que o compunha. Por isso, era necessário questionar aos colegas de equipe e aos superiores, o que atrapalhava o meu trabalho e o das outras pessoas.

1.2 Contexto e Problema

O processo de desenvolvimento mudou bastante com o passar dos anos. Atualmente, as novas empresas de software competem por fatias de mercado através do lançamento de produtos inovadores, usando tecnologias de ponta e práticas ágeis de desenvolvimento de software (KUHRMANN et al., 2021). A maior parte dessas empresas lança pequenas versões dos seus produtos ao mercado para coletar *feedback* de possíveis consumidores iniciais. Nesse contexto, ideias e funcionalidades do sistema mudam constantemente, assim como os requisitos que devem ser atendidos (KLOTINS; UNTERKALMSTEINER; GORSCHEK, 2019).

Essas empresas gastam tempo e recursos no rápido desenvolvimento de protótipos. Nesse sentido, aspectos de qualidade como planejamento da arquitetura do sistema e testes automatizados têm menor prioridade, devido ao contexto altamente incerto em que estão inseridas (GIARDINO et al., 2016). Além disso, documentar o conhecimento sobre o produto é considerado desperdício, visto que não adiciona valor ao produto final e a produtividade da empresa é medida em termos da quantidade de software desenvolvido, o que torna a documentação uma tarefa contra-produtiva (THEUNISSEN; HEESCH; AVGERIOU, 2022).

No entanto, quando a empresa cresce, ela precisa definir processos mais estruturados de desenvolvimento para manter e evoluir o sistema, além de contratar mais pessoas e atrair mais investimentos (KLOTINS; UNTERKALMSTEINER; GORSCHEK, 2019). Nesse contexto, muitas vezes, a arquitetura desenvolvida não foi planejada da melhor forma e o seu conhecimento, antes restrito à mente dos fundadores e funcionários da empresa, precisa ser aprendido e man-

tido pelos novos membros da equipe (GIARDINO et al., 2016). Além disso, o alto nível de conhecimento tácito torna a tomada de decisões sobre a arquitetura do sistema mais difícil (DASANAYAKE et al., 2015) e pode criar discrepâncias entre o que é entendido pela equipe sobre o sistema e como ele realmente funciona (KLOTINS et al., 2018).

A partir do contexto e do problema relatado anteriormente, a questão de pesquisa que guia este trabalho é: **quais abordagens de documentação da arquitetura de sistemas são viáveis em contextos ágeis de desenvolvimento de software e como utilizá-las?**

1.3 Objetivos

O objetivo geral desta pesquisa é avaliar quais abordagens de documentação de arquitetura de software são mais viáveis para aplicação em contextos ágeis de desenvolvimento. Os objetivos específicos desta pesquisa são:

- Entender quais são as dificuldades enfrentadas em contextos ágeis de desenvolvimento relacionadas a documentação de sistemas, em especial, relacionada à arquitetura de software.
- Selecionar um conjunto de características relevantes em abordagens de documentação, focado no contexto estudado.
- Criar um corpo de conhecimento sobre as abordagens disponíveis na literatura para documentação e registro de conhecimento da arquitetura de software.

- Investigar a viabilidade das abordagens coletadas para documentação da arquitetura de software, em contextos ágeis de desenvolvimento de software.

1.4 Metodologia de Pesquisa

A metodologia de pesquisa está presente na Figura 1 e é descrita abaixo.

Figura 1 – Metodologia de pesquisa adotada neste trabalho.

1. **Revisão da literatura** – Estudo para compreender o contexto ágil de desenvolvimento de software, os problemas enfrentados por empresas presentes nesse contexto e coletar abordagens de descrição de arquitetura de software. As abordagens coletadas serviram como ferramentas candidatas na *Feature Analysis*, realizada na etapa seguinte. Além disso, artigos sobre diretrizes para construção de arquiteturas ágeis e descrições úteis e consistentes também foram coletados e utilizados para definição das *features*.

2. ***Feature Analysis* de modelos arquiteturais** – Estudo realizado através de uma análise teórica de características sobre abordagens de descrição de arquitetura de software, para verificar a viabilidade de implementação em contextos ágeis de desenvolvimento de software, segundo as orientações propostas por Kitchenham, Linkman e Law (1996). As duas abordagens com as melhores pontuações (C4 e 4+1) foram utilizadas como instrumento no estudo de viabilidade, descrito na etapa a seguir.

3. **Estudo de viabilidade *in vitro*** – Estudo realizado com a participação de alunos da disciplina de Análise e Projeto de Sistemas, do curso de Bacharelado em Ciência da Computação, da Universidade Federal do Amazonas (UFAM). A turma foi dividida em equipes e cada uma utilizou duas abordagens de documentação de arquitetura para planejar e registrar o conhecimento sobre a arquitetura de seus sistemas, idealizados em disciplinas anteriores do curso. As abordagens utilizadas durante o estudo foram as duas melhores pontuadas na *Feature Analysis*, realizada na etapa anterior. As percepções dos participantes sobre a facilidade ou dificuldade de uso, utilidade e futura adoção guiaram a etapa seguinte da pesquisa, além de mostrar se os resultados da *Feature Analysis* são válidos e se alguma das abordagens utilizadas é suficiente para implementação em contextos ágeis de desenvolvimento de software.

4. **Experimento Controlado** – Estudo experimental realizado com alunos de uma turma da disciplina de Arquitetura de Software, do curso de Bacharelado em Ciência da Computação, da Universidade Federal do Ceará (UFC) – Campus Russas. O objetivo deste estudo foi avaliar a corretude de visões construídas com as mesmas abordagens de documentação utilizadas no estudo anterior, em um ambiente controlado e com descrições de sistemas previamente definidas.

5. **Estudo em Ciclo de Vida** – Estudo experimental realizado com alunos da disciplina de Práticas em Engenharia de Software, do curso de Bacharelado em Engenharia de Software, da Universidade Federal do Amazonas. O objetivo deste estudo é coletar indícios sobre a utilização das duas abordagens melhor pontuadas na *Feature Analysis*, de forma semelhante

ao primeiro estudo experimental, além de obter relatos sobre o impacto da tarefa de documentar a arquitetura de software no processo de desenvolvimento de software de equipes ágeis de desenvolvimento.

1.5 Resultados Esperados

A partir da realização desses estudos, espera-se obter indícios sobre quais abordagens e, especificamente, que visões são mais viáveis em contextos ágeis de desenvolvimento de software, além de informações sobre em que fases do ciclo de desenvolvimento e de que forma cada uma deve ser aplicada.

1.6 Organização do Trabalho

O Capítulo 2 apresenta o Referencial Teórico sobre os conceitos de desenvolvimento ágil de software, arquitetura ágil, além das suas correlações e trabalhos relacionados. O Capítulo 3 apresenta as abordagens utilizadas para a realização da *Feature Analysis*. O Capítulo 4 descreve a análise de abordagens de descrição de arquitetura de software, no contexto ágil de desenvolvimento de software. O Capítulo 5 relata o estudo experimental in vitro, realizado na turma de Análise e Projeto de Sistemas, na Universidade Federal do Amazonas. O Capítulo 6 descreve o experimento controlado, realizado na turma de Arquitetura de Software, da Universidade Federal do Ceará. O Capítulo 7 apresenta o estudo em ciclo de vida, realizado com os alunos de Práticas em Engenharia de Software, da Universidade Federal do Amazonas. O Capítulo 8 contém as considerações finais desta pesquisa, assim como as contribuições e indicações de trabalhos fu-

turos.

2

REFERENCIAL TEÓRICO

2.1 Desenvolvimento Ágil de Software

O Manifesto Ágil (BECK et al., 2001) guia a construção de sistemas para atender aos clientes de forma rápida e através de entregas contínuas. Os requisitos são dinâmicos e incorporados ao processo de desenvolvimento. A produtividade é medida pela quantidade de software entregue e algumas práticas ágeis são o desenvolvimento iterativo e incremental, com a entrega de novas versões em intervalos curtos e a refatoração de código de versões anteriores do sistema (YANG; LIANG; AVGERIOU, 2016).

Além disso, algumas metodologias agregam essas práticas e ajudam organizações em seus processos de desenvolvimento como o Scrum (SCHWABER, 1997), através de *sprints*, para construir versões menores do sistema em períodos de uma ou duas semanas e reuniões diárias (*daily meetings*), para acompanhar o progresso do desenvolvimento; e o XP, com a programação em pares, integração contínua e refatoração (BECK, 1999).

O contexo ágil pode ser encontrado em diversas empresas, como as startups de software (KLOTINS et al., 2021), por exemplo, devido à pressão do mer-

cado, a necessidade de conquistar clientes e a incerteza sobre as funcionalidades e o público-alvo dos produtos de software. Pantiuchina et al. (2017) também afirmam que as *startups* de software também aplicam práticas ágeis, com o objetivo de acelerar as entregas aos clientes, como lançamentos frequentes e planejamento a curto prazo. No entanto, aspectos relacionados à qualidade do sistema, como refatoração e testes automatizados são negligenciados (PANTIUCHINA et al., 2017).

2.2 Documentação no Contexto Ágil de Desenvolvimento de Software

Os desafios enfrentados em contextos ágeis de desenvolvimento com relação à documentação de software são a dificuldade em entender documentações informais, a medição de produtividade pela quantidade de software entregue, a falta de sincronia entre a documentação e o sistema e, por fim, o pouco valor dado à documentação (THEUNISSEN; HEESCH; AVGERIOU, 2022).

Nesse sentido, muitas empresas se baseiam em comunicação verbal e em artefatos de código como documentação (TDD e BDD). Além disso, relatos da literatura mostram que requisitos e *user stories* são as informações mais documentadas em contextos ágeis de desenvolvimento de software (ISLAM; HASAN; EISTY, 2023).

2.3 Arquitetura de Software

A arquitetura de software, de acordo com a ISO 42010/2011, é "o conjunto de conceitos ou propriedades fundamentais de um sistema, composto pelos seus elementos, relações e princípios de projeto e evolução" (ISO/IEC/IEEE..., 2011). Ela também é composta pelas decisões tomadas para conciliar os desejos dos *stakeholders* e a construção do sistema, como a escolha das tecnologias empregadas, da distribuição do sistema e de que forma ele será construído (SOMMERVILLE, 2010).

A arquitetura de software é representada através de visões, que contém as estruturas presentes no software e no hardware do sistema (BASS; CLEMENTS; KAZMAN, 2021). Essas estruturas podem ser modulares, comportamentais e de alocação dos recursos físicos. A primeira contém a divisão das responsabilidades dentro do sistema dentre os módulos e componentes do sistema. As visões comportamentais mostram o sistema e a sua arquitetura em tempo de execução, através dos processos e atividades realizadas nele. Por fim, as visões de alocação descrevem como o sistema será implantado no hardware disponível (BASS; CLEMENTS; KAZMAN, 2021).

Nesse contexto, para definir a arquitetura de software de um sistema, algumas atividades são realizadas e, segundo Hofmeister et al. (2007), são elas:

- Análise arquitetural - Consiste na definição dos problemas que a arquitetura deve resolver.
- Síntese arquitetural - Propõe possíveis soluções de arquitetura, de acordo com os requisitos mais significativos.

- Avaliação da arquitetura - Garante que as decisões sobre a arquitetura são corretas, através de avaliações de diferentes alternativas.

Além dessas atividades, outras ações relacionadas à arquitetura de software são a sua manutenção, para correções de falhas (ISO/IEC/IEEE..., 2011) e evolução da arquitetura, para se adequar a mudanças de requisitos (YANG; LIANG; AVGERIOU, 2016; WATERMAN; NOBLE; ALLAN, 2015).

A arquitetura de software evoluiu, junto ao desenvolvimento de sistemas, e tornou-se objeto de pesquisa a partir da década de 1990 (KRUCHTEN; OBBINK; STAFFORD, 2006). Abordagens de descrição de arquitetura como 4+1 (KRUCHTEN, 1995), Siemens' 4 Views (SONI; NORD; HOFMEISTER, 1995) e BAPO/CAFCR (AMERICA; ROMMES; OBBINK, 2003) foram apresentadas, além de métodos de avaliação e análise da arquitetura como o ATAM (KAZMAN; KLEIN; CLEMENTS, 2000), por exemplo.

Com o desenvolvimento de sistemas distribuídos e a utilização de metodologias ágeis, a arquitetura de software precisou se adaptar a esse contexto (ERDER; PUREUR; WOODS, 2021), planejada de forma progressiva e iterativa, junto ao ciclo contínuo de lançamentos do sistema.

2.4 Arquitetura Ágil de Software

A arquitetura de software ágil é, segundo Waterman, Noble e Allan (2015), satisfaz a definição descrita na Seção 2.3 e responde melhor às mudanças de requisitos, fatores externos e contextos de alta incerteza. Ela deve ser fácil de modificar e projetada de maneira iterativa e incremental.

Waterman, Noble e Allan (2015) formularam uma teoria sobre as forças

que atuam sobre uma arquitetura ágil e as estratégias adotadas pelos engenheiros de software para lidar com elas. Algumas forças detectadas são a instabilidade dos requisitos, o valor entregue ao consumidor de forma rápida e a experiência dos integrantes da equipe.

Os requisitos do sistema podem ser insuficientes e alterados, durante o processo de desenvolvimento. Essa incerteza faz com que os desenvolvedores de software evitem, ao máximo, a geração de artefatos e documentos da arquitetura do sistema pois, a qualquer momento, os requisitos podem ser alterados, provocando o desperdício do trabalho realizado e dos recursos empregados.

Essa atitude também serve para entregar valor ao cliente o quanto antes. Entretanto, ela provoca um aumento do esforço despendido pela empresa pois a arquitetura, emergente ou não, evolui a cada iteração. O custo de desenvolvimento cresce e, se arquitetura gerada até a iteração não for mais viável, ela deverá ser refeita. Ainda assim, o custo desse retrabalho é aceito pelas empresas ágeis, pois ele surge nas fases futuras do desenvolvimento (WATERMAN; NOBLE; ALLAN, 2015).

Para responder às forças descritas acima, Waterman, Noble e Allan (2015) descrevem quatro estratégias, presentes entre as cinco listadas abaixo:

- S1 – **Adaptar-se a mudanças**: Manter os projetos de arquitetura simples, validar o código com a arquitetura de forma iterativa, utilizar boas práticas de projeto, adiar tomadas de decisão e planejar alternativas

- S2 – **Endereçar riscos**: Planejar e tomar decisões sobre aspectos que estão relacionados ao sistema como um todo (como escolha de tecnologias para a construção do sistema)

- S3 – **Arquitetura emergente**: Planejar antecipadamente o mínimo possível de decisões sobre a arquitetura, como os padrões de arquitetura do sistema, tecnologias a serem utilizadas, entre outros.

- S4 – **Planejamento completo antes do desenvolvimento**: Essa estratégia é contrária à S3, pois ela sugere a elicitação de todos os requisitos e planejamento completo do sistema antes do seu desenvolvimento. Essa estratégia também não é desejável, pois ela reduz a capacidade da equipe e do sistema se adaptar a mudanças (S1).

- S5 – **Usar frameworks e as arquiteturas que eles implementam**: Essas ferramentas possuem padrões arquiteturais que restringem os sistemas implementados com elas a esses padrões. O ponto positivo é a utilização de padrões estabelecidos para solucionar problemas comuns, diminuindo a necessidade dos engenheiros de software de tomar decisões nesse sentido para o sistema. Isso reduz o esforço e permite o desenvolvimento das soluções de forma mais rápida.

Yang, Liang e Avgeriou (2016) também estudaram a combinação da arquitetura de software com o desenvolvimento ágil, através de um mapeamento sistemático. Como resultados, os autores identificaram abordagens como desenvolvimento incremental, uso de *backlogs* e planejamento apenas do que for suficiente sobre a arquitetura. Além disso, Yang, Liang e Avgeriou (2016) afirmam que o foco no projeto da arquitetura deve existir o quanto antes, os *stakeholders* devem decidir quando "congelar" a arquitetura, para lançar uma nova versão e a modelagem da arquitetura ágil é melhor conduzida quando feita de forma incremental, com o envolvimento dos clientes.

2.5 Descrições de Arquitetura de Software Úteis e Consistentes

Para endereçar a necessidade de equilibrar o planejamento da arquitetura feito antes do desenvolvimento com as mudanças que surgem em contextos ágeis, Wohlrab et al. (2019) desenvolveram um conjunto de diretrizes para construir descrições de arquitetura de software mais consistentes e, por consequência, mais úteis para seus usuários. Dessa forma, decisões importantes sobre a arquitetura podem ser tomadas, mantendo espaço disponível para adaptação aos novos requisitos.

Essas diretrizes foram extraídas após a realização de dois *surveys*. O primeiro serviu para investigar as inconsistências presentes em descrições de arquitetura e entre a documentação e o código do sistema e, o segundo, para validar as diretrizes propostas.

Wohlrab et al. (2019) listaram alguns tipos de inconsistências presentes em descrições de arquitetura de software. São elas:

- Especificação não sincronizada com o código
- Regras definidas na arquitetura mas não implementadas
- Padrões e diretrizes que devem ser seguidos, mas não são reforçados como regra
- Linguagem inconsistente

Além disso, Wohlrab et al. (2019) verificaram que as inconsistências em descrições de arquitetura acontecem por falta de tempo e de conhecimento sobre o sistema, junto à falta de interesse e valor dado a esses artefatos. Nesse

contexto, os autores propuseram as seguintes diretrizes para construir descrições de arquitetura consistentes e úteis:

- G1 – Para definir uma descrição de arquitetura, deve-se descrever de forma clara o seu público-alvo e o propósito desse artefato.
- G2 – Deve existir uma clara distinção entre a arquitetura corrente e a planejada para o futuro.
- G3 – Minimizar a quantidade de elementos no planejamento da arquitetura o máximo possível.
- G4 – Antes de criar a descrição da arquitetura, avaliar cuidadosamente as decisões sobre a arquitetura.
- G5 – Integrar os arquitetos de software à coleta de feedback com os desenvolvedores, para identificar inconsistências e aspectos emergentes sobre o sistema
- G6 – Utilizar apenas uma fonte de informação acessível, para tornar a descrição da arquitetura rastreável.

3

ABORDAGENS DE DESCRIÇÃO DE ARQUITETURA DE SOFTWARE

Este capítulo apresenta seis abordagens de descrição de arquitetura de software. Elas foram encontradas durante a revisão da literatura e serviram como ferramentas candidatas da *Feature Analysis* descrita no Capítulo 4.

Um trabalho prévio de comparação de abordagens de documentação de arquitetura foi feito por Hofmeister et al. (2007), que avaliaram as abordagens Attribute Driven Design, Siemens'4 Views, 4+1, BAPO/CAFCR e Architecture Separation of Concerns (ASC) com o objetivo de construir um modelo genérico, gerado a partir das semelhanças entre as cinco abordagens. Dentre essas cinco abordagens, apenas as quatro primeiras foram utilizadas na *Feature Analysis*, pois o artigo original que descreve a abordagem ASC não foi encontrado, impossibilitando a sua análise.

As abordagens C4 Model e C3A Agile Architecture foram extraídas dos trabalhos de Brown (2017) e Hadar e Silberman (2008). A primeira surgiu da busca por abordagens de descrição de arquitetura utilizadas na indústria, mas

sem descrição formal na literatura explorada durante a revisão. A abordagem C3A foi extraída das referências utilizadas no mapeamento sobre a combinação entre arquitetura de software e métodos ágeis, realizado por Yang, Liang e Avgeriou (2016).

As abordagens estão apresentadas de forma ordenada por data de criação, da mais antiga para a mais recente.

3.1 Siemens' 4 Views

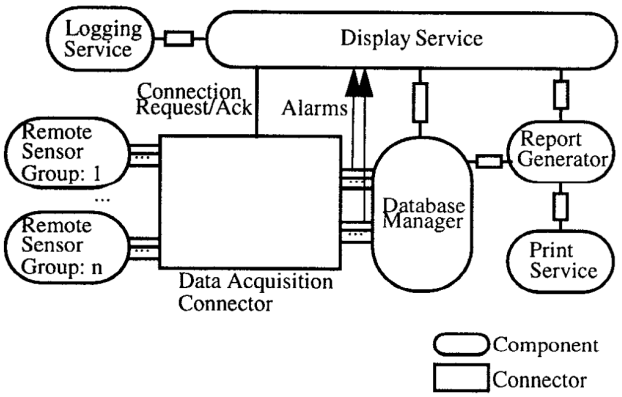
A abordagem Siemens' 4 Views (S4V) surgiu a partir da avaliação feita por Soni, Nord e Hofmeister (1995) sobre como indústrias descreviam e executavam a arquitetura de seus sistemas, através de um *survey* com 11 sistemas que variavam entre processadores de sinais e imagens, sistemas operacionais, sistemas de controle e operação e sistemas de comunicação. Além do *survey*, os arquitetos responsáveis por esses sistemas foram entrevistados para entender a estrutura dos sistemas e os seus usos.

A análise dos sistemas estudados levou à classificação das estruturas dos sistemas em quatro categorias: arquitetura **conceitual**, arquitetura de **módulos interconectados**, arquitetura de **execução** e arquitetura de **código**. Essas categorias, aqui chamadas de visões, estão melhor descritas a seguir.

A visão **conceitual** descreve os elementos principais do sistema. As partes do sistema são representadas como componentes e as relações entre eles, através de conectores. Ela é independente das decisões de implementação e enfatiza os protocolos de interação entre os elementos. Seus usos mais comuns são o projeto de sistemas usando componentes específicos de determinado con-

texto, análise de segurança e confiabilidade, compreensão da configuração estática e dinâmica do sistema, além da estimativa de desempenho (SONI; NORD; HOFMEISTER, 1995). A Figura 2 mostra um exemplo de visão conceitual, presente em (SONI; NORD; HOFMEISTER, 1995).

Figura 2 – Exemplo de visão conceitual da abordagem Siemens 4 Views (SONI; NORD; HOFMEISTER, 1995).



A visão de **módulos interconectados** reflete as decisões de implementação, independente da linguagem escolhida. Ela também pode ser vista como a implementação ideal do sistema. Os autores sugerem que ela seja utilizada em todas as tomadas de decisão do sistema. Seus usos envolvem a gerência e controle de interfaces e de configurações, além da análise do impacto de mudanças no sistema. Essa visão possui duas estruturas:

- A **decomposição funcional**, que captura a forma que o sistema é logicamente disposto (subsistemas, módulos e unidades abstratas) além das relações entre os componentes importados e exportados dentro do sistema. Um exemplo de decomposição funcional está presente na Figura 3.
- A **distribuição em camadas**, que descreve as decisões de projeto baseadas nas relações e restrições das interfaces. Os autores afirmam que ela reduz e

isola dependências e facilita a construção de subsistemas de forma *bottom-up*. A Figura 4 apresenta um exemplo de distribuição em camadas.

Figura 3 – Exemplo de decomposição funcional da abordagem Siemens 4 Views (SONI; NORD; HOFMEISTER, 1995).

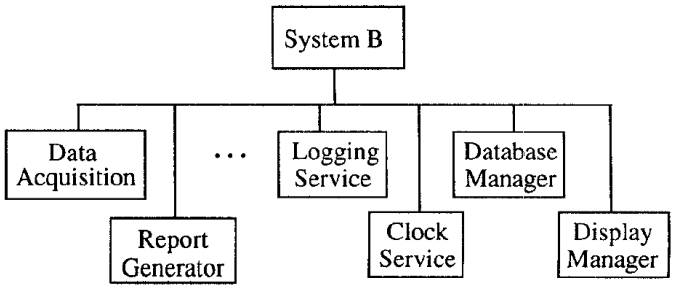
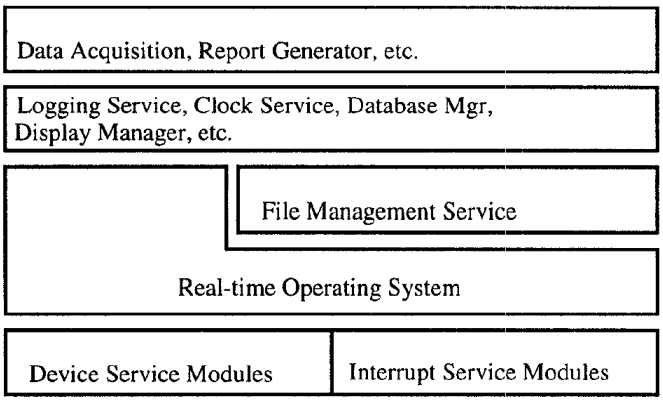


Figura 4 – Exemplo de distribuição em camadas da abordagem Siemens 4 Views (SONI; NORD; HOFMEISTER, 1995).



A visão de **execução** é utilizada para descrever a estrutura dinâmica do sistema, em termos dos elementos de execução. Alguns exemplos desses elementos são tarefas, processos, mecanismos de comunicação e recursos alocados. Essa visão pode ser usada para analisar o desempenho do sistema e a portabilidade do sistema para diferentes ambientes de execução. Nessa visão, os aspectos não funcionais como desempenho, segurança e ambiente de execução também são endereçados.

Por fim, a visão de **código** descreve a organização dos módulos no sistema representados por bibliotecas, arquivos de código-fonte e diretórios. A linguagem escolhida para implementar o sistema tem forte influência sobre essa visão e seu objetivo principal é auxiliar na construção, instalação e gerenciamento do sistema, além de reforçar as restrições especificadas na visão de módulos.

3.2 Kruchten's 4+1

A abordagem apresentada por Kruchten (1995) é composta por cinco visões da arquitetura de um sistema, criadas para endereçar os aspectos do sistema, que interessam públicos distintos como alocação de recursos, tecnologias utilizadas para o desenvolvimento, quais componentes estão presentes no sistema e como essas partes se relacionam.

Kruchten (1995) afirma que o modelo é genérico e que notações, ferramentas e métodos de projeto diferentes podem ser utilizados. O autor, no artigo original, utiliza uma notação derivada da notação de Booch (1990), mas também é possível desenhar as visões utilizando a linguagem UML[1].

A visão **lógica** é responsável por exibir como o sistema atende aos requisitos funcionais elicitados, através de uma abstração em forma de objetos que o integram. Ela pode ser construída através de diagramas de classes ou de pacotes UML. As Figuras 5 e 6 mostram exemplos de visões lógicas.

A visão de **processos** compreende as relações entre diferentes partes do sistema, levando em conta requisitos não funcionais como desempenho, con-

[1] https://www.uml.org/what-is-uml.htm

Figura 5 – Exemplo de visão lógica da abordagem 4+1, construída com um diagrama de classes UML. Figura do autor.

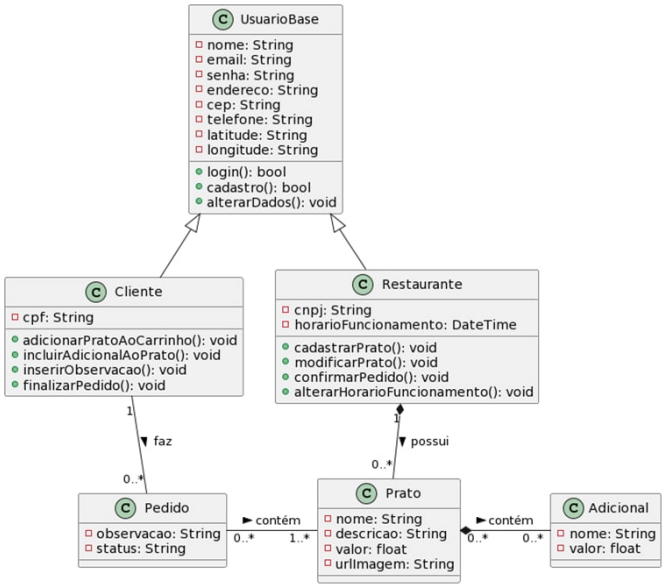
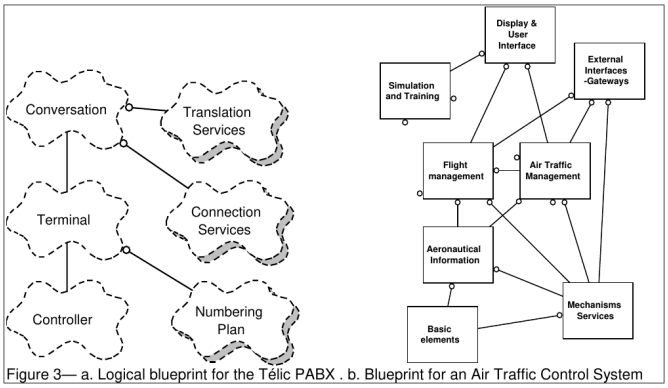


Figura 6 – Exemplos de visões lógicas da abordagem 4+1, construídas com a notação de Booch (KRUCHTEN, 1995).



corrência, disponibilidade e tolerância a falhas. Ela pode ser descrita em diferentes níveis de abstração. Cada processo é um conjunto de tarefas que podem ser executadas de forma independente. Eles se relacionam através de chamadas de procedimentos, envio de mensagens, execução de funções e atividades internas dentro do sistema. Essa visão pode ser construída com diagramas de

comunicação, de sequência e de atividades UML.

A visão de **desenvolvimento** ilustra a organização de módulos dentro do sistema, as suas relações e hierarquias. Ela separa os módulos em camadas empilhadas, com interfaces definidas de comunicação com as camadas adjacentes. Essa visão está relacionada a requisitos internos como desenvolvimento e gerenciamento do software, além de restrições de linguagens e *frameworks*. Ela pode ser construída com diagramas de componentes e pacotes UML. As Figuras 7 e 8 mostra um exemplo da visão de desenvolvimento.

Figura 7 – Exemplo de visão de desenvolvimento da abordagem 4+1, construída com o diagrama de pacotes UML. Figura do Autor.

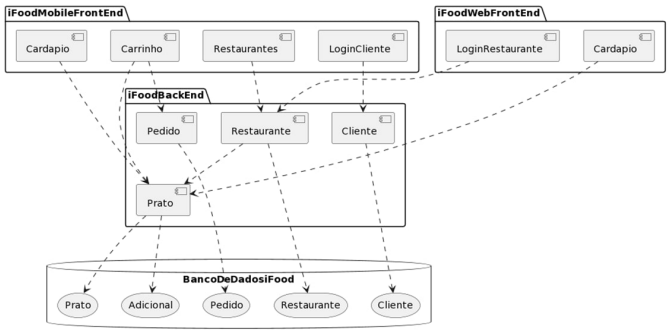


Figura 8 – Exemplo de visão de desenvolvimento da abordagem 4+1 (KRUCHTEN, 1995).

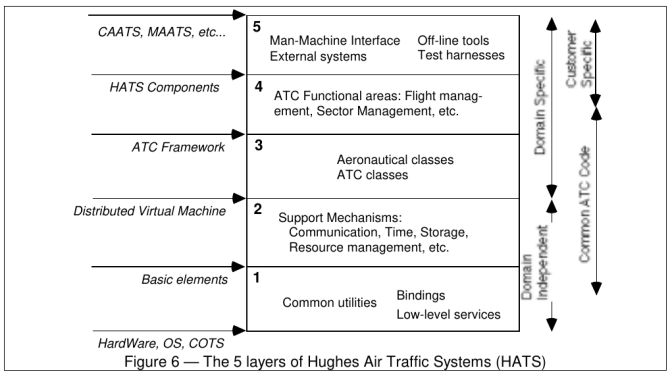


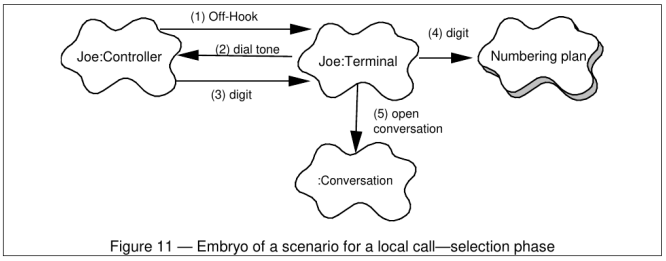
Figure 6 — The 5 layers of Hughes Air Traffic Systems (HATS)

A visão de baixo-nível do sistema é chamada **física**. Nela, a infraestru-

tura do sistema está descrita pelos nós de processamento (servidores, computadores, redes de nós) e a relação entre eles. Nesse artefato, é possível endereçar os requisitos não funcionais do sistema como desempenho, disponibilidade, escala, segurança, entre outros. O diagrama de implantação UML pode ser utilizado para criar essa visão.

Por último, a visão de **cenários** reúne as visões anteriormente descritas para contemplar os principais casos de uso do sistema. Nela, é possível enxergar quais módulos do sistema resolvem o caso escolhido, como ele está organizado na hierarquia, em que nós de processamento ele será executado e quais processos se relacionam com ele para executar a ação pretendida. O diagrama de casos de uso UML pode ser utilizado para construir essa visão. Um exemplo de visão de cenários pode ser vista na Figura 9.

Figura 9 – Exemplo de visão de cenários da abordagem 4+1 (KRUCHTEN, 1995).



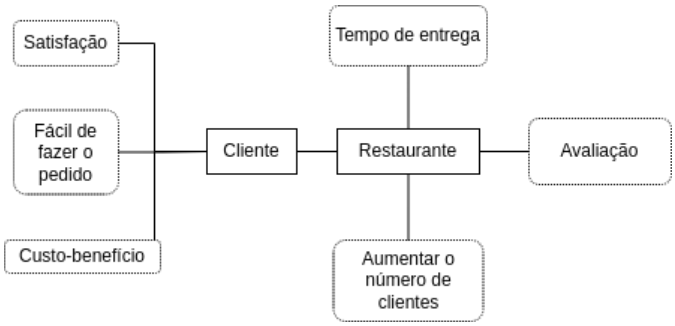
3.3 BAPO/CAFCR

A abordagem BAPO/CAFCR, proposta por America, Rommes e Obbink (2003), foi criada para preencher as lacunas entre as necessidades, os objetivos e os desejos dos clientes e a sua implementação com a tecnologia disponível.

Ela possui cinco visões acerca de diferentes aspectos do sistema como proposta de valor ao cliente, cenários de uso, lista de funcionalidades, decomposição do sistema e decisões tecnológicas.

A visão **comercial** compreende as necessidades do cliente do sistema. O artefato gerado contém um diagrama de contexto, que descreve as partes interessadas do sistema (*stakeholders*), as suas preocupações e outros aspectos que direcionam a utilização do produto. O objetivo dessa visão é mostrar o que o cliente deseja e auxiliar na construção da proposta de valor entregue pela solução. Um exemplo de visão comercial pode ser visto na Figura 10.

Figura 10 – Exemplo de visão comercial da abordagem BAPO/CAFCR (AMERICA; ROMMES; OBBINK, 2003).



A visão de **aplicação** busca mostrar como o sistema pode ser usado para atender às necessidades dos clientes, presentes na visão descrita acima. Sua função principal é auxiliar na construção dos casos de uso e pode ser criada em relação a contextos diferentes como a visão geral do sistema (diagrama de implementação UML), o trabalho realizado no sistema (diagrama de atividades UML, diagramas BPMN) e os domínios da aplicação (diagrama de classes UML).

A visão **funcional** descreve as principais *features* do sistema e o seu comportamento. Os artefatos gerados podem ser um dicionário das *features* que impactam na arquitetura de forma resumida ou uma matriz funcionalidade-valor,

que contém os aspectos que direcionam o uso descritos na visão comercial relacionados com cada *feature* do sistema.

As visão **conceitual** contempla as partes principais do sistema, os princípios e padrões utilizados no sistema e como ele funciona. Diversos são os artefatos que podem ser gerados nesta visão como a decomposição do sistema em diagramas UML, descrições de princípios em linguagem natural e modelos de informação para descrever como os dados são armazenados.

Por fim, a visão de **realização** apresenta as tecnologias utilizadas para implementar o sistema. Além disso, ela descreve os mecanismos, convenções e metas de lançamento de versões.

3.4 Attribute Driven Design (ADD)

A abordagem Attribute Driven Design, proposta por Wojcik et al. (2006), auxilia no projeto da arquitetura do sistema com foco nos atributos de qualidade relevantes para os *stakeholders* e para a arquitetura.

As etapas principais podem ser divididas entre **planejamento**, em que os atributos de qualidade influenciam na escolha dos elementos da arquitetura; **execução**, quando os elementos são instanciados para atender aos requisitos funcionais e aos atributos de qualidade e; **verificação**, que define se os requisitos foram atendidos. Esse processo deve ser feito enquanto os atributos de qualidade não forem completamente atendidos.

As visões que podem ser utilizadas para descrever a arquitetura gerada com o ADD são a de módulos, de componente-conector e de alocação, sem notação definida.

A visão de **módulos** compreende a decomposição do sistema em módulos, os relacionamentos entre eles, a disposição em camadas ou a organização das classes dentro do sistema. A visão de **componente-conector** apresenta os aspectos dinâmicos do sistema como processos, concorrência, consumo e escrita de dados e a comunicação entre servidores. A visão de **alocação** descreve a estrutura de implementação e como os elementos são mapeados para a estrutura física disponível. Em outros casos, ela pode ajudar a implementar e integrar os módulos do sistema para as diversas equipes dentro de uma empresa.

A abordagem ADD propõe as seguintes atividades, que devem ser seguidas a cada iteração:

1. A priorização dos requisitos pelos *stakeholders*.
2. Confirmar se há informações suficientes para a construção da arquitetura e ranquear todos os requisitos em ordem de importância para os *stakeholders*).
3. Escolher e decompor um elemento do sistema, de acordo com a necessidade.
4. Identificar os requisitos que direcionam o desenvolvimento e ordená-los de acordo com o impacto na arquitetura e a importância para os *stakeholders*.
5. Escolher um conceito de projeto que satisfaça os *drivers* arquiteturais, avaliar e resolver inconsistências no conceito de projeto.
6. Instanciar elementos arquiteturais e alocar responsabilidades nas visões de módulos, componente-conector e alocação, apresentadas por Bass, Clements e Kazman (2021).

7. Exercitar os requisitos funcionais instanciados no passo 6, observando as entradas e saídas dos elementos e documentando a interface para cada elemento.

8. Verificar e refinar os requisitos.

3.5 C3A Agile Architecture

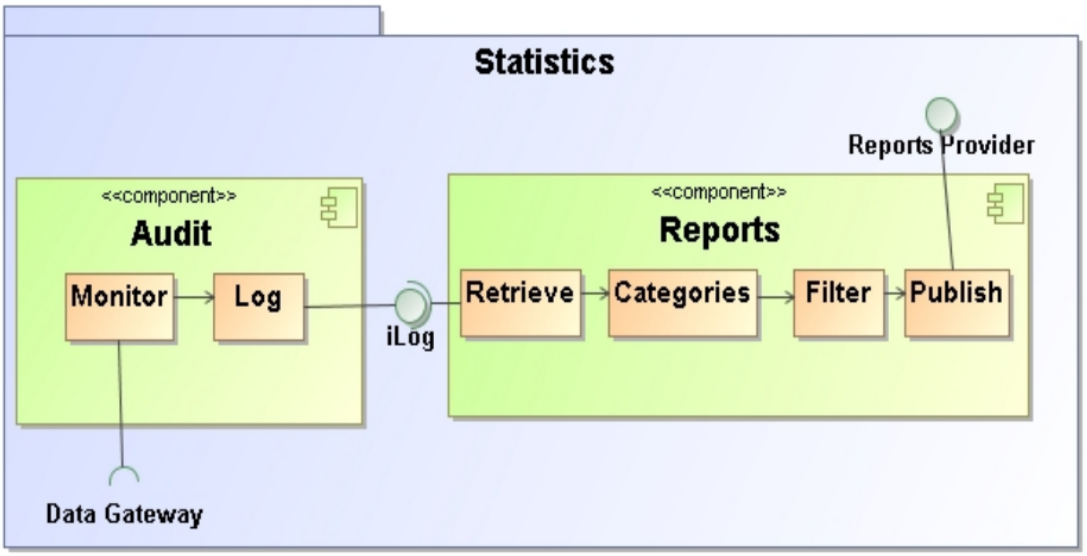
A abordagem C3A Agile Architecture, proposta por Hadar e Silberman (2008), foi criada para permitir a combinação de metodologias ágeis e as atividades de projeto da arquitetura de software. Nesse sentido, os autores definiram apenas dois níveis obrigatórios de abstração para a arquitetura.

No primeiro nível, o nível 0, os **módulos** do sistema estão descritos. Um módulo é um componente do sistema logicamente separado dos demais e com responsabilidades próprias. Para criar a visão desse nível, o diagrama de pacotes UML pode ser utilizado e, para manter o conhecimento sobre os módulos, cada um deles deve ter um contrato associado. O contrato contém informações relevantes sobre os módulos como nome, dono, responsabilidades, dependências, como foi implementado, aspectos de escalabilidade e desempenho, entre outros.

O segundo nível, o nível 1, é formado pelos **componentes** do sistema que integram os módulos. Nele, estão presentes as interfaces de comunicação, protocolos e troca de mensagens entre os componentes. Além disso, cada um deles deve ser capaz de ser substituído sem precisar reinstalar o módulo que ele integra. Os componentes também devem ter um contrato ligado a cada um deles, descrevendo os mesmos aspectos mencionados anteriormente. Os auto-

res sugerem, de forma opcional, a criação de visões para níveis mais profundos de abstração para expor padrões de projeto, tecnologias e o código utilizado em cada componente nível 1. A Figura 11 exibe um exemplo de visão da arquitetura com os elementos níveis 0 e 1. Nela, o pacote *Statistics* representa o módulo de mesmo nome (Nível 0). Dentro dele, existem dois componentes (Nível 1): *Audit* e *Reports*, com outros elementos que os integram, mas que não são auto-implementáveis.

Figura 11 – Exemplo de visão da arquitetura com a abordagem C3A. (HADAR; SILBERMAN, 2008)



Outra característica do C3A Agile Architecture é a separação da arquitetura em duas instâncias: uma **arquitetura de referência** e outra de **implementação**. A primeira é composta pelos níveis 0 e 1, no estado corrente do sistema. Ela serve para guiar a equipe de desenvolvimento na construção de grandes *releases* e na compreensão do estado atual do sistema. Por outro lado, a arquitetura de implementação deve conter as mudanças presentes em futuras (e pequenas)

versões do sistema, descritas em formato de módulos e componentes integrados à arquitetura de referência, sem prejudicar o funcionamento atual do sistema.

3.6 C4 Model

A abordagem C4, proposta por Brown (2017), é voltada ao registro do conhecimento da arquitetura de software, em diferentes níveis de abstração. Ela surgiu para padronizar e remover ambiguidades e inconsistências presentes nas representações da arquitetura utilizadas na indústria.

Para isso, a notação proposta é composta por caixas e linhas que podem representar qualquer elemento. Para distinguir os elementos, o autor exige que eles sejam bem descritos, com clareza e o mínimo de ambiguidades. Nesse sentido, cada elemento deve ter um **nome**, uma indicação do seu **tipo** (ator, sistema externo, componente, entre outros) e uma **descrição** sobre o seu papel dentro da visão projetada. As relações entre os elementos também devem ser rotuladas, descrevendo o que elas fazem. Além disso, cada visão deve ter uma **legenda** associada, descrevendo os elementos. A notação da abordagem C4 pode ser vista na Figura 12.

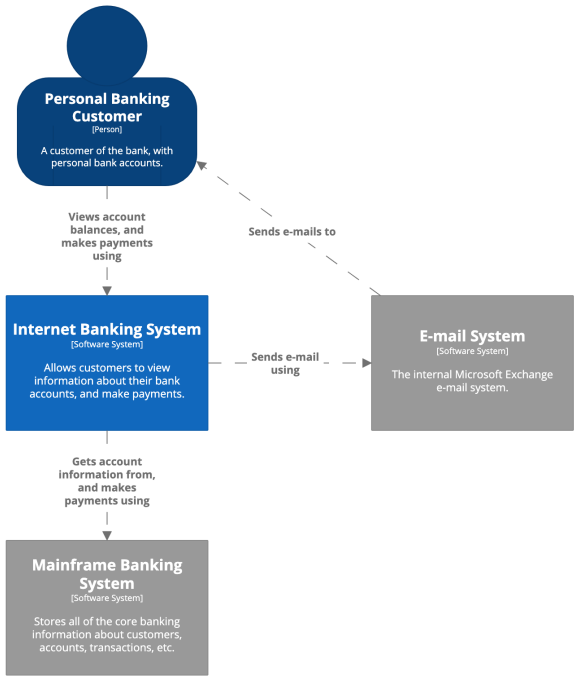
Figura 12 – Notação proposta por para a abordagem C4 (BROWN, 2017).

Ela possui quatro visões principais e três complementares. As visões principais são a de **contexto do sistema**, **contêineres**, **componentes** e **código**.

As visões complementares são a *system landscape*, a dinâmica e a de implementação.

A primeira visão é a visão de **Contexto do Sistema**. Nela, estão descritos apenas o sistema abordado, os atores e outros sistemas interagem com ele. Seu objetivo é mostrar uma ideia simples, porém correta do sistema a um público externo à equipe de desenvolvimento, sem tanto conhecimento técnico de implementação. Por exemplo, um sistema de *internet banking* pode se comunicar com o sistema de um servidor e ser utilizado pelo funcionário do banco e pelo cliente que possui uma conta corrente. A Figura 13 exibe a visão de contexto do sistema para o exemplo descrito anteriormente.

Figura 13 – Exemplo de visão de contexto do sistema, presente na abordagem C4 (BROWN, 2017).



System Context diagram for Internet Banking System
The system context diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)

A visão de **Contêineres** compreende as grandes partes que integram o sistema projetado. Cada uma delas é uma unidade executável e implementável de código ou de banco de dados. Nessa visão, é possível descrever quais tecnologias foram utilizadas para construir os contêineres e quais são as suas responsabilidades, além das relações deles com os atores e sistemas externos. Alguns exemplos de contêineres são APIs REST, aplicativos *mobile* e uma aplicação *web*. A Figura 14 exibe um exemplo de visão de contêineres.

As partes que integram os contêineres podem ser vistas na visão de **Componentes**. Ela é uma visão voltada aos arquitetos e desenvolvedores do sistema e exibe as relações entre os componentes com os outros contêineres e sistemas externos, além das características de implementação. A Figura 15 exibe um exemplo de visão de componentes.

Figura 14 – Exemplo de visão de contêineres, presente na abordagem C4 (BROWN, 2017).

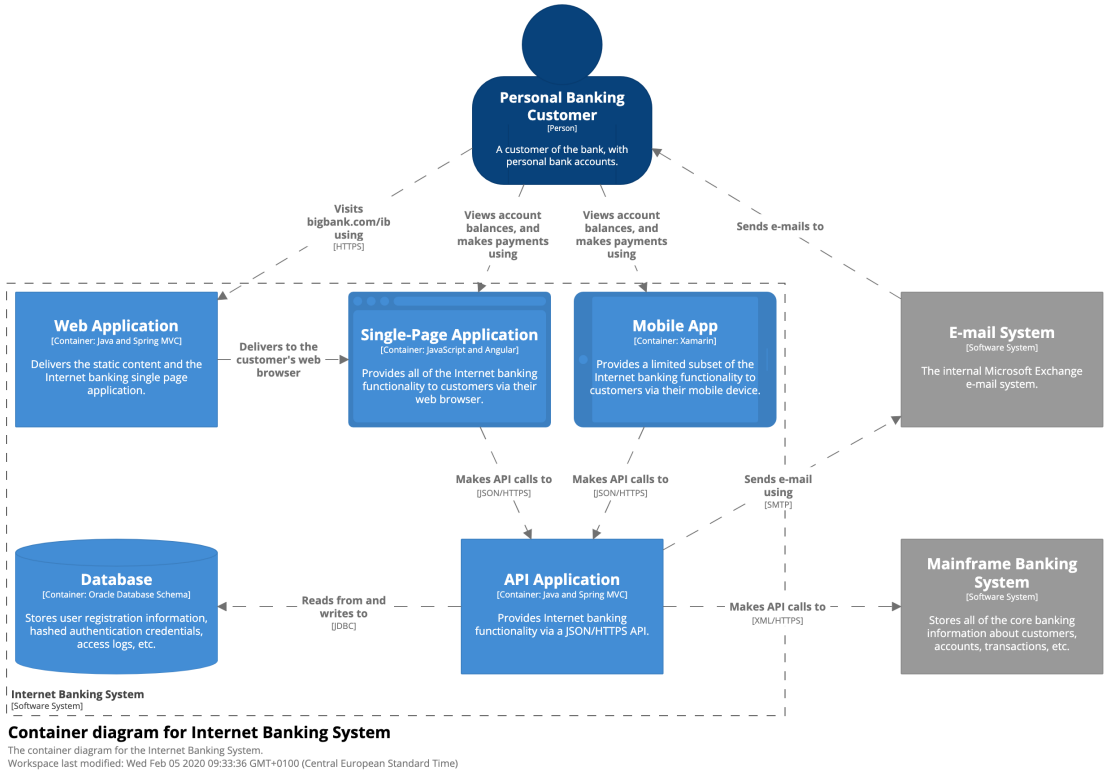
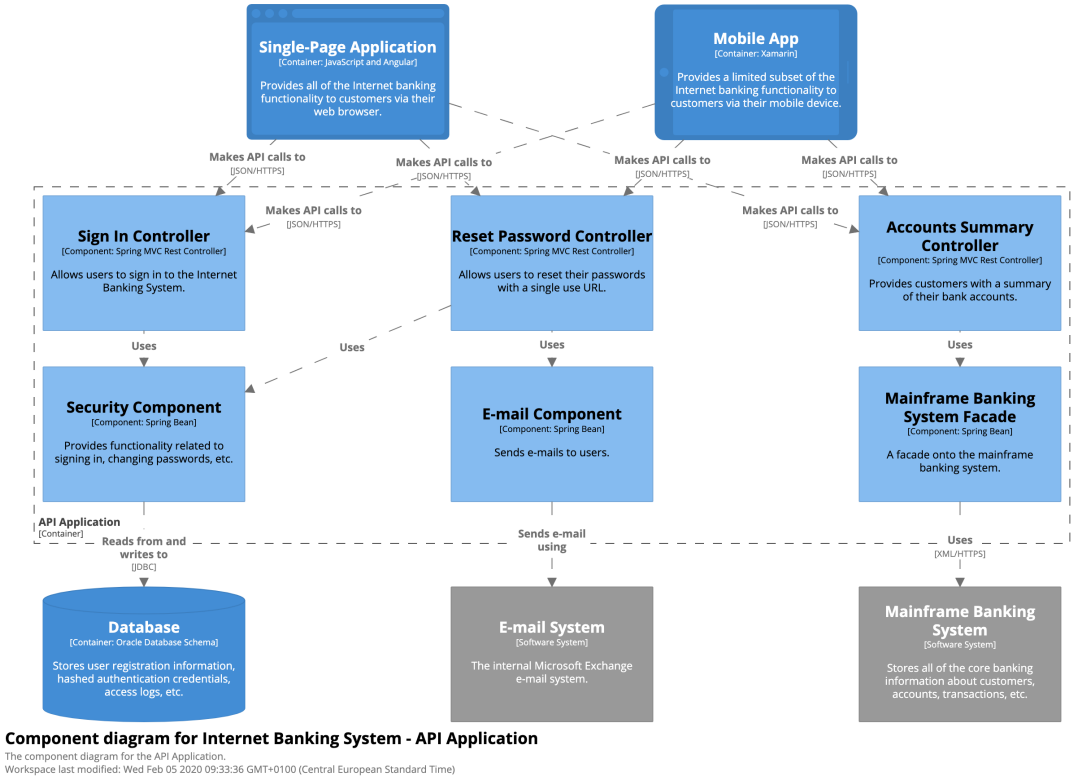


Figura 15 – Exemplo de visão de componentes, presente na abordagem C4 (BROWN, 2017).



Por fim, a última visão principal da abordagem C4 é a visão de **Código**, que descreve como cada componente é criado. Ela pode ser descrita na forma de diagramas UML, Entidade-Relacionamento e, assim como a visão de componentes, é opcional e só deve ser criada em casos extremamente necessários.

As visões complementares são a **Dinâmica** e a de **Desenvolvimento**. A primeira descreve como os elementos implementam, em tempo de execução, uma *user story*, um cenário ou um caso de uso, por exemplo. As interações entre os elementos são feitas com setas direcionadas e descrições numeradas, para indicar a sequência de acontecimentos. A visão de Desenvolvimento descreve como os elementos estão dispostos na infraestrutura disponível. Ela é inspirada

nos diagramas de implementação UML e utiliza os nós de *deployment* para comportar os elementos em termos de servidores, sistemas de balanceamento de carga, máquinas virtuais, entre outros.

3.7 Discussões

As abordagens apresentadas tratam a arquitetura do sistema através de variados níveis de abstração, que endereçam aspectos diferentes e são úteis para pessoas diferentes. Elas compartilham visões semelhantes entre si, que tratam sobre os mesmos aspectos da arquitetura do sistema.

A disposição dos elementos principais do sistema e as suas relações podem ser desenhadas com as visões Lógica (4+1), Módulos Interconectados (S4V), Conceitual (BAPO/CAFCR), Módulos (ADD), Elementos Nível 0 (C3A) e Contêineres (C4).

O mapeamento do sistema para a sua implementação pode ser apresentada com as visões Física (4+1), Execução (S4V), Realização (BAPO/CAFCR), Alocação (ADD) e Implementação (C4 – visão complementar). A abordagem C3A não possui uma visão específica para a implementação dos sistemas, porém essas informações devem ser descritas nos contratos de cada elemento.

O comportamento do sistema também pode ser descrito por visões das abordagens apresentadas. As visões Processo e Cenários (4+1), Execução (S4V), Aplicação (BAPO/CAFCR), Componente-conector (ADD) e Dinâmica (C4 – visão complementar) descrevem o sistema em termos de processos, concorrência e atividades internas. A abordagem C3A não possui elementos ou visões que contemplam o comportamento do sistema.

A organização de código também é mapeada em visões por algumas das abordagens através das visões Desenvolvimento (4+1) e Código (S4V e C4). As abordagens BAPO/CAFCR, ADD e C3A não possuem visões específicas para esse aspecto.

Assim como as abordagens possuem suas semelhanças, elas também têm características únicas. A abordagem BAPO/CAFCR foca no consumidor desde o começo do planejamento da arquitetura com a visão comercial, contendo as dores e desejos dos clientes. A abordagem ADD possui foco maior na qualidade do sistema do que as outras abordagens. Isso pode ser visto no processo extenso, recursivo e iterativo de elicitação e priorização de requisitos de qualidade, seleção de alternativas e validação dos artefatos gerados, com os *stakeholders*. A abordagem C4 Model apresenta uma nova notação para descrever a arquitetura com elementos simples, descrições precisas e sem ambiguidades.

4

FEATURE ANALYSIS DE ABORDAGENS DE DESCRIÇÃO DA ARQUITETURA DE SOFTWARE

Após a coleta e extração das abordagens, uma modelagem inicial de um sistema exemplo foi realizada, para familiarização com as abordagens. Os artefatos gerados estão presentes no Apêndice B e os requisitos do sistema utilizado como exemplo foram extraídos de um repositório no GitHub[1].

Em seguida, para entender se alguma das abordagens descritas pode ser viável para utilização em contextos ágeis de desenvolvimento, um processo de *Feature Analysis* foi realizado. Esse processo é descrito em detalhes na Seção 4.1, os resultados apresentados na Seção 4.2 e as discussões e conclusões expostas nas Seções 4.3 e 4.4, respectivamente.

[1] Requisitos iFood – Priorização de requisitos Moscow. Disponível em: <https://github.com/Requisitos-2018-1-iFood/iFood/wiki/Prioriza%C3%A7%C3%A3o-de-requisitos—Moscow>

4.1 Metodologia

Para verificar a viabilidade de abordagens de documentação de arquitetura de software em contextos ágeis de desenvolvimento de software, uma *Feature Analysis* (FA) foi realizada. Ela é uma das técnicas listadas por Kitchenham, Linkman e Law (1996) para a avaliação qualitativa de métodos, abordagens e outras ferramentas. Um exemplo de uso da *Feature Analysis* para avaliar técnicas é o trabalho de Parizi et al. (2020), em que os autores compararam três recomendadores de técnicas de Design Thinking.

A *Feature Analysis* é baseada em um conjunto previamente definido de *features* relevantes para determinado público-alvo. Os passos para a sua realização envolvem a seleção dos candidatos, a definição das *features* e *subfeatures*, níveis e pesos de importância para cada *Feature Set* e a pontuação das abordagens, através de escalas de julgamento.

4.1.1 Candidatas selecionadas para a avaliação

A seleção das abordagens ocorreu após uma revisão exploratória nos *journals* TOSEM, JSS, TSE, IST e TOIS, buscando por artigos que apresentem abordagens de descrição de arquitetura de software. Dentre os artigos encontrados, Hofmeister et al. (2007) apresentam um *framework* para construção de abordagens de descrição de arquitetura. Ele é derivado de cinco abordagens de descrição: 4+1, Siemens' 4 Views, Attribute Driven Design (ADD), BAPO/CAFCR e Architecture Separation of Concerns (ASC). As quatro primeiras estão presentes na *Feature Analysis* e a ASC não foi incluída, devido à ausência do artigo original, não sendo possível avaliá-la.

Yang, Liang e Avgeriou (2016) apresentam um mapeamento entre métodos ágeis e arquitetura de software. Um dos artigos selecionados pelos autores descreve uma abordagem que atendia ao critério de inclusão, a C3A Agile Architecture, também avaliada. Por fim, uma busca por abordagens foi feita em portais (*grey literature*), para encontrar abordagens utilizadas na indústria de software, porém não descritas em *journals* acadêmicos. Dessa forma, a abordagem C4 Model (BROWN, 2017) foi encontrada e utilizada para avaliação. As abordagens avaliadas são listadas a seguir e, melhor descritas, no Capítulo 3.

- **Siemens' 4 Views** – A abordagem proposta por Soni, Nord e Hofmeister (1995) possui quatro visões principais: conceitual, de módulos, de execução e de código, que descrevem elementos, comportamentos e tecnologias utilizadas no sistema.

- **Kruchten's 4+1** – É composta por cinco visões da arquitetura de um sistema, cada uma relacionada a diferentes aspectos do sistema. As visões presentes são a lógica, de processos, de desenvolvimento, física e de cenários (KRUCHTEN, 1995).

- **BAPO/CAFCR** – A abordagem BAPO/CAFCR possui cinco visões principais acerca de diferentes aspectos do sistema: comercial, aplicação, funcional, conceitual e realização. Ela também possui variações dessas visões, para lidar com os cenários de uso (AMERICA; ROMMES; OBBINK, 2003).

- **Attribute Driven Design** – Propõe a construção da arquitetura de software através de diversos passos recursivos, feitos de forma iterativa, com foco nos atributos de qualidade relevantes para os *stakeholders* e para a ar-

quitetura. As visões geradas são a de módulos, de componentes-conectores e de alocação (WOJCIK et al., 2006).

- **C3A Agile Architecture** – Essa abordagem possui dois níveis obrigatórios de abstração para a arquitetura e a sua separação em uma versão de referência, que trata das grandes versões do sistema e outra de implementação, que lida com as próximas iterações (HADAR; SILBERMAN, 2008).

- **C4 Model** – A abordagem C4 possui quatro visões principais, compostas apenas por caixas e linhas que podem representar qualquer elemento. As visões são: contexto de sistema, contêineres, componentes e código. Além delas, o autor apresenta outras visões complementares como a visão dinâmica e a de desenvolvimento (BROWN, 2017).

4.1.2 Definição das *features*, pesos e níveis de importância

As *features* e os pesos de importância foram derivados das características descritas na literatura sobre contextos ágeis de desenvolvimento de software (PATERNOSTER et al., 2014; GIARDINO et al., 2016; KLOTINS et al., 2019), arquitetura de software ágil (WATERMAN; NOBLE; ALLAN, 2015) e consistência e utilidade de descrições de arquitetura (WOHLRAB et al., 2019).

Nesse sentido, cada um dos conjuntos de *features* (*Feature Set – FS*) possui um **peso de importância (PI)**, definido e revisado pelos autores de forma iterativa, buscando a adequação ao contexto estudado. Os pesos descrevem a relevância do FS e, baseado no trabalho de Marshall, Brereton e Kitchenham (2014), a soma dos pesos dos FS foi igual a 1,0. A Tabela 1 apresenta os FS com a descrição e os pesos associados a cada um deles.

Tabela 1 – *Feature Sets* e *subfeatures* utilizados na análise.

ID	*Feature Set*	Descrição da *Subfeature*	Nível de Importância (NI)	Peso de Importância (PI)
FS1	Documentação	A abordagem deve apoiar o registro do conhecimento sobre a arquitetura do sistema.	Desejável (2)	0,15
		A abordagem deve permitir a documentação de uma quantidade pequena de elementos no planejamento inicial.	Altamente Desejável (3)	
		A abordagem deve possuir diferentes descrições da arquitetura para diferentes públicos e propósitos.	Interessante (1)	
		A abordagem deve ser capaz de separar as arquiteturas correntes e futuras.	Interessante (1)	
FS2	Análise de decisões	A abordagem deve facilitar a análise de decisões arquiteturais.	Interessante (1)	0,05
FS3	Sistema	A abordagem deve permitir a modularidade da arquitetura do sistema.	Desejável (2)	0,10
FS4	Requisitos	A abordagem deve ser flexível e adaptável às mudanças de requisitos e decisões arquiteturais.	Obrigatória (4)	0,35
FS5	Custo	A abordagem deve permitir a construção da arquitetura com a menor quantidade de tarefas obrigatórias.	Obrigatória (4)	0,35

O FS relacionado à **documentação** (FS1 – $PI_{FS_1} = 0,15$) tem o objetivo de avaliar o apoio ao registro do conhecimento sobre a arquitetura de software, fornecido pelas abordagens candidatas. Esse conjunto foi construído, também, para verificar se a abordagem avaliada atende aos critérios G1 (Descrever de forma clara, o público-alvo), G2 (Distinguir a arquitetura corrente e futura) e G3 (Minimizar a quantidade de elementos presentes nas descrições de arquitetura de software) propostos por Wohlrab et al. (2019) para criar uma descrição útil e consistente, além da diretriz S1 (Adaptar-se a mudanças), apresentada por Waterman, Noble e Allan (2015), para construção de um arquitetura ágil: minimizar a quantidade de elementos planejados na arquitetura (FS1-02), a separação de públicos e propósitos das visões da arquitetura (FS1-03) e a diferenciação

entre as instâncias corrente e futuras da arquitetura (FS1-04).

O FS2 ($PI_{FS_2} = 0,05$) compreende a capacidade de analisar as **decisões arquiteturais** através da abordagem candidata, o que inclui o registro das decisões tomadas em artefatos, assim como o raciocínio realizado durante a tomada da decisão. Esse FS foi escolhido para atender ao critério G4 (Avaliar cuidadosamente as decisões sobre a arquitetura), proposto por Wohlrab et al. (2019), para criar descrições úteis e consistentes. Além disso, equipes ágeis de desenvolvimento adiam a tomada de decisões arquiteturais o máximo possível, acumulando débito técnico a ser pago, quando a empresa tiver mais recursos humanos e financeiros. Entretanto, na maioria das vezes, o produto se torna complexo e difícil de corrigir (KLOTINS et al., 2019). Dessa forma, facilitar a análise e a tomada de decisões nas fases iniciais pode contribuir na evolução do produto, reduzindo o débito que será acumulado.

O FS3 ($PI_{FS_3} = 0,10$) tem o objetivo de avaliar os candidatos quanto à **capacidade de projetar sistemas modulares**. Esse *Feature Set* foi definido pois a construção modular dos sistemas é uma característica encontrada em contextos ágeis de desenvolvimento, ao permitir a remoção ou alteração de funcionalidades, em um contexto de necessidades incertas (PATERNOSTER et al., 2014). O quarto *Feature Set* (FS4 – $PI_{FS_4} = 0,35$) avalia a **flexibilidade e adaptabilidade às mudanças de requisitos**, característica muito presente no contexto estudado nessa pesquisa (GIARDINO et al., 2016).

Por fim, o FS5 (FS05 – $PI_{FS_5} = 0,35$) avalia o **custo de aplicação das abordagens**, medido pela quantidade de tarefas que devem ser realizadas de forma obrigatória, para projetar a arquitetura. Esse *Feature Set* foi definido para a avaliação, pois equipes ágeis atuam com recursos escassos, sendo o tempo o

mais importante deles (GIARDINO et al., 2016).

O tempo de desenvolvimento e a adaptação aos requisitos dinâmicos têm muita importância para o desenvolvimento ágil de software (GIARDINO et al., 2016; KLOTINS et al., 2019; KLOTINS; UNTERKALMSTEINER; GORSCHEK, 2019). Por isso, os FS relacionados a eles (FS4 e FS5) têm peso igual a $0,35$, maior que o peso dos outros FS. Como o desenvolvimento modular (FS3) é um aspecto mais encontrado do que a tomada de decisões arquiteturais (FS2), o peso de FS3 (0,10) é maior que o de FS2 (0,05). O peso de FS1 (0,15) é maior, pois tem *subfeatures* mais importantes, como a presença de poucos elementos no planejamento inicial (FS1-02), ideal em um contexto de requisitos dinâmicos.

O **nível de importância da *subfeature* (NI)** contempla a relevância da presença de uma *feature* na ferramenta avaliada, para o público-alvo. Os NIs possuem um valor multiplicador, utilizado no cálculo da pontuação de cada candidata. Os níveis são: 4 - Obrigatória, 3 - Altamente Desejável, 2 - Desejável e 1 - Interessante.

4.1.3 Pontuação das abordagens

Para avaliar as abordagens candidatas, cada *subfeature* é julgada segundo uma escala pré-definida e a pontuação é chamada de Julgamento da *Subfeature* (JSF_{FS_X}). Neste estudo, duas escalas foram utilizadas. Para os FS1, FS2, FS3 e FS4, a escala é:

- **Sim –** $JSF = 1$, se a *subfeature* for completamente atendida pela abordagem.
- **Parcialmente –** $JSF = 0,5$, se a *subfeature* não estiver completamente pre-

sente ou for implementada de forma diferente pela candidata.

- **Não –** $JSF = 0$, se a *subfeature* não for atendida.

Para o FS5, a pontuação foi atribuída de forma comparada entre as candidatas, com pontuações de 1 a 6. As abordagens foram ordenadas de forma crescente, segundo a quantidade de tarefas obrigatórias. O primeiro lugar teve pontuação 6, o segundo 5 e assim por diante. Em caso de empate, as candidatas tiveram a mesma avaliação. Dessa forma, a abordagem com menos tarefas teria a melhor pontuação. Quando a abordagem não apresentar as tarefas de forma clara, será contado o número de visões obrigatórias, apresentadas pelos autores da abordagem candidata.

A pontuação máxima (PM) do FS (PM_{FS}) é a soma dos valores máximos das suas *subfeatures*. Por exemplo, o FS1 possui quatro *subfeatures*. A FS1-01 foi classificada como Desejável ($NI_{FS1_{01}} = 2$). A FS01-02 foi considerada Altamente Desejável ($NI_{FS1_{02}} = 3$). As duas *subfeatures* seguintes foram classificadas como Interessantes ($NI_{FS1_{03}} = NI_{FS1_{04}} = 1$). Nesse caso, será $PM_{FS1} = ((2 \times 1) + (3 \times 1) + (1 \times 1) + (1 \times 1)) = 7$.

A pontuação do *Feature Set* x ($\%PFS_x$) e a pontuação geral (PG) das candidatas são calculadas pelas Equações 4.1 e 4.2:

$$\%PFS_{FS_x} = \frac{\sum_{x=1}^{n}(NI_{FS_x} \times JSF_{FS_x})}{PM_{FS_x}} \tag{4.1}$$

$$\%PG = \sum_{x=1}^{n}(\%PFS_{FS_x} \times PI_{FS_x}) \tag{4.2}$$

No final da análise, as abordagens de documentação avaliadas foram ordenadas de forma decrescente, com base na pontuação geral (PG) de cada uma.

As pontuações foram definidas pelo autor deste trabalho, sob supervisão e revisão da orientadora, e armazenadas em planilhas disponibilizadas no Apêndice A.

4.2 Resultados

Nesta seção, os resultados da *Feature Analysis* estão descritos e explicados, com base nos artigos originais que apresentam as abordagens avaliadas.

4.2.1 Siemens' 4 Views

A abordagem Siemens' 4 Views (Tabela 2) apoia a documentação da arquitetura ($JSF_{FS1-01} = 1$), mas não discute sobre quantos elementos devem estar presentes no planejamento e no projeto inicial da arquitetura ($JSF_{FS1-02} = 0$) (SONI; NORD; HOFMEISTER, 1995). Segundo os autores, as quatro visões endereçam preocupações diferentes, pois cada uma contém os resultados das decisões influenciadas por diferentes fatores organizacionais e técnicos e são utilizadas por equipes variadas (arquitetos de software, desenvolvedores, equipe operacional, entre outros) ($JSF_{FS1-03} = 1$). Por fim, a abordagem não fornece nenhum tipo de separação entre a arquitetura corrente e a futura ($JSF_{FS1-04} = 0$), além de não discutir sobre como documentar as decisões arquiteturais ($JSF_{FS2-01} = 0$) ou como uma análise das tomadas de decisões pode ser feita.

Essa abordagem permite a construção de uma arquitetura modular através da decomposição funcional e da disposição em camadas. A primeira captura a forma como o sistema é distribuído dentre os módulos, subsistemas e

unidades abstratas, além das relações entre os componentes, através de interfaces que podem ser exportadas e importadas. A disposição em camadas reflete as decisões de projeto baseadas nas relações de importação e exportação, as restrições criadas, além de reduzir e isolar as dependências internas e externas ($JSF_{FS3-01} = 1$).

Os autores afirmam que a visão de execução é a que tende a ser modificada de forma mais constante, devido à necessidade de comportar a evolução do software e do hardware ($JSF_{FS4-01} = 1$). A abordagem não descreve quais tarefas devem ser feitas para criar as descrições da arquitetura. Nesse caso, contam-se as visões obrigatórias propostas (quatro). Na escala adotada para o FS5, a pontuação para JSF_{FS5-01} foi calculada como descrito abaixo.

$$JSF_{FS5-01} = 5 \therefore \%PFS_{FS5} = \frac{5 * 4}{24} = \frac{20}{24} = 83,33\%$$

Tabela 2 – Resultados da FA para a abordagem Siemens' 4 Views.

			S4V			
Feature Set	**Subfeature ID**	**Peso de Importância (PI)**	**Pontuação Máxima (PM)**	**Julgamento da SF (JSF)**	**Pontuação FS (PFS)**	**% Pontuação FS (%PFS)**
FS1	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 0 1 0	3,00	42,86%
FS2	FS2-01	1	1	0	0,00	0,00%
FS3	FS3-01	2	2	1	2,00	100,00%
FS4	FS4-01	4	4	1	4,00	100,00%
FS5	FS5-01	4	24	5	20,00	83,33%
				Pontuação Geral (PG)		**80,60%**

4.2.2 Kruchten's 4+1

A abordagem 4+1 (Tabela 3) apoia a documentação da arquitetura do sistema ($JSF_{FS1-01} = 1$) baseada em cenários que, de forma cíclica, gera as visões para pequenos conjuntos ($JSF_{FS1-02} = 1$). Em seguida, os artefatos são validados pelos *stakeholders* e o ciclo se repete, com os cenários ainda não tratados (KRUCHTEN, 1995). Cada uma das visões possui propósitos e públicos distintos ($JSF_{FS1-03} = 1$): os engenheiros do sistema utilizam mais as visões Física e de Processos; os usuários finais, clientes e especialistas de dados, a visão Lógica; e os gerentes de projeto e a equipe de implementação enxergam o sistema através da visão de Desenvolvimento. Quanto à separação de instâncias da arquitetura corrente e futura, o autor não descreve qualquer abordagem nesse sentido ($JSF_{FS1-04} = 0$) (KRUCHTEN, 1995). A visão de cenários serve para combinar as outras visões e auxiliar na validação do que foi projetado para determinado cenário. O documento de diretrizes de projeto armazena as principais decisões tomadas, os objetivos arquiteturais, os cenários, os atributos de qualidade, entre outros aspectos do sistema ($JSF_{FS2-01} = 1$) (KRUCHTEN, 1995). A abordagem 4+1 permite a descrição de arquiteturas modulares através da visão de desenvolvimento, que contém a organização modular do sistema ($JSF_{FS3-01} = 1$). Nela, os subsistemas são organizados em uma hierarquia de camadas, cada uma fornecendo uma interface bem definida de comunicação com as camadas superiores. Para se adaptar a mudanças, a abordagem deve ser aplicada de forma iterativa pois, após a criação das primeiras versões das visões, pouco se sabe ainda para validar a arquitetura. A cada iteração, será possível inserir ou remover elementos da arquitetura e atender às mudanças que surgirem por qualquer razão (KRUCHTEN, 1995) ($JSF_{FS4-01} = 1$).

Por fim, as funcionalidades mais importantes do sistema são capturadas na forma de cenários. Nesse sentido, a abordagem descreve quatro passos para construir a arquitetura:

1. Definir de um pequeno conjunto de cenários, baseado nos seus riscos e importâncias.
2. Criar abstrações que atendem aos cenários (classes, subsistemas, etc).
3. Dispor os elementos da arquitetura nas quatro visões principais.
4. Implementar, testar e validar o sistema, detectando falhas que devem ser corrigidas.

Na escala adotada para o FS5, a pontuação para JSF_{FS5-01} foi calculada como descrito abaixo.

$$JSF_{FS5-01} = 5 \therefore \%PFS_{FS5} = \frac{5 * 4}{24} = \frac{20}{24} = 83,33\%$$

Tabela 3 – Resultados da FA para a abordagem 4+1.

4+1						
Feature Set	**Subfeature ID**	**Peso de Importância (PI)**	**Pontuação Máxima (PM)**	**Julgamento da SF (JSF)**	**Pontuação FS (PFS)**	**% Pontuação FS (%PFS)**
FS1	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 1 1 0	6,00	85,71%
FS2	FS2-01	1	1	1	1,00	100,00%
FS3	FS3-01	2	2	1	2,00	100,00%
FS4	FS4-01	4	4	1	4,00	100,00%
FS5	FS5-01	4	24	5	20,00	83,33%
				Pontuação Geral (PG)		**92,02%**

4.2.3 BAPO/CAFCR

A abordagem BAPO/CAFCR (Tabela 4) (AMERICA; ROMMES; OBBINK, 2003) permite a construção da arquitetura do sistema ($JSF_{FS1-01} = 1$) através de cinco visões, usando apenas os cenários principais para construir o planejamento inicial ($JSF_{FS1-02} = 1$). A visão comercial está relacionada com o valor proposto ao cliente do sistema, as suas motivações e necessidades. A visão de aplicação descreve os cenários de uso do sistema, através de fluxos de atividades, requisitos de qualidade e modelos de domínio. A visão funcional descreve as propriedades de um sistema de forma concisa utilizando, por exemplo, uma lista de *features* importantes do sistema ou uma matriz que as mapeie para o custo de implementá-las. A visão conceitual documenta os conceitos essenciais do sistema, com o objetivo de mostrar as partes do sistema e como elas cooperam entre si, além dos padrões de projeto e os princípios que regem o desenvolvimento. Por fim, a visão de realização contém as tecnologias utilizadas para implementar o sistema projetado. Essas visões mais técnicas são voltadas à equipe de desenvolvimento ($JSF_{FS1-03} = 1$).

A abordagem não distingue as arquiteturas corrente e futura ($JSF_{FS1-04} = 0$). A análise de decisões arquiteturais pode ser feita através das cinco visões principais e pelas suas variações, a qualquer momento do projeto da arquitetura ($JSF_{FS2-01} = 1$). Sistemas modulares podem ser descritos através da visão conceitual, com os elementos principais e seus relacionamentos apresentados na decomposição do sistema, que compõe a visão conceitual ($JSF_{FS3-01} = 1$).

As variações apresentadas servem para lidar com os diferentes cenários de uso do sistema, conter os requisitos emergentes e representar futuros (ou novos) impactos na arquitetura. As mudanças na arquitetura devem ser tratadas

como novos cenários e modeladas como os cenários já existentes. ($JSF_{FS4-01} = 1$). Os autores não definem tarefas a ser seguidas para construir a arquitetura (FS5-01). Nesse caso, contam-se as visões obrigatórias apresentadas. Nesse exemplo, são 5 visões (comercial, aplicação, funcional, conceitual, realização). Na escala adotada para avaliar a SF5-01, a abordagem CAFCR teve o JSF_{FS5-01} calculado como descrito abaixo.

$$JSF_{FS5-01} = 4 \therefore \%PFS_{FS5} = \frac{4 * 4}{24} = \frac{16}{24} = 66,67\%$$

Tabela 4 – Resultados da FA para a abordagem BAPO/CAFCR.

BAPO/CAFCR						
Feature Set	**Subfeature ID**	**Peso de Importância (PI)**	**Pontuação Máxima (PM)**	**Julgamento da SF (JSF)**	**Pontuação FS (PFS)**	**% Pontuação FS (%PFS)**
FS1	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 1 1 0	6,00	85,71%
FS2	FS2-01	1	1	1	1,00	100,00%
FS3	FS3-01	2	2	1	2,00	100,00%
FS4	FS4-01	4	4	1	4,00	100,00%
FS5	FS5-01	4	24	4	16,00	66,67%
				Pontuação Geral (PG)		**86,19%**

4.2.4 Attribute Driven Design

A abordagem Attribute Driven Design (ADD) (Tabela 5) auxilia na documentação da arquitetura do sistema ($JSF_{FS1-01} = 1$), com foco nos atributos de qualidade. O valor de $JSF_{FS1-02} = 0,5$, pois todos os requisitos devem ser priorizados pelos *stakeholders*, antes do início projeto da arquitetura. Entretanto, os

autores também afirmam que isso dificilmente será possível e recomendam que o arquiteto trabalhe com os requisitos que tiver à disposição (WOJCIK et al., 2006). Durante o projeto da arquitetura, três visões são geradas com propósitos diferentes, porém sem públicos definidos. Dessa forma, a SF1-03 é parcialmente atendida e, por isso, $JSF_{FS1-03} = 0,5$. A visão de módulos serve para documentar as propriedades estáticas do sistema. A visão de componente-conector diz respeito ao comportamento de execução do sistema. A visão de alocação mapeia os elementos de software para os componentes de hardware. A abordagem não possui uma separação da arquitetura corrente e a futura ($JSF_{FS1-04} = 0$).

A abordagem possui um passo a passo para a construção da arquitetura de software e, a partir da quarta etapa, os autores afirmam que as decisões de projeto começam a ser tomadas ($JSF_{FS2-01} = 1$). Elas envolvem a escolha dos conceitos de projeto do sistema, elementos, recursos alocados, dependências internas e externas e a validação dos atributos de qualidade, comparado ao que foi implementado. O ADD permite a construção de sistemas modulares ($JSF_{FS3-01} = 1$), cuja documentação é feita na visão de módulos, que contém os elementos principais do sistema. A flexibilidade e adaptação a mudanças no projeto são tratadas no final de cada tarefa. Deve haver uma análise para verificar se o sistema projetado está de acordo com o que foi planejado e, se necessário, alterar e refinar a arquitetura gerada ($JSF_{FS4-01} = 1$).

Sobre o custo da abordagem ADD, ela possui 8 atividades, que devem ser seguidas a cada iteração e estão descritas na Seção 3.4. Por ter sido a abordagem com mais tarefas, na escala adotada para o SF5-01, ele obteve a pontuação mais baixa.

$$JSF_{FS5-01} = 2 \therefore \%PFS_{FS5} = \frac{2 * 4}{24} = \frac{8}{24} = 33,33\%$$

Tabela 5 – Resultados da FA para a abordagem ADD.

ADD						
Feature Set	**Subfeature ID**	**Peso de Importância (PI)**	**Pontuação Máxima (PM)**	**Julgamento da SF (JSF)**	**Pontuação FS (PFS)**	**% Pontuação FS (%PFS)**
FS1	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 0,5 0,5 0	4,00	57,14%
FS2	FS2-01	1	1	1	1,00	100,00%
FS3	FS3-01	2	2	1	2,00	100,00%
FS4	FS4-01	4	4	1	4,00	100,00%
FS5	FS5-01	4	24	2	8,00	33,33%
				Pontuação Geral (PG)		**70,24%**

4.2.5 C3A Agile Architecture

A C3A (Tabela 6) serve para construir a arquitetura ($JSF_{FS1-01} = 1$) do sistema em contextos ágeis, com uma quantidade mínima de elementos ($JSF_{FS1-02} = 1$). Existem duas visões principais: a de referência (AR), que mostra uma visão geral do sistema, com os módulos principais e os seus componentes; e a de implementação (AI), descrevendo as mudanças na arquitetura para as novas e menores versões, além de mais detalhes sobre como desenvolver o sistema. A arquitetura de referência contém as camadas da arquitetura, com os elementos níveis 0 e 1, os visionários e, de forma opcional, os elementos nível 2. A camada de negócios é utilizada por analistas de negócio para registrar a integração do sistema com sistemas externos. As interfaces entre sistemas e com os usuários estão descritas na camada funcional, usada por arquitetos de software e gerentes de projeto. A camada de arquitetura do sistema, utilizada pelos arquitetos de

software, compreende as partes internas do sistema e serve para implementar as funcionalidades descritas na camada anterior ($JSF_{FS1-03} = 1$). A abordagem C3A possui uma separação clara entre a arquitetura corrente ($JSF_{FS1-04} = 1$), representada pela AR e as versões futuras, projetadas na AI.

Para registrar as decisões arquiteturais, deve existir um contrato para cada elemento dos níveis 0 e 1 da arquitetura, com informações relevantes para auxiliar na análise das decisões: Nome do elemento, Responsável, Responsabilidades, Dependências, Implementação, API, Estrutura de dados, Tecnologias, entre outros ($JSF_{FS2-01} = 1$). A construção de sistemas modulares é permitida através da AR e da AI, que contêm os elementos níveis 0 e 1, chamados de módulos e componentes, respectivamente. O sistema é descrito nos seus módulos principais e decompostos em componentes menores, que explicam melhor o funcionamento do sistema ($JSF_{FS3-01} = 1$).

A abordagem possui componentes visionários ou estratégicos representando novas funcionalidades ou alterações, que podem ou não ser adicionadas ao sistema. Sua presença ajuda a alinhar a posição e o valor das novas *features* com a arquitetura geral do sistema, além do provável impacto causado. Esses componentes só são adicionados à AR quando são considerados maduros pela equipe ($JSF_{FS4-01} = 1$).

O passo a passo descrito para construir a arquitetura possui 6 etapas (HADAR; SILBERMAN, 2008):

1. Coletar documentos arquiteturais, publicações técnicas, manuais e conhecimento tácito, além de capturar a primeira arquitetura em uma arquitetura de referência, com a definição dos componentes nível 0.

2. Validar o projeto criado até o momento, detalhando o status dos módu-

los da próxima arquitetura de referência mantendo a AR corrente, para prevenir que ela seja impactada por componentes incertos.

3. Para cada módulo nível 0, definir os componentes nível 1 e implementar os pequenos lançamentos das AIs.

4. Mapear ou prover a análise das lacunas entre os próximos lançamentos e a arquitetura atual, ajustar a AI sem impactar na AR. Se não for possível, mitigar o impacto na AR.

5. Modificar os planejamentos de lançamentos menores, para se adequar à arquitetura atual e, então, executar e implementar a AI.

6. Estimar os efeitos da AI no próximo lançamento da AR, propagando as lacunas encontradas para a última AI. Se for necessário, voltar ao passo I, senão, voltar ao passo IV.

No ranking adotado para pontuar o FS5-01 de cada candidata, a C3A teve o JSF_{FS5-01} calculado como descrito abaixo.

$$JSF_{FS5-01} = 3 \therefore \%PFS_{FS5} = \frac{3 * 4}{24} = \frac{12}{24} = 50,00\%$$

Tabela 6 – Resultados da FA em cada subfeature para a abordagem C3A.

C3A						
Feature Set	**Subfeature ID**	**Peso de Importância (PI)**	**Pontuação Máxima (PM)**	**Julgamento da SF (JSF)**	**Pontuação FS (PFS)**	**% Pontuação FS (%PFS)**
FS1	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 1 1 1	7,00	100,00%
FS2	FS2-01	1	1	1	1,00	100,00%
FS3	FS3-01	2	2	1	2,00	100,00%
FS4	FS4-01	4	4	1	4,00	100,00%
FS5	FS5-01	4	24	3	12,00	50,00%
				Pontuação Geral (PG)		**82,50%**

4.2.6 C4 Model

A abordagem C4 (Tabela 7) permite a documentação da arquitetura de sistemas através de quatro visões principais ($JSF_{FS1-01} = 1$). Com um conjunto pequeno de informações, é possível construir as visões de contexto e contêiner, gerando um desenho inicial e resumido do sistema com os atores e componentes principais ($JSF_{FS1-02} = 1$).

As visões possuem públicos e propósitos distintos ($JSF_{FS1-03} = 1$): a visão de contexto envolve apenas o sistema em questão, os sistemas externos e outros atores que interagem com ele, utilizada por pessoas externas ou integrantes da equipe de desenvolvimento; a visão de contêineres contém os subsistemas do sistema maior projetado e os atores que interagem com eles, utilizada pela equipe de desenvolvimento, operacional e arquitetos de software; as visões de componentes e de código são mais técnicas e descrevem os elementos

que integram os contêineres, além de descrições em linhas de código ou diagramas UML com a implementação do sistema, voltadas aos desenvolvedores e engenheiros de *software*.

A abordagem não faz distinção entre a arquitetura corrente e arquiteturas futuras ($JSF_{FS1-04} = 0$) e não propõe nenhum método de análise de decisões arquiteturais ($JSF_{FS2-01} = 0$).

A C4 permite a construção de arquiteturas modulares com a visão de contêineres ($JSF_{FS3-01} = 1$), descrevendo os módulos (subsistemas) do sistema, suas relações entre si e com os atores. Da mesma forma, os contêineres são decompostos em módulos menores chamados de componentes. O autor afirma que cada diagrama será alterado em velocidades diferentes ($JSF_{FS4-01} = 1$), sendo a de componentes a visão que sofrerá mais mudanças, com a adição ou remoção desses elementos pela equipe.

Não há tarefas definidas para construir a arquitetura. Nesse caso, contam-se as visões obrigatórias. A abordagem possui quatro visões, mas o autor afirma que somente as duas primeiras são obrigatórias e que serão utilizadas na maior parte das empresas e dos sistemas (BROWN, 2017). Na escala utilizada para avaliar o FS5-01, a C4 teve o JSF_{FS5-01} calculado como descrito abaixo.

$$JSF_{FS5-01} = 6 \therefore \%PFS_{FS5} = \frac{6 * 4}{24} = \frac{24}{24} = 100\%$$

Tabela 7 – Resultados da FA para a abordagem C4.

C4						
Feature Set	**Subfeature ID**	**Peso de Importância (PI)**	**Pontuação Máxima (PM)**	**Julgamento da SF (JSF)**	**Pontuação FS (PFS)**	**% Pontuação FS (%PFS)**
FS1	FS1-01 FS1-02 FS1-03 FS1-04	2 3 1 1	7	1 1 1 0	6,00	85,71%
FS2	FS2-01	1	1	0	0,00	0,00%
FS3	FS3-01	2	2	1	2,00	100,00%
FS4	FS4-01	4	4	1	4,00	100,00%
FS5	FS5-01	4	24	6	24,00	100,00%
				Pontuação Geral (PG)		**92,86%**

4.3 Discussões

Esta seção apresenta a discussão dos resultados na subseção 4.3.1, com o destaque para os resultados gerais em cada *Feature Set*. Além disso, as limitações do estudo também são discutidas na subseção 4.3.2.

4.3.1 Resultados

Giardino et al. (2016) relatam que o desenvolvimento acelerado faz com que atividades de planejamento e qualidade sejam postergadas em contextos ágeis de desenvolvimento. Elas aceitam o acúmulo de débito técnico, incluindo decisões arquiteturais, em favor do lançamento de novas versões. Entretanto, quando a empresa progride e o produto fica complexo, o foco passa a ser o pagamento do débito acumulado em vez do lançamento de novas *features*, com grande carga

de retrabalho para corrigir partes do código.

Esses fatores são os que mais impactam negativamente no *time-to-market* da empresa e Klotins et al. (2019) os reforçam, mostrando que decisões arquiteturais ruins fazem parte do débito técnico acumulado, cujo pico ocorre na fase de crescimento, afetando o *time-to-market*.

Nesse sentido, uma abordagem que permita a construção da arquitetura, em um contexto de poucas informações, requisitos dinâmicos e incertos, sob a pressão do *time-to-market* pode ser considerada uma boa solução.

Tabela 8 – Classificação geral das abordagens avaliadas.

Posição	1^a	2^a	3^a	4^a	5^a	6^a
Modelo	C4	4+1	CAFCR	C3A	S4V	ADD
Pontuação Geral	92,86%	92,02%	86,19%	82,50%	80,60%	70,24%

Como visto na Tabela 8, as abordagens melhor avaliadas que, possivelmente, se encaixem na definição acima são as abordagens C4 ($PG = 92,86\%$) e 4+1 ($PG = 92,02\%$), embora a diferença percentual entre elas seja muito pequena e a ordem entre elas pode ser alterada, caso outras análises sejam realizadas.

De forma geral, a maior parte das abordagens permite a **descrição da arquitetura com poucos elementos** (SF1-02). Entretanto, a ADD é a única que solicita a elicitação de todos os requisitos do sistema, antes do projeto da arquitetura, embora os autores afirmem que esse contexto é quase inalcançável (WOJCIK et al., 2006).

Quanto à **separação de propósitos e públicos** (SF1-03) para as visões desenhadas, a maioria das abordagens explica quais partes interessadas se beneficiam de cada uma das visões e os elementos do sistema presentes em cada uma delas. As abordagens CAFCR, C4 e 4+1 se destacam ao descrever tanto visões

abstratas quanto mais específicas. A visão Comercial (AMERICA; ROMMES; OBBINK, 2003) e a de Contexto (BROWN, 2017) são mais voltadas ao público geral, com poucos elementos internos e técnicos, descrevendo as motivações e necessidades dos usuários, além dos principais elementos do sistema e as partes que interagem com ele. Na abordagem 4+1 (KRUCHTEN, 1995), as visões detalham diferentes aspectos do sistema que, na visão de cenários, são integrados para descrever e validar como o sistema atenderá aos diferentes cenários de uso do sistema. Sobre a **separação de instâncias corrente e futuras da arquitetura** (SF1-04), a abordagem C3A (HADAR; SILBERMAN, 2008) foi a única a atender a esse quesito.

As abordagens ADD (AMERICA; ROMMES; OBBINK, 2003), C3A Agile Architecture (HADAR; SILBERMAN, 2008) e 4+1 (KRUCHTEN, 1995) se destacam no quesito de **análise de decisões arquiteturais** (SF2-01). A primeira (AMERICA; ROMMES; OBBINK, 2003) propõe a validação do que foi projetado, perante as necessidades do sistema e das partes interessadas, em metade das tarefas necessárias para construir a arquitetura. A C3A (HADAR; SILBERMAN, 2008) sugere a criação de contratos para os elementos do sistema, com informações relevantes dos módulos e componentes. A 4+1 (KRUCHTEN, 1995) também orienta a criação de um documento chamado Diretrizes de Projeto, com as principais decisões tomadas durante o projeto.

Todas as abordagens avaliadas fornecem meios de **construir arquiteturas modulares** (SF3-01) para os sistemas. A descrição dos elementos pode ser feita através de contratos, diagramas com caixas e setas genéricas ou na linguagem UML. O destaque é dado às abordagens Siemens' 4 Views (SONI; NORD; HOFMEISTER, 1995) e 4+1 (KRUCHTEN, 1995) que, em 1995, já possuíam a

descrição da arquitetura de módulos de sistemas. A primeira abordagem permite essa descrição através da decomposição funcional e da organização em camadas, enquanto que a segunda abordagem organiza os módulos na visão lógica da arquitetura. Sobre a flexibilidade e a capacidade de adaptar a arquitetura a possíveis mudanças, todas as abordagens se mostraram capazes de serem modificadas, devido a mudanças provocadas por qualquer razão.

As abordagens C3A e CAFCR apresentam formas semelhantes de **adaptação da arquitetura do sistema** (SF4-01). A primeira possui uma arquitetura de implementação contendo os componentes visionários, cuja função é descrever as novas funcionalidades ou alterações, avaliadas quanto ao impacto na arquitetura até atingirem a maturidade para inclusão na arquitetura de referência. A CAFCR (AMERICA; ROMMES; OBBINK, 2003) propõe os cenários de uso para armazenar os requisitos emergentes e representar futuros ou novos impactos na arquitetura.

Em relação ao **custo de aplicação** das abordagens (SF5-01), as abordagens consideradas mais custosas são a ADD, com 8 etapas principais (WOJCIK et al., 2006) e a C3A (HADAR; SILBERMAN, 2008), que possui 6 tarefas. As tarefas envolvem, por exemplo, a coleta de informações sobre o sistema e as suas funcionalidades, a definição das primeiras versões da arquitetura, o detalhamento e a combinação dos módulos que estão ou serão planejados para as próximas instâncias da arquitetura com a versão corrente do sistema.

As abordagem C4 foi avaliada com o menor custo por ter menos visões obrigatórias (duas, segundo Brown (2017)). Mesmo assim, a sua pontuação geral ficou muito próxima da abordagem 4+1 (KRUCHTEN, 1995), com quatro tarefas que devem ser realizadas.

4.3.2 Limitações

As limitações deste estudo se encontram na escolha das *features* avaliadas, na definição dos pesos e na avaliação do custo das abordagens candidatas. O risco gerado por essas limitações é que os pesos utilizados podem não refletir a real importância dos fatores e os resultados não podem ser generalizados. Para minimizar esse risco, as *features* e os pesos utilizados foram extraídos das características relatadas na literatura, por trabalhos realizados com entrevistas e *surveys*, em contextos semelhantes ao estudado nesta pesquisa.

Além disso, o custo de implementar uma abordagem pode estar relacionado à complexidade de cada tarefa. Entretanto, a sua medição pode diferir, dependendo de quem mensurar a tarefa. Por isso, para avaliar esse aspecto de forma mais objetiva, apenas a quantidade de tarefas obrigatórias foi utilizada. Ainda assim, não foi possível medir a quantidade de tarefas obrigatórias de todas as candidatas (FS5), devido à ausência dessa informação nos artigos originais. Para contornar esse problema, utilizou-se a contagem de visões obrigatórias da arquitetura, visto que são os artefatos gerados e o local em que o esforço da equipe seria alocado.

4.4 Conclusões

Esse estudo avaliou seis abordagens propostas para a construção da arquitetura de sistemas de *software*. Ele foi realizado através de uma *Feature Analysis*, com a definição de um conjunto de *features* que as abordagens devem possuir, segundo o contexto ágil de desenvolvimento de software descrito na literatura.

As contribuições deste trabalho são mostrar se as abordagens avaliadas

possuem características que apoiam a adoção no ambiente estudado e fornecer alternativas para estudos futuros.

As abordagens C4 (BROWN, 2017) e 4+1 (KRUCHTEN, 1995) obtiveram as maiores pontuações e foram utilizadas como instrumento no estudo *in vitro*, descrito no Capítulo 5.

5

Estudo Experimental In Vitro

A primeira avaliação das abordagens que melhor pontuaram na Feature Analysis, descrita no Capítulo 4, foi realizada com o objetivo de entender a experiência durante a sua utilização em um ambiente acadêmico, com participantes aprendendo sobre o processo de desenvolvimento de software. As questões de pesquisa que guiaram a execução e a análise dos achados são:

1. Quais são as facilidades e dificuldades ao utilizar as abordagens 4+1 e C4 Model?
2. Quais são as utilidades percebidas para as abordagens 4+1 e C4 Model?
3. Quando e de que forma as abordagens 4+1 e C4 Model seriam adotadas pelos participantes?

5.1 Metodologia

Nesse sentido, foi realizado um estudo exploratório com a participação de alunos da disciplina de Análise e Projeto de Sistemas (sexto período), do curso de

bacharel em Ciência da Computação, da Universidade Federal do Amazonas. Nela, os alunos aprendem tópicos sobre projetos de sistemas, através de modelos UML e conceitos de arquitetura de software. Assim, durante o estudo, o desenho da arquitetura do sistema foi feito sobre os sistemas já elicitados e com os casos de uso definidos pelas equipes, durante o andamento da disciplina.

5.1.1 Demografia dos Participantes

A turma foi dividida em equipes de quatro a seis integrantes e cada uma deveria desenhar a arquitetura do sistema com as duas abordagens, utilizando todas as visões e cada aluno deveria, obrigatoriamente, desenhar pelo menos uma visão de cada abordagem. A ordem de utilização das abordagens foi definida por cada equipe. Antes da realização do estudo, os participantes tiveram aulas sobre arquitetura de software e receberam treinamento sobre as duas abordagens de documentação de arquitetura.

A participação dos alunos neste estudo ocorreu através da realização de um trabalho prático. Além de entregar os desenhos no prazo estipulado, os alunos deveriam preencher um formulário com a sua experiência, durante a construção das visões e de acordo com os seguintes aspectos: Facilidade, Dificuldade, Utilidade e Adoção, disponibilizado como material suplementar [1]. Além disso, um Termo de Consentimento Livre e Esclarecido foi entregue a todos os alunos e somente os relatos dos alunos que o assinaram foram utilizados neste estudo, resultando em um total de 20 participantes.

[1] Material Complementar – Estudo in Vitro: https://figshare.com/s/5a858f2b1c70d0803b25

5.1.2 Compilação das Respostas

As respostas obtidas foram extraídas do formulário preenchido e colocadas em um documento separado para análise, com identificadores numerados para cada participante e mantido apenas no computador do pesquisador, tornando-os anônimos. A análise qualitativa das respostas foi feita através da análise temática (CRUZES; DYBA, 2011) dos relatos dos participantes, com a criação de códigos fundamentados nas citações; a construção das relações entre eles (está associado com, é causa de, contradiz, é solução para) e a organização dos códigos de acordo com os aspectos avaliados (facilidade, dificuldade, utilidade e adoção). Esse processo foi realizado com o apoio da ferramenta Atlas.TI 8 [2] e com a contribuição de duas pesquisadoras da área de engenharia de software.

5.2 Resultados 4+1

Nesta seção, os resultados do estudo estão descritos e justificados, através dos códigos extraídos após a análise dos relatos dos participantes sobre a utilização da abordagem 4+1 (KRUCHTEN, 1995). As divisões das visões utilizadas pelos participantes para cada abordagem estão descritas nas tabelas 9 e 10.

5.2.1 Visão de Cenários

Alguns participantes relataram que a visão é fácil de construir, por já ter sido construído anteriormente e que a visão se resume ao diagrama de casos de uso UML, como pode ser visto nos relatos dos participantes P18, P8 e P10, respecti-

[2] https://atlasti.com/pt

Tabela 9 – Divisão dos participantes entre as visões da abordagem 4+1 utilizadas no estudo.

4+1				
Cenários	Lógica	Processos	Desenvolvimento	Física
P18	P18	P12	P7	P19
P8	P9	P20	P17	P11
P10	P14	P4	P16	P15
P13	P1		P5	P5
P3			P2	

Tabela 10 – Divisão dos participantes entre as visões da abordagem C4 utilizadas no estudo.

C4			
Contexto	Contêineres	Componentes	Código
P18	P17	P7	P19
P14	P8	P20	P11
P10	P5	P9	P12
	P1	P13	P2
		P15	
		P16	
		P6	
		P4	
		P3	

vamente: *"Por ter sido uma reciclagem do diagrama de casos de uso do último trabalho, a visão foi extremamente fácil de fazer."*; *"Como já tínhamos feito diagrama de casos de uso, foi fácil definir os atores e as ações de cada um"* e *"Não houve [dificuldades], já que [a visão] segue a mesma ideia dos casos de uso."*.

Como contrapontos, estão as dificuldades em encontrar inconsistências no diagrama, delimitar os atores do sistema e as suas ações. Isso pode ser visto nos seguintes relatos: *"Encontrar inconsistências no diagrama, e por conta disso teve que ser remodelado várias vezes"* (P8), *"...No começo foi um pouco difícil separar o ator*

usuário dos outros" (P13) e *"Por vezes é comum querer indicar como autor um banco de dados ou o sistema, porém refletir que autor está ativando determinada ação pode levar algum tempo mesmo que a resposta seja simples"* (P3).

Além disso, os participantes P8 e P13 destacaram a extensão e a compreensão dessa visão quando todos os requisitos são levados em conta, visto pelos relatos *"Mostrar todos os requisitos deixou o diagrama muito extenso"* (P13) e *"Por ser UML, por mais que acredito que programadores, analistas de sistema, engenheiros de software, etc possam entender, acredito que não está ajudando na visualização do sistema por outros setores do projeto, como designers. O que deveria ser feito ao meu ver, seria de alguma forma "diminuir"a quantidade de coisas que aparecem."*.

Sobre a utilidade dessa visão, os participantes que a construíram no experimento tiveram opiniões divididas. Os participantes P18, P10 afirmaram que não consideram a visão de cenários útil, como pode ser visto nos relatos a seguir: *"Como a única coisa que essa visão fez foi ilustrar as relações de herança e requisitos, ela não parece ser particularmente útil"* (P18), *"Acho pouco útil, pois outras visões acabam detalhando o fluxo mais completo que usam dos requisitos e estruturas do sistema."* (P10). Em contrapartida, outros participantes a consideraram útil pois *"a visão ajuda a ver a necessidade de algumas funcionalidades e também a excluir algumas que se mostram desnecessárias. Também é boa para determinar o MVP do sistema"* (P13), *"No geral, esta visão é útil. Acredito que não vale a pena fazê-la se tomar muito tempo ou recurso, pois é uma visão bem rasa, no entanto, pode ser uma boa ideia caso seja algo rápido de ser feito"* (P3) e *"Essa visão é totalmente útil para o sistema, visto que todas as ações dentro do sistemas podem ser moldadas de acordo com ele"* (P8). A Figura 16 mostra a rede construída com os códigos obtidos através dos relatos dos participantes sobre a utilidade da visão de cenários.

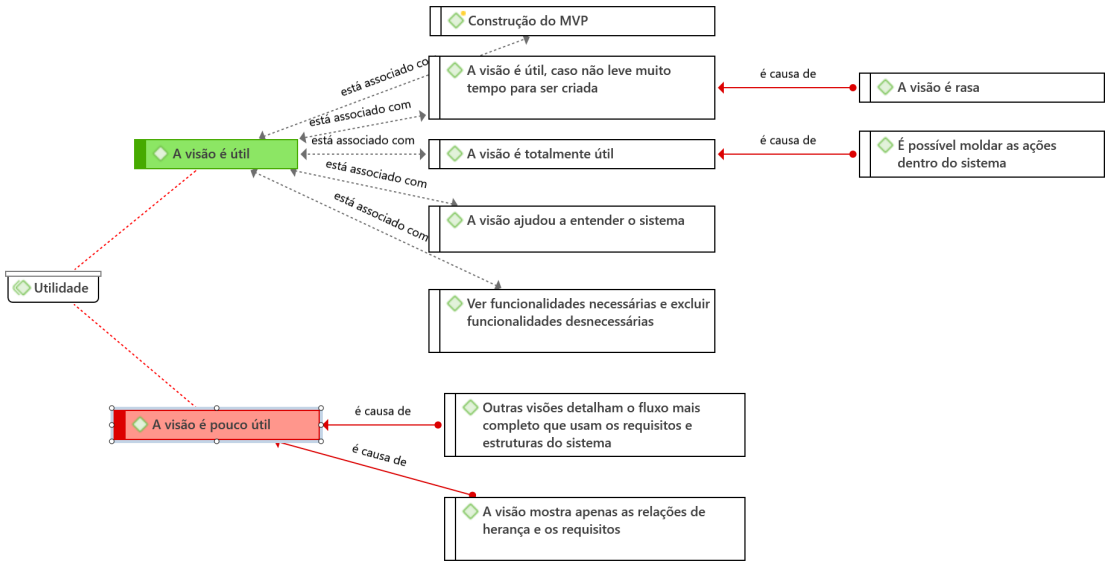


Figura 16 – Utilidade da visão de cenários de acordo com o relato dos participantes que a utilizaram durante o estudo.

Por fim, na perspectiva dos participantes, essa visão seria adotada apenas *"... para o MVP, que facilitaria a inclusão ou exclusão de requisitos para este"* (P13) e *"se fosse para um sistema mais simples, utilizaria"* (P3). Os outros participantes afirmaram que não a utilizariam, como visto nos relatos a seguir: *"Acho pouco útil, pois outras visões acabam detalhando o fluxo mais completo que usam dos requisitos e estruturas do sistema."* (P10), *"Jamais adotaria para representar arquitetura do sistema, pois acredito ser puramente UML, e que deveria existir a fase de UML e a fase de arquitetura de sistema"* (P8) e *"Como a única coisa que essa visão fez foi ilustrar as relações de herança e requisitos, ela não parece ser particularmente útil"* (P18). A Figura 17 mostra os códigos obtidos sobre a adoção da visão de cenários pelos participantes do estudo.

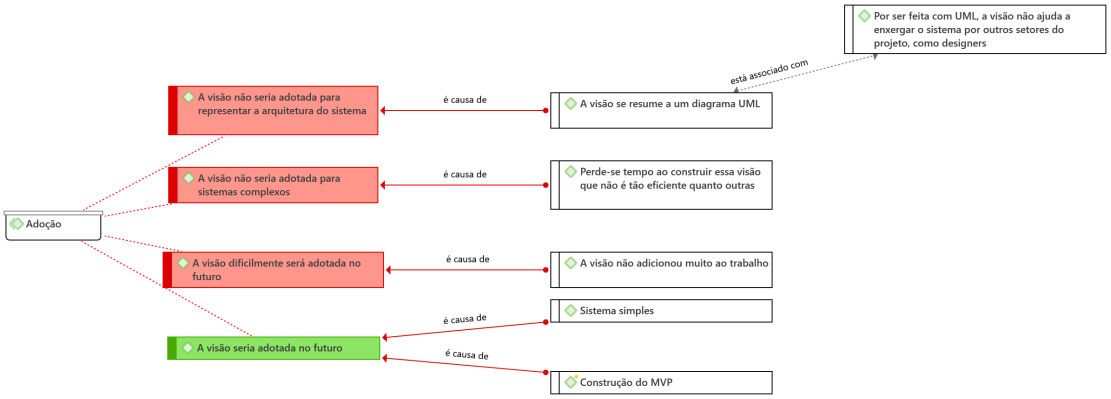


Figura 17 – Adoção da visão de cenários, segundo os participantes.

5.2.2 Visão Lógica

De forma semelhante à visão de cenários, por utilizar a UML e o diagrama de classes, foi possível reaproveitar partes de diagramas construídos anteriormente ao estudo. Isso pode ser visto nos relatos a seguir: *"Como a visão lógica faz uso do UML e classes, foi possível reutilizar partes do diagrama de classes do trabalho anterior"* (P18), *"...foi uma visão fácil de construir, dado a similaridade com o diagrama de classes que já havia sido anteriormente feito no projeto"* (P9) e *"Foi fácil concluir a visão lógica, pois já havia uma versão do Diagrama de Classes do sistema feita de um trabalho anterior"* (P14). Outra facilidade relatada foi *"...separar as classes..."* (P1) e isso ajudou a identificar o que é necessário para implementar no lançamento do MVP do sistema.

Mesmo com o aproveitamento de diagramas construídos anteriormente, os participantes afirmaram ter dificuldades com a construção da visão *"...devido à sua complexidade"* (P18). Além disso, de acordo com o participante P9, *"a princípio, foi um pouco difícil de entender o desenvolvimento dessa visão"* e, segundo o participante P14: *"foi difícil para identificar possíveis correções e mudanças nele, de-*

vido a não termos acesso a correção do trabalho anterior e também por conta de já estar habituada com aquela versão do diagrama". Por fim, o participante P1 afirmou ter sido*"difícil de visualizar todos os métodos e entradas que cada componente precisa e qual a relação entre eles"*.

Ainda assim, a visão lógica foi considerada útil, por quase todos os participantes, para entender o funcionamento (P18, P14) do sistema e o auxílio à equipe no desenvolvimento do código do sistema (P1). Apenas o participante P9 não a considerou útil *"...por sua similaridade com o diagrama que a equipe havia realizado antes. Como se trata de um sistema mais simples também, talvez não fosse preciso."*

Por fim, a visão lógica seria adotada em sistemas maiores (P9), devido aos motivos descritos acima (P14) e por ser *"a forma mais perto (sic) de identificar o que precisa ser implementado"* (P1). O participante P18 afirma ser difícil dizer se adotaria a visão lógica pois *"como a visão lógica existe em conjunto com as outras visões do 4+1, minha adoção dependeria da praticidade das outras visões que eu não fiz..."*. A Figura 18 mostra os códigos obtidos durante a análise dos relatos, relacionados à utilidade e adoção da visão lógica pelos participantes.

5.2.3 Visão de Desenvolvimento

Com relação à visão de desenvolvimento, os participantes relataram ter facilidade *"...em enxergar as responsabilidades de cada parte do sistema..."* (P7) e na *"...organização do que irá aparecer no front de cada app"* (P2). Além disso, é fácil construir essa visão se o esquema do banco de dados já estiver definido (P17) ou as decisões necessárias para a sua construção já tenham sido tomadas anteriormente

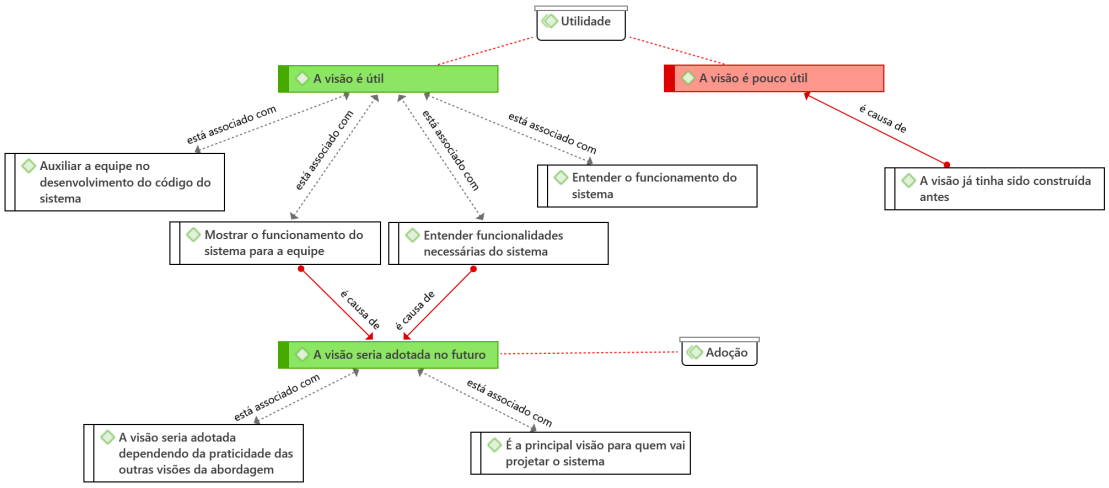


Figura 18 – Utilidade e adoção da visão lógica de acordo com os relatos dos participantes do estudo.

(P5).

Entretanto, as dificuldades enfrentadas foram enxergar os relacionamentos entre as partes do sistema, definir quais delas devem ser modularizadas, lidar com a poluição visual, construir essa visão sem o código implementado e utilizar as ferramentas de modelagem UML. Essas dificuldades podem ser vistas nos relatos a seguir: *"Enxergar detalhadamente os relacionamentos entre todas as partes do sistema e quais partes modularizar e utilizar uma mesma interface e quais partes devem ser componentes separados* (P7), *"Se as classes do front-end não estiverem bem definidas, fica muita poluição visual na tela e isso dificulta a montagem"* (P17), *"Creio que é um pouco difícil e sem propósito pensar em um código imaginário para colocar no diagrama."* (P5) e *"Foi um pouco complicado associar as ferramentas do UML à primeira vista"* (P2).

A visão de desenvolvimento foi considerada útil pelos participantes, o que pode ser visto nos seguintes relatos: *"conseguimos enxergar os componentes e*

suas interações, sem necessidade de pegar em código ou ler longos textos para visualizar como o sistema se comporta" (P7). Além disso, ela também *"Dá (sic) uma visão geral sobre o sistema"* (P17), ajuda a *"visualizar os fluxos de informações e planejar a melhor forma de modelá-los de acordo com as classes disponíveis"* (P16) e *"organizar como será feito o desenvolvimento da parte de código do sistema"* (P2). Todavia, o participante P5 afirma que ela é útil *"se [o diagrama] for feito automaticamente pela IDE a cada alteração de código. Se for feito antes do código, ele é inútil* (P5). Segundo os participantes, a visão de desenvolvimento seria adotada para sistemas de médio porte ou maior (P2) pois *"necessitam de grandes interações, sistemas que se comunicam com APIs externas, ou até mesmo sistemas com quantidades grandes de tabelas e entidades"* (P7). Além disso, ela seria usada para resumir o sistema, para explicá-lo a outras pessoas (P16 e P17), sendo gerada, preferencialmente, de forma automática (P5). As Figuras 19 e 20 mostram as redes de códigos criadas com os relatos dos participantes sobre a utilidade e a adoção da visão de desenvolvimento.

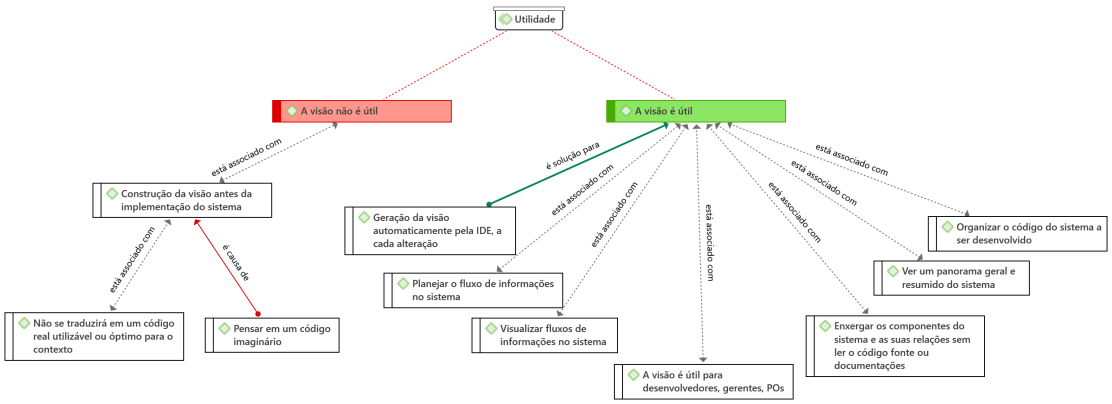


Figura 19 – Utilidade da visão de desenvolvimento, segundo o relato dos participantes que utilizaram essa visão durante o estudo.

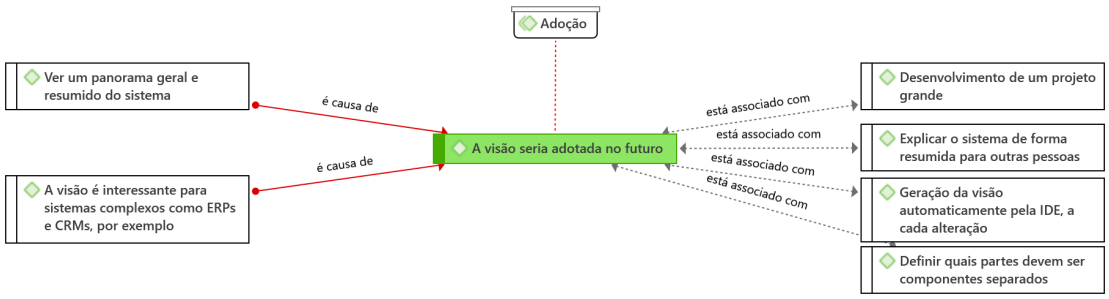


Figura 20 – Adoção da visão de desenvolvimento, de acordo com o relato dos participantes que utilizaram essa visão durante o estudo.

5.2.4 Visão de Processos

Com relação à visão de processos, todos os participantes que a construíram utilizaram o diagrama de atividades (UML) e nenhum deles relatou ter dificuldades em sua construção. Como facilidades relatadas, o participante P12 afirma que *"Foi fácil definir as entidades que estariam envolvidas no processo e qual seria a sequência das atividades"*, o participante P20 disse *"Por ser uma visão que aborda uma dinâmica, construir ela acabou sendo mais fácil, principalmente, por já conhecer os casos de uso, diagramas de estado, sequência e atividade, o que acabou facilitando o entendimento dessa visão"* e P4 afirma que *"A facilidade encontrada foi a visão ser simples, não necessita pensar muito ao fazê-la"*.

Os participantes também foram unânimes em afirmar que a visão de processos é útil, pois ela mostra as entidades envolvidas nas atividades e as etapas realizadas para executar uma atividade. Isso pode ser visto nos relatos a seguir: *"...mostra quem são as entidades envolvidas em determinada funcionalidade...Além disso, é útil em mostrar se determinada funcionalidade está com uma sequência de passos muito longa, nos dando a oportunidade de conseguir melhorar e torná-lo mais fácil e mais rápido para o usuário"* (P12), *"Facilita a ver a interação*

usuário-sistema de maneira rápida, porém minimamente detalhada" e *"...a visualização dos processos do sistema faz com que fique mais claro as etapas a serem seguidas, principalmente por quem não entende muito a linguagem técnica"*.

Por fim, apenas um dos participantes afirma que não adotaria a visão de processos, contradizendo o que falou sobre a sua utilidade, pois para o P20: *"Não adotaria por não parecer ter tanta utilidade para entender a arquitetura do sistema"*. Por outro lado, os participantes P4 e P12 adotariam essa visão, pois *"ajuda a enxergar a usabilidade da aplicação e fácil de fazer"* (P4) e *"Se o protótipo do sistema ainda não estiver pronto, acredito que essa visão facilitaria bastante"* (P12). Os códigos obtidos na análise ligados à utilidade e adoção da visão de processos estão dispostos na Figura 21.

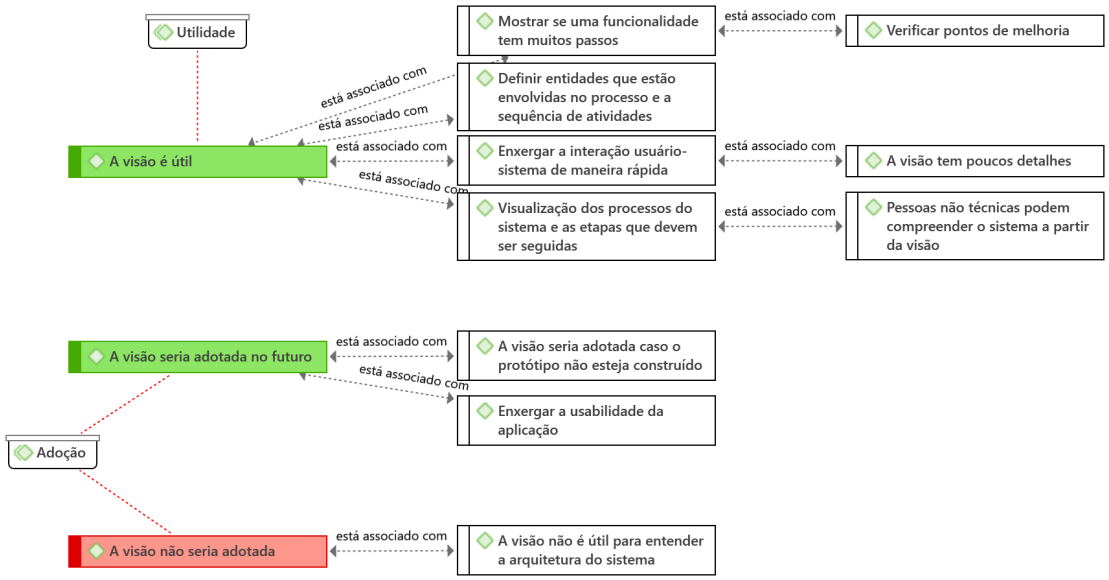


Figura 21 – Utilidade e adoção da visão de processos, segundo os participantes.

5.2.5 Visão Física

Sobre a visão física, os participantes que a utilizaram afirmaram que ela é uma visão simples e fácil de criar. Isso pode ser visto pelos relatos a seguir: *"É muito simples"* (P19), *"a ideia dessa visão foi fácil de criar, pois foi um processo natural vindo da Visão de Processos, como uma evolução do já havia sido pensado, considerando os aspectos mais operacionais"* (P11), *"O Diagrama é bem simples de ser desenhado"* (P15) e *"A sintaxe do diagrama é fácil. Ele também é fácil de interpretar."* (P5).

Entre as dificuldades encontradas, estão a avaliação das alternativas disponíveis no mercado para implantação do sistema, a falta de experiência nesse tema e o nível maior de detalhes que estão presentes na visão. O participante P11 afirma: *"O primeiro ponto é avaliar as alternativas de mercado para o processo de implantação, pois diversas arquiteturas são usadas e com uma gama grande de ferramentas, então é complicado pensar em ferramentas que sirvam ao propósito ao mesmo tempo, em que não sejam exageradas. O segundo ponto é a falta de familiaridade da minha parte com esse aspecto, visto que estou mais acostumado com desenvolver para frontend"*.

Com relação à utilidade, a maioria dos participantes que a utilizaram consideram a visão física útil, pois *"a partir dela é possível fazer uma checagem na visão de desenvolvimento, verificar pontos de melhoria na arquitetura do sistema, além de permitir saber sobre custo de uso das ferramentas"* (P11), *"É bem útil para visualização de alguém menos familiarizado com alguns conceitos de arquitetura de software, pois separa bem cada componente"* (P15) e *"apresenta os componentes físicos e serviços do sistema em uma forma fácil de digerir para pessoas técnicas...ao mesmo tempo não é muito difícil de criar e manter"* (P5). Na Figura 22, é possível ver os códigos relacionados à utilidade da visão física, segundo o relato dos participantes.

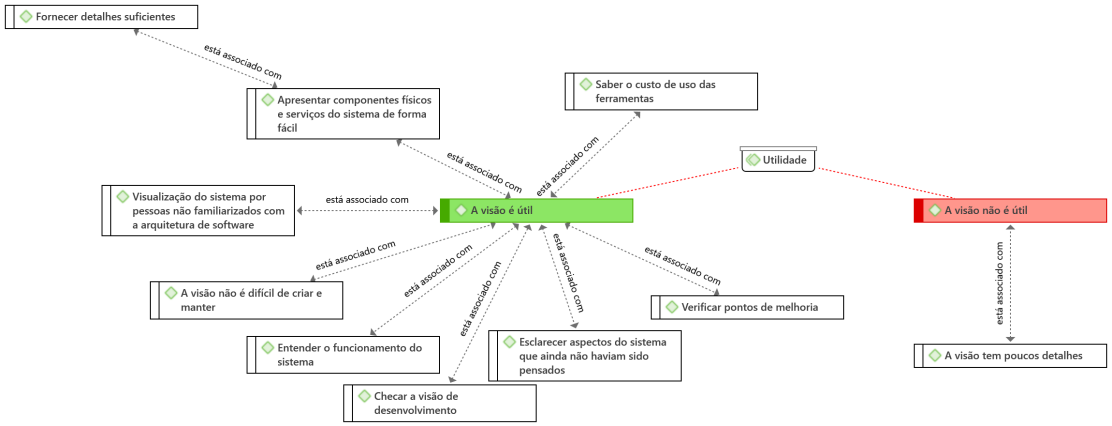


Figura 22 – Utilidade da visão física, segundo os participantes.

Todos os participantes que utilizaram a visão física concordaram que a adotariam no futuro, por razões diferentes como a rapidez para construí-la (P19), a visualização macro do sistema (P15) e a descrição detalhada do sistema para pessoas novas na equipe (P5). O participante P11 também afirma que *"adotaria isso [a visão] antes dos processos, visto que modificar a topologia do sistema pode trazer impacto para o funcionamento do código e para o negócio"*. A Figura 23 mostra os códigos resultantes da análise dos relatos dos participantes quanto à adoção da visão física.

5.3 Resultados C4 Model

Nesta seção, os resultados do estudo estão descritos e justificados, através dos códigos extraídos após a análise dos relatos dos participantes sobre a utilização da abordagem C4 Model (BROWN, 2017).

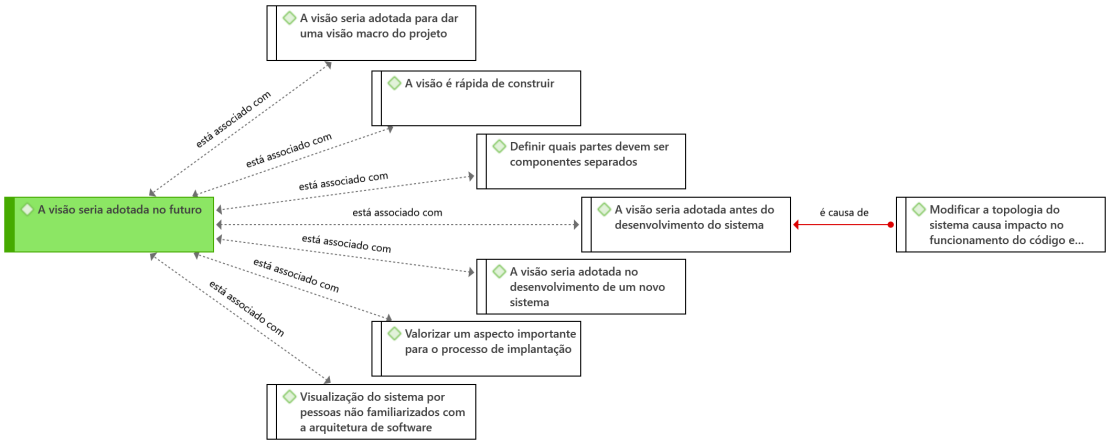


Figura 23 – Adoção da visão física relatada pelos participantes que a utilizaram durante o estudo.

5.3.1 Visão de Contexto

Quanto à visão de contexto, os participantes que a utilizaram relataram mais facilidades do que dificuldades com a construção dessa visão, com destaque para a simplicidade e a facilidade de construção, como pode ser visto pelo relato dos participantes P18 e P10: *"a visão é simples e direta, tornando a atividade de modelagem fácil e rápida"* (P18) e *"Fácil de construir"* (P10). Outro fator que auxiliou na construção dessa visão é a definição prévia dos atores e a não interação com outros sistemas, como descrito pelo participante P14: *"Foi fácil pensar no ponto de partida porque já havia sido definido os atores do sistema e o MVP não requisitava a interação com outros sistemas"*.

Dentre as dificuldades relatadas, o participante P14 afirma: *"Foi difícil pensar se haveria outros sistemas que iriam interagir com o principal, também foi difícil descrever a relação dos usuários com o sistema de forma sucinta"*. Por fim, mesmo com mais facilidades do que dificuldades, apenas um dos participantes que utilizaram a visão de contexto a considera útil, pois de acordo com o participante P14:

"É útil porque pessoas que não conhecem nenhuma técnica podem compreender o sistema a partir dele.". Embora seja uma visão fácil de construir, para o participante P10: *"...a ideia é tão básica que, ao meu ver, não agrega muito nas demais visões."* e para o P18: *"essa visão não acrescentou muito ao trabalho, pois apenas ilustrou o óbvio"*.

Os relatos dos participantes P10 e P18 mostram que eles não voltariam a utilizar a visão de contexto: *"Não adotaria, pois não achei útil para as demais visões, além disso é simples e genérica demais."* (P10) e *"Provavelmente não voltaria a usar essa visão a menos que houvesse múltiplos usuários distintos, que fizessem uso do sistema"* (P18). Os resultados da análise também podem ser vistos na Figura 24.

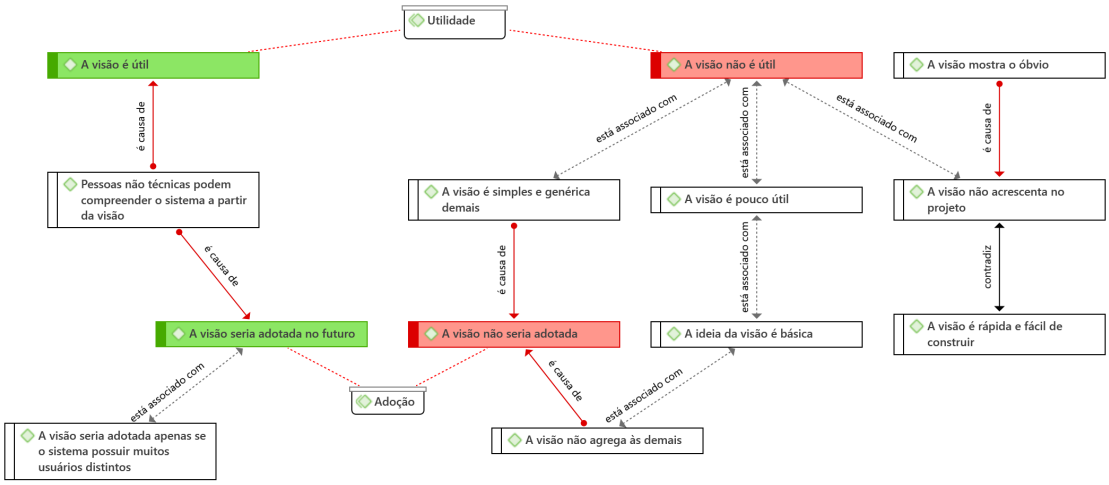


Figura 24 – Utilidade e adoção da visão de contexto, segundo o relato dos participantes do estudo.

5.3.2 Visão de Contêineres

Grande parte dos participantes que utilizaram a visão de contêineres afirmam que ela é uma visão simples e fácil de construir, de acordo com os relatos a

seguir: *"O diagrama em si só ilustra o óbvio...* (P17), *"[A visão é] Muito simples de fazer, entender e visualizar..."* (P8) e *"Ele é pequeno (para esse sistema) e a sintaxe é fácil"* (P5).

Em contrapartida, os participantes também tiveram dificuldades com a visão de contêineres. Os participantes P17 e P1 relataram dificuldades com a definição dos contêineres, como visto a seguir: *"A única dificuldade que eu senti foi para definir os containers de Sistema Web e Backend, pois eu não tenho experiência com processos"* (P17) e *"Foi difícil definir qual componente se comunica com outro"*. O participante P8 teve dificuldades com a ferramenta usada na construção da visão.

Sobre a utilidade da visão de contêineres, o participante P17 afirma que não viu utilidade alguma pela mesmo motivo que foi fácil construir a visão: *"...pois ele [a visão] só ilustra o óbvio"*. Por outro lado, os outros participantes consideram essa visão útil para delimitar os usuários e as partes do sistema que eles terão acesso (P8), ajudar na divisão do trabalho e visualização da equipe (P8). Os participantes P5 e P1 entendem que *"O diagrama não demora muito tempo para fazer, mas permite ter uma ideia básica de como o sistema funciona"* (P5) e *"Essa visão é útil para conseguir mostrar para outros membros da equipe como vai funcionar o MVP do sistema de forma funcional"*.

Com relação à adoção da visão de contêineres, as opiniões são conflitantes. O participante P17 prefere *"fazer um diagrama UML mais detalhado do que isso, pois eu não vi serventia alguma"* e o participante P5 disse que usaria a visão física (4+1) em vez da de contêineres *"já que tem mais detalhes de processos, mas acho que valeria a pena fazer um se a equipe não for composta apenas de desenvolvedores"*. Como contraponto, os participantes P8 e P1 adotariam a visão de contêineres,

pois *"acredito que ajuda na visualização e divisão integral do time, mas principalmente por ser mais simples* (P8) e *"para fazer com que toda a equipe fique por dentro do que vai ser feito"* (P1). Os códigos relacionados à utilidade e à adoção da visão de contêineres podem ser vistos na Figura 25.

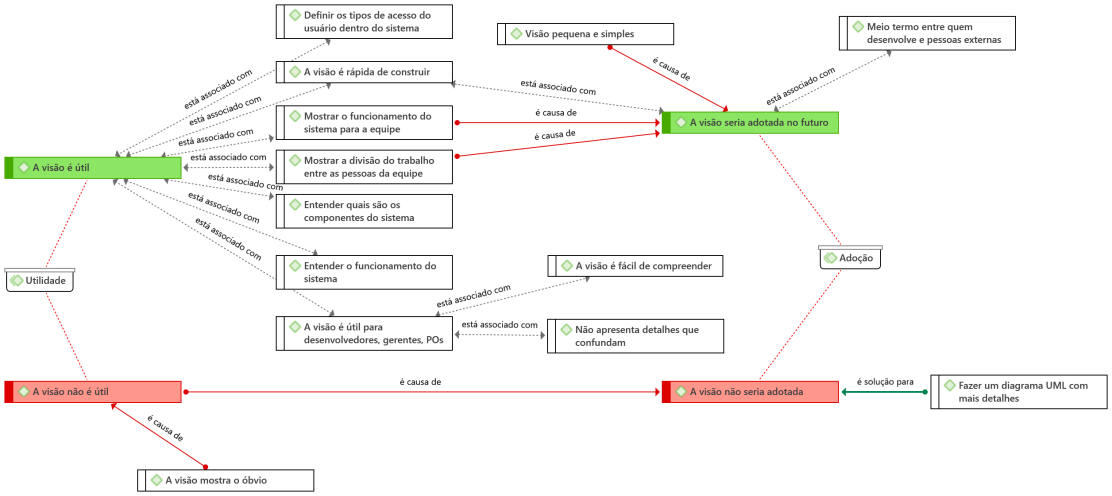


Figura 25 – Utilidade e adoção da visão de contêineres, segundo o relato dos participantes do estudo.

5.3.3 Visão de Componentes

Sobre as facilidades com a visão de componentes, os participantes descreveram diferentes fatores: *"...essa visão ajuda a separarmos que componentes serão responsáveis por certas lógicas do sistema"* (P7), *"...visualizar as interações entre os componentes"* (P20), *"...visualizar as conexões entre os componentes do sistema e as tecnologias utilizadas."* (P16), *"...determinar quais telas seriam as principais e deveriam estar no container para o MVP"* (P13), *"...visão fácil de ser compreendida teoricamente."* (P9), *"...utilização de poucos componentes visuais para desenhar."* (P15).

Embora tenha muitos relatos ligados à facilidade na construção, o uso da visão de componentes também pode ser dificultada por algumas razões. Segundo o participante P7, *"essa visão requer bastante noção de separação de componentes do que já vai ser feito para poder identificar corretamente o quê colocar..."* (P7). Para o participante P20, *"no começo, foi difícil visualizar quais seriam os componentes necessários para a criação dessa visão, pois como ela é mais detalhada, em um primeiro momento, foram muitas informações para processar e organizar..."* (P20). Em adição, outros participantes tiveram dificuldade em raciocinar sobre o que cada componente faria (P3, P15), entender o fluxo de dados (P4, P13), recordar o caminho percorrido pelo usuário no sistema (P9) e tomar as decisões necessárias sobre o sistema (P6).

A visão de componentes (BROWN, 2017) é útil segundo os participantes, de acordo com os seguintes relatos: *"útil para documentação do projeto, manutenção, ter uma visão técnica das responsabilidades"* (P7), *"essa visão ajudou a analisar melhor como os componentes do sistema que vão ser organizados, em como eles serão representados, em quais serão suas interações (ainda que de forma mais abstrata) e com quais outros componentes haverá interação na hora de transformar o sistema em código"* (P20), *"...visão útil para registrar quais ferramentas usar, quais são os componentes que temos disponíveis. Facilita a passagem de conhecimento caso entre no projeto alguém que não esteve nele desde o início do desenvolvimento"* (P9), *"é útil para organizar a estrutura mais detalhada de uma parte do sistema e para ver como uma equipe pode se dividir no trabalho e entender como seu trabalho pode influenciar o do colega"* (P13), *"é bastante útil para um desenvolvedor ter uma visão relativamente macro do sistema"* (P15) e *"visualizar com clareza os componentes do sistema web para proprietário no nível de abstração que pode ser mostrado e discutido com membros da equipe"* (P16),

"...é uma visão útil para entendermos melhor de onde vem os dados" (P4) e *"esta visão tem a capacidade de simplificar todo o backend de uma forma mais simples"* (P3). Os motivos descritos anteriormente também podem ser vistos na Figura 26.

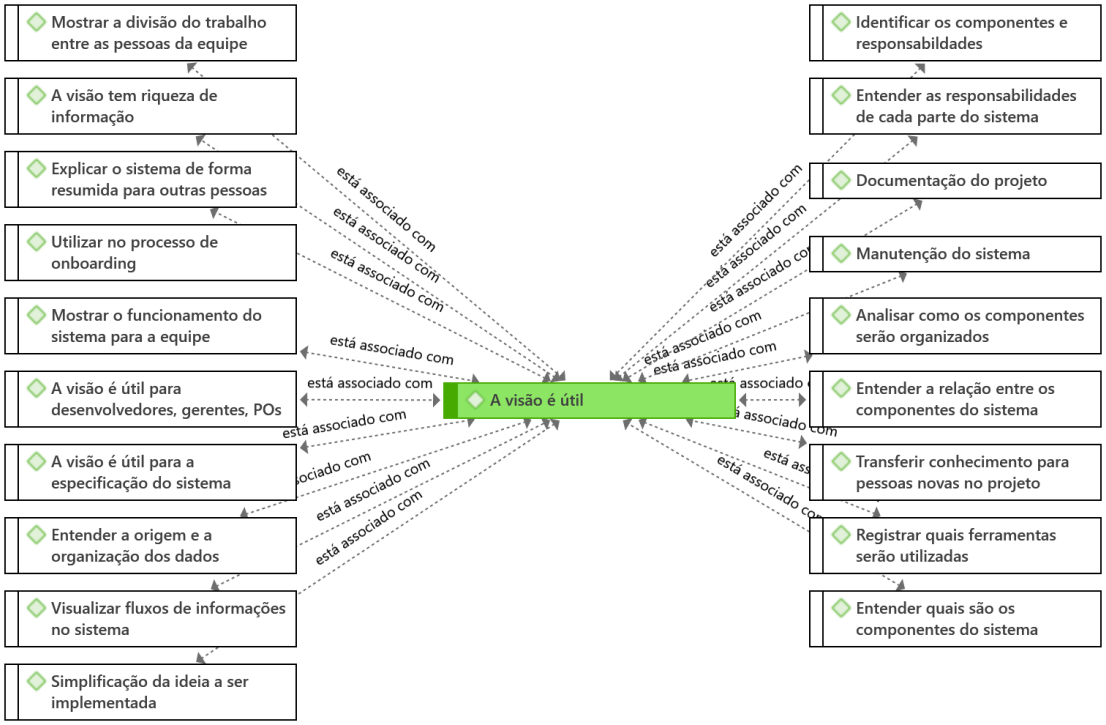


Figura 26 – Utilidade da visão de componentes, segundo o relato dos participantes do estudo.

Por fim, quase todos os participantes afirmaram que adotariam a visão de componentes pelos motivos descritos no parágrafo anterior. Os destaques (que também podem ser encontrados na Figura 27) são o participante P20 ao afirmar que *"Não adotaria novamente por ser muito trabalhoso e, apesar de todo esse trabalho, a visão ainda me parece abstrata para valer a pena tanto esforço"* e como contraponto, o participante P15 que descreve a visão de componentes como *"...uma excelente maneira de diagramar parte de um sistema como documentação, e*

também utilizá-lo no processo de onboarding".

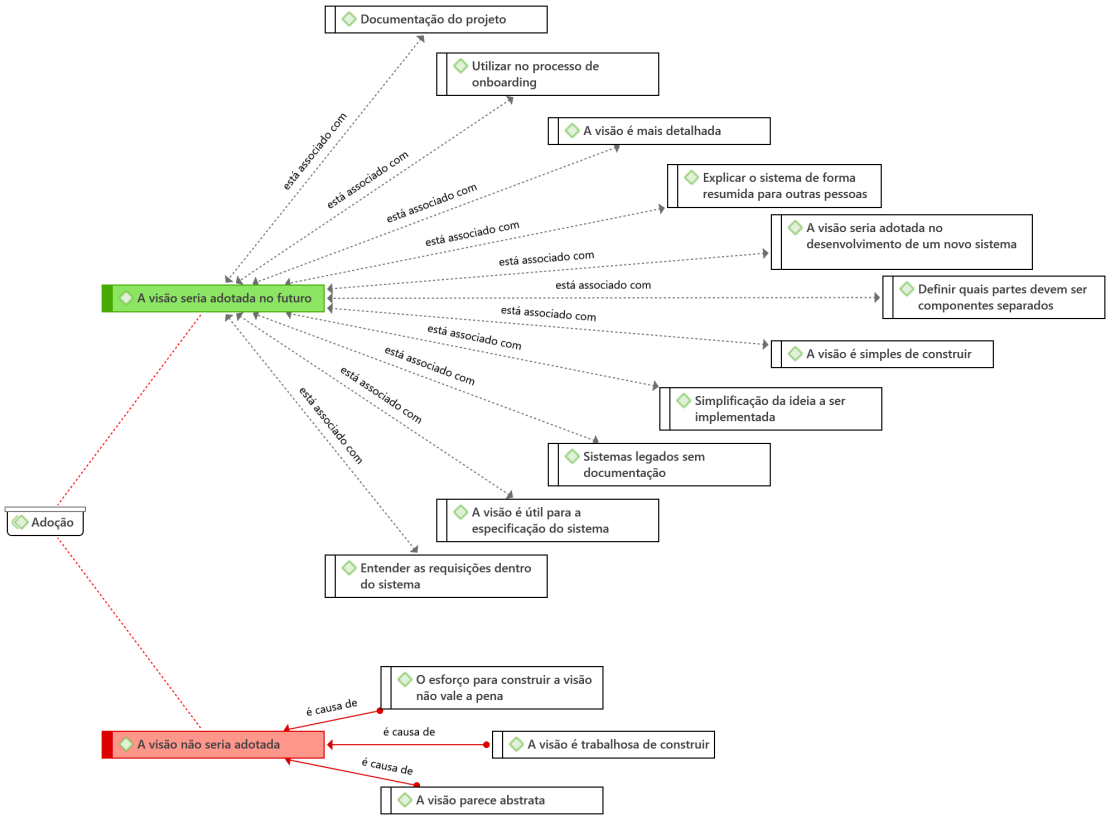


Figura 27 – Adoção da visão de componentes, segundo o relato dos participantes do estudo.

5.3.4 Visão de Código

A visão de código é *"...uma visão muito simples considerando só um componente"* (P19) e, para o participante P12: *"Foi fácil fazer a representação gráfica (sic)"*. Entretanto, para o participante P2 *"...não me pareceu tão simples, precisaria "codar" (sic) primeiro antes de usar"*. Essa necessidade de codificar o sistema antes de desenhar a visão pode ser relacionada com a facilidade de construção para o participante

P11, que afirmou ter facilidade na atividade devido à sua familiaridade com o processo mobile (*"essa visão foi fácil de construir devido à minha familiaridade com o desenvolvimento mobile"*).

Como dificuldades enfrentadas durante a construção da visão de códigos estão o tempo gasto na organização dos elementos dentro da visão (P19), a compreensão da teoria que embasa a visão (P11, P2) e encontrar material de apoio sobre a visão em outros locais (P11).

Com relação à utilidade, apenas o participante P12 afirma que a visão de código é útil para *"...enxergar as responsabilidades e dependências de cada componente"*. Os outros participantes consideram que a visão não é útil *"porque gerar essa estrutura é um processo natural para quem vai desenvolver, não trazendo tantos insights e visões sobre o sistema que vai ser desenvolvido...além de que, conforme o sistema cresce, manter esse diagrama vai ser tornando inviável"* (P11), *"não agregou informação a mais sobre o sistema e sobre o componente minimamente"* (P19) e para o participante P2, *" [a visão] parece bem útil se eu tivesse compreendido um pouco melhor"*. Por esses motivos, apenas o participante P12 adotaria essa visão pois, para ele, *"[a visão] ajuda na compreensão do fluxo da aplicação e torna mais fácil identificar o componente que precisa ser alterado para um incremento ou uma correção"*.

5.4 Discussões

Como visto na Seção 5.2, os participantes relataram ter facilidade ou não com as visões e enxergar a utilidade e a possível adoção por diferentes motivos, codificados após a análise dos resultados. Dentre os códigos, é possível enxergar similaridades e agrupá-los, de acordo com as suas características. Dessa forma,

dois grupos de códigos surgiram: códigos relacionados aos **aspectos técnicos da construção da visão** e aos **aspectos alheios à construção da visão**. Assim, é possível discutir quais motivos levam à utilidade enxergada e a futura adoção das abordagens ou de subconjuntos de suas visões (Seções 5.4.1 e 5.4.2) e em que momento do ciclo de desenvolvimento construir as visões presentes nas duas abordagens, como descrito também, de forma resumida, nas Tabelas 11 e 12.

5.4.1 4+1

De acordo com os resultados obtidos, a visão considerada menos útil e que não seria adotada é a visão de cenários. Embora seja considerada uma visão simples de construir, ela não é considerada valiosa, dificulta o entendimento do sistema por pessoas que não conhecem a UML e **deve ser feita apenas se não levar muito tempo para construí-la**. Com relação às quatro visões restantes, o entendimento sobre a utilidade e a adoção delas não é unânime.

A visão lógica é considerada útil por 75% dos participantes que a construíram, durante o experimento, e seria adotada no futuro por 50% deles, pois ela pode ajudar a equipe de desenvolvimento, mostrando o funcionamento do sistema. Entretanto, **ela será mais útil se for construída durante o desenvolvimento do sistema**, pois ela requer conhecimento sobre as classes do sistema e o controle da complexidade da visão, pois isso pode torná-la difícil de manter. Entende-se que as dificuldades enfrentadas pelos participantes, na construção dessa visão, se devem ao fato de o sistema desenhado por eles ainda não ter sido implementado, durante a realização do estudo.

Com relação à visão de processos, 100% dos participantes que a cons-

truíram não afirmaram ter dificuldades com a sua construção. Além disso, ela é considerada útil para mostrar se uma *feature* possui muitos passos, definir entidades e a sequência de atividades, verificar pontos de melhoria, além de ser entendida por pessoas não técnicas, de acordo com os participantes. Por fim, ela seria adotada, por ser fácil de construir e ajudaria no entendimento do sistema, caso o protótipo não tenha sido criado e **pode ser construída em qualquer momento do processo de desenvolvimento do sistema**.

A visão física é considerada útil para apresentar os componentes físicos do sistema de forma fácil, com detalhes suficientes, verificar pontos de melhoria e saber o custo das ferramentas usadas no sistema. A sintaxe utilizada pela visão e a criação após a visão de desenvolvimento facilitam esse processo. Entretanto, as dificuldades enfrentadas pelos participantes envolvem a avaliação de alternativas de mercado e a tomada de decisões sobre a infraestrutura do sistema, devido à falta de familiaridade com esse aspecto. Ainda assim, **a visão física seria adotada para construir um novo sistema e, em especial, antes do desenvolvimento**.

Por fim, a visão de desenvolvimento é considerada útil para organizar o código do sistema, visualizar fluxos de informação e enxergar os componentes do sistema sem precisar ler o código-fonte, sendo adotada, de acordo com os participantes, na construção de sistemas grandes e gerada de forma automática pelo ambiente de desenvolvimento. Dessa forma, **essa visão deve ser construída durante o desenvolvimento do sistema**, pois alguns participantes tiveram dificuldade em pensar no código do sistema antes de implementá-lo, assim como definir quais partes do sistema devem ser componentes separados. Assim como na visão lógica, isso se deve ao fato de o sistema desenhado pelos

participantes não ter sido implementado, durante a realização do estudo.

Tabela 11 – Resumo das discussões sobre as visões presentes na abordagem 4+1.

4+1			
Visão	**Quando Usar?**	**Por Quê?**	**Cuidados Necessários**
Lógica	D	Pode ajudar a equipe de desenvolvimento, mostrando o funcionamento do sistema.	Ter conhecimento das classes do sistema
Processos	AQM	Pode mostrar se uma funcionalidade tem muitos passos, ajudar a definir entidades e a sequência de atividades dentro do sistema.	Definir os casos de uso do sistema antes de montar esta visão
Física	A	Ajuda a mostrar os componentes do sistema de forma fácil, saber o custo das ferramentas utilizadas para implantar o sistema e verificar pontos de melhoria.	Ter experiência de desenvolvimento ou de implantação de sistemas e capacidade de tomar decisões
Desenvolvimento	D	É útil para organizar o código-fonte do sistema e talvez sirva mais para sistemas maiores, sendo gerada de forma automática por IDEs.	Ter começado a desenvolver o sistema antes de construir a visão, para ter mais certeza do que deve ser desenhado
Cenários	N	Construir somente se não levar muito tempo e quando estritamente necessário, pois talvez não adicione tanto valor ao trabalho realizado e não seja tão útil para pessoas não técnicas.	Definir os casos de uso do sistema antes de montar esta visão

Legenda: A - Antes do desenvolvimento; AD - Antes e durante o desenvolvimento; D - Durante o desenvolvimento; AQM - A qualquer momento; N - Não construir

5.4.2 C4 Model

A visão de contexto não foi considerada útil por mostrar o óbvio e por ser simples e genérica demais. Ainda assim, ela é uma visão rápida e fácil de construir e pode ajudar pessoas não técnicas a compreender o sistema. **Essa visão deve ser construída antes e durante do desenvolvimento do sistema**, pois é preciso pensar na interação com outros sistemas e descrever a relação dos usuários com o sistema, dificuldades enfrentadas pelos participantes do estudo.

Os participantes consideram a visão de contêineres útil para definir os tipos de acesso dos usuários no sistema, além de mostrar o funcionamento do sistema e a divisão do trabalho para a equipe de desenvolvimento. Em adição, ela foi vista como um um diagrama que pode ser útil para pessoas técnicas e não

técnicas como gerentes, desenvolvedores e outras pessoas da equipe, por não apresentar detalhes que confundem o leitor, além de ser fácil de compreender. Entretanto, é necessário que quem a utilizar tenha um pouco de experiência com desenvolvimento, para separar os contêineres e entender a relação entre eles, visto que essas foram dificuldades relatadas no estudo. Dessa forma, **a visão de contêineres deve ser criada antes e mantida durante o desenvolvimento do sistema** pois, com o crescimento e evolução do produto, o conhecimento continua sendo repassado à equipe e, de acordo com os participantes, essa visão é boa para esse objetivo.

Sobre a visão de componentes, todos os participantes que a utilizaram durante o estudo experimental a consideram útil, em especial, para aspectos alheios ao processo de construção da visão. Segundo os resultados do estudo, essa visão é útil para documentar o projeto, transferir conhecimento para novas pessoas da equipe, entender o fluxo e a origem dos dados presentes no sistema, identificar as partes importantes e as suas responsabilidades e também seria adotada para documentar sistemas legados e novos. Entretanto, ela requer conhecimento sobre separação e comunicação de componentes, capacidade de tomada de decisão e experiência de desenvolvimento, visto que os participantes tiveram dificuldade nesses pontos. Outro fato que pode ter influenciado na dificuldade com a construção da visão é que o sistema desenhado ainda não tinha sido implementado, durante a realização do estudo. Dessa forma, os participantes tiveram que pensar e idealizar o sistema apenas de forma abstrata e isso pode ter atrapalhado a realização da tarefa. Portanto, entende-se que **a visão de componentes deve ser construída apenas durante o desenvolvimento do sistema e não antes**, pois as abstrações maiores (contêineres) já estarão pla-

nejadas (na visão de contêineres) e os componentes serão desenhados junto ao crescimento do sistema.

Por fim, a visão de códigos não foi considerada útil por nenhuma pessoa que a construiu, durante o experimento. Entende-se que a circunstância que atrapalhou a construção da visão de componentes (sistema ainda não desenvolvido) influenciou ainda mais a experiência com a visão de códigos, pois cada visão de código é uma descrição detalhada de um componente específico do sistema, logo se o componente foi planejado apenas de forma teórica, o código que o implementa também foi pensado assim. Dessa forma, a experiência ou não com desenvolvimento influencia diretamente na construção da visão. Além disso, para os participantes, a visão de código é um processo natural para o desenvolvedor, não traz insights ou ideias novas sobre o sistema e sua manutenção pode se tornar inviável, com o crescimento do sistema. Esses relatos corroboram a descrição da visão feita pelo autor da abordagem e entende-se que **a visão de código deve ser criada apenas para componentes complexos e durante o desenvolvimento do sistema**, cujo código não consegue ser explicado facilmente ou precisa ser entendido por muitas pessoas.

5.5 Limitações deste Estudo

Uma limitação do estudo experimental é a sua realização com equipes compostas por alunos de graduação, com graus de experiência com desenvolvimento diferentes. Alem disso, os sistemas documentados não haviam sido implementados. Isso foi levado em consideração pelos participantes e influenciou na experiência deles com algumas das visões, como de componentes (C4) e de desen-

Tabela 12 – Resumo das discussões sobre as visões presentes na abordagem C4 Model.

C4 Model			
Visão	**Quando Usar?**	**Por Quê?**	**Cuidados Necessários**
Contexto	AD	É uma visão simples e ajuda pessoas não técnicas a compreender o sistema.	Pensar na interação de atores e sistemas externos
Contêineres	AD	É uma visão útil para definir os tipos de acesso dos usuários no sistema, mostrar o funcionamento do sistema e pode descrever a separação de trabalho dentro da equipe, importante para gerentes, POs e desenvolvedores.	Ter experiência com desenvolvimento para separar os contêineres e definir as relações entre eles.
Componentes	D	Ajuda a documentar o projeto, transferir conhecimento para pessoas novas, entender o fluxo e origem dos dados, identificar as partes importantes e suas responsabilidades.	Ter capacidade de tomar decisões e experiência de desenvolvimento para separar os componentes
Código	N	Somente para componentes complexos, pois é um processo natural para o desenvolvedor, não traz insights ou ideias novas sobre o sistema e sua manutenção pode se tornar inviável, com o crescimento do sistema.	Garantir que o componente escolhido precisa ser explicado para muitas pessoas ou é tão complexo que não pode ser entendido facilmente

Legenda: A - Antes do desenvolvimento; AD - Antes e durante o desenvolvimento
D - Durante o desenvolvimento; N - Não construir

volvimento (4+1) e esse fator foi considerado para o planejamento dos estudos seguintes.

5.6 Conclusões

Esse estudo avaliou a experiência de uso de duas abordagens de documentação de arquitetura de software em um contexto de equipes em estágio inicial de desenvolvimento de sistemas. Os resultados mostram que a experiência foi influenciada por fatores ligados aos aspectos técnicos da construção do sistema (inserir elementos, utilizar linguagens específicas de modelagem) e aspectos alheios à construção da visão como a experiência com desenvolvimento, tomada de decisão, avaliação de alternativas de mercado e o estágio de desenvolvimento do sistema (antes, durante e após a implementação).

Dessa forma, é possível extrair informações relevantes sobre as duas

abordagens e inferir em que momento cada visão será mais útil de ser construída, como discutido na Seção 5.4. Entretanto, dado ao contexto fechado de execução, este estudo carece de mais indícios de validade e esta necessidade é endereçada nos estudos realizados e apresentados nos Capítulos 6 e 7.

6

Experimento Controlado

A inserção de defeitos durante a utilização de abordagens de documentação pode influenciar na escolha e na adoção de alguma delas. Dessa forma, é importante verificar se há diferenças significativas entre a quantidade de defeitos encontrados nos diagramas, após a documentação da arquitetura de um sistema com as abordagens estudadas.

Essa quantidade de defeitos pode ser encontrada na presença ou não de informações importantes, na descrição precisa ou ambígua dos conceitos presentes nas visões ou no acréscimo de características irrelevantes sobre o sistema. Todas essas consequências podem dificultar a transmissão do conhecimento sobre o sistema, através das visões, e, assim, não alcançar o objetivo que se busca ao utilizar essas abordagens.

Nesse sentido, um estudo experimental foi realizado com a participação de 18 (dezoito) estudantes do curso de Ciência da Computação, da Universidade Federal do Ceará – Campus Russas, com o objetivo de responder a seguinte pergunta: **existe diferença significativa na quantidade de defeitos encontrados ao utilizar a abordagem C4 Model e a 4+1?**

Os resultados apontam que não há diferença significativa entre a quantidade de defeitos encontrados após a utilização das duas abordagens e podem ser vistos com mais detalhes nas seções a seguir, junto à metodologia implementada.

6.1 Planejamento

Para entender a metodologia, é importante também conhecer os participantes. Por isso, na Seção 6.1.1, estão descritas informações sobre o grau de conhecimento sobre arquitetura de software, além do nível de experiência profissional com desenvolvimento de software. Em seguida, os resultados estão descritos de forma geral e distribuídos por participante na Seção 6.2.1, com as médias das quantidades de defeitos e explicações dos testes de hipótese aplicados. Por fim, os defeitos foram agrupados por categoria, para permitir uma análise mais profunda, presente na Seção 6.2.2.

6.1.1 Demografia dos participantes

Dezoito pessoas concordaram em participar deste estudo experimental, através do preenchimento do Termo de Consentimento Livre e Esclarecido (TCLE). Eles possuíam diferentes graus de conhecimento sobre arquitetura de software e níveis de experiência profissional de desenvolvimento de software. As Figuras 28, 29 e 30 a seguir mostram a distribuição dos participantes de acordo com o nível de experiência com desenvolvimento de software, a sua duração e o grau de experiência sobre a área de arquitetura de software.

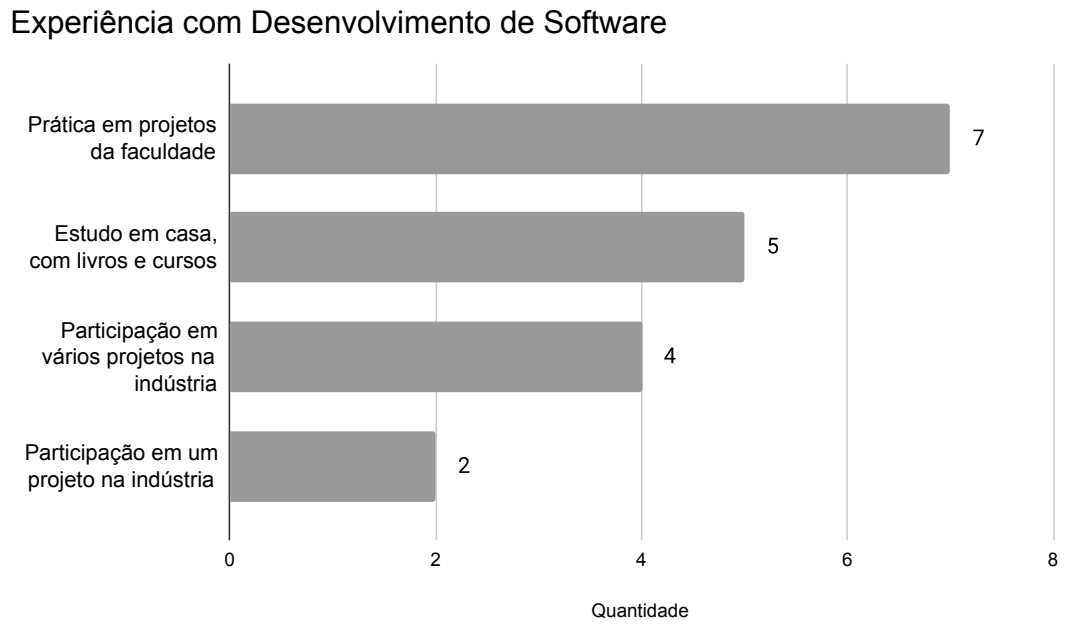


Figura 28 – Distribuição dos participantes de acordo com a experiência com desenvolvimento de software.

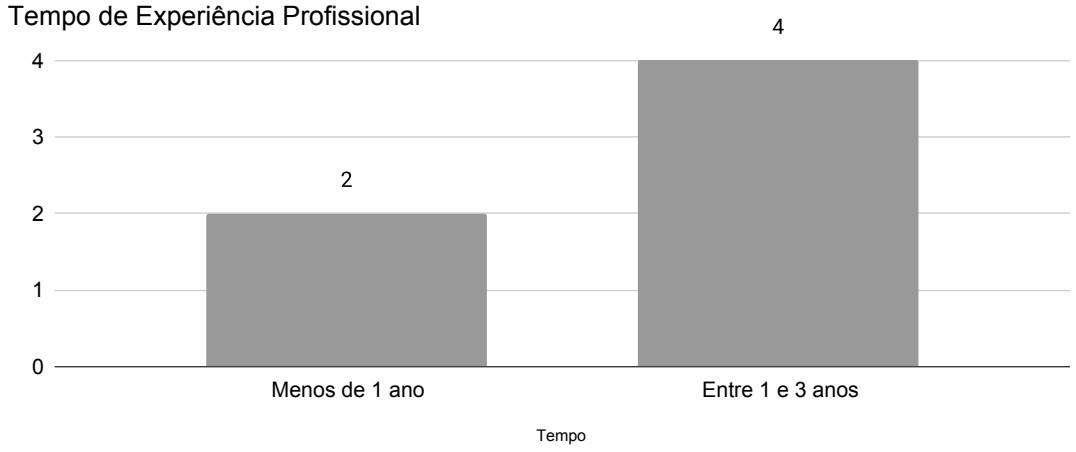


Figura 29 – Distribuição dos participantes que possuem algum grau de experiência profissional de acordo com a sua duração.

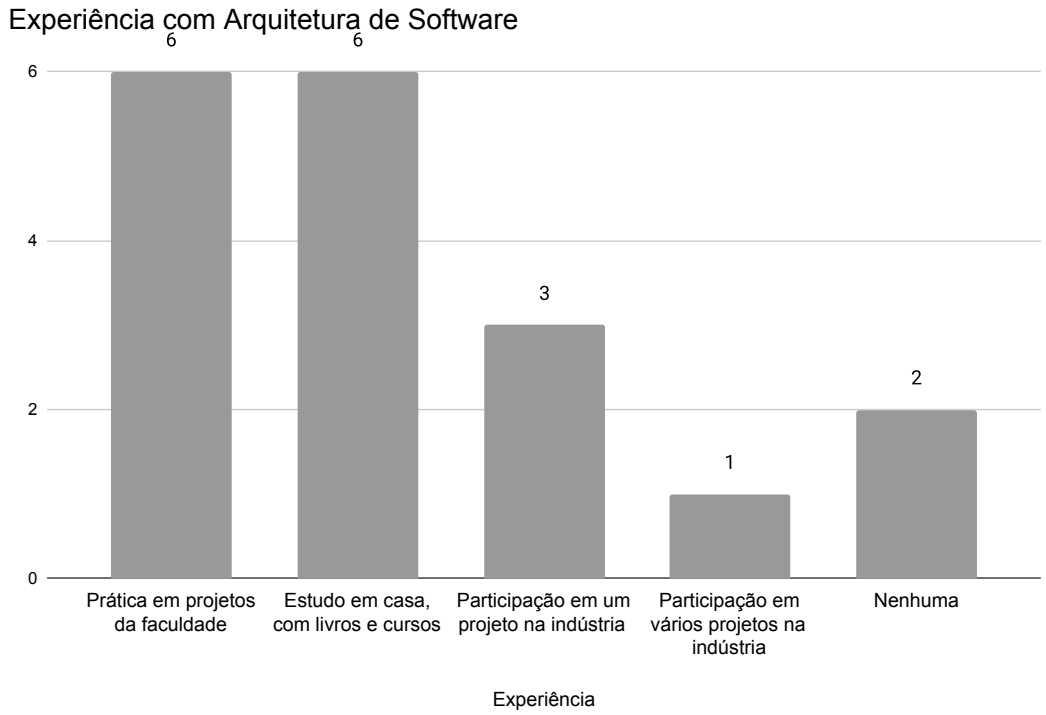


Figura 30 – Distribuição dos participantes de acordo com a experiência com a área de arquitetura de software.

6.1.2 Execução

Os participantes foram divididos em dois grupos e eles realizaram o experimento em duas etapas. A tarefa dos participantes durante as duas etapas foi **documentar a arquitetura de um sistema de software, utilizando uma das duas abordagens estudadas (C4 e 4+1)**. Eles deveriam construir três visões da abordagem atribuída a eles, na etapa em questão. Essa quantidade foi definida para não haver diferença no número de artefatos entregues, visto que a abordagem 4+1 possui mais visões que a C4. Ademais, no contexto da pesquisa, em que os participantes não iriam implementar os sistemas modelados, não seria neces-

sário construir as visões de desenvolvimento (4+1) e de código (C4). Por isso, elas não foram utilizadas no estudo. Para a abordagem 4+1, foram definidas as visões Lógica, de Processos e Física e, para a abordagem C4, as visões de Contexto, Contêineres e Componentes.

Na primeira etapa, cada participante de um dos grupos deveriam documentar, individualmente, a arquitetura de um sistema utilizando a abordagem C4, enquanto os integrantes do outro grupo fariam o mesmo, entretanto utilizando a abordagem 4+1. Na segunda etapa, as equipes se inverteram: o participante que utilizou a abordagem C4 na primeira etapa, deveria construir a documentação do sistema com a abordagem 4+1 e vice-versa. A Tabela 13 e a Figura 31 mostram a divisão dos participantes entre os grupos e quais abordagens foram utilizadas por cada equipe, em cada etapa do experimento, respectivamente. Essa divisão foi feita com a atribuição de um número a cada participante e ao sorteio do grupo em que o número seria alocado.

Tabela 13 – Divisão dos participantes entre os grupos, para realização do estudo controlado.

Grupo	Participantes									
A	P03	P06	P08	P10	P12	P13	P14	P15	P17	P18
B	P01	P02	P04	P05	P07	P09	P11	P16	P19	P20

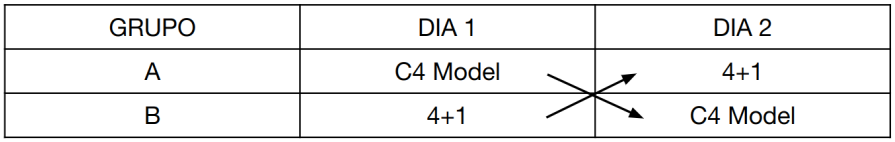


Figura 31 – Divisão das abordagens entre os grupos, em cada etapa de execução do estudo controlado.

Os sistemas foram fornecidos aos participantes através de uma lista de

requisitos funcionais e não funcionais, além de um diagrama de casos de uso. Dessa forma, não seria possível construir a visão de cenários (4+1), não solicitada durante o experimento. Os sistemas entregues aos participantes foram validados previamente por duas professoras que lecionam disciplinas da área de engenharia de software, com experiência profissional e doutorado na área. A validação prévia serviu para verificar se os sistemas possuíam complexidades semelhantes. Além disso, o sistema utilizado na segunda etapa é diferente do sistema fornecido na primeira etapa (requisitos parecidos em um produto diferente).

As listas de requisitos dos dois sistemas estão disponíveis como material complementar [1] e também presentes no Apêndice C.

6.1.3 Avaliação dos resultados

Os modelos construídos foram avaliados pelo primeiro autor desta pesquisa e revisados pela orientadora Tayana Conte. A partir dos modelos, construiu-se uma base de defeitos relacionados a cada participante. Os defeitos possuem pesos, para diferenciar o grau de severidade do erro presente no modelo, de acordo com a Tabela 14:

Peso	**Severidade**
1	Leve
2	Moderada
3	Grave
5	Extrema

Tabela 14 – Graus de severidade dos defeitos encontrados durante o estudo quantitativo.

[1] Material Complementar – Estudo Quantitativo: https://figshare.com/s/5a858f2b1c70d0803b25

A hipótese nula que guiou a análise quantitativa é: **a diferença entre as médias das quantidades de defeitos na construção de modelos usando as abordagens C4 e 4+1 é não significativa.**

Em seguida, a pontuação de cada abordagem foi calculada através da soma ponderada dos defeitos. Ela também foi analisada de forma separada por participante e utilizada no teste de hipótese, realizado com o T-Test Pareado (ROSNER, 1982), visto que as amostras possuem distribuição normal, verificada com o teste de Shapiro-Wilk (para a abordagem C4: $p = 0,2681 > 0,05$; para a abordagem 4+1: $p = 0,5363 > 0,05$).

6.2 Resultados

6.2.1 Resultados Gerais

A média da quantidade de defeitos para a abordagem C4 foi 12,55, a mediana = 11 e o desvio padrão = 7,1191. Para a abordagem 4+1, a média da quantidade de defeitos foi 14,20, a mediana = 14,50 e o desvio padrão = 5,227458. Essas informações também podem ser vistas na Figura 32.

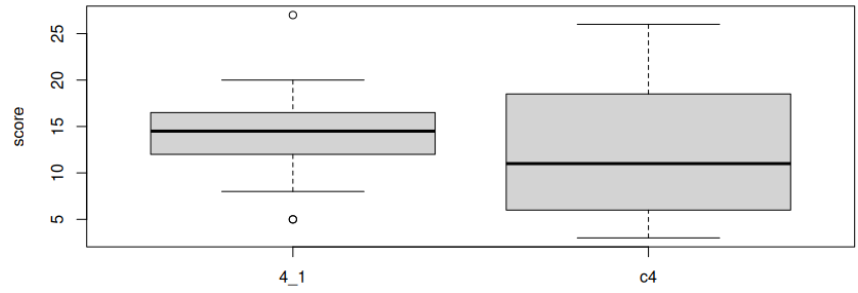


Figura 32 – Boxplot dos resultados obtidos durante a utilização da abordagens estudadas.

Os resultados dos testes estatísticos apontam que **não há diferença estatística significativa entre as médias das quantidades de defeitos encontradas com o uso da abordagem C4 e da 4+1** ($p - value = 0,1865 > 0.05$). Dessa forma, a hipótese nula **não deve ser rejeitada**, o que significa, que não há evidências matemáticas que comprovem que a utilização de uma abordagem leva a maior inserção de defeitos do que a outra. Apesar disso, os defeitos encontrados podem ser agrupados de acordo com o seu tipo e essa divisão foi utilizada para uma segunda análise (qualitativa) para verificar se há alguma relação entre as abordagens de documentação da arquitetura e os tipos de defeitos encontrados durante a sua utilização.

6.2.2 Análise dos defeitos por categoria

Os defeitos encontrados foram classificados em três categorias, listadas a seguir, e as suas quantidades na Tabela 15. Essa categorização foi feita durante a correção dos diagramas entregues, antes das análises quantitativa e qualitativa.

- Falta de informação importante – Ocorre quando um requisito funcional ou não funcional não é descrito em um diagrama. Por exemplo, a ausência de um ator que foi descrito nos requisitos ou a não inclusão de um componente necessário para atender aos requisitos.

- Ambiguidade – É identificada quando um elemento do diagrama (um componente, por exemplo) não foi descrita de forma exata, permitindo interpretações diferentes sobre o mesmo elemento. Um exemplo disso é a ligação de um ator chamado Usuário a uma parte do sistema, quando o

sistema possui diferentes tipos de usuário e, assim, não é possível identificar qual deles faz parte da relação.

- Informação incorreta inserida sobre o sistema – Ocorre quando aspectos não necessários para a visão são inseridos nela ou são descritos de forma incorreta como descrever um protocolo de comunicação como um componente (C4), o que está conceitualmente incorreto.

Tabela 15 – Quantidades de defeitos separados por participante e por categoria para as duas abordagens estudadas (4+1 e C4)

	Falta de informação importante		**Ambiguidade**		**Informação incorreta inserida sobre o sistema**		**Visão não construída**	
	C4	**4+1**	**C4**	**4+1**	**C4**	**4+1**	**C4**	**4+1**
P01	1	5	3	0	1	6	0	0
P02	2	3	0	1	0	2	0	0
P03	1	5	0	0	0	1	0	0
P04	5	3	0	0	3	1	0	1
P05	1	5	3	2	3	1	0	0
P06	3	3	2	1	1	1	2	1
P07	4	0	0	0	1	0	0	1
P08	5	1	0	0	0	0	0	2
P09	3	5	0	0	1	3	0	0
P10	1	5	0	1	0	0	0	0
P11	5	2	0	0	1	0	0	0
P12	2	2	0	0	0	1	0	0
P13	2	4	0	1	0	0	0	0
P14	1	4	1	0	1	0	2	1
P15	1	4	0	0	1	0	1	1
P16	1	4	0	0	1	0	0	0
P17	2	4	0	0	1	0	2	1
P18	5	5	3	1	0	0	0	0
P19	1	2	0	0	1	1	1	1
P20	1	2	0	0	2	4	2	0
Total	**48**	**68**	**11**	**7**	**18**	**21**	**10**	**9**

6.2.2.1 Falta de informação importante

A maior parte dos defeitos encontrados nos diagramas estão relacionados à falta de informações importantes que estavam nas especificações entregues aos participantes, mas não foram desenhadas nas visões. Isso aconteceu em maior intensidade dentro dos diagramas feitos com a abordagem 4+1 (68 defeitos).

Alguns exemplos desses defeitos são a ausência da representação de entidades importantes na visão lógica como classes e atributos presentes nos requisitos funcionais. Além disso, componentes que seriam responsáveis por algumas funcionalidades também não estiveram presentes na sua devida visão. A Figura 33 mostra um exemplo de visão com falta de informações importantes sobre o sistema. Nela, o ator Recepcionista do Hotel **não** está representado na visão de contexto. Dessa forma, entende-se que somente o Cliente utilizará o sistema, o que não está correto.

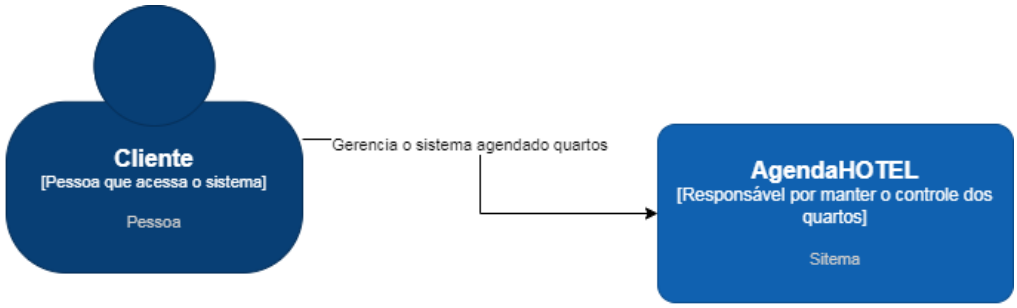


Figura 33 – Exemplo de diagrama de contexto que possui defeito de falta de informacao importante sobre o sistema.

Uma possível razão para a aparição deste tipo de defeitos é a **não compreensão do conceito de entidades**. A visão lógica pede que nela estejam os domínios importantes da aplicação, extraídos dos requisitos funcionais. Dessa forma, seria necessário constar nas visões lógicas relacionadas ao sistema da pri-

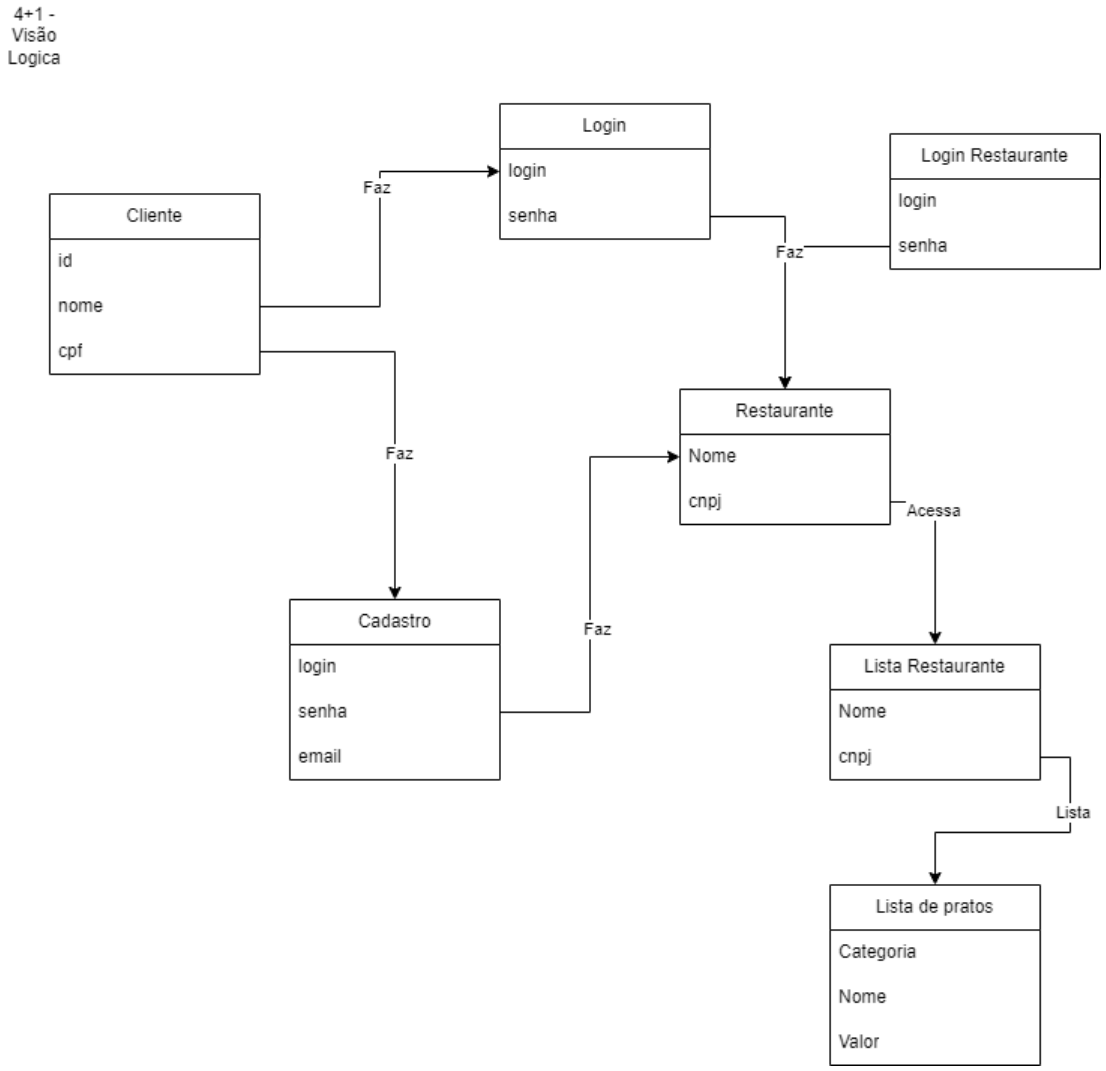


Figura 34 – Exemplo de visão lógica que possui defeito de falta de informacao importante sobre o sistema.

meira etapa as seguintes entidades: Hotel, Quarto, Reserva, Cliente, Avaliação. De maneira semelhante, a visão lógica relativa ao sistema entregue na segunda etapa deveria ter as entidades Cliente, Restaurante, Pedido, Prato e Avaliação.

6.2.2.2 Informação incorreta inserida sobre o sistema

Sobre informações incorretas inseridas no sistema, isso se caracteriza pela descrição incorreta de entidades e atributos e a inserção de características externas como parte do sistema. Alguns exemplos são a descrição do protocolo de comunicação entre containers como um container próprio (Figura 35), a descrição de entidades como atividades (visão de processos - 4+1) e eles podem ser vistos na Figura 36. Nela, a visão de processos contém todos os casos de uso do sistema. Entretanto, cada caso de uso deveria ter uma visão, descrevendo com detalhes o que deve ser feito.

6.2.2.3 Ambiguidade

Com relação à ambiguidade, os defeitos presentes nessa categoria apresentam características que podem levar a interpretações diversas sobre o mesmo conhecimento, devido à falta de informações que diferenciem os elementos ou à presença de dados que aumentem o grau de confusão. Essa foi a única categoria de defeitos em que a abordagem C4 Model se sobressaiu à 4+1, na quantidade de defeitos encontrados.

Alguns exemplos de defeitos encontrados considerados ambíguos estão a definição de atores diferentes com o mesmo nome, visto que não é possível saber qual ator utilizará o componente/container que está ligado a ele. Da mesma forma, elementos sem descrição de tipo não conseguem ser diferenciados, principalmente, quando têm o mesmo formato, cor e tamanho. Os defeitos citados podem ser vistos na Figura 37. Nela, há apenas um ator chamado “Person”. Isso está incorreto pois, na especificação, os atores foram separados em Cliente

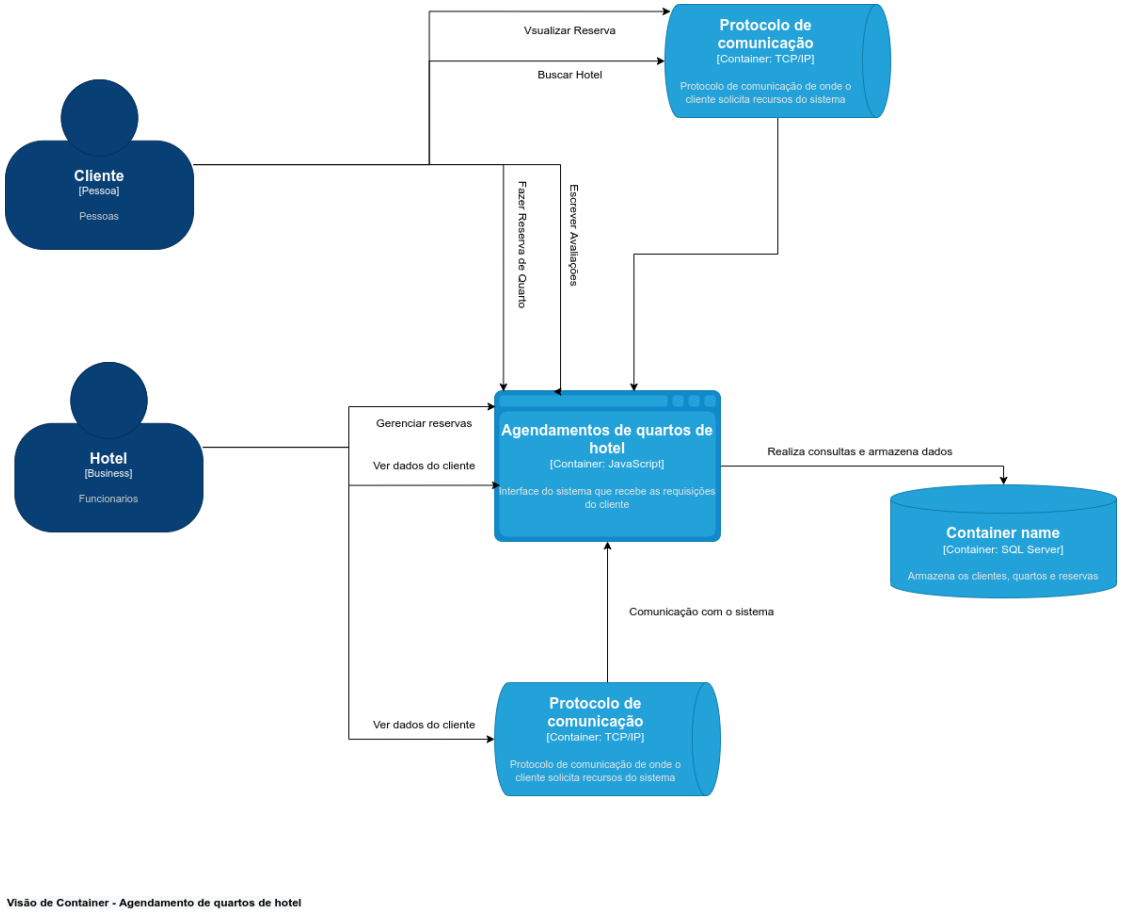


Figura 35 – Exemplo de visão de contêineres que possui defeito de informações incorretas inseridas sobre o sistema.

e Restaurante. Essa separação deveria constar na visão entregue.

6.3 Ameaças à Validade

As ameaças se concentram na validade de construção, visto que a quantidade de defeitos encontrada durante o experimento foi influenciada por fatores externos aos instrumentos utilizados. Para atenuar esse efeito, os participantes ti-

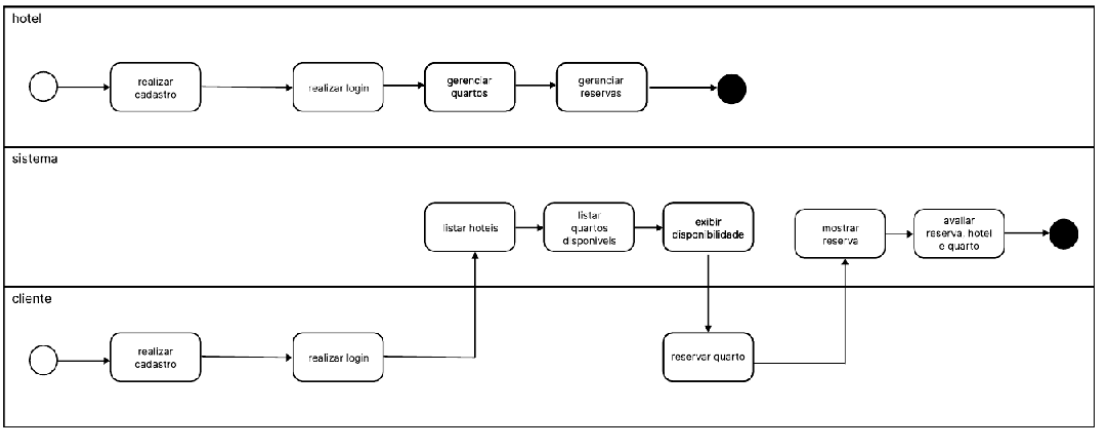


Figura 36 – Exemplo de visão de processos que possui defeito de informações incorretas inseridas sobre o sistema.

veram acesso às listas de requisitos apenas no dia da execução do experimento, assim, todos tiveram espaços de tempo semelhantes para extrair as informações necessárias para desenhar os diagramas solicitados.

Além disso, outra ameaça foi o viés de conhecimento sobre os instrumentos entre as etapas de realização do experimento. Por isso, com o objetivo de diminuir o viés, os participantes foram divididos em duas equipes, cada uma utilizando um instrumento diferente. Durante a segunda etapa de execução, os participantes que utilizaram a abordagem C4 anteriormente deveriam utilizar a abordagem 4+1 e vice-versa.

6.4 Conclusões

É possível extrair, a partir deste estudo, indícios embrionários sobre os defeitos encontrados durante a utilização de duas abordagens de documentação de

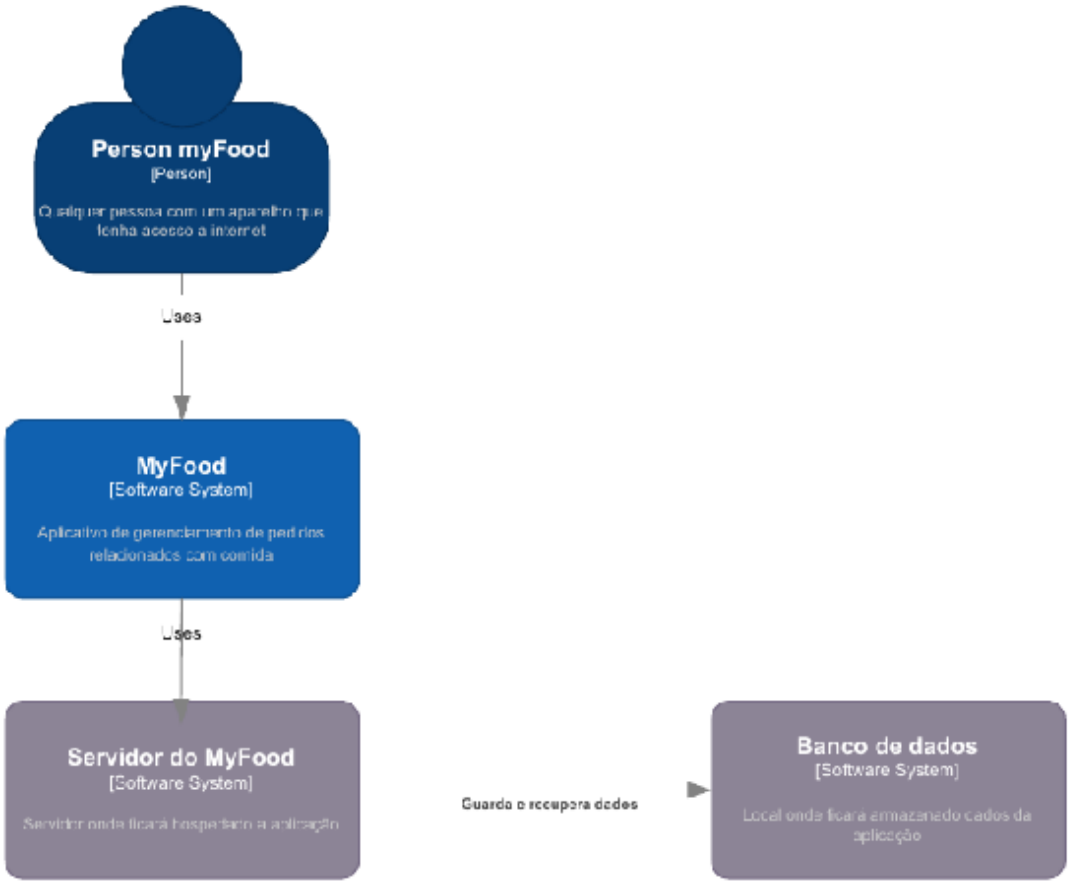


Figura 37 – Exemplo de visão de contexto que possui defeito de ambiguidade.

arquitetura em contextos ágeis.

Os resultados estatísticos mostram que não há evidência significativa de diferença na inserção de defeitos ao utilizar uma ou a outra abordagem.

Por isso, para encontrar mais informações sobre como são esses defeitos, outra análise foi realizada, com caráter qualitativo, e verificou-se que a falta de informações importantes sobre o sistema, a ambiguidade das descrições dos elementos e a inserção de dados irrelevantes podem influenciar na qualidade

dos diagramas desenvolvidos.

Nesse sentido, mais estudos podem ser realizados para avaliar o impacto do conhecimento sobre o sistema e sobre as abordagens de documentação para reforçar a hipótese que fatores externos influenciam mais na inserção de defeitos ou se a experiência com as abordagens (ou a falta dela) impede os participantes de expressarem o conhecimento sobre a arquitetura dos sistemas da melhor forma possível.

7

Estudo em Ciclo de Vida

Nesta seção, é apresentado um estudo experimental realizado para coletar evidências sobre o uso das abordagens de documentação estudadas durante o ciclo de desenvolvimento, além de indícios sobre aspectos externos às abordagens como a divisão da tarefa de documentar a arquitetura dentro da equipe, assim como os critérios utilizados para a escolha da abordagem. O objetivo deste estudo foi descrito abaixo, seguindo o modelo GQM (CALDIERA; ROMBACH, 1994):

Analisar duas abordagens de documentação de arquitetura de software,

Com o propósito de coletar indícios,

A respeito da experiência de utilização, organização da equipe e critérios de escolha,

Pelo ponto de vista dos participantes e dos pesquisadores,

No contexto de estudantes de graduação, divididos em equipes de desenvolvimento ágil de software.

7.1 Metodologia

Para obter indícios sobre a utilização de abordagens de documentação de arquitetura de software, um estudo experimental foi realizado com 20 alunos de graduação do curso de Bacharelado em Engenharia de Software, da Universidade Federal do Amazonas. Os participantes foram escolhidos devido à semelhança da sua atuação na disciplina de Projeto de Engenharia de Software com contextos ágeis de desenvolvimento.

Essa disciplina é ministrada durante o sétimo período do curso de Bacharelado em Engenharia De Software da Universidade Federal do Amazonas. Nela, os alunos participam de atividades práticas relacionadas ao processo de desenvolvimento de software, através da metodologia de Aprendizado Baseado em Projetos (ABP) (SANTOS et al., 2007).

Os alunos foram divididos em cinco equipes e deveriam construir produtos de software utilizando novas tecnologias, seguindo o framework Scrum, para organização das atividades e entregas (SCHWABER, 1997), com *sprints* de 15 dias de duração. Durante a disciplina, os alunos devem elicitar os requisitos, projetar e desenvolver seus sistemas e, no final da disciplina, entregar a documentação do sistema. Eles podem ter clientes externos, mas a maioria tinha como gerente a professora da disciplina como gerente, com quem tinham as reuniões para apresentar o sistema durante as diferentes *sprints* e coletar *feedbacks*.

A documentação que deve ser entregue, ao fim da disciplina, não continha originalmente, aspectos da arquitetura do sistema. Por isso, para realizar o estudo, foi solicitado aos alunos que construíssem a documentação da arquitetura, utilizando uma abordagem dentre as duas avaliadas: C4 e 4+1.

Para realizar essa tarefa, os alunos receberam treinamento durante duas aulas da disciplina. Nesse treinamento, os alunos receberam uma introdução aos conceitos de arquitetura de sistemas e documentação da arquitetura, assim como a apresentação das duas abordagens. Em seguida, as equipes deveriam construir a documentação da arquitetura do sistema em um prazo de 10 (dez) dias.

Após a apresentação das documentações criadas, os alunos participaram de uma entrevista coletiva, para coletar relatos sobre a experiência de utilização de abordagens de documentação de arquitetura de software, além do processo de realização dessa tarefa. Os participantes assinaram um Termo de Consentimento Livre e Esclarecido, autorizando a utilização de seus relatos para o estudo e a entrevista foi realizada de forma remota.

A entrevista foi gravada para transcrição posterior e foi utilizada para a análise dos resultados. As documentações criadas pelos alunos não foram utilizadas no estudo e não interferiram em seus resultados.

As perguntas realizadas durante a entrevista estão listadas abaixo:

1. O que vocês acharam da atividade?
2. Qual foi a maior dificuldade de vocês durante a atividade?
3. Quais critérios vocês utilizaram para escolher a abordagem utilizada?
4. Como foi a organização da equipe para usar/implementar a abordagem escolhida? Por quê?
5. O uso da abordagem ajudou vocês a alterarem o sistema de alguma forma? O que mudou?

6. Como o uso da abordagem influenciou nas outras atividades da equipe?

7. Teve alguma visão da abordagem escolhida que vocês usariam mais ou consideram mais interessante? Por quê?

8. Vocês tiveram dificuldade em criar alguma visão da abordagem escolhida? Qual? Por quê?

9. Vocês acham que a abordagem que vocês não escolheram pode ajudá-los de alguma forma que a abordagem escolhida não foi capaz?

10. No futuro, vocês escolheriam quais visões das duas abordagens para documentar a arquitetura de um sistema? Por quê?

7.2 Resultados

As seções a seguir descrevem os resultados obtidos e divididos de acordo com os temas abordados na entrevista: critérios de escolha da abordagem utilizada, organização da equipe, dificuldades enfrentadas e visões mais interessantes ou que seriam adotadas futuramente.

As respostas dadas durante a entrevista foram analisadas através da análise temática (CRUZES; DYBA, 2011), com a criação de códigos fundamentados nas citações, a construção e organização de relações entre eles. Esse processo foi realizado com o apoio da ferramenta Atlas TI 8 e revisado pela orientadora desta pesquisa, além da professora da disciplina de Práticas em Engenharia de Software, Dra. Ana Oran.

7.2.1 Critérios de escolha da abordagem

Os participantes, em cada equipe, deveriam escolher uma das duas abordagens estudadas para documentar a arquitetura dos seus sistemas. Os critérios utilizados para fazer essa escolha estão presentes na Figura 38. A familiaridade com a UML foi um fator levado em consideração para escolher a abordagem que seria utilizada. Algumas equipes escolheram a abordagem 4+1 por já conhecerem os diagramas UML, aprendidos e utilizados em disciplinas anteriores, como visto nos relatos do participantes P2 e P6: *"...eu acho que porque eu já conhecia boa parte dos diagramas. Então, ficou mais fácil para fazer dessa forma, basicamente isso..."* (P2) e *"A gente fez também toda a modelagem dessa [abordagem], então a gente já tinha familiaridade com os diagramas..."* (P6). Entretanto, o uso da UML e as suas regras também levaram alguns participantes a escolher a abordagem C4. Isso pode ser visto no relato do participante P4: *"a UML, assim, eu penso que é muito restritiva, né? Os diagramas têm que seguir aquelas regras, usar bonequinho, fazer o diagrama de sequência, tem que colocar a setinha certa. O C4 não. É só usar aquelas caixinhas e é mais fácil de fazer assim, né?"*. Além do uso da UML, a legibilidade e a compreensão por outras pessoas também foram considerados durante a escolha da abordagem. Por fim, uma das equipes fez uma votação entre os seus integrantes para definir a abordagem para documentar a arquitetura, embora o resultado tenha sido unânime (escolha da C4). Dentre as cinco equipes, três escolheram utilizar a abordagem 4+1 e duas decidiram aplicar a abordagem C4 Model.

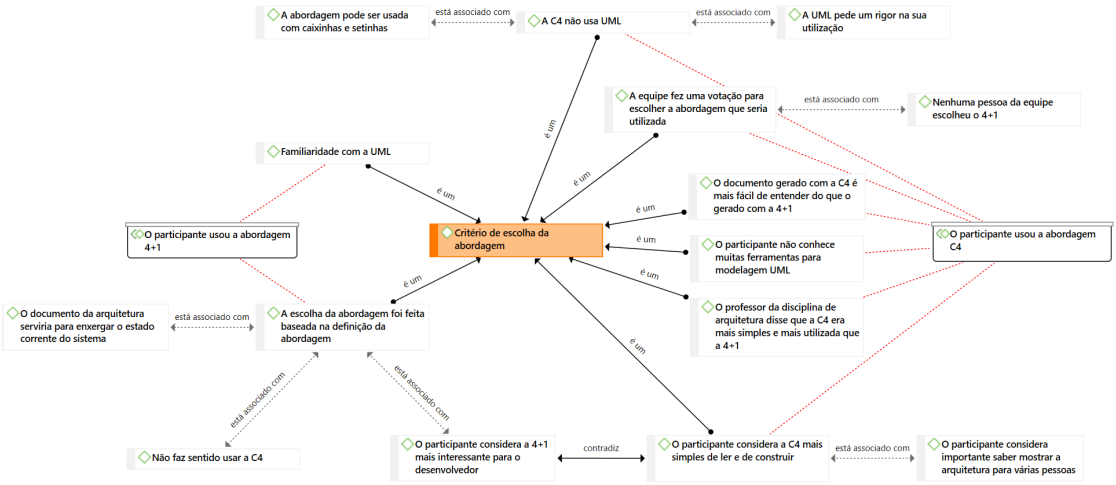


Figura 38 – Códigos relacionados aos critérios de escolha da abordagem utilizada na documentação da arquitetura dos sistemas.

7.2.2 Organização da equipe

Com relação à organização da equipe para a documentação da arquitetura, todas as equipes definiram uma pessoa para realizar essa tarefa, pois pelo menos um participante de cada equipe estava com disponibilidade para documentar a arquitetura. Essa tarefa foi feita com a construção da arquitetura junto à coleta do *feedback* dos colegas de equipe. Os relatos dos participantes P2 e P4 reafirmam como foi feita a documentação da arquitetura do sistema: *"Bom, basicamente a gente definiu uma pessoa, que no caso fui eu e, como o pessoal que está desenvolvendo também já tem um conhecimento ali do sistema em si..."* (P2) e *"Eu fiz e mandei para o pessoal para ver o que eles achavam que eles achavam, se tinham alguma sugestão..."* (P4). Em contrapartida, uma das equipes teve mais de uma pessoa documentando a arquitetura, pois o integrante previamente designado não possuía conhecimento sobre todas as partes do sistema, como pode ser visto no relato do participante P5: *"Eu adicionei nas quatro visões tudo o que eu sabia do back*

e aí eu pedi para Participante 7 complementar o restante das visões, com as informações do front...". Assim, outro integrante da equipe completou a documentação, incluindo o conhecimento sobre o sistema que tinha domínio. A Figura 39 mostra os códigos relacionados à organização da equipe para a documentação do sistema, a partir da análise dos relatos dos participantes do estudo.

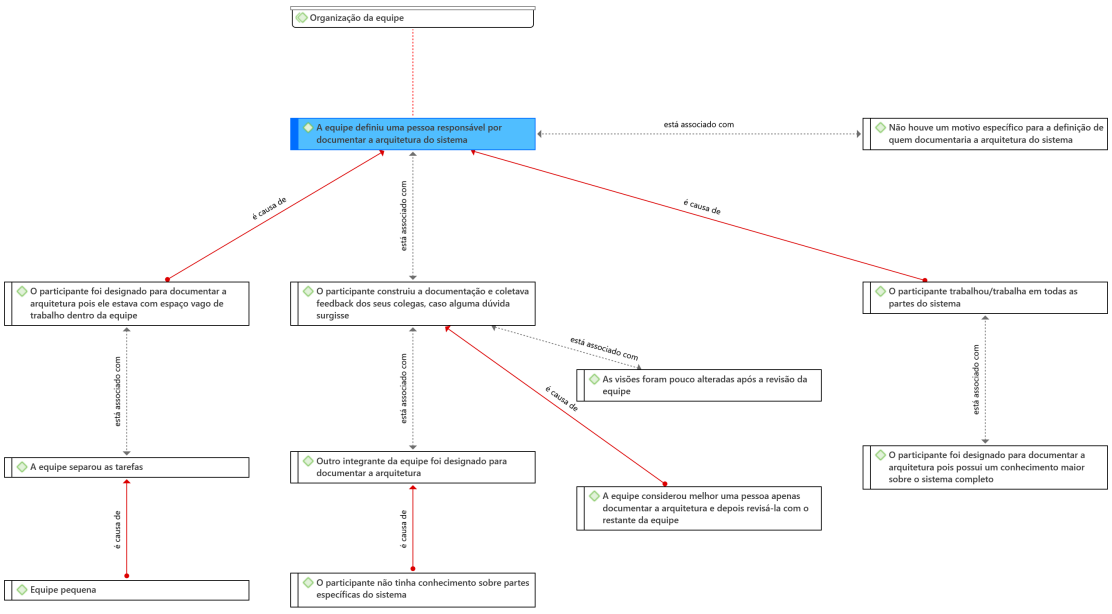


Figura 39 – Códigos relacionados ao processo de organização da equipe para a documentação da arquitetura dos sistemas.

7.2.3 Dificuldades

As dificuldades enfrentadas pelos participantes (que podem ser vistas na Figura 40) estão relacionadas, principalmente, à notação utilizada para documentar a arquitetura e em definir os componentes do sistema. As equipes que utilizaram a abordagem 4+1 e, por consequência, a linguagem UML, precisaram revisar os

conceitos e os diagramas da UML. Isso pode ser visto nos relatos dos participantes P2 e P6: *"eu acho que relembrar grande parte dos diagramas, principalmente, o de sequência"* (P2) e *"A gente teve que revisar bastante coisa, né? ... Por exemplo, diagrama de atividades, não é? Tem aquelas barras que dividem o fluxo, né? Todas essas coisas assim é tipo extensão, diagrama de classe, né? O que que é uma extensão, sobreposição, essas coisas, tudo a gente teve que dar uma revisada para poder refrescar a memória"* (P6). Por outro lado, algumas equipes que utilizaram a abordagem C4 tiveram dificuldade em definir os componentes, como visto no relato dos participantes P4 e P7: *"Uma dificuldade que a gente teve foi nela, um diagrama de componentes, além do de código também, em como definir corretamente componentes, né? Por exemplo, aquela questão de novo, de usar um framework. Em um front end, os componentes não necessariamente estão, assim, na mesma pasta, no mesmo componente, né? Quer dizer, só na mesma pasta, dentro do projeto não é? E a mesma coisa vale para o backend..."* (P4) e *"Quando a gente foi fazer a parte da diagramação lá do front end, eu tive um pouco de dificuldade em definir, assim como o Participante 4, o que de fato, seria um componente ali dentro do framework e tudo mais. Então foi uma dificuldade de fato e também eu não encontrei muito material relacionado."* (P7).

7.2.4 Visões que seriam adotadas

Sobre as visões que foram consideradas mais interessantes (Figura 41) ou que seriam adotadas futuramente (Figura 42), os resultados apontam para um conjunto final composto por algumas visões das duas abordagens, de acordo com os relatos dos participantes. O participante P8, por exemplo, afirma que *"...pela simplicidade tanto de ler quanto de escrever documentação com o C4, eu escolheria ele ...*

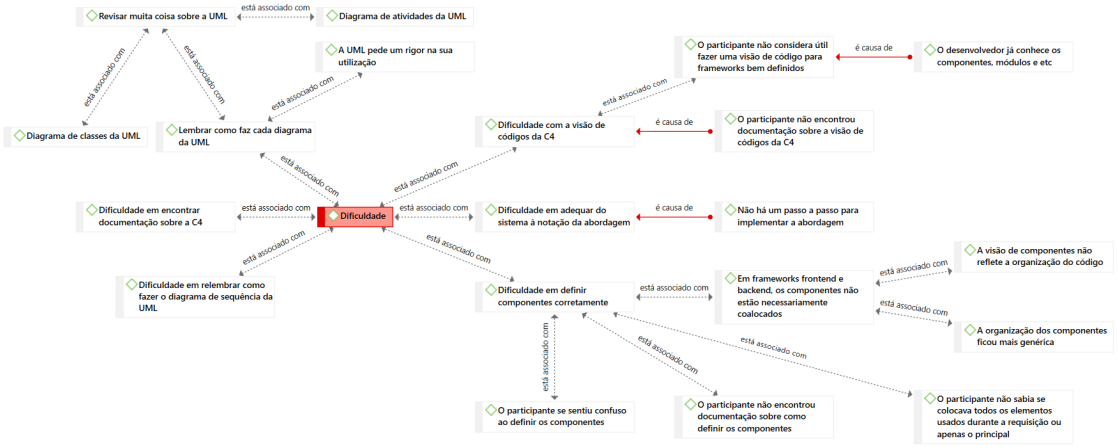


Figura 40 – Códigos relacionados às dificuldades da equipe durante a documentação da arquitetura dos sistemas.

pegaria um pouco da parte física do 4 + 1, né? Que foi uma coisa que a gente sentiu falta quando estava escrevendo a documentação do nosso sistema, então talvez ficaria legal.". O participante P2 escolheria apenas a visão lógica (4+1) pois, no seu contexto de trabalho, *"a gente cria jogos (sic). Tem muitas funções necessariamente que precisam de classe, né?"* e usaria as outras *"... dependendo do que eu faria no sistema."* (P2). Como contraponto à abordagem C4, o participante P4 afirma que *"...escolheria a visão de contexto e de contêiner, porque oferecem uma visão mais alto nível do sistema. É mais fácil uma pessoa ler e entender o sistema nessas visões de mais alto nível do que as outras duas, de componentes e de código."*. Por fim, de forma semelhante ao participante P2, o participante P5 relata que devido ao contexto de testes em que está inserido, ele *"... não está no dia a dia a parte de desenvolver diagrama e tal e mexer com UML. Então, eu ia priorizar as visões do C4, contexto e de contêiner, e do 4+1 pegaria física, por exemplo, porque é uma preocupação..."*.

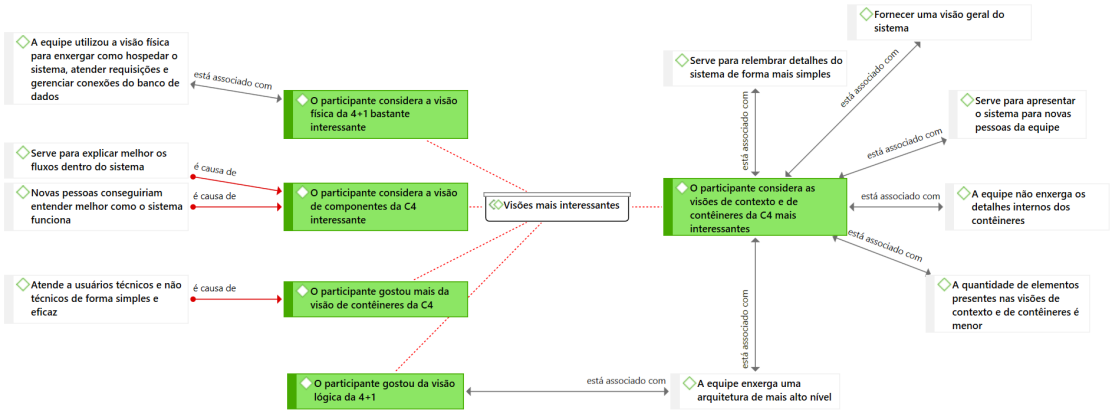


Figura 41 – Visões consideradas mais interessantes para a documentação da arquitetura dos sistemas.

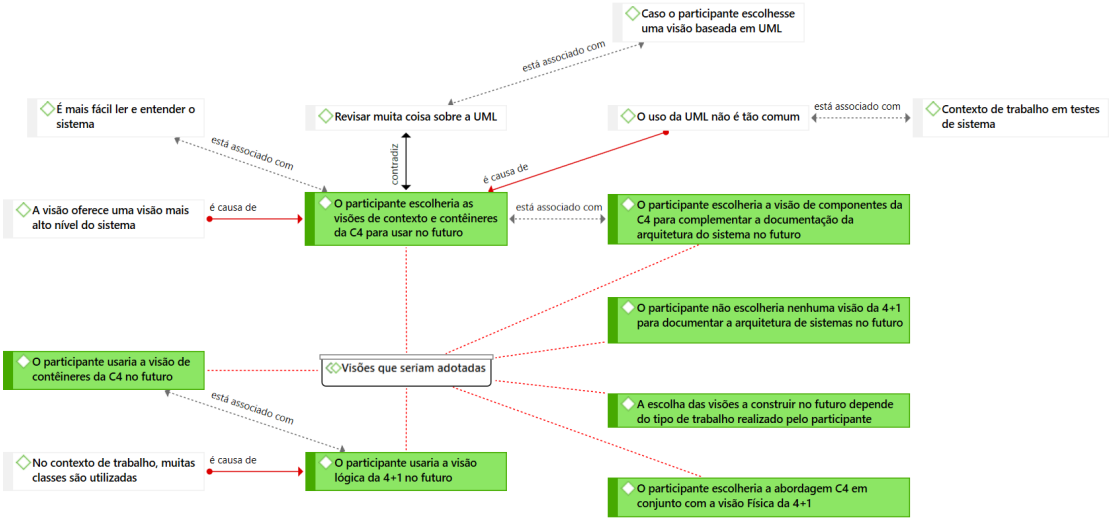


Figura 42 – Visões que seriam adotadas no futuro para a documentação da arquitetura dos sistemas.

7.3 Discussões

A partir dos resultados obtidos, é possível identificar informações sobre a experiência de utilização das abordagens, organização da equipe e critérios utiliza-

dos para escolher a abordagem.

O tamanho reduzido das equipes leva-as à distribuição de tarefas e isso também foi refletido durante a realização do experimento. Todas as equipes definiram uma pessoa para realizar a tarefa de documentação e a escolha foi feita baseada na vacância de trabalho ou no nível de conhecimento sobre o sistema. Além disso, a pessoa escolhida mantinha contato constante com a equipe para tirar dúvidas e validar o desenho que fazia. Dessa forma, a equipe não paralisava o seu processo de desenvolvimento e a tarefa era realizada em paralelo. Essa estratégia pode ter influenciado nos relatos obtidos sobre o impacto no processo de desenvolvimento, relatado por quase todas as equipes como nulo ou quase nulo. Aqueles que afirmaram ter parado de desenvolver para desenhar a arquitetura disseram que a atividade foi rápida e não influenciou nas entregas ou no desenvolvimento do produto. Assim, pode-se inferir que **definir uma pessoa da equipe para realizar a tarefa de documentação da arquitetura** seja uma boa prática, desde que ela tenha espaço de trabalho disponível ou conhecimento aprofundado do sistema. Isso vai de encontro com a característica de equipes novas e pequenas de desenvolvimento de software, em que os membros possuem diversos papéis e trabalham focando na entrega rápida de valor pelo produto de software (KLOTINS et al., 2019). A falta de domínio sobre partes do sistema levou, por exemplo, o participante P5 a pedir ajuda de outras pessoas da equipe, para incluir os conceitos sobre o sistema que elas conheciam mais.

Sobre os critérios de escolha das abordagens, a familiaridade com uma notação pode influenciar na decisão de que ferramenta utilizar. No contexto estudado, o fato de já ter utilizado a UML levou alguns participantes a escolhe-

rem a abordagem 4+1 para documentar a arquitetura de seus sistemas, embora saber das restrições e regras de utilização também foi algo que, de forma contrária, afastou alguns participantes da 4+1 e os fizeram escolher a abordagem C4. Por isso, **o critério de escolha da abordagem de documentação deve ser a experiência prévia com as ferramentas/notações necessárias para uso**.

Com relação às visões mais úteis para documentar a arquitetura em contextos ágeis, entende-se que a melhor opção é criar um conjunto com algumas visões das duas abordagens. A partir da abordagem 4+1, as visões física e lógica foram consideradas mais interessantes. A primeira pode ajudar os profissionais a enxergar como hospedar seus sistemas, atender requisições e gerenciar outros aspectos de implementação. A visão lógica (4+1) pode ser útil em contextos em que existem muitas classes para manter e para enxergar o sistema de forma mais alto nível. De forma semelhante, com relação à abordagem 4+1, as visões de contexto e componentes também foram consideradas úteis para mostrar uma visão geral do sistema. Além disso, a visão de componentes pode servir na exibição dos fluxos dentro do sistema.

Dessa forma, os resultados obtidos neste estudo indicam que o conjunto de visões que apresentam maiores contribuições na documentação da arquitetura em contextos ágeis são:

- Visão Lógica (4+1)
- Visão Física (4+1)
- Visão de Contexto (C4)
- Visão de Contêineres (C4)
- Visão de Componentes (C4)

7.4 Ameaças à Validade

A ameaça à validade deste estudo está ligada à validade de conclusão. As discussões feitas neste estudo são baseadas nos resultados obtidos a partir do relato de 9 (nove) participantes, com pelo menos um participante por equipe. Nesse sentido, caso outro aluno de cada equipe tivesse participado do estudo e realizado a tarefa, os resultados poderiam ter sido diferentes dos apresentados neste estudo. Essa circunstância reflete a realidade de projetos de software, onde existem papéis e responsabilidades definidas. Embora os relatos tenham sido individuais, o contexto de trabalho da equipe se assemelha ao escopo utilizado neste estudo: equipes ágeis de desenvolvimento de software, com entregas rápidas e pouco tempo para desenvolver. Dessa forma, essa circunstância pode ser interpretada como uma ameaça à validade dos resultados, porém é um extrato importante sobre como equipes com características semelhantes se organizam para realizar determinada tarefa.

7.5 Conclusões

Esse estudo avaliou a experiência de uso de abordagens de documentação de arquitetura de software em um contexto de equipes ágeis de desenvolvimento e a forma como essa tarefa foi realizada. Os resultados mostram que essas equipes determinam uma pessoa para documentar a tarefa, geralmente, com espaço vago de trabalho. Dessa forma, a documentação é gerada junto à coleta de *feedback* com os outros colegas de equipe. Para escolher a abordagem, a experiência prévia com notações e ferramentas é levada em consideração. Além disso, um conjunto ideal de visões da arquitetura de software serve para mostrar uma vi-

são geral do sistema, seus fluxos e aspectos de implementação, neste estudo, descrita como a combinação de visões das duas abordagens.

8

Considerações Finais

Esta pesquisa buscou responder à seguinte questão: **quais abordagens de documentação da arquitetura de sistemas são viáveis em contextos ágeis de desenvolvimento de software e como utilizá-las?** Para encontrar possíveis respostas, foram realizados uma revisão da literatura, uma *Feature Analysis*, um estudo *in vitro*, um experimento controlado e um estudo em ciclo de vida, com diferentes objetivos específicos, mas que, respondem à questão de pesquisa e permitem a discussão sobre em que momento utilizar as visões presentes nas abordagens e como fatores externos ao ato de documentar o sistema podem influenciar na realização dessa atividade.

Em primeiro lugar, os resultados da Feature Analysis mostram que as duas abordagens melhor avaliadas, dentre as selecionadas para esta pesquisa, são a abordagem 4+1 e a C4 Model. Além disso, as pontuações dessas abordagens estão muito próximas, o que indicou, de forma inicial, que não há grandes diferenças entre utilizar uma ou a outra durante a documentação da arquitetura de software em contextos ágeis.

O estudo in vitro permitiu avaliar aspectos qualitativos do uso das abor-

dagens 4+1 e C4, durante a documentação de sistemas. A partir dele, foram obtidos indícios sobre a facilidade, a utilidade percebida e a futura adoção de cada visão pelos participantes e, também, a influência que aspectos alheios à construção delas influenciam na sua experiência como o conhecimento sobre o sistema, tomadas de decisão e avaliação de alternativas de mercado, por exemplo

O experimento controlado, cujo objetivo era verificar se há diferença de corretude entre os diagramas presentes nas visões, mostra que não há diferença estatística entre a quantidade de defeitos encontrados ao utilizar a abordagem C4 Model comparada com o uso da abordagem 4+1. Entretanto, foi possível verificar os diferentes tipos de defeitos encontrados e discutir possíveis causas para sua inserção nos diagramas.

Os resultados da análise qualitativa do estudo em ciclo de vida mostram que as visões tanto da abordagem C4 Model quanto da abordagem 4+1 devem ser construídas em momentos diferentes ou nem devem ser utilizadas, em contextos ágeis de desenvolvimento de software. Isso vai depender do nível de maturidade do sistema, da quantidade de conhecimento disponível, assim como da prioridade dada aos diferentes tipos de informação. Além disso, este estudo também explorou de que forma os participantes se organizaram para a realização da documentação da arquitetura dos seus sistemas.

Portanto, ao responder à questão de pesquisa, este trabalho fornece as seguintes contribuições:

- Identificação de abordagens de documentação de arquitetura de software
- Evidências sobre a sua utilização em contextos ágeis de desenvolvimento de software

- Indícios sobre a distribuição da tarefa de documentar a arquitetura de software em contextos ágeis de desenvolvimento.

Por fim, ainda há espaço para pesquisa relacionada à documentação de arquitetura de software, em especial, na indústria, como a replicação dos estudos descritos nesta pesquisa, com diferentes equipes e empresas. Isso servirá para aumentar a escala dos estudos e obter mais respostas para os questionamentos apresentados, visto que os resultados encontrados nesta pesquisa foram obtidos em contextos acadêmicos, o que pode influenciar na validade dos indícios descritos no texto.

A

PLANILHAS GERADAS NA *Feature Analysis*

Neste capítulo, estão as tabelas geradas durante a *Feature Analysis* das seis abordagens, descrita no Capítulo 4.

A.1 Siemens 4 Views

Tabela 16 – Análise da abordagem Siemens 4 Views.

ID	Feature Set	Sub Feature Description	Level of importance	Obs.	Judgment Interpretation
FS1	Documentação	A abordagem deve apoiar o registro do conhecimento sobre a arquitetura do sistema.	Highly Desirable		1
		A abordagem deve permitir a documentação de uma quantidade pequena de elementos no planejamento inicial.	Highly Desirable	Os autores não discutem a quantidade de elementos presentes no planejamento.	0
		A abordagem deve possuir diferentes descrições da arquitetura para diferentes públicos e propósitos.	Nice to Have	Cada uma das 4 visões descritas pelo modelo possui um público-alvo e um propósito específico.	1
		A abordagem deve ser capaz de separar as arquiteturas correntes e futuras.	Nice to Have	Não há essa separação.	0
FS2	Análise de decisões	A abordagem deve facilitar a análise de decisões arquiteturais.	Nice to Have	O modelo não aborda a descrição das decisões arquiteturais.	0
FS3	Sistema	A abordagem deve permitir a modularidade da arquitetura do sistema.	Desirable	A abordagem permite a construção de sistemas modulares e a sua descrição na "arquitetura de módulos"	1
FS4	Requisitos	A abordagem deve ser flexível e adaptável às mudanças de requisitos e decisões arquiteturais.	Mandatory	A arquitetura de execução é a que tende a ser modificada de forma mais constante, para comportar a evolução do software e do hardware.	1
FS5	Custo	A abordagem deve permitir a construção da arquitetura com a menor quantidade de tarefas obrigatórias.	Mandatory	O modelo não descreve quais tarefas devem ser feitas para criar as descrições da arquitetura. Nesse caso, conta-se quantas visões obrigatórias são propostas. No modelo S4V, são 4 visões.	5

A.2 4+1

Tabela 17 – Análise da abordagem 4+1.

ID	Feature Set	Sub Feature Description	Level of importance	Obs.	Judgment Interpretation
FS1	Documentação	A abordagem deve apoiar o registro do conhecimento sobre a arquitetura do sistema.	Highly Desirable		1
		A abordagem deve permitir a documentação de uma quantidade pequena de elementos no planejamento inicial.	Highly Desirable	O autor apresenta uma abordagem baseada nos cenários que, de forma cíclica, gera as visões para um pequeno conjunto de cenários, valida com as partes interessadas e, a partir das lições aprendidas, repete o ciclo atendendo aos cenários ainda não tratados.	1
		A abordagem deve possuir diferentes descrições da arquitetura para diferentes públicos e propósitos.	Nice to Have	Cada uma das 5 visões possui um público-alvo e um propósito diferente em relação à arquitetura.	1
		A abordagem deve ser capaz de separar as arquiteturas correntes e futuras.	Nice to Have	Não há essa separação no modelo 4+1	0
FS2	Análise de decisões	A abordagem deve facilitar a análise de decisões arquiteturais.	Nice to Have	A visão de cenários serve para juntar as outras descrições e verificar se o que foi planejado atende o que se pede no cenário estudado. É uma forma de analisar as decisões arquiteturais. Além disso, o documento de diretrizes de projeto armazena as principais decisões tomadas que devem ser respeitadas para manter a integridade do sistema.	1
FS3	Sistema	A abordagem deve permitir a modularidade da arquitetura do sistema.	Desirable	A abordagem permite a descrição de arquiteturas modulares através da visão de desenvolvimento, ainda que de forma hierárquica.	1
FS4	Requisitos	A abordagem deve ser flexível e adaptável às mudanças de requisitos e decisões arquiteturais.	Mandatory	O autor afirma que o modelo deve ser aplicado de forma iterativa pois, após a criação das visões, pouco se sabe ainda para validar a arquitetura. Em cada iteração, será possível inserir/remover elementos da arquitetura e atender às mudanças que aparecerem.	1
FS5	Custo	A abordagem deve permitir a construção da arquitetura com a menor quantidade de tarefas obrigatórias.	Mandatory	O autor, no artigo original que apresenta o modelo, afirma que as funcionalidades mais importantes do sistema são capturadas na forma de cenários. Nesse sentido, ele descreve uma série de passos a seguir para construir a arquitetura usando o 4+1. São eles: - Um pequeno conjunto de cenários é definido, baseado nos riscos e na importância deles. Eles podem ser sintetizados em requisitos de usuário. - Uma arquitetura com bonecos de palito é criada e os cenários são descritos para identificar as principais abstrações (classes, mecanismos, processos, subsistemas, etc). - Os elementos da arquitetura descobertos são dispostos nas 4 visões: lógica, processos, desenvolvimento e física. - A arquitetura gerada é implementada, testada e mensurada. Sua análise deve ajudar a detectar melhorias e falhas que devem ser corrigidas. Além disso, as lições aprendidas devem ser registradas. São, ao todo, 4 atividades.	5

A.3 BAPO/CAFCR

Tabela 18 – Análise da abordagem BAPO/CAFCR.

ID	Feature Set	Sub Feature Description	Level of importance	Obs.	Judgment Interpretation
FS1	Documentação	A abordagem deve apoiar o registro do conhecimento sobre a arquitetura do sistema.	Highly Desirable		1
		A abordagem deve permitir a documentação de uma quantidade pequena de elementos no planejamento inicial.	Highly Desirable	Os autores afirmam que apenas os cenários principais podem ser utilizados para construir o planejamento inicial.	1
		A abordagem deve possuir diferentes descrições da arquitetura para diferentes públicos e propósitos.	Nice to Have	O modelo possui cinco visões diferentes. Cada uma delas descreve elementos diferentes, para diversos públicos e com propósitos variados.	1
		A abordagem deve ser capaz de separar as arquiteturas correntes e futuras.	Nice to Have	Não há a exata divisão de arquitetura atual e futura.	0
FS2	Análise de decisões	A abordagem deve facilitar a análise de decisões arquiteturais.	Nice to Have	A análise das decisões pode ser feita através das descrições sugeridas na visão conceitual do sistema e na variações das visões.	1
FS3	Sistema	A abordagem deve permitir a modularidade da arquitetura do sistema.	Desirable	A modularidade pode ser descrita na visão conceitual proposta pela abordagem.	1
FS4	Requisitos	A abordagem deve ser flexível e adaptável às mudanças de requisitos e decisões arquiteturais.	Mandatory	A abordagem possui variações de cada uma das visões para atender a possíveis mudanças. Um exemplo é a variação da visão de aplicação.	1
FS5	Custo	A abordagem deve permitir a construção da arquitetura com a menor quantidade de tarefas obrigatórias.	Mandatory	Os autores não definem tarefas a ser seguidas para construir a arquitetura. Nesse caso, contam-se as visões obrigatórias apresentadas. Nesse exemplo, são 5 visões (comercial, aplicação, funcional, conceitual, realização).	4

A.4 ADD

Tabela 19 – Análise da abordagem ADD.

ID	Feature Set	Sub Feature Description	Level of importance	Obs.	Judgment Interpretation
FS1	Documentação	A abordagem deve apoiar o registro do conhecimento sobre a arquitetura do sistema.	Highly Desirable		1
		A abordagem deve permitir a documentação de uma quantidade pequena de elementos no planejamento inicial.	Highly Desirable	O primeiro passo é verificar se os stakeholders priorizaram os requisitos e se os atributos de qualidade estão definidos. Ainda assim, os autores afirmam que quase nunca isso será possível. Portanto, o arquiteto deve trabalhar com a lista de requisitos que possuir no momento.	0,5
		A abordagem deve possuir diferentes descrições da arquitetura para diferentes públicos e propósitos.	Nice to Have	Os autores não citam a separação de públicos e propósitos para os documentos.	0
		A abordagem deve ser capaz de separar as arquiteturas correntes e futuras.	Nice to Have	O modelo não possui uma separação da arquitetura corrente e a futura.	0
FS2	Análise de decisões	A abordagem deve facilitar a análise de decisões arquiteturais.	Nice to Have	A análise das decisões (tecnologia, arquitetura, etc) são solicitadas durante todas as etapas do ADD.	1
FS3	Sistema	A abordagem deve permitir a modularidade da arquitetura do sistema.	Desirable	Os autores recomendam a criação de visões de módulos do sistema. Dessa forma, a abordagem premite a modularidade do sistema.	1
FS4	Requisitos	A abordagem deve ser flexível e adaptável às mudanças de requisitos e decisões arquiteturais.	Mandatory	Os autores não restringem a arquitetura para a descrita de forma inicial. Além disso, as análises das decisões durante as iterações permitem que o sistema seja alterado, se necessário.	1
FS5	Custo	A abordagem deve permitir a construção da arquitetura com a menor quantidade de tarefas obrigatórias.	Mandatory	O modelo possui um conjunto de 8 etapas que devem ser seguidas a cada iteração. São elas: 1 - Priorização dos requisitos por parte dos stakeholders, de acordo com os objetivos de negócio. 2 - Confirmar se há informações suficientes para a construção da arquitetura (ranquear todos os requisitos, considerar os elementos arquiteturais em ordem de importância para os stakeholders). 3 - Escolher um elemento do sistema para decompor baseado no conhecimento corrente da arquitetura, riscos e dificuldades, critérios de negócio e organizacionais. 4 - Identificar os requisitos arquiteturais que direcionam o desenvolvimento (drivers) (ordenar os aspectos da arquitetura que têm grande impacto e são importantes para as partes interessadas). 5 - Escolher um conceito de projeto que satisfaz os drivers arquiteturais e avaliar e resolver inconsistências no conceito de projeto. 6 - Instanciar elementos arquiteturais e alocar responsabilidades (quantos elementos devem ser instanciados, quais são as relações estruturais que eles possuem, como deve ser disposta a infraestrutura, comunicação entre elementos, recursos). Isso deve ser disposto em 3 visões: módulos, componente-conector, alocação. 7 - Definir interfaces para os elementos instanciados (exercitar os requisitos funcionais instanciados no passo 6, observar as informações produzidas e consumidas pelos elementos, registrar os achados em uma documentação de interface para cada elemento) 8 - Verificar e refinar os requisitos, tornando restrições para os elementos instanciados.	2

A.5 C3A

Tabela 20 – Análise da abordagem C3A.

ID	Feature Set	Sub Feature Description	Level of importance	Obs.	Judgment Interpretation
FS1	Documentação	A abordagem deve apoiar o registro do conhecimento sobre a arquitetura do sistema.	Highly Desirable		1
		A abordagem deve permitir a documentação de uma quantidade pequena de elementos no planejamento inicial.	Highly Desirable	O modelo foi proposto para construir a arquitetura do sistema com uma quantidade mínima de elementos.	1
		A abordagem deve possuir diferentes descrições da arquitetura para diferentes públicos e propósitos.	Nice to Have	Existem duas visões principais: a de referência, que mostra uma visão geral do sistema e a de implementação, que mostra os novos lançamentos e versões menores, além de como desenvolver o sistema. Cada uma delas serve a um público-alvo diferente.	1
		A abordagem deve ser capaz de separar as arquiteturas correntes e futuras.	Nice to Have	A arquitetura corrente é a de referência. A arquitetura futura é descrita na de implementação.	1
FS2	Análise de decisões	A abordagem deve facilitar a análise de decisões arquiteturais.	Nice to Have	As decisões arquiteturais são descritas no contrato de cada elemento dos níveis 0 e 1 da arquitetura.	1
FS3	Sistema	A abordagem deve permitir a modularidade da arquitetura do sistema.	Desirable	A abordagem permite a descrição de módulos na arquitetura nível 0.	1
FS4	Requisitos	A abordagem deve ser flexível e adaptável às mudanças de requisitos e decisões arquiteturais.	Mandatory	As mudanças nos requisitos e nas decisões são tratadas na arq. de implementação e, no futuro, integradas à arq. de referência.	1
FS5	Custo	A abordagem deve permitir a construção da arquitetura com a menor quantidade de tarefas obrigatórias.	Mandatory	O passo a passo descrito pelos autores possui 7 etapas. São elas: 1 - Coletar documentos arquiteturais, publicações técnicas, manuais e conhecimento tácito. 2 - Capturar a primeira arquitetura em uma arquitetura de referência, com a definição dos componentes nível 0. 3 - Validar o projeto criado até o momento. 4 - Detalhar o status dos módulos da próxima arquitetura de referência, mantendo a RA corrente para prevenir que ele seja impactado por componentes incertos. 5 - Para cada módulo nível 0, definir os componentes nível 1. 6 - Mapear ou prover a análise das lacunas entre os próximos lançamentos e a arquitetura atual. 7 - Modificar os planejamentos de lançamento para se adequar à arquitetura atual, executar e implementar.	3

A.6 C4

Tabela 21 – Análise da abordagem C4.

ID	Feature Set	Sub Feature Description	Level of importance	Obs.	Judgment Interpretation
FS1	Documentação	A abordagem deve apoiar o registro do conhecimento sobre a arquitetura do sistema.	Highly Desirable	Sim	1
		A abordagem deve permitir a documentação de uma quantidade pequena de elementos no planejamento inicial.	Highly Desirable	Com as visões de contexto e container, a abordagem permite o desenho inicial do sistema, sem tantos detalhes específicos.	1
		A abordagem deve possuir diferentes descrições da arquitetura para diferentes públicos e propósitos.	Nice to Have	Cada visão é uma abstração mais profunda da anterior. Dessa forma, diferentes públicos e propósitos são atendidos.	1
		A abordagem deve ser capaz de separar as arquiteturas correntes e futuras.	Nice to Have	Não há essa separação.	0
FS2	Análise de decisões	A abordagem deve facilitar a análise de decisões arquiteturais.	Nice to Have	O modelo em si não. O autor sugere e suporta o uso de ADL em sua ferramenta Structurizr.	0
FS3	Sistema	A abordagem deve permitir a modularidade da arquitetura do sistema.	Desirable	A visão de containers contém a separação e organização do sistema em módulos.	1
FS4	Requisitos	A abordagem deve ser flexível e adaptável às mudanças de requisitos e decisões arquiteturais.	Mandatory	O autor afirma que o diagrama que será alterado mais constantemente é o de componentes, à medida que o sistema é construído.	1
FS5	Custo	A abordagem deve permitir a construção da arquitetura com a menor quantidade de tarefas obrigatórias.	Mandatory	O autor não define tarefas que devem ser seguidas para construir a arquitetura. Nesse caso, contam-se as visões obrigatórias. O modelo possui 4 visões mas o autor afirma que apenas duas visões (contexto e contêineres) são obrigatórias.	6

B

MODELAGEM DE UM SISTEMA COM AS ABORDAGENS ANALISADAS

Neste capítulo, estão as modelagens de um sistema, com as seis abordagens avaliadas no estudo descrito no Capítulo 4. Os requisitos utilizados para embasar a modelagem estão descritos na Seção B.1 e foram extraídos de um repositório no GitHub[1].

B.1 Requisitos utilizados na modelagem

- RF1 – O sistema deve ser capaz de intermediar compras entre os usuários e os restaurantes
- RF2 – O sistema deve ser capaz de permitir o cadastro dos seus clientes
- RF3 – O sistema deve listar os restaurantes de acordo com a sua localização inserida

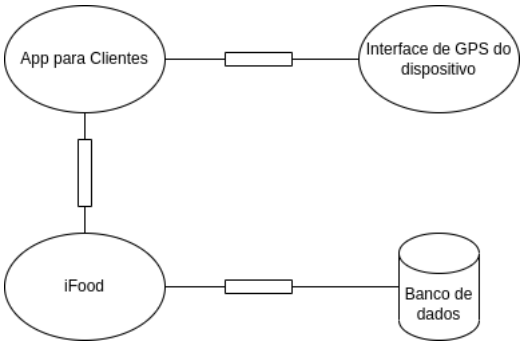
[1] Requisitos iFood. Disponível em: <https://github.com/Requisitos-2018-1-iFood/>

- RF4 – O sistema deve listar o cardápio do restaurante
- RF5 – O sistema deve permitir que os restaurantes modifiquem seus cardápios
- RF7 – O sistema deve identificar a disponibilidade do restaurante
- RF21 – O usuário deve ser capaz de editar seus dados
- RF25 – O usuário deve ser capaz de visualizar o status do pedido em: Realizado, Não Realizado
- RF27 – O sistema deve mostrar o "carrinho", contendo a lista de comidas selecionadas e a opção de adicionar mais itens.
- RF28 – Ao visualizar o carrinho, o sistema deve mostrar a opção de confirmação do pedido.
- RF32 – Ao realizar o pedido, o sistema deve permitir ao usuário incluir adicionais ao seu pedido.
- RF33 – Antes de adicionar ao carrinho, o sistema deve permitir a inclusão de observações ao pedido.
- RF34 – O sistema deve permitir a alteração de quantidades dos itens pedidos no carrinho
- RF37 – O sistema deve permitir ao restaurante incluir imagem e descrição dos pratos ofertados.
- RF43 – O sistema deve permitir ao usuário o login.

B.2 Siemens 4 Views

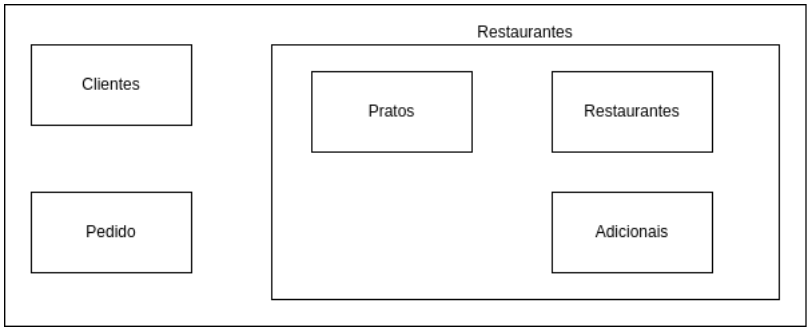
B.2.1 Visão Conceitual

Figura 43 – Visão conceitual (S4V) do sistema exemplo modelado.



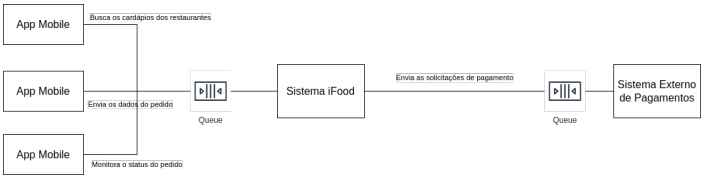
B.2.2 Visão de Módulos

Figura 44 – Visão de módulos (S4V) do sistema exemplo modelado.



B.2.3 Visão de Execução

Figura 45 – Visão de execução (S4V) do sistema exemplo modelado.



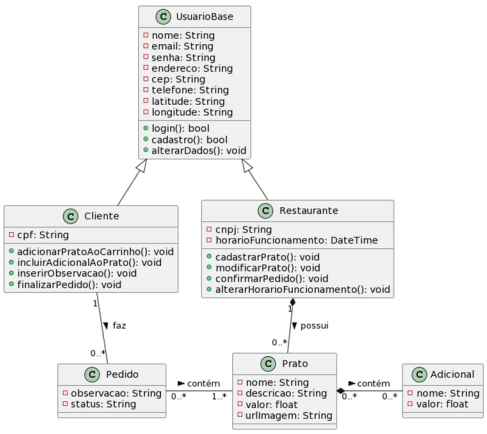
B.2.4 Visão de Código

A visão de código não foi gerada, pois o sistema não foi implementado e não seria possível desenhar a estrutura do código.

B.3 4+1

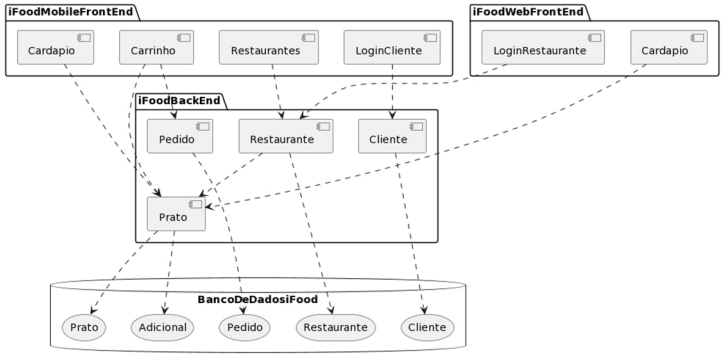
B.3.1 Visão Lógica

Figura 46 – Visão lógica (4+1) do sistema exemplo modelado.



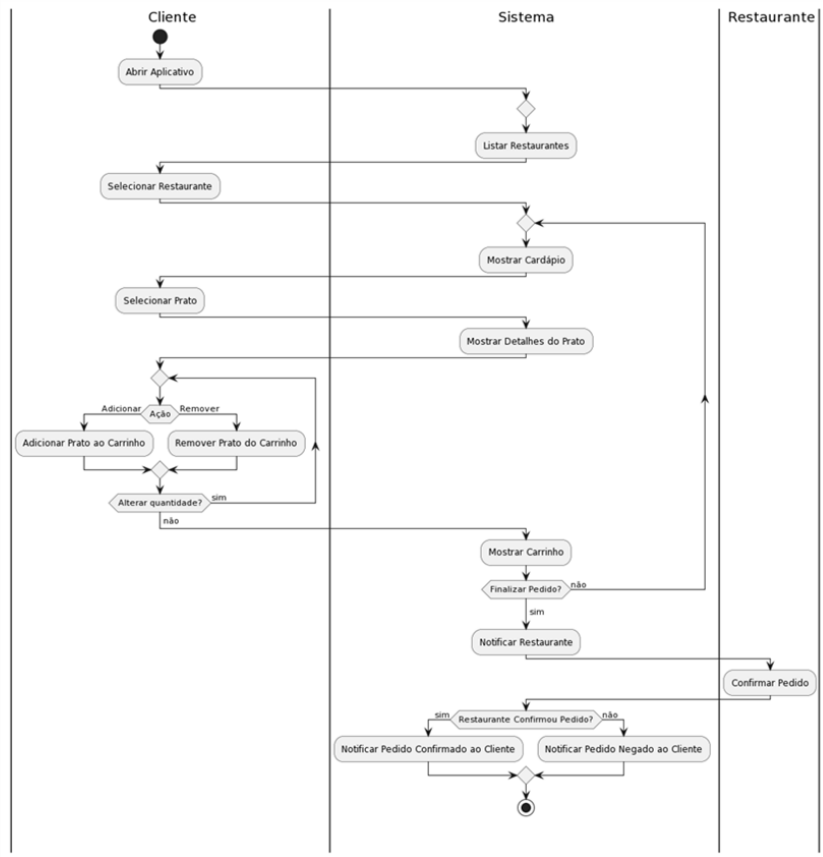
B.3.2 Visão de Desenvolvimento

Figura 47 – Visão de desenvolvimento (4+1) do sistema exemplo modelado.



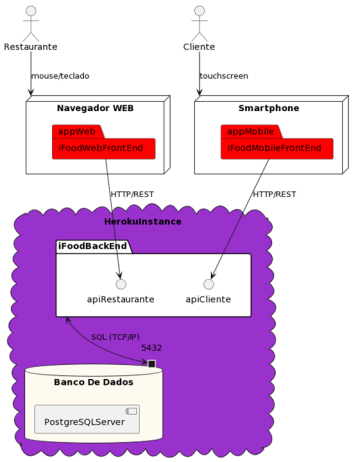
B.3.3 Visão de Processos

Figura 48 – Visão de processos (4+1) do sistema exemplo modelado.



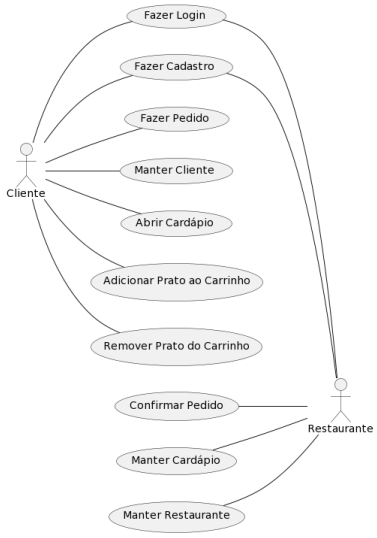
B.3.4 Visão Física

Figura 49 – Visão física (4+1) do sistema exemplo modelado.



B.3.5 Visão de Cenários

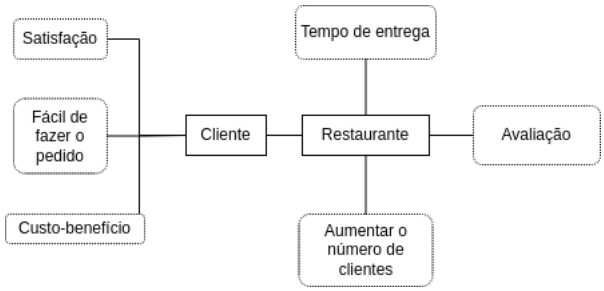
Figura 50 – Visão de cenários (4+1) do sistema exemplo modelado.



B.4 BAPO/CAFCR

B.4.1 Visão Comercial

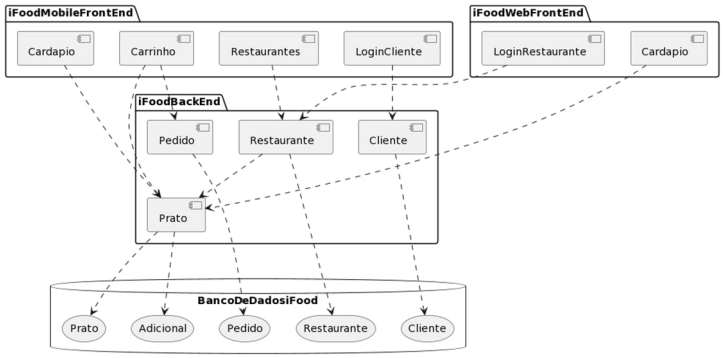
Figura 51 – Visão comercial (BAPO/CAFCR) do sistema exemplo modelado.



B.4.2 Visão de Aplicação

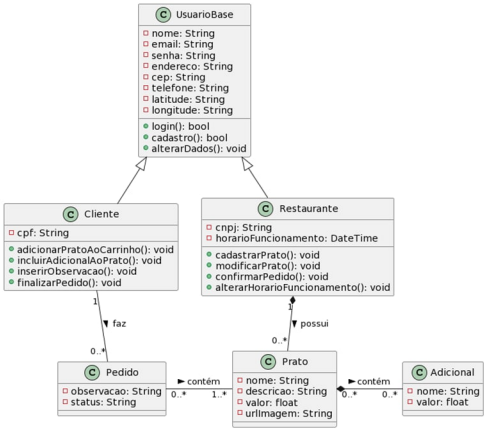
B.4.2.1 Contexto do sistema

Figura 52 – Visão de contexto do sistema, dentro da visão de aplicação (BAPO/CAFCR) do sistema exemplo modelado.



B.4.2.2 Modelo de domínios

Figura 53 – Visão de modelo de domínios, dentro da visão de aplicação (BAPO/CAFCR) do sistema exemplo modelado.



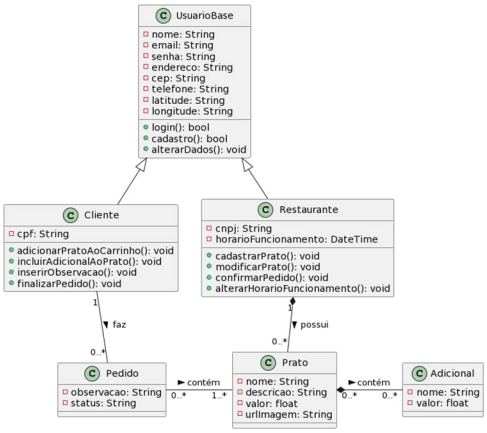
B.4.3 Visão Funcional

Tabela 22 – Matriz de Funcionalidade/Valor que integra a visão funcional (BAPO/CAFCR) do sistema exemplo modelado.

	Tempo de entrega	Avaliação	Tempo de preparação	Número de clientes	Facilidade de uso
Pedido de lanche por aplicativo				++	+
Avaliação de pratos				+	
Custo-benefício		+/-	+/-		+/-

B.4.4 Visão Conceitual

Figura 54 – Visão conceitual (BAPO/CAFCR) do sistema exemplo modelado.



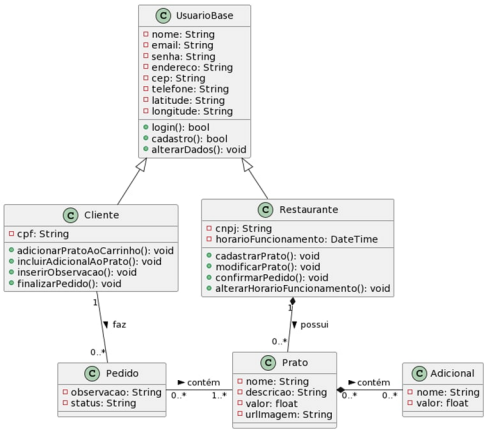
B.4.5 Visão de Realização

Essa visão possui um mapeamento das tecnologias que serão utilizadas na construção do sistema. Neste caso, as tecnologias utilizadas são Javascript com NodeJS, PostgreSQL, ReactJS, Kotlin e Swift.

B.5 Attribute Driven Design

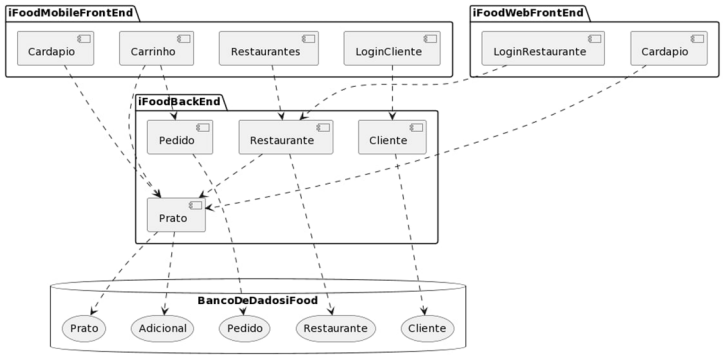
B.5.1 Visão de Módulos

Figura 55 – Visão de módulos (ADD) do sistema exemplo modelado.



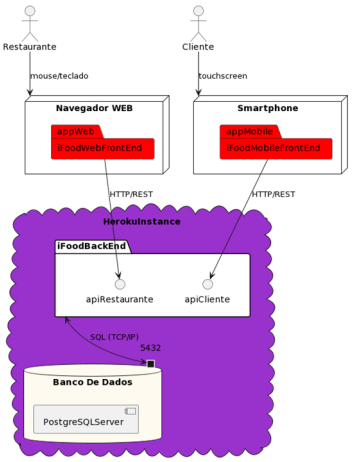
B.5.2 Visão de Componente-conector

Figura 56 – Visão de componente-conector (ADD) do sistema exemplo modelado.



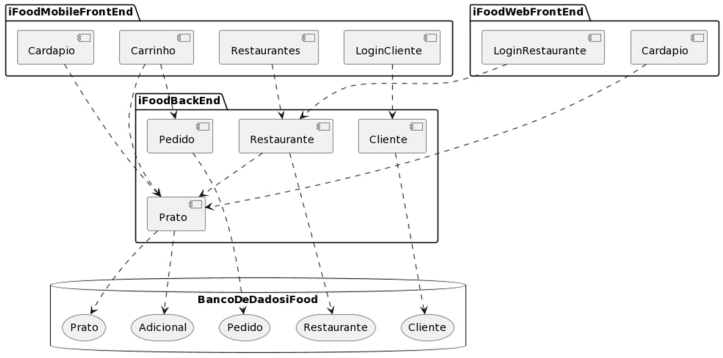
B.5.3 Visão de Alocação

Figura 57 – Visão de alocação (ADD) do sistema exemplo modelado.



B.6 C3A Agile Architecture

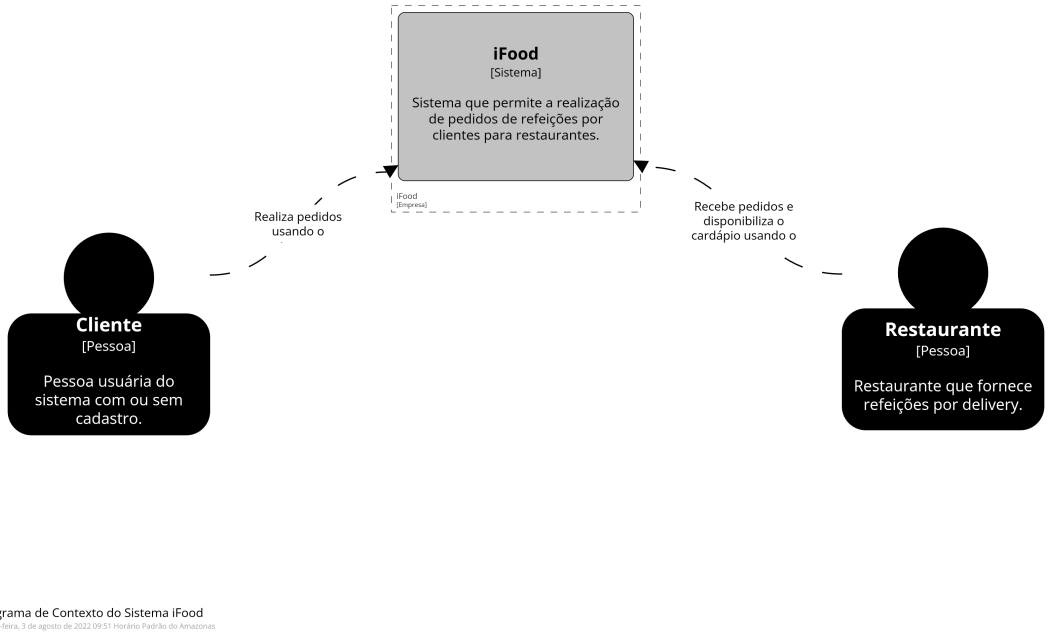
Figura 58 – Visão da arquitetura de referência (C3A) do sistema exemplo modelado. Os pacotes são os elementos nível 0 (módulos) e os componentes, os elementos nível 1 (componentes).



B.7 C4 Model

B.7.1 Visão de Contexto

Figura 59 – Visão de contexto de sistema (C4) do sistema exemplo modelado.



B.7.2 Visão de Contêineres

Figura 60 – Visão de contêineres (C4) do sistema exemplo modelado.

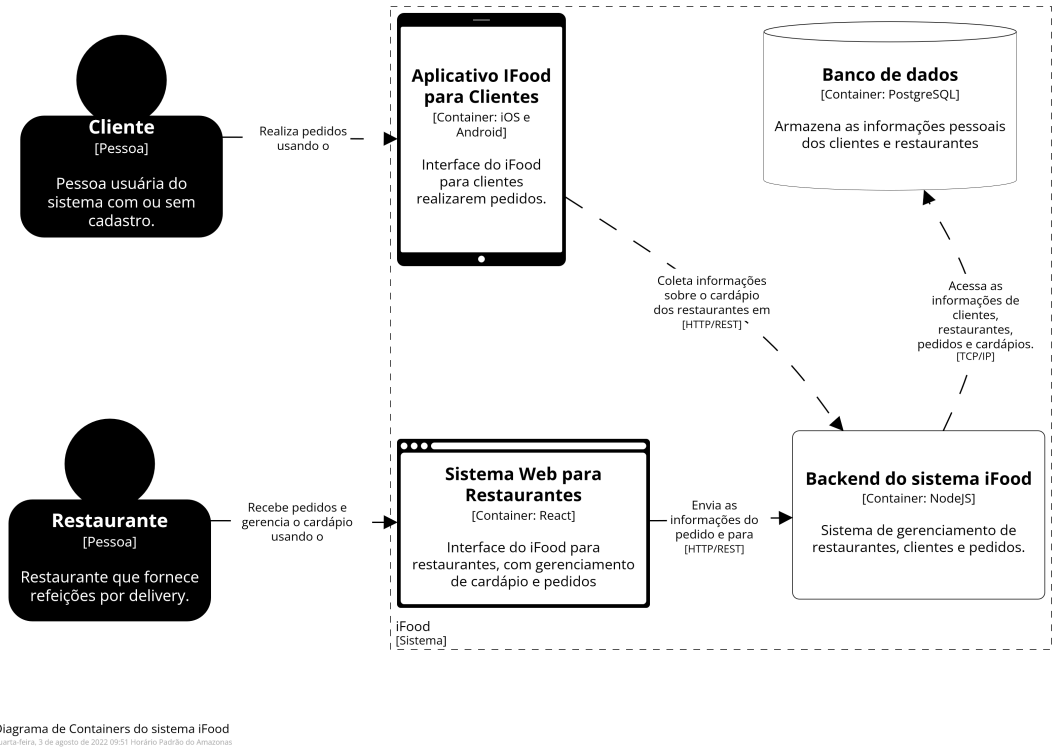
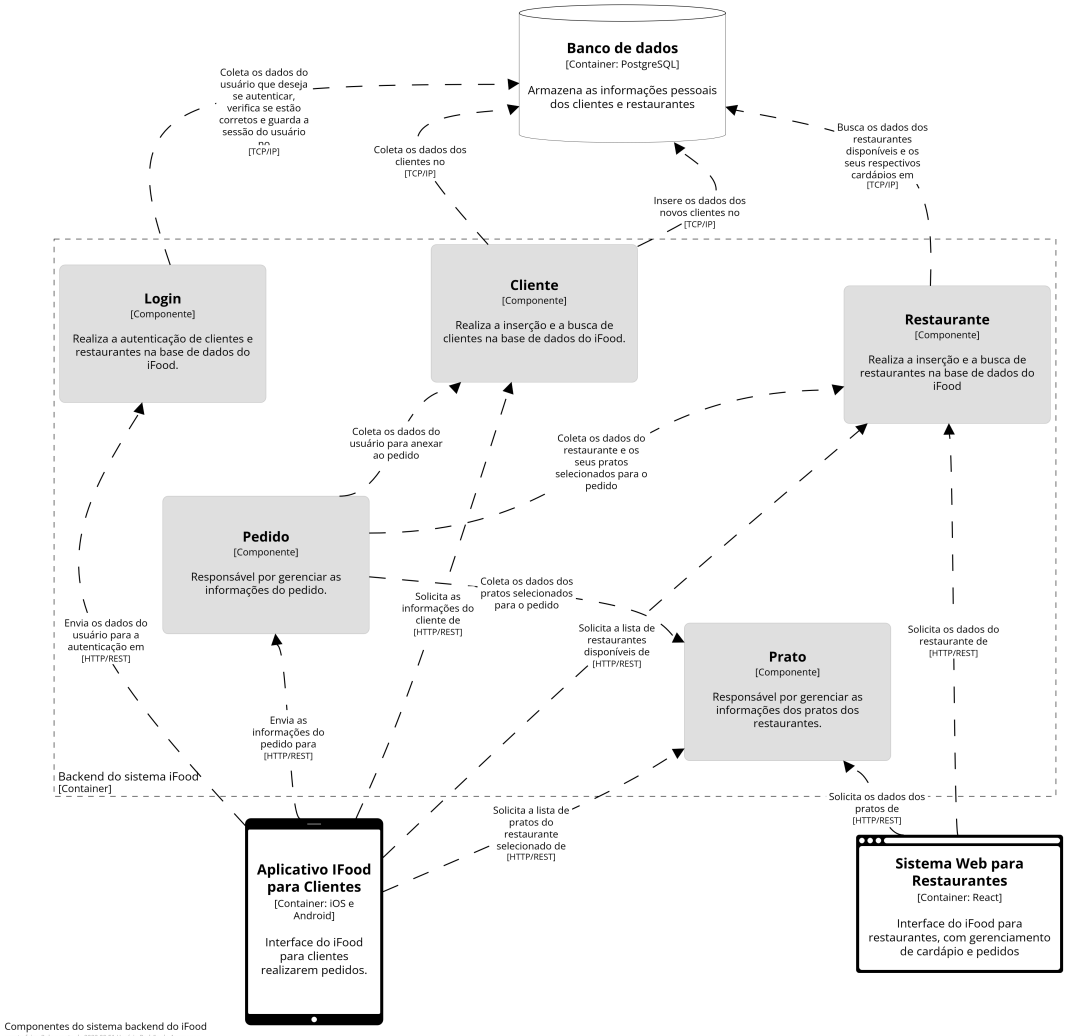


Diagrama de Containers do sistema iFood
quarta-feira, 3 de agosto de 2022 09:51 Horário Padrão do Amazonas

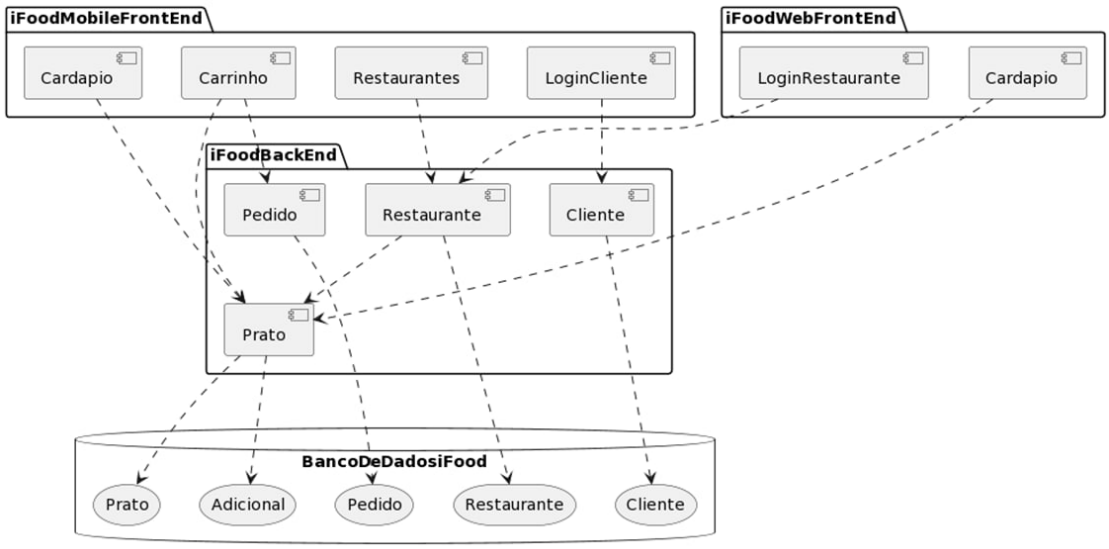
B.7.3 Visão de Componentes

Figura 61 – Visão de componentes (C4) do sistema exemplo modelado.



B.7.4 Visão de Código

Figura 62 – Visão de código (C4) do sistema exemplo modelado.



C

INSTRUMENTOS UTILIZADOS NO EXPERIMENTO CONTROLADO

C.1 Especificações de sistemas

Nesta seção, estão as especificações dos sistemas entregues aos participantes do estudo para a realização da tarefa (documentar a arquitetura dos sistemas).

C.1.1 Sistema AgendaHotel

C.1.1.1 Perspectiva de produto

O sistema a ser projetado se chama AgendaHotel. O sistema a ser projetado será utilizado para gerenciar um serviço de reserva de quartos de hotel.

C.1.1.2 Interfaces

- Usuário – Para o hotel e para o cliente - sistema Web, que será acessado através do navegador do computador.
- Hardware – Computador com teclado, mouse e monitor; Notebook
- Software – Banco de dados relacional; Navegador WEB
- Sistema operacional – PC: Windows, Linux, MacOS
- Comunicação – Conexão com a internet TCP/IP
- Memória – Servidor para armazenamento dos dados do sistema

C.1.1.3 Funcionalidades

- O sistema deve permitir ao cliente o cadastro e edição dos seus dados (nome, email, senha, CPF, telefone).
- O sistema deve permitir ao hotel o cadastro e edição dos seus dados (nome, endereço, cidade, estado, telefone, CNPJ) e dos dados dos quartos (nome, descrição, imagem, preço, itens disponíveis).
- O sistema deve permitir o login ao cliente e ao hotel.
- O sistema deve listar para o cliente os hotéis de acordo com a localização inserida no campo de pesquisa (cidade, estado).
- O sistema deve listar para o cliente os quartos disponíveis de um hotel selecionado.

- O sistema deve permitir ao hotel gerenciar os quartos disponíveis para reserva.
- O sistema deve identificar e exibir para o cliente a disponibilidade do quarto (disponível, indisponível).
- O sistema deve permitir ao cliente escrever avaliações sobre a reserva, o quarto e o hotel.
- O sistema deve permitir ao hotel gerenciar, aprovar ou reprovar as reservas solicitadas.
- O sistema deve permitir ao cliente/visitante reservar um quarto para um determinado intervalo de datas.
- O sistema deve para o cliente mostrar a reserva, contendo o nome, a descrição do quarto, o intervalo de datas que o cliente reservou, os itens disponíveis no quarto e o status da reserva em: Solicitada, Aprovada, Não Aprovada.
- O sistema deve listar ao hotel os dados do cliente que solicitou uma determinada reserva (nome e telefone)

C.1.1.4 Restrições

- O sistema deve se mostrar disponível 24 horas por dia, 7 dias na semana.
- O cliente só poderá fazer uma reserva se estiver cadastrado no sistema.
- O sistema deve ser responsável pela segurança e privacidade dos dados dos clientes.

- O pagamento será feito pessoalmente no hotel, no momento do check-in.
- O sistema deve informar o cliente sobre eventuais falhas na conexão.
- O sistema deve permitir ao cliente fazer reservas em qualquer lugar.
- Apenas hotéis com CNPJ válidos poderão se cadastrar.

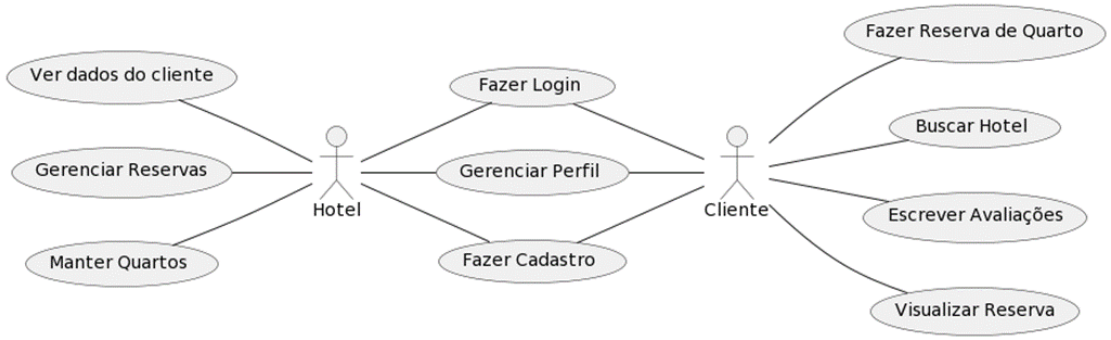


Figura 63 – Diagrama de Casos de Uso do sistema AgendaHotel, fornecido aos participantes do estudo experimental quantitativo.

C.1.2 Sistema Delivery

C.1.2.1 Perspectiva de produto

O sistema a ser projetado se chama myFood. O sistema será utilizado para gerenciar um serviço de delivery para restaurantes e lanchonetes.

C.1.2.2 Interfaces

- Sistema – GPS do smartphone do cliente

- Usuário – Para o restaurante - sistema Web, que será acessado através do navegador do computador; Para o cliente - aplicativo de celular
- Hardware – Computador com teclado, mouse e monitor; Notebook; Smartphone
- Software – Banco de dados relacional; Navegador WEB
- Sistema operacional – PC: Windows, Linux, MacOS; Smartphone: iOS, Android
- Comunicação – Conexão com a internet TCP/IP
- Memória – Servidor para armazenamento dos dados do sistema

C.1.2.3 Funcionalidades

- O sistema deve permitir ao cliente o login, o cadastro e a edição de seus dados (nome, CPF, email, senha, telefone, endereço, cidade, estado).
- O sistema deve permitir ao restaurante o login, o cadastro, a edição de seus dados (nome, telefone, endereço, cidade, estado, CNPJ, horario de funcionamento) e a manutenção dos pratos (nome, descrição, preço, imagem) do cardápio.
- O sistema deve listar os restaurantes para o cliente, de acordo com a localização inserida no campo de pesquisa.
- O sistema deve listar os pratos para o cliente, de acordo com a palavra-chave inserida no campo de pesquisa.

- O sistema deve identificar a disponibilidade do restaurante, de acordo com o horário de funcionamento.
- O sistema deve permitir ao cliente fazer um pedido de prato(s) e escolher a localização de entrega do pedido: Usar a localização atual (GPS); Inserir a localização manualmente (CEP e número); Usar o endereço cadastrado pelo cliente.
- O sistema deve permitir ao cliente escrever avaliações sobre o pedido e sobre o restaurante.
- O sistema deve permitir ao cliente visualizar o status do pedido em: Realizado, Aguardando Aprovação, Não Realizado.
- O sistema deve permitir ao cliente visualizar "o carrinho", contendo a lista de comidas selecionadas e a opção de alterar os itens do carrinho ou confirmar/finalizar o pedido.
- O sistema deve listar, para o restaurante, os dados do cliente de um determinado pedido (nome, telefone e endereço).
- O sistema deve permitir ao restaurante a aprovação e o gerenciamento dos pedidos feitos.

C.1.2.4 Restrições

- O sistema deve se conectar com o sistema de localização do aparelho do cliente.
- O sistema deve se mostrar disponível 24 horas por dia, 7 dias na semana.

- O cliente só poderá fazer um pedido se estiver cadastrado no sistema.
- O sistema deve ser responsável pela segurança e privacidade dos dados bancários dos clientes.
- O pagamento do pedido deve ser realizado com o entregador (dinheiro ou cartão).
- O sistema deve informar o cliente sobre eventuais falhas na conexão.
- O sistema deve permitir ao cliente realizar pedidos em qualquer lugar.
- O restaurante deve ter sistema próprio de entregas e CNPJ válido.

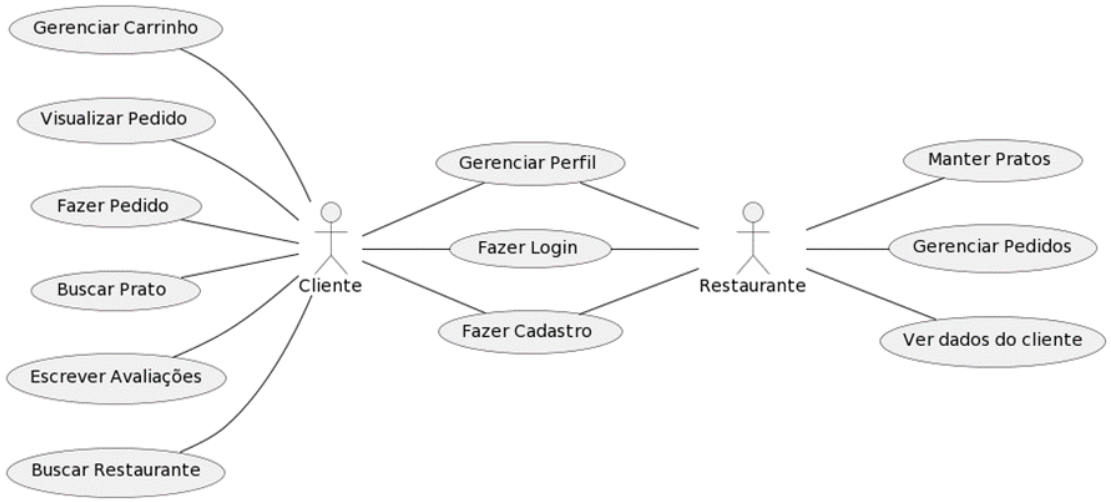


Figura 64 – Diagrama de Casos de Uso do sistema Delivery, fornecido aos participantes do estudo experimental quantitativo.

REFERÊNCIAS

AMERICA, P.; ROMMES, E.; OBBINK, H. Multi-view variation modeling for scenario analysis. In: SPRINGER. *International Workshop on Software Product-Family Engineering*. [S.l.], 2003. p. 44–65. 12, 34, 46, 47, 61, 71, 81, 82

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. Fourth. [S.l.]: Addison-Wesley Professional, 2021. 33, 49

BECK, K. Embracing change with extreme programming. *Computer*, v. 32, n. 10, p. 70–77, 1999. 31

BECK, K. et al. *Manifesto for Agile Software Development*. 2001. Disponível em: <http://www.agilemanifesto.org/>. 31

BOOCH, G. *Object oriented design with applications*. [S.l.]: Benjamin-Cummings Publishing Co., Inc., 1990. 43

BROWN, S. Software architecture for developers - volume 2: Visualise, document and explore your software architecture. *Ebook*, 2017. 12, 39, 52, 53, 55, 56, 61, 62, 78, 81, 82, 84, 98, 103

CALDIERA, V. R. B.-G.; ROMBACH, H. D. Goal-question-metric paradigm. *Encyclopedia of software engineering*, v. 1, n. 528-532, p. 6, 1994. 130

CRUZES, D. S.; DYBA, T. Recommended steps for thematic synthesis in software engineering. In: *2011 International Symposium on Empirical Software Engineering and Measurement*. [S.l.: s.n.], 2011. p. 275–284. 87, 133

DASANAYAKE, S. et al. Software architecture decision-making practices and challenges: an industrial case study. In: IEEE. *2015 24th Australasian Software Engineering Conference*. [S.l.], 2015. p. 88–97. 26

ERDER, M.; PUREUR, P.; WOODS, E. *Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps*. ADDISON WESLEY Publishing Company Incorporated, 2021. (A Vaughn Vernon signature book). ISBN 9780136523567. Disponível em: <https://books.google.com.br/books?id=fnUOzgEACAAJ>. 34

GIARDINO, C. et al. Software development in startup companies: The greenfield startup model. *IEEE Transactions on Software Engineering*, v. 42, n. 6, p. 585–604, 2016. 25, 26, 62, 64, 65, 79

HADAR, E.; SILBERMAN, G. M. Agile architecture methodology: long term strategy interleaved with short term tactics. In: *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*. [S.l.: s.n.], 2008. p. 641–652. 12, 39, 50, 51, 62, 75, 81, 82

HOFMEISTER, C. et al. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, v. 80, n. 1, p. 106–126, 2007. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121206001634>. 33, 39, 60

ISLAM, M.; HASAN, R.; EISTY, N. U. Documentation practices in agile software development: A systematic literature review. IEEE Computer Society, Los Alamitos, CA, USA, p. 266–273, may 2023. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/SERA57763.2023.10197828>. 32

ISO/IEC/IEEE Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, p. 1–46, 2011. 33, 34

KAZMAN, R.; KLEIN, M.; CLEMENTS, P. *ATAM: Method for Architecture Evaluation*. Pittsburgh, PA, 2000. Disponível em: <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5177>. 34

KITCHENHAM, B.; LINKMAN, S.; LAW, D. Desmet: A method for evaluating software engineering methods and tools. *Keele University*, 1996. 27, 60

KLOTINS, E. et al. Exploration of technical debt in start-ups. In: IEEE. *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. [S.l.], 2018. p. 75–84. 26

KLOTINS, E. et al. A progression model of software engineering goals, challenges, and practices in start-ups. *IEEE Transactions on Software Engineering*, Ieee, 2019. 62, 64, 65, 80, 140

KLOTINS, E. et al. Use of agile practices in start-up companies. *e-Informatica Software Engineering Journal*, v. 15, n. 1, 2021. 31

KLOTINS, E.; UNTERKALMSTEINER, M.; GORSCHEK, T. Software engineering in start-up companies: An analysis of 88 experience reports. *Empirical Software Engineering*, Springer, v. 24, n. 1, p. 68–102, 2019. 25, 65

KRUCHTEN, P.; OBBINK, H.; STAFFORD, J. The past, present, and future for software architecture. *IEEE Software*, v. 23, n. 2, p. 22–30, 2006. 34

KRUCHTEN, P. B. The 4+ 1 view model of architecture. *IEEE software*, Ieee, v. 12, n. 6, p. 42–50, 1995. 11, 34, 43, 44, 45, 46, 61, 69, 81, 82, 84, 87

KUHRMANN, M. et al. What makes agile software development agile? *IEEE Transactions on Software Engineering*, Ieee, v. 48, n. 9, p. 3523–3539, 2021. 25

MARSHALL, C.; BRERETON, P.; KITCHENHAM, B. Tools to support systematic reviews in software engineering: a feature analysis. In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. [S.l.: s.n.], 2014. p. 1–10. 62

PANTIUCHINA, J. et al. Are software startups applying agile practices? the state of the practice from a large survey. In: SPRINGER, CHAM. *International Conference on Agile Software Development*. [S.l.], 2017. p. 167–183. 32

PARIZI, R. et al. A design thinking techniques recommendation tool: An initial and on-going proposal. In: VIANA, D.; SCHOTS, M. (Ed.). *19th Brazilian Symposium on Software Quality, SBQS 2020, São Luís, Brazil, December, 2020*. Acm, 2020. p. 36. Disponível em: <https://doi.org/10.1145/3439961.3439997>. 60

PATERNOSTER, N. et al. Software development in startup companies: A systematic mapping study. *Information and Software Technology*, Elsevier, v. 56, n. 10, p. 1200–1218, 2014. 62, 64

ROSNER, B. A generalization of the paired t-test. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, v. 31, n. 1, p. 9–13, 1982. Disponível em: <https://rss.onlinelibrary.wiley.com/doi/abs/10.2307/2347069>. 120

SANTOS, D. M. B. et al. Integrando as disciplinas de engenharia de software, análise e projeto de sistemas e banco de dados utilizando pbl. In: *XV Workshop sobre Educação em Computação–Anais do XXVII Congresso da Sociedade Brasileira de Computação*. [S.l.: s.n.], 2007. p. 66–75. 131

SCHWABER, K. *Scrum development process*. [S.l.]: Springer, 1997. 117–134 p. 31, 131

SOMMERVILLE, I. *Software Engineering*. 9th. ed. Usa: Addison-Wesley Publishing Company, 2010. ISBN 0137035152. 33

SONI, D.; NORD, R. L.; HOFMEISTER, C. Software architecture in industrial applications. In: IEEE. *1995 17th International Conference on Software Engineering*. [S.l.], 1995. p. 196–196. 11, 34, 40, 41, 42, 61, 67, 81

THEUNISSEN, T.; HEESCH, U. van; AVGERIOU, P. A mapping study on documentation in continuous software development. *Inf. Softw. Technol.*, v. 142, p. 106733, 2022. Disponível em: <https://doi.org/10.1016/j.infsof.2021.106733>. 25, 32

WATERMAN, M.; NOBLE, J.; ALLAN, G. How much up-front? a grounded theory of agile architecture. In: IEEE. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. [S.l.], 2015. v. 1, p. 347–357. 34, 35, 62, 63

WOHLRAB, R. et al. Improving the consistency and usefulness of architecture descriptions: Guidelines for architects. In: IEEE. *2019 IEEE International Conference on Software Architecture (ICSA)*. [S.l.], 2019. p. 151–160. 37, 62, 63, 64

WOJCIK, R. et al. *Attribute-driven design (ADD), version 2.0*. [S.l.], 2006. 48, 62, 73, 80, 82

YANG, C.; LIANG, P.; AVGERIOU, P. A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, Elsevier, v. 111, p. 157–184, 2016. 31, 34, 36, 40, 61