INSTITUTO FEDERAL DE CIÊNCIA E TECNOLOGIA DE PERNAMBUCO

Campus Recife

Curso Superior Tecnólogo em Análise e Desenvolvimento de Sistemas

JOÃO VICTOR VASCONCELOS CORRÊA DE OLIVEIRA

RETROSPECTIVA DE PROJETOS DE EVOLUÇÃO DE SOFTWARE: COMPARANDO TEORIA E PRÁTICA

Recife

2021

JOÃO VICTOR VASCONCELOS CORRÊA DE OLIVEIRA

RETROSPECTIVA DE PROJETOS DE EVOLUÇÃO DE SOFTWARE: COMPARANDO TEORIA E PRÁTICA

Trabalho de Conclusão de Curso apresentado ao curso de Tecnologia em Análise e Desenvolvimento de Sistemas, como pré-requisito para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco IFPE, campus Recife.

Orientador: Prof. Dr. Vilmar Santos Nepomuceno

Recife

2021

Ficha elaborada pela bibliotecária Maria do Perpétuo Socorro Cavalcante Fernandes CRB4/1666

O48r
2021 Oliveira, João Victor Vasconcelos Corrêa de.
Retrospectiva de projetos de evolução de software: comprando teoria e prática./ João Victor Vasconcelos Corrêa de Oliveira. --- Recife: O autor, 2021.
59f. il. Color.

TCC (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Pernambuco, Departamento Acadêmico de Controle de Sistemas Eletrônicos - DASE, 2021.

Inclui Referências e Apêndices.

Orientador: Professor Dr Vilmar Santos Nepomuceno

1. Software - Manutenção. 2.Evolução desoftware. 3. Modernização de software. 4. Metodologia. I. Nepomuceno, Vilmar Santos (orientador). II. Instituto Federal de Pernambuco. III. Título.

CDD 005.16 (21ed.)

Trabalho de Conclusão de Curso apresentado pelo(a) estudante João Victor Vasconcelos Corrêa De Oliveira à coordenação de Tecnologia em Análise e Desenvolvimento de Sistemas, do Instituto Federal de Pernambuco, sob o título de "RETROSPECTIVA DE PROJETOS DE EVOLUÇÃO ENTRE VERSÕES DE SOFTWARE: COMPARANDO TEORIA E PRÁTICA", orientado pelo Prof. Dr(a). Vilmar Santos Nepomuceno e aprovado pela banca examinadora formada pelos professores:

Recife, 23de Setembro de 2021.

Prof. Dr(a). Vilmar Santos Nepomuceno
CTADS/DASE/IFPE

Prof. Dr(a). Renata Freire de Paiva Neves
CTADS/DASE/IFPE

Prof(a). Dr(a). Bruno Falcão de Souza Cartaxo
IFPE - Campus Paulista

Aluno(a): João Victor Vasconcelos Corrêa De Oliveira

AGRADECIMENTOS

Primeiramente a minha família, sem o apoio e colaboração tudo na minha vida se tornaria mais difícil.

A todos os amigos que fiz durante o curso, foram muitas conversas de grande importância para juntos superarmos os desafios propostos pelo curso.

Ao professor Vilmar Santos Nepomuceno, por toda orientação na elaboração deste trabalho.

RESUMO

A manutenção de software é uma atividade muito comum e importante que as empresas precisam dar atenção caso queiram manter a relevância dos seus produtos e a satisfação dos seus usuários. Essa manutenção pode ser entendida como pequenas atualizações que possuem objetivos de corrigir erros, adaptar para integrar com serviços externos ou adicionar novas funcionalidades. Faz parte também a necessidade de migrar o produto para uma nova versão (muitas vezes com outras tecnologias), pois a versão anterior acaba ficando muito complexa, por ter sido desenvolvida com uma tecnologia que se tornou obsoleta ou por decisão estratégica. Com base nisso, este trabalho se propôs a investigar na literatura algumas metodologias específicas para evolução de software e comparar com as atividades realizadas em dois projetos reais de modernização. Para tal, uma lista das principais características extraídas das metodologias foi utilizada para criar um questionário a ser respondido por pessoas que fizeram parte dos projetos de modernização avaliados. Após obter as respostas do questionário, os dados foram relacionados com os desafios e as características encontradas na literatura. Desse modo, foi constatada a presença de tais desafios e características nos projetos, embora não tenham utilizado nenhuma metodologia da literatura específica para evolução de software.

Palavras-chave: Manutenção de software. Evolução de software. Modernização de software. Metodologia.

ABSTRACT

The software maintenance is a very common activity that the companies need to be atention if want keep the relevance of product and user satisfaction. This maintenance could be understanded like smalls updates that have purpose of correct errors, adapt for integration with external services and add new functionalities. Is part too the need of migration the product to new version (often with others technologies), because the previously version ends up getting very complex, may have been developed with one technology wich became obsolete or by strategic decision. Based on this, this job set out to investigate the literature some specific methodologies for software evolution and to compare with the activities carried out in two real modernization projects. For this, a list of main features extracted from the methodologies was used for create a questionnaire to be responded for people who were part of the evaluated modernization projects. After getting the answers, the data was related with the challenges and characteristics founded in the literature. At the end of the analysis, was found that the projects have gone through the challenges and have the characteristics, although they have not used any specific methodology for the scenario.

Keywords: Software maintenance. Software evolution. Software modernization. Metodology.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AOM - Arquitetura Orientada a Modelos

XIRUP - eXtreme end-User dRIven Process (Processo eXtremo diRigido ao Usuário final)

SMART - Service-Oriented Migration and Reuse Technique (Técnica de Reutilização e Migração Orientada a Serviços)

SMP - Software Maintenance Process (Processo de Manutenção de Software)

AOS - Arquitetura Orientada a Serviços

SMIG - Service Migration Interview Guide (Guia de Entrevista de Migração de Serviço)

XP - eXteme Programming (Programação eXtrema)

TDD - Test Driven Development (Desenvolvimento Dirigido por Testes)

API - Application Programming Interface (Interface de Programação de Aplicativos)

SUMÁRIO

1 INTRODUÇÃO

Atualmente, nota-se a existência de uma grande variedade de softwares em execução, bem como uma grande variedade de tecnologias, e, a cada dia, novas ferramentas, frameworks, linguagens e conceitos são criados. Então, diante dessas inovações, o que acontece com os softwares que já foram desenvolvidos?

Segundo Sommerville (2011) o desenvolvimento de um software não termina quando é entregue. O produto precisa de atenção para continuar sendo útil para seus usuários, cumprindo seu objetivo e trazendo vantagem competitiva.

No mercado de trabalho vários profissionais irão se deparar com oportunidades de emprego para manter e atualizar aplicações já existentes, pois, de acordo com Durelli et al. (2014), a atividade mais comum de engenharia de software talvez seja a de manter um software, e isso vai muito além de apenas corrigir erros.

De acordo com Sommerville (2011), as organizações tem muito interesse em manter seu software em execução, uma vez que as empresas gastaram dinheiro para desenvolvimento de uma versão, o software torna-se crítico para o negócio, e então é normal que ainda mais dinheiro seja gasto para mantê-lo.

Diante desse cenário de manter e atualizar um software, metodologias de projeto especializadas nessa atividade foram surgindo com a finalidade de facilitar o trabalho das equipes dando mais atenção a pontos específicos.

Desta forma, este trabalho apresenta algumas dessas metodologias encontradas na literatura para comparar com casos reais de projetos de modernização, e assim concluir se essa teoria foi seguida na prática.

A comparação tem o objetivo de entender se os desafios e as soluções estudadas estão de acordo com os cenários reais vividos pelas equipes. E além disso, como objetivo específico, destacar se existem problemas nos projetos que poderiam ser melhor solucionados caso se conhecesse a literatura.

Baseado nos objetivos apresentados, foram elaboradas as seguintes perguntas de pesquisa:

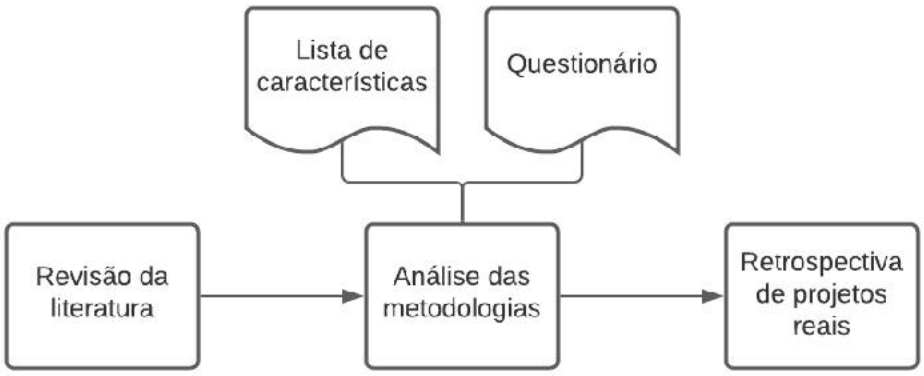
- Quais as semelhanças entre teoria e prática em projetos de evolução de software?

- Quais são as características de um projeto de evolução de software na prática?

1.1 Metodologia

Para responder às perguntas de pesquisa apresentadas anteriormente, foram realizados os seguintes passos ilustrados na figura 1:

Figura 1 - Metodologia



Fonte: desenvolvido pelo autor, 2021.

1. Revisão da literatura

Foi pesquisado na literatura, estudos que abordassem o tema de migração e modernização de software para inicialmente entender quais os pontos relevantes para esse cenário.

2. Análise das metodologias

Foram encontradas algumas metodologias que trouxeram boas práticas, diferentes perspectivas e evidenciaram desafios para tentar padronizar e facilitar essa atividade.

Após analisar as metodologias citadas anteriormente, percebeu-se que certas características estavam presentes em mais de uma metodologia. Assim, uma lista das características foi criada para destacar essas semelhanças.

A lista serviu como inspiração para elaborar um questionário a ser aplicado em pessoas que participaram de projetos que tinham objetivo de migrar a versão de um software.

3. Retrospectiva de projetos reais

Com as respostas do questionário, o último passo foi analisar essas respostas comparando com as metodologias presentes na literatura.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Ciclo de vida do software

De acordo com Sommerville (2011) quando fazemos referência ao desenvolvimento podemos estar falando de um produto feito a partir do zero como também uma extensão ou modificação de um software já existente.

Fuentes-Fernández, Pavón e Garijo (2012) dizem que a grande importância e o impacto dos sistemas para as organizações, somados com a velocidade com que as tecnologias atualmente se tornam obsoletas, resultam em uma necessidade contínua de criar e atualizar funcionalidades ou recursos que vão ficando ultrapassados. Na mesma linha de pensamento, Khan, Lo e Skramstad (2001) já afirmavam que era necessário que os sistemas passassem por mudanças e evoluções, como também era inevitável que as alterações degradassem o código com o passar do tempo.

A inevitabilidade de manutenção no software, para refletir as necessidades do negócio, gera degradação no código e isso está de acordo com as leis da evolução de software que foram refinadas por Lehman, Perry e Ramil (1998). As leis trazem características como:

- Atualizações no software são inevitáveis. Caso não seja feita, com o passar do tempo, usuários vão ficando cada vez mais insatisfeitos;
- Na medida em que o software é atualizado sua estrutura inicial é danificada, pois aumenta-se a complexidade do código, a menos que um trabalho extra seja feito para diminuir os impactos;
- De modo geral, os incrementos, com o passar do tempo, tendem a ser num volume constante ou ir diminuindo;
- Os processos de evolução possuem uma dinâmica própria. Ou seja, cada cenário é exclusivo e os feedbacks vão conduzindo os processos.

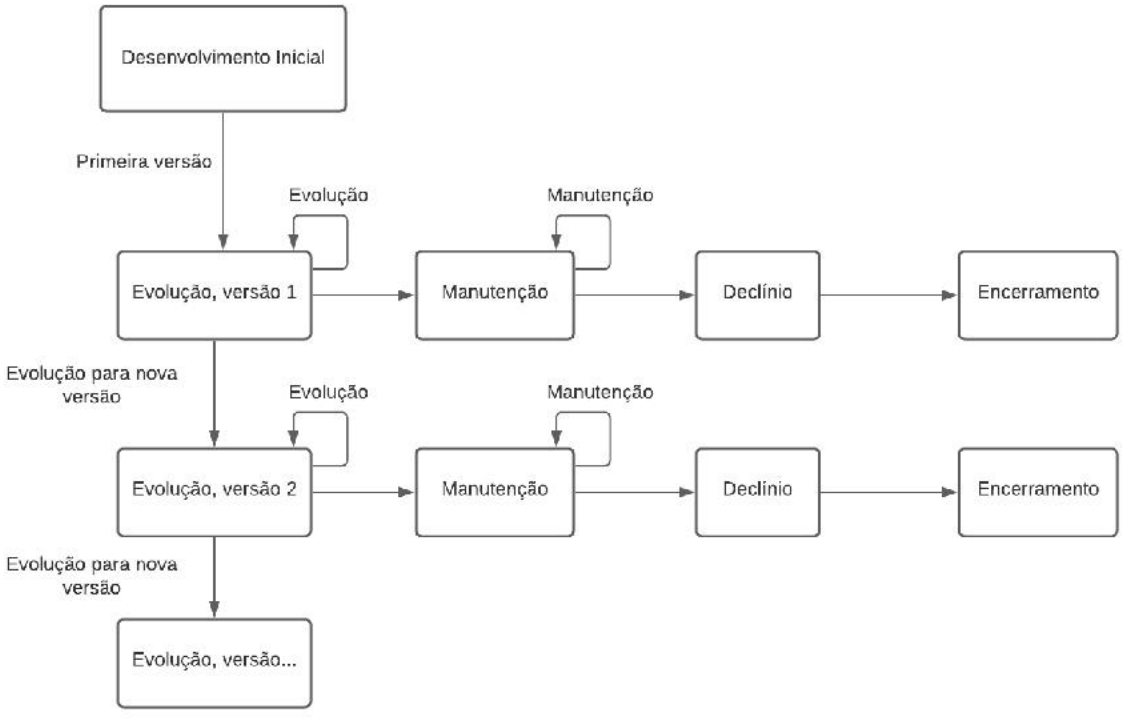
Desse modo, o software ao longo do tempo vai ganhando mais funcionalidades e por consequência ganha também mais complexidade e espaço para falhas. A consequência dessa complexidade é analisada por Ahmed E. Hassan (2005) em que afirma que uma vez incorporados os requisitos, a equipe se sente desencorajada para reescrever grandes partes do software para se adequar melhor

a novas funcionalidades que estão sendo inseridas, portanto as aplicações precisam ser migradas quando a estrutura inicial não é mais suportada.

Além dos problemas da deterioração do software, Rajlich e Bennett (2000) também dão foco na perda da experiência da equipe. Já que é algo natural o código ir se tornando cada vez mais complexo, há uma necessidade dos engenheiros se especializarem mais para entender o código que estão mantendo, e essa experiência adquirida cotidianamente é quase que impossível de se documentar. O que acontece na prática é que com o passar do tempo as pessoas da equipe tendem a sair e são substituídas por pessoas com menos experiência, o que resulta em uma degradação mais acelerada.

Um ciclo de vida produzido por Rajlich e Bennett (2000) propõe uma separação entre os estágios de evolução e manutenção além de uma ideia de versionamento:

Figura 2 - Ciclo de vida do Software



Fonte: Rajlick e Bennett, 2000.
Adaptado de: A Staged Model for Software Life Cycle

Na fase de “Desenvolvimento inicial” (Figura 2) é onde a equipe vai adquirir conhecimento do domínio do negócio e do problema, além de conhecimentos técnicos dos componentes, tecnologia e as funcionalidades do sistema.

Com uma primeira versão lançada, as evoluções (Figura 2) ocorrem de maneira iterativa, podem trazer modificações e/ou exclusões de funcionalidades. Pressões competitivas e demandas do cliente podem trazer grandes mudanças na estrutura do produto. Vale destacar que o limite entre evolução e manutenção pode ser um marco indefinido e arbitrário.

Na manutenção (Figura 2), as mudanças já são menores e por vezes vão ficando mais caras e complexas para serem feitas, então muitas vezes a abordagem é encapsular componentes antigos deixando esse código intocado, mas mesmo assim contribuindo para a degradação.

Na fase de declínio (Figura 2) a empresa não realiza mais a manutenção e tenta extrair o que ainda for possível do software. Os usuários ainda usam o sistema, mas sem alterações ou correções. Dificilmente se retorna ao estágio anterior pelo grande acúmulo das mudanças e suas complexidades.

A última fase, encerramento (Figura 2), a empresa desliga o software e redireciona seus usuários para o sistema substituto, caso exista. Na medida em que vai entrando nessa fase é de responsabilidade do gerente planejar e organizar a migração dos dados.

2.2 Evolução de Software

A transição que ocorre em “Evolução para nova versão” (Figura 2) será analisada com mais detalhes. Na literatura foram encontradas diferentes abordagens e diferentes termos com o objetivo de classificar esse estágio.

No trabalho de Stavru, Krasteva, Ilieva (2013) o termo “Modernização” foi o escolhido para fazer referência às modernizações no software que devem ser feitas pelas empresas em seus sistemas legados dada a competitividade do mercado. “Modernização” também foi utilizado por Fuentes-Fernández, Pavón e Garijo (2012) para explicar um modelo capaz de realizar de forma eficiente o desenvolvimento de novas funcionalidades de forma a substituir os recursos mais antigos.

Já no estudo de Lewis (2005) a palavra “Migração” é empregada para explicar a técnica que ajuda as organizações analisarem seus sistemas e determinarem se os mesmos podem ser expostos em uma arquitetura orientada a serviços.

Para o presente estudo, os termos “evolução”, “migração” e “modernização” são similares e, em todos os casos analisados, o objetivo consiste em resolver o problema do código que já foi muito alterado ou que não acompanhou a evolução do mercado para uma tecnologia, arquitetura ou linguagem que possa trazer alguma vantagem para o negócio.

É válido destacar que a decisão de se passar de uma versão para outra pode ser feita a qualquer momento, sem necessariamente a versão ter passado por todos os estágios do ciclo (Figura 2). De acordo com Rajlich e Bennett (2000) o versionamento se refere a mudanças estratégicas durante a evolução.

Os projetos de evolução para nova versão tendem a apresentar algumas peculiaridades, que foram destacadas por Fuentes-Fernández, Pavón e Garijo (2012):

- Requisitos novos são frequentemente bem conhecidos e planejados, são geralmente os objetivos principais do projeto;
- Uma outra versão do sistema já existe e pode servir como estudo para a equipe;
- O design do sistema mais antigo depende muito da arquitetura utilizada. Os engenheiros realizam atividades específicas como redesenhar partes do sistema;
- O processo pode incluir componentes novos ou melhorar implementações já existentes;
- Atividades de implementação são parcialmente dedicadas a fazer com que os componentes modernos se comuniquem com sistemas antigos.

2.3 Sistemas Legados

Quando se fala de uma modernização de software muitas pessoas possivelmente imaginam um sistema legado. De fato, existem metodologias que partem da premissa de se recuperar um sistema legado, como é o caso de Durelli et al. (2014), em que diz que a proposta da Arquitetura Orientada a Modelos (AOM) serve de base para outras metodologias de migração, e essa concentra-se principalmente em modernizar sistemas legados para outras plataformas ou

arquiteturas. Para ele, os legados são softwares que têm seus custos com a manutenção mais elevados do que os níveis desejados pela empresa.

No entanto, ainda não temos uma definição clara para esse tipo de sistema, como pode ser percebido no trabalho de Chervenski, Bordin e Dos Santos (2017), que apresenta diferentes definições encontradas na literatura e a proposta deles foi fazer uma lista das características que mais se repetiram entre os mais de 7000 artigos encontrados no período de 1995 até 2015. Abaixo as cinco características mais frequentes (não necessariamente precisam estar em conjunto):

1. Desenvolvido com tecnologia obsoleta
2. Vital para a organização
3. Documentação ausente ou desatualizada
4. Manutenção difícil
5. Desenvolvido há muito tempo

A evolução de um sistema para uma nova versão não acontece necessariamente por ele ser considerado legado, como foi dito na seção 2.2, as evoluções podem ocorrer a qualquer momento e vai de acordo com as estratégias do negócio. Mas, problemas podem se acentuar nos casos onde o projeto é baseado em sistemas mais antigos, como, por exemplo: um projeto em que necessariamente o código fonte precisa ser interpretado e esse código fonte está escrito em uma linguagem obsoleta, onde existem poucos profissionais disponíveis no mercado.

No estudo de Stavru, Krasteva e Ilieva (2013) vários desafios acerca da migração de sistemas legados foram identificados e divididos em dois grupos (Tabela 1 e Tabela 2):

Tabela 1 - Desafios Organizacionais

Definição do contexto do negócio	Para o projeto ser considerado bem sucedido precisa ter o objetivo e a estratégia alinhados com o negócio.
Falta de compromisso comercial	O sucesso está relacionado ao esforço, e o esforço requer motivação.
Resistência a mudanças	Os sistemas são críticos ao negócio e a mudança traz altos riscos.
Aquisição de competências e conhecimentos	Usuários e desenvolvedores podem precisar de treinamento, cada um no seu campo de atuação.
Aumento de risco devido a	As tecnologias devem ser avaliadas para viabilidade

nova tecnologia	técnica.
Falta de conhecimento profundo	Pode ser que haja uma falta de conhecimentos dos requisitos funcionais e não funcionais do sistema legado.
Falta de suporte	O cliente pode estar pouco presente durante o processo.

Fonte: Stavru, Krasteva, Ilieva, 2013.
Adaptado de: Challenges of Model-driven Modernization An Agile Perspective

Tabela 2 - Desafios Técnicos

Extrair conhecimentos comercial e técnico do sistema legado	A necessidade de extrair conhecimento pode ser problemático devido a arquitetura inadequada ou desatualizada, documentação de baixa qualidade ou código fonte de baixa qualidade.
Garantir equivalência comportamental entre sistemas	São necessários esforços adicionais para garantir completa equivalência entre sistemas.
Coexistência de sistemas	Problemas adicionais podem existir quanto a integração com outros sistemas e na utilização dos dados.
Superar tecnologias obsoletas	O sistema pode ser baseado em tecnologias antigas que dificultem o trabalho dos engenheiros.

Fonte: Stavru, Krasteva, Ilieva, 2013.
Adaptado de: Challenges of Model-driven Modernization An Agile Perspective

3 ANÁLISE DAS METODOLOGIAS

3.1 Metodologias de evolução de software

Não necessariamente em um projeto de modernização de software faz-se necessário uma metodologia específica, como explicado por Fuentes-Fernández, Pavón e Garijo (2012), às soluções para esse tipo de projeto variam entre versões personalizadas de processos genéricos e processos centrados na modernização. Os mesmos autores ainda afirmam que as metodologias que são focadas possuem particularidades para planejamento e avaliação de riscos, diferentemente de outras mais genéricas.

Fazendo jus às abordagens específicas para migração de software, Stavru, Krasteva e Ilieva (2013) destacam que a Arquitetura Orientada a Serviços (AOS) resultava em uma modernização promissora, o que resultou no surgimento de várias metodologias orientadas a serviços. Além dessa, outras técnicas, como orientadas a modelos e ferramentas também foram surgindo como alternativa para automatizar e agilizar atividades (extrair conhecimento do software ou implementação para uma plataforma específica), mas essas ainda são escassas e possuem pouco apoio.

Como forma de diversificar a investigação das características, o estudo foi baseado em metodologias de abordagens distintas: orientação a modelos (XIRUP), arquitetura orientada a serviços (SMART), framework para manutenção de software (Processo de Manutenção de Software) e uma perspectiva de métodos ágeis em cenário de modernização (Perspectiva Ágil).

3.1.1 XIRUP

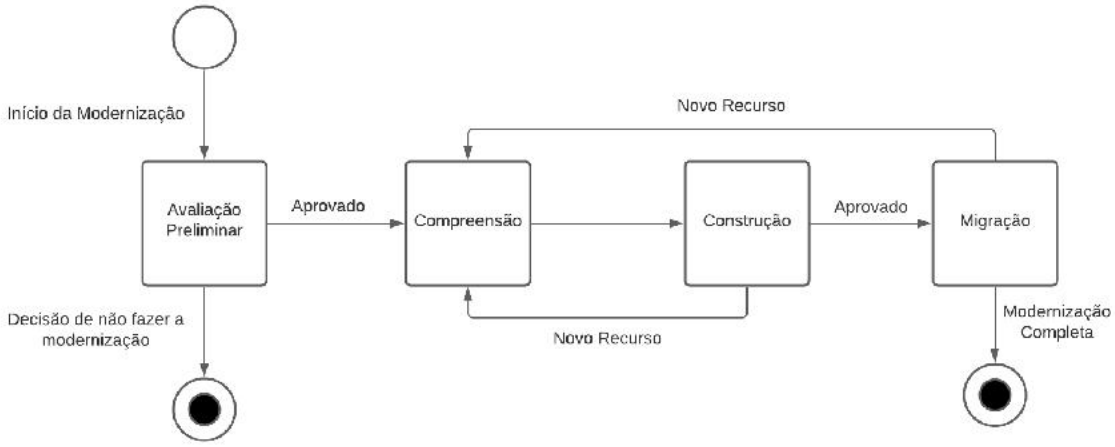
De acordo com Fuentes-Fernández, Pavón e Garijo (2012) a metodologia XIRUP foi concebida em 2008 juntamente com seu ferramental chamado de XIRUP Suite. Esse foi o resultado do projeto MOMOCS, um projeto financiado pela União Europeia centrado na modernização de sistemas.

A proposta segue os princípios da Arquitetura Orientada a Modelos (AOM) propondo uma modernização baseada em componentes de sistema, como explicado por Fuentes-Fernández, Pavón e Garijo (2012). Eles ainda completam citando os 3 pilares essenciais: possuir uma linguagem de modelagem e ferramentas específicas,

processos iterativos com o objetivo de antecipar o conhecimento e levar em consideração experiências em projetos anteriores.

Segundo Fuentes-Fernández, Pavón e Garijo (2012), a metodologia envolve dois grupos de papéis principais: o cliente e o provedor de serviços. Esses stakeholders interagem entre si por 4 fases: Avaliação Preliminar, Compreensão, Construção e Migração (Figura 3).

Figura 3 - Modelo de processo do XIRUP



Fonte: Fuentes-Fernández, Pavón e Garijo, 2012
Adaptado de: A model-driven process for the modernization of component-based systems

As fases são baseadas em recursos, e para o XIRUP um recurso é definido por: “Um conjunto coeso de requisitos de modernização que produzem um incremento significativo nos artefatos do projeto.” (Fuentes-Fernández, Pavón e Garijo, 2012).

A primeira fase é a fase da Avaliação Preliminar, se baseia principalmente na relação de custo-benefício, requer uma análise sobre o problema entendendo o sistema existente e a plataforma de destino. O resultado dessa fase é decidir se a evolução deve ser feita e, caso positivo, decidir quais recursos devem ser considerados.

Com a aprovação da fase 1, o próximo passo é a Compreensão, o objetivo na fase 2 é identificar os principais componentes do sistema e as adaptações que serão necessárias para atender os requisitos.

A fase de Construção faz todas as transformações de modelos dos componentes do sistema tomando como base o conhecimento adquirido na fase anterior. Após uma iteração, os engenheiros podem partir para outro recurso ou continuar até a migração desse recurso específico já modelado.

Na última fase, de migração, é onde começa a ocorrer a transição entre sistemas, onde é abordada a implantação de novos componentes. Após uma iteração da fase 4, se ainda faltarem recursos para serem modelados, os engenheiros voltam à fase de compreensão.

Entre cada fase é necessária a validação e verificação de seus resultados com o cliente.

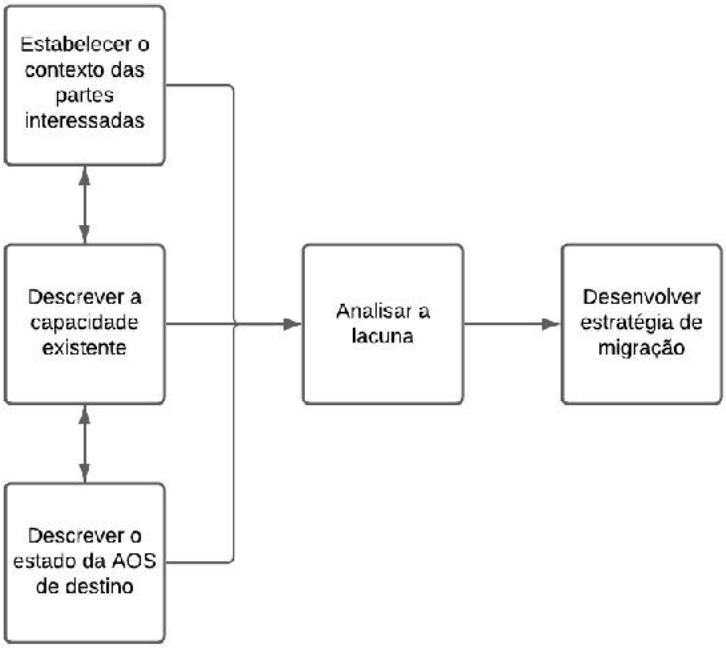
3.1.2 SMART

A Técnica de Reutilização e Migração Orientada a Serviços (SMART) tem o objetivo de auxiliar empresas com seus sistemas legados para determinar se as funcionalidades podem ser expostas como serviços em uma AOS (Lewis, 2005).

Um serviço, de acordo com Lewis (2005), pode ser definido como uma entidade detectável e independente que interage com outros aplicativos com fraco acoplamento, geralmente de forma assíncrona. Lewis ainda diz que um conjunto de serviços com interfaces bem definidas e comunicação compartilhada é caracterizada como uma AOS.

O SMART tem uma premissa de que alguns cenários de migração podem depender pouco de ferramentas semiautomáticas para codificar e mais de análises cuidadosas e completas da viabilidade e do esforço (Lewis, 2005). Desse modo, ele reúne informações sobre os componentes selecionados para produzir uma estratégia de migração através de 5 atividades principais (Figura 4):

Figura 4 - Modelo de atividades do SMART



Fonte: Lewis, 2005.
Adaptado de: Service-oriented migration and reuse technique (smart)

Estabelecer o contexto das partes interessadas: as partes interessadas incluem o proprietário do software, os usuários finais e potenciais usuários do serviço migrado. O objetivo é identificar quem mais tem conhecimento sobre o sistema e entender as funcionalidades utilizadas e o grau de importância para identificar se há demanda para tal esforço.

Descrever a capacidade existente: obter dados descritivos sobre os componentes do sistema legado como arquitetura empregada, linguagem, complexidade do código, documentação, interfaces dos módulos, possíveis pendências, histórico de mudanças e satisfação do usuário.

Descrever o estado da AOS de destino: organizar as informações para planejar possíveis serviços a partir do estado atual, além disso, planejar como os próprios serviços irão interagir entre si.

Analisar a lacuna: a partir das etapas anteriores, identificar a lacuna existente entre o software atual e a solução nova, determinando esforço, riscos e custo. Atividades de suporte para essa análise podem incluir inspecionar o código fonte.

Desenvolver estratégia de migração: a última atividade não pretende entregar o produto final. Mas tem o objetivo de recomendar estratégias e apresentar as

conclusões da equipe SMART para a organização. Após isso é dever da empresa decidir o que fazer.

As 3 primeiras fases que estão mais focadas em coleta de informação são direcionadas pelo SMIG (Guia de entrevista de migração de serviço), que Lewis (2005) afirma ser um guia que serve para direcionar as reuniões e garante a cobertura de diversos assuntos: Informações das partes interessadas, problemas gerais de migração, dados sobre componentes, riscos e questões específicas do componente, serviços potenciais e características da AOS alvo.

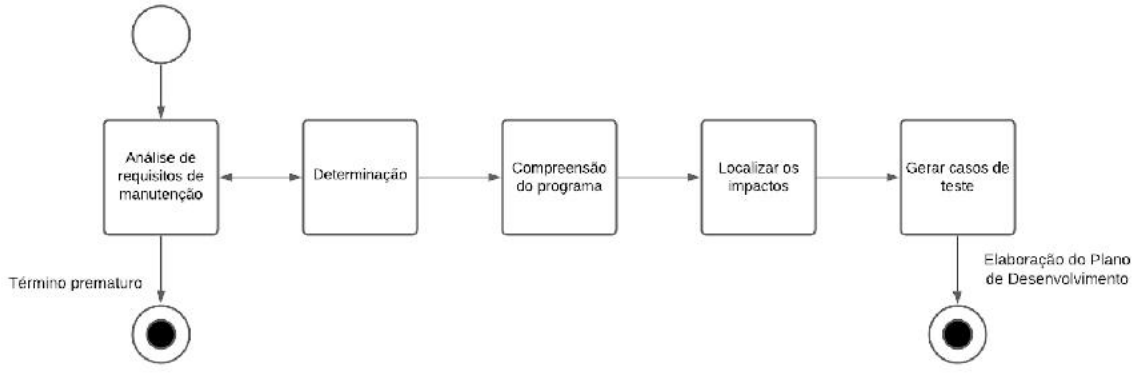
3.1.3 Processo de Manutenção de Software (SMP)

O Processo de Manutenção de Software tem o propósito de aumentar a eficiência da manutenção. Khan, Lo e Skramstad (2001) afirmam que o processo de manutenção pode ser tão complexo a ponto de ser necessária uma separação entre manutenção e desenvolvimento, mas consideram que ambos os processos possuem interseções e devem ser usados juntos.

Nessa metodologia, Khan, Lo e Skramstad (2001) destacam 3 pontos principais: pessoas envolvidas, tarefas de suporte e conhecimento sobre o software. Em diferentes ocasiões é normal que algum desses componentes possa ter maior relevância em comparação a outro.

A estrutura do framework definida por Khan, Lo e Skramstad (2001) tem foco nas tarefas de suporte (Figura 5), organizadas em uma sequência de tarefas bem definidas e que possuem objetivos, informações de entrada, informações de saída, método de execução, ferramentas, feedbacks e devolvem um status do projeto (Figura 5).

Figura 5 - Modelo de Processo de Manutenção de Software



Fonte: Khan, Lo e Skramstad, 2001.
Adaptado de: Tasks and Methods for Software Maintenance: a process oriented framework

Análise de requisitos de manutenção: é o gatilho do processo, são recebidas e analisadas quanto à adição, modificação ou exclusão de funcionalidades, correção de falhas ou migração do sistema para outras tecnologias.

Determinação: examina a viabilidade técnica e econômica das mudanças e seus benefícios. Os requisitos são refinados e caso necessário pode-se ter um ciclo com a fase anterior para um melhor detalhamento e então decidir se segue com a manutenção ou encerra de forma prematura.

Compreensão do programa: na maioria dos casos, onde não há documentação ou mesmo uma falta de confiança na documentação, é necessário o entendimento do código por parte dos programadores para auxiliar no entendimento da tarefa.

Localizar os impactos: localizar exatamente os locais no código que sofrerão as mudanças e tentar identificar possíveis efeitos colaterais que surgirão com essa tarefa a fim de prevenir a entrada de erros com a manutenção.

Gerar casos de teste: projetar os casos de teste. Os caminhos dos testes e os procedimentos para um teste de regressão são documentados.

Ao final da última etapa não é o objetivo da metodologia fazer a implementação das mudanças, mas entregar um plano de desenvolvimento. Portanto, entende-se que após o plano estabelecido começa a fase de desenvolvimento.

3.1.4 Perspectiva Ágil

Esse último na realidade não se trata de uma metodologia, é um estudo que teve o objetivo de identificar e solucionar os problemas encontrados em uma revisão de literatura sobre Desenvolvimento Orientado a Modelos e Modernizações de Software.

Após o levantamento dos problemas, Stavru, Krasteva, Ilieva (2013) conseguiram realizar questionários com especialistas da área de métodos ágeis para avaliar e julgar qual técnica ágil era melhor para solucionar cada problema.

Mesmo não se tratando de uma metodologia, foi selecionado aqui, pois escala técnicas baseadas em XP e SCRUM como possíveis soluções para os desafios encontrados em cenários de modernização (Tabelas 1 e 2).

Com os resultados de Stavru, Krasteva, Ilieva (2013), algumas das técnicas ágeis que são possíveis soluções para a maior parte dos problemas são: pequenas entregas, Sprints e equipes multifuncionais. Pequenas entregas e Sprints são para tornar o feedback rápido, incluindo feedbacks sobre os processos, o andamento do produto e pessoas, além de permitir mudanças menos custosas à organização. Para equipes multifuncionais os argumentos foram de espalhar competências e conhecimentos e com isso reduzir riscos, além de dar ênfase em questões sociais

O bom aproveitamento das técnicas no estudo de Stavru, Krasteva, Ilieva (2013) resultou na recomendação de utilização de métodos ágeis para extrair total efetividade dos recursos. Os especialistas mostraram dar ênfase em: envolvimento ativo do cliente, maior colaboração da equipe, aumento do valor do cliente, risco reduzido de falha, apoio e compromisso das partes interessadas, esclarecimento das funções, detecção precoce de preocupações.

3.2 Características das metodologias

A partir da análise das metodologias explicadas anteriormente na seção 3.1, características foram extraídas dos processos, e julgadas importantes para um cenário de evolução de software a partir de reuniões com o professor orientador. O objetivo era extrair pontos em comum nas diferentes abordagens e fazer comparações entre as mesmas, as características estão listadas abaixo:

1. Gerenciamento do conhecimento do negócio

Característica de buscar conhecer o contexto/domínio no qual a aplicação está inserida, o problema que o sistema resolve e as ideias propostas pelo cliente junto a necessidade para sua modernização.

2. Análise dos stakeholders

Característica de levar em consideração as necessidades dos stakeholders na utilização do sistema antes mesmo de começar o projeto. Stakeholders podem ser outros sistemas, usuários finais, pessoas do ambiente do cliente e a equipe que desenvolve.

3. Estudo de viabilidade

Característica de fazer uma análise prévia dos objetivos da modernização levando em conta a tecnologia do sistema atual, a tecnologia para modernização, a

equipe, o prazo, o esforço, o risco e o custo. Análise que também deve levar em consideração os motivos que levaram o cliente buscar a solução e os recursos disponíveis para manter o projeto.

4. Requer especialista

Característica de necessitar de uma pessoa certificada, treinada ou com experiência na metodologia ou tecnologia para conseguir obter total efetividade dentro do projeto.

5. Equipes multifuncionais

Característica de que os integrantes da equipe do projeto estão aptos a possuir diferentes responsabilidades e papéis para obter total efetividade da metodologia.

6. Geração de código

Característica que faz a implementação do produto. Considera-se que a escrita do código fonte pode ser manual, semiautomática ou automática por meio de ferramentas. O objetivo é entregar o produto final funcionando para o cliente.

7. Teste de software

Característica de possuir a atividade de teste em alguma fase dentro da metodologia. Testes vão garantir que o comportamento do produto esteja de acordo com a implementação do teste. Testes e validações por parte do cliente também são considerados.

8. Pequenas entregas

Característica de entregar pequenas partes do software em funcionamento para o cliente, e ir entregando incrementos até chegar ao produto final completo. Com pequenas entregas de valor antecipadas pode-se ter a diminuição dos riscos através de detecção precoce de possíveis problemas.

9. Adaptação a mudanças

Característica de a metodologia estar preparada para responder a mudanças. Pode ser uma correção, adaptação, melhoria ou um novo requisito que foi identificado depois da fase de coleta de informação. A mudança trará impacto, mas essa característica deve minimizar o máximo possível esse impacto.

10. Processo iterativo

Característica de processos que buscam sucessivas iterações, as fases da metodologia se relacionam entre si e podem entrar em ciclos. O processo é

dinâmico e pode repetir em várias estâncias ou mesmo uma fase pode retornar a fase anterior.

11. Criação de artefatos

Característica de gerar algum tipo de documentação: modelagem, backlog, plano de desenvolvimento. Auxiliam a comunicação com o cliente e também entre a própria equipe, são úteis durante e após o desenvolvimento.

12. Relacionamento explícito com o cliente

Característica de a metodologia possuir atividades direcionadas para trazer o cliente para dentro do projeto com objetivo de aumentar o relacionamento dele com a equipe. Aqui o cliente e a equipe podem dar e receber feedbacks e discutir o caminho que o projeto vai seguir.

13. Conjunto de ferramentas

Característica de a metodologia depender de ferramentas específicas. Ferramentas podem ser produtos feitos sob medida para o cenário específico da modernização ou externas a metodologia, podem ser softwares ou guias.

14. Inspeção do código fonte

Característica de analisar o código fonte do produto. Pode ser usado para compreender lógica das funções, uso de bibliotecas externas, protocolos de comunicação, formato de dados, complexidade do código, arquitetura, linguagem e os paradigmas relacionados.

15. Explícita análise do sistema legado

Característica que considera a existência de um produto já desenvolvido e através dele aproveita para adquirir conhecimentos para o novo projeto. O aprendizado da equipe pode ser através da observação do sistema legado em funcionamento e/ou pelo estudo da documentação. O importante é a dedicação em analisar o sistema antigo para mapear suas funcionalidades e restrições.

16. Reuso

Característica que leva em consideração a possibilidade de reuso de alguns componentes, o novo software pode ser implementado aproveitando partes do outro. Componentes podem ser entendidos como trechos de código, lógica dos testes ou arquitetura, e podem ser reutilizados de forma completa ou de forma parcial.

3.3 Classificação

3.3.1 Tabela comparativa

Verificou-se que apesar das metodologias possuírem características em comum, essas podem ser aplicadas em diferentes momentos. Os momentos foram classificados como “Antes da modernização” e “Durante a modernização”, o marco é relativo à decisão de confirmar o projeto. Também foi considerado que as metodologias podem possuir a característica em ambas as ocasiões, resultando assim na classificação de “A qualquer momento” (Tabela 3).

No entanto, algumas das metodologias não possuem certas características exatamente como está definido na seção 3.2, mas algo próximo, desse modo, por entender que isso não deve ser desconsiderado, a característica também pode estar presente de forma “Parcial”.

Tabela 3 - Relação entre as características e as metodologias

	Característica	XIRUP	SMART	SMP	Perspectiva Ágil
	Antes da modernização				
1	Gerenciamento do conhecimento do negócio	X	X	X	X
2	Análise dos stakeholders		X	Parcial	
3	Estudo de viabilidade	X	X	X	X
4	Requer especialista		X		
5	Equipes multifuncionais				X
	Durante a modernização				
6	Geração de código	X			X
7	Teste de software	X		X	X
8	Pequenas entregas	X			X
	A qualquer momento				
9	Adaptação a mudanças	X	Parcial	Parcial	X
10	Processo iterativo	X		Parcial	X
11	Criação de artefatos	X	X	X	X
12	Relacionamento explícito com clientes	X	X	Parcial	X
13	Conjunto de ferramentas	X	X	Parcial	Parcial
14	Inspeção do código fonte	X	X	X	X

15	Explícita análise do sistema legado	X	X	X	X
16	Reuso	X	X		X

Fonte: desenvolvido pelo autor, 2021.

3.3.2 Características do XIRUP

Foi desenvolvido juntamente com seu conjunto de ferramentas (13) chamado XIRUP suite, que tem papel fundamental nas transformações de modelos.

A metodologia propõe uma abordagem ágil e iterativa (10) com adaptação a mudanças (9). Suas fases podem ter repetições quantas vezes forem necessárias se ainda houver recursos pendentes (com exceção da primeira fase).

A primeira fase é a avaliação preliminar, em que é feito estudo de viabilidade (3), análise do sistema legado (15) e obtém-se conhecimento do negócio (1). Ao final, é tomada a decisão junto com o cliente (12) se o projeto deve ou não continuar.

Para a segunda fase busca-se uma análise do sistema legado (15) mais detalhada que a anterior com inspeção do código fonte (14), e dependendo do tipo de modernização o reuso (16) dos componentes pode ser considerado.

A Terceira fase serve para gerar e validar as transformações entre modelos. São aplicados testes (7) sobre os modelos e há também uma validação por parte do cliente (12).

A última fase é a implantação do código gerado para a plataforma específica. Gera-se o código (6) do sistema modernizado. Também são aplicadas as atividades de testes (7) e validação do cliente (12) como na etapa anterior.

Pelo processo iterativo ser orientado a recurso, cada recurso é migrado por vez, caracterizando pequenas entregas (8).

Os artefatos (11) correspondem aos modelos, que devem ser mantidos e gerenciados.

Não foi citada a análise dos stakeholders (2) na fase inicial de avaliação, apenas é afirmado que existem dois grupos de pessoas: o cliente e o provedor de serviço. Necessidade de especialistas (4) ou de equipes multifuncionais (5) também não foram mencionadas.

3.3.3 Características do SMART

Suas 3 fases iniciais levantam informações necessárias para uma análise do sistema legado (15), que compreende em:

1 - Análise dos stakeholders (2), entendendo os cenários de uso e principais funcionalidades do sistema, obtendo entendimento do negócio (1).

2 - Inspeção do código fonte (14), levantando informações como linguagem, funções e idade do código. Busca entender junto ao time de desenvolvimento suas particularidades.

3 - Descreve uma ideia de solução, reunindo o máximo de evidências para planejar as características do novo sistema. Inclusive definindo os componentes principais do sistema e definindo aqueles componentes que podem ter reuso (16).

Todas as 3 etapas iniciais são feitas em reuniões com a participação do cliente (12) e demais stakeholders (2), tais reuniões são guiadas pela ferramenta (13) SMIG.

A quarta etapa preenche o gap criado da atividade 2 e atividade 3. Neste momento é realizado um estudo de viabilidade (3) e dependendo dos resultados obtidos em relação a esforço, risco, tecnologia, custo e prazo, mudanças (9) podem ser sugeridas na migração (mudanças são apenas sugeridas ao cliente para elaboração do plano).

Na última etapa é entregue um plano de implementação, não possuindo o objetivo de produzir código (6). Tal artefato (11) deve ser analisado pelo cliente, e somente ele pode concluir se deve fazer uma migração ou não.

A abordagem não possui um processo iterativo (10) e as características de testes (7) e pequenas entregas (8) não fazem parte do escopo.

De acordo com Lewis (2005) há um analista SMART presente na equipe, uma pessoa com conhecimento em design de software e manutenção e que requer 3 dias de treinamento no método além de experiência em projeto SMART. Portanto entende-se que a abordagem precisa de especialistas (4).

Não foram dados mais detalhes sobre a equipe de um projeto SMART, portanto não é possível afirmar a necessidade de uma equipe multifuncional (5).

3.3.4 Características do Processo de Manutenção de Software

O SMP corresponde a um processo de manutenção sem propósito de geração de código (6). A ideia central é focar no framework de manutenção que pode se relacionar com abordagens de desenvolvimento.

As duas fases iniciais correspondem a entender os requisitos da manutenção em conjunto de uma análise de viabilidade técnica (análise do sistema legado (15)) e

econômica do projeto (estudo de viabilidade (3)) também obtendo conhecimento sobre o negócio (1). Nessas duas fases podem ocorrer diversas iterações para então decidir continuar o projeto, portanto, sendo exclusividade dessas fases iniciais há existência de um processo iterativo (10) e com adaptação a mudanças (9).

Ainda nas fases iniciais, o uso de ferramentas (13) para estimativa de custos é considerado pela metodologia como opcional.

Quando decidido continuar com a modernização, a próxima fase apresenta inspeção do código (14) para refinar ainda mais o entendimento acerca do sistema legado (15), sendo opcional o uso de ferramentas (13) de qualquer tipo (automática ou não) para auxiliar na tarefa.

Depois dos impactos avaliados é levado em consideração os cenários de teste (7) para verificar o resultado das mudanças.

Todas essas etapas geram artefatos (11) (documentos de saída das atividades, como por exemplo: especificação dos requisitos, filtro dos requisitos, artefatos de design, nome de funções, especificação dos fluxos de teste) que são armazenados e podem servir para consultas futuras.

A abordagem apresenta a participação do cliente (12) apenas na fase 2 para decidir iniciar a manutenção.

Não foi localizado no estudo características da equipe atuante no projeto, deixando em aberto questões como a participação de especialistas (4) ou a necessidade de uma equipe multifuncional (5).

Apesar deste estudo focar nas tarefas de suporte, o autor ao menos menciona que outro ponto importante é a análise dos stakeholders (2).

As características de pequenas entregas (8) e reuso (16) não fazem parte do escopo dessa abordagem.

3.3.5 Características da Perspectiva Ágil

O foco é o desenvolvimento de código (6) seguindo as boas práticas dos métodos ágeis em uma mistura das metodologias XP e SCRUM.

Sprints e pequenas entregas (8) são fortemente recomendadas pois garantem rápido feedback com participação do cliente (12) resultando em uma diminuição de riscos e auxiliando no conhecimento do negócio (1) que tende a aumentar cada vez mais à medida que o tempo passa. Além disso, contribuem para processo iterativo (10) e incremental com adaptação a mudanças (9).

Também considera as atividades de programação em pares, reuniões diárias e priorização de backlog como auxiliadoras na extração de conhecimentos técnicos do código legado (inspecionar o código legado (14)), um problema que foi mencionado no estudo de Stavru, Krasteva, Ilieva (2013).

A produção dos artefatos (11) também é uma tarefa necessária, consideram-se artefatos, por exemplo, os documentos de backlog do produto e backlog da sprint, da metodologia SCRUM.

Não foi mencionada análise dos stakeholders (2). Mas por outro lado problematiza riscos tecnológicos e a compreensão da estratégia da empresa (estudo de viabilidade (3)), concluindo que atividades de Planning e backlog do produto podem auxiliar na solução desses problemas.

As ferramentas (13) são necessárias, mas não específicas, são ferramentas principalmente para gerenciamento da equipe, do backlog e da sprint.

O reuso (16) foi citado, embora dito como limitado e um desafio quando considerado a partir de tecnologias obsoletas.

Os testes (7) fazem parte dessa abordagem, pois TDD é apontado como uma solução para diversos problemas. Uma outra solução ágil que também soluciona vários desafios é a adoção de equipes multifuncionais (5).

Não foi mencionada a necessidade de especialistas (4). E no caso da explícita análise do sistema legado (15), pode-se entender que mesmo de forma indireta existe, pois há a preocupação em extrair conhecimentos técnicos e de negócio além de difundir isso com a equipe.

4 RETROSPECTIVA DE PROJETOS REAIS DE EVOLUÇÃO

4.1 Descrição dos projetos

Ao todo, dois projetos que eu pude participar e que possuíam o objetivo de evoluir o software entre versões foram analisados neste estudo. Ambos foram realizados pela mesma empresa (no modelo fábrica de software) prestando um serviço terceirizado.

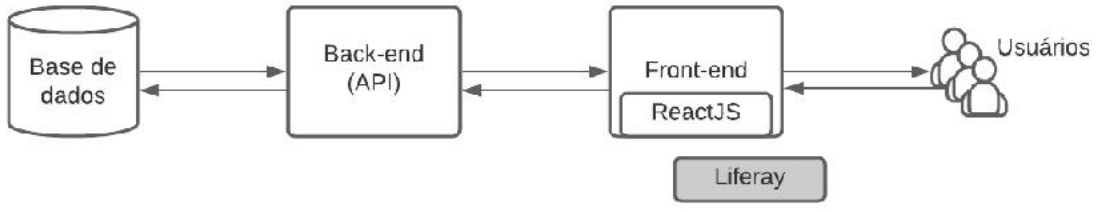
No primeiro deles, que levou cerca de cinco meses até a implantação em produção, a empresa responsável pela migração já estava trabalhando na manutenção do código Java Liferay que herdou de outra organização que havia desenvolvido o produto anteriormente.

Com a decisão da migração, novos profissionais foram contratados para formar um time com alguns daqueles que já conheciam o produto e tinham trabalhado na manutenção. A equipe foi formada por um gerente, um testador, três analistas e ao todo sete desenvolvedores passaram pelo projeto.

O time se comprometeu em fazer reuniões diárias, se organizavam em sprints, existia um scrum master para priorizar o backlog e tal backlog era composto por histórias de usuários. Além disso, a cada sprint era entregue um conjunto de funcionalidades para ser verificada pelo cliente.

A proposta do projeto foi de utilizar a tecnologia ReactJS para reconstruir todo o front-end da aplicação WEB. Os serviços da API e a base de dados continuaram os mesmos, além disso, o layout do front-end deveria permanecer igual ou o mais parecido possível com a antiga versão. A arquitetura e a mudança proposta estão representadas na figura 6:

Figura 6 - Modelo de arquitetura dos projetos reais



Fonte: desenvolvido pelo autor, 2021

No outro projeto, que durou cerca de três meses até ser publicado, pedido pelo mesmo cliente, a ideia era aproveitar da experiência de parte da equipe do projeto anterior para fazer algo similar. Uma outra aplicação Liferay deveria ter todo o front-end refeito para código ReactJS mantendo toda identidade visual, e mantendo também a integração com a mesma API e banco de dados.

Diferentemente do primeiro projeto, a equipe foi formada por 1 gerente e 3 desenvolvedores, e ninguém conhecia previamente o produto. Os desenvolvedores se auto organizavam para fazer os requisitos por demanda seguindo histórias de usuário que eram adicionadas no backlog, algumas dessas pediam por melhorias no produto.

Vale destacar que ambos os projetos já foram concluídos com sucesso e a mesma empresa responsável pela migração segue fazendo o trabalho de manter o código até o momento.

4.2 Descrição do questionário

A seção atual apresenta o trabalho relacionado à construção do instrumento de pesquisa (questionário), obtenção dos dados e a validade, as ideias foram baseadas no guia de Shull, Singer e Sjoberg (2008).

4.2.1 Objetivo do questionário

Com o presente questionário busca-se comparar as características (seção 3.2) extraídas como resultado da análise das metodologias de modernização, migração e manutenção de software que foram encontradas na literatura com cenários reais de projetos de evolução entre versões de software que foram bem sucedidos. A seguinte pergunta norteou o estudo:

- Quais as semelhanças entre teoria e prática em projetos de evolução de software?

4.2.2 Desenho de pesquisa

De acordo com Shull, Singer e Sjoberg (2008) o desenho foi do tipo transversal, pois os participantes da pesquisa forneceram informações de um ponto fixo no tempo. Responderam sobre suas experiências e observações em projetos que já haviam sido finalizados. O formulário de pesquisa foi aplicado no modelo

autoaplicável, em que o participante de forma individual e remota respondia às perguntas propostas em um formulário online do Google.

4.2.3 Construção do instrumento de pesquisa

Inicialmente foram identificados artigos que falassem sobre migração de software, e para melhorar os resultados da busca por estudos, outras palavras-chave foram utilizadas como “Modernização”, “Evolução” e “Manutenção” em engenharia de software. Após selecionados alguns estudos para referencial teórico, notou-se que algumas metodologias foram citadas, como expostas no capítulo 3.

Comparando as metodologias entre si e destacando as características para o cenário de migração de software, a lista da seção 3.2 foi gerada.

Cada item da lista serviu como tema para elaborar uma ou mais perguntas (Apêndice A). As exceções foram as perguntas iniciais (quatro perguntas), que não foram baseadas nas características, mas serviram para direcionar os participantes para o tema geral em questão (evolução de software) e a pergunta final, que foi inserida para caso o entrevistado quisesse complementar alguma informação. No apêndice B é apresentada a relação da característica com suas respectivas perguntas.

As perguntas receberam uma classificação conforme sua característica temática, e foram agrupadas respectivamente em: “Antes da modernização” (12 perguntas), “Durante a modernização” (6 perguntas) e “A qualquer momento” (16 perguntas).

Quanto aos tipos, as perguntas podem ser classificadas como:

- Múltipla escolha: o entrevistado deve escolher apenas uma alternativa dentre as listadas.
- Caixa de seleção: podem ser marcadas quantas opções forem necessárias (0, 1 ou mais de uma alternativa).
- Escala linear: a resposta deve ser dada atribuindo uma nota de 1 a 10.
- Aberta: a resposta deve ser escrita pelo entrevistado. Há casos de respostas curtas, em que é apresentado um campo menor de texto, e outros de resposta longa onde o tamanho do campo de texto é maior.

Antes de aplicá-lo nos desenvolvedores, gerentes e analistas um filtro foi feito dependendo do perfil. A partir de um formulário raiz foram criados dois formulários para serem aplicados. Algumas perguntas foram feitas para todos os perfis, mas

perguntas mais técnicas foram deixadas apenas para os desenvolvedores, enquanto que algumas perguntas sobre o negócio foram deixadas apenas para gerentes e analistas. Foi feito assim por entender que talvez determinado perfil não soubesse responder ou interpretasse mal alguma pergunta.

Na aplicação dos questionários foi explicado que nenhuma questão era obrigatória, ou seja, caso não soubesse responder à pergunta, a mesma poderia ser deixada em branco.

4.2.4 Avaliação da confiabilidade e validade do instrumento de pesquisa

Os formulários, depois de preparados, passaram por algumas iterações de avaliação do professor orientador a fim de conferir a validade dos instrumentos.

Essas revisões tinham objetivo de corrigir erros de português, melhorar construção dos enunciados, reordenar perguntas e até mesmo excluir aquelas julgadas redundantes ou que somavam pouco para a pesquisa.

Além dos dois formulários que seriam aplicados, também foram mantidos outros dois documentos, sendo eles: o formulário raiz com todas as perguntas (Apêndice A) e um documento que relacionava a pergunta com sua característica temática (Apêndice B).

4.2.5 Obtenção de dados válidos

Os escolhidos para responder a pesquisa são engenheiros de software, analistas e gerentes que participaram de ao menos um projeto de evolução de software com foco em evolução entre versões.

Ao todo seis engenheiros de software diferentes passaram por ao menos um projeto em algum momento, dentre eles, três não estavam mais na empresa no período em que o questionário foi aplicado. As respostas foram obtidas dos três desenvolvedores que ainda trabalhavam na empresa e de um dos envolvidos que já não estava mais presente. Vale ressaltar que duas das três pessoas que ainda estavam na empresa participaram dos dois projetos que foram estudados, portanto, responderam duas vezes ao formulário.

O nível de senioridade não foi levado em consideração, estagiários e profissionais de senioridades maiores receberam as mesmas perguntas de acordo com seu perfil de atuação no projeto.

Considerando os dois projetos, um deles possuía três analistas, enquanto que no outro não havia nenhum. Porém, durante o período de aplicação do questionário apenas dois ainda se encontravam na empresa e foram somente esses que enviaram suas respostas.

Apenas uma pessoa detinha o papel de gerente, e o mesmo respondeu o formulário duas vezes, levando em conta cada um dos projetos em que participou.

4.3 Resultados

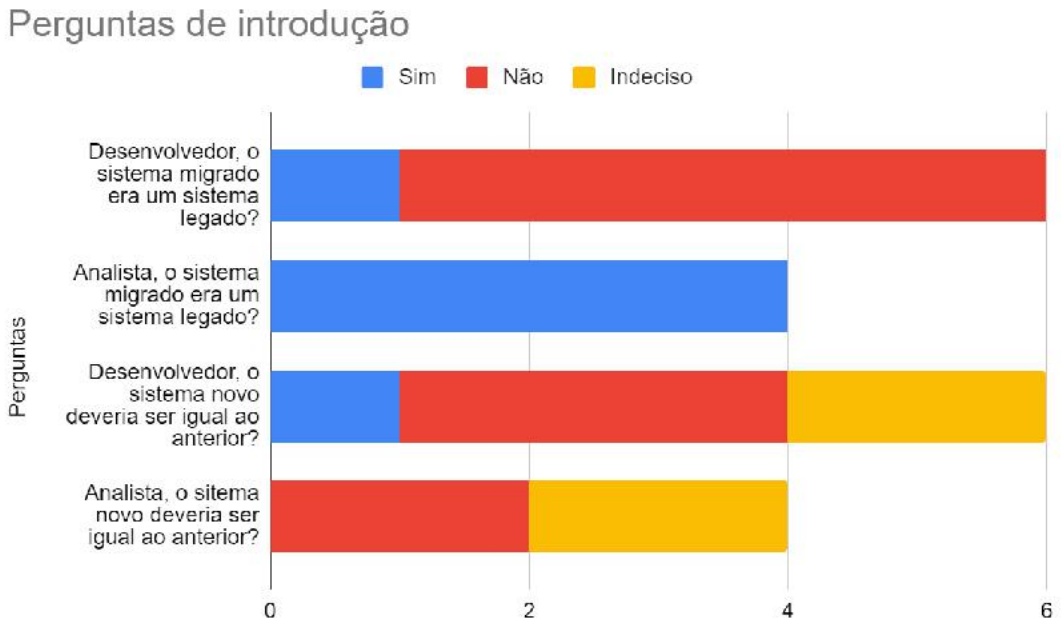
Os resultados serão apresentados de acordo com as seções do questionário aplicado.

4.3.1 Introdução

Questionando os entrevistados sobre qual metodologia foi utilizada para guiar o desenvolvimento, oito responderam Scrum, um respondeu Kanban e um respondeu “mistura de Scrum e Kanban”. Os mesmos avaliaram se as metodologias foram seguidas corretamente com todas as obrigações e atribuíram notas quatro (duas ocorrências), cinco (duas ocorrências), seis (duas ocorrências), sete (três ocorrências) e nove (uma ocorrência).

As demais estão ilustradas no gráfico a seguir separadas pelo tipo de perfil:

Figura 7 - Perguntas de introdução



Fonte: desenvolvido pelo autor, 2021

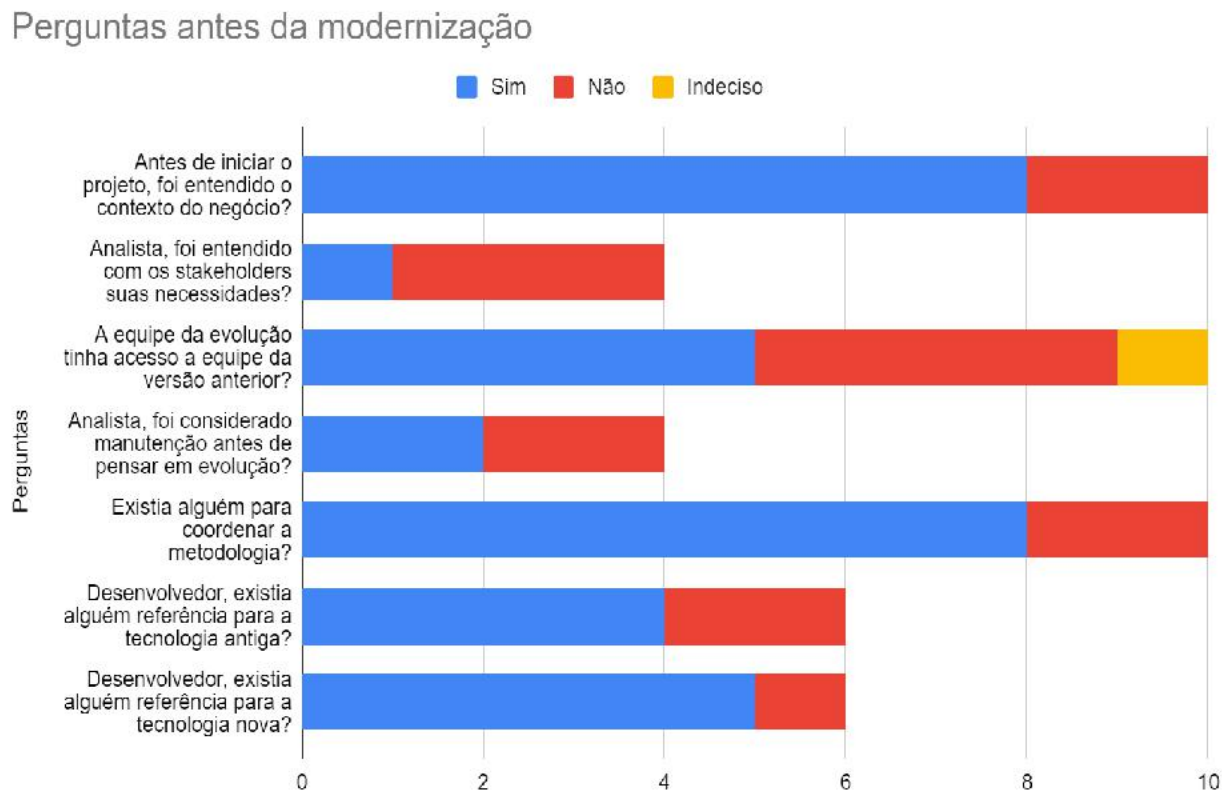
4.3.2 Antes da modernização

Para a pergunta sobre os objetivos do projeto, as respostas dos desenvolvedores e analistas informam que os principais eram de melhorar performance, reduzir custos com a plataforma custosa e reduzir complexidade da manutenção.

Foi perguntado sobre os critérios para a escolha das tecnologias para o projeto e as respostas foram por ser as tecnologias mais modernas, foi determinada pelo cliente, por serem bem difundidas no mercado e duas pessoas não souberam responder.

Demais perguntas de sim ou não foram resumidas no gráfico a seguir:

Figura 8 - Perguntas antes da modernização



Fonte: desenvolvido pelo autor, 2021

Um espaço de texto foi adicionado ao final desta seção para o entrevistado adicionar, caso achasse necessário, alguma informação complementar, e os

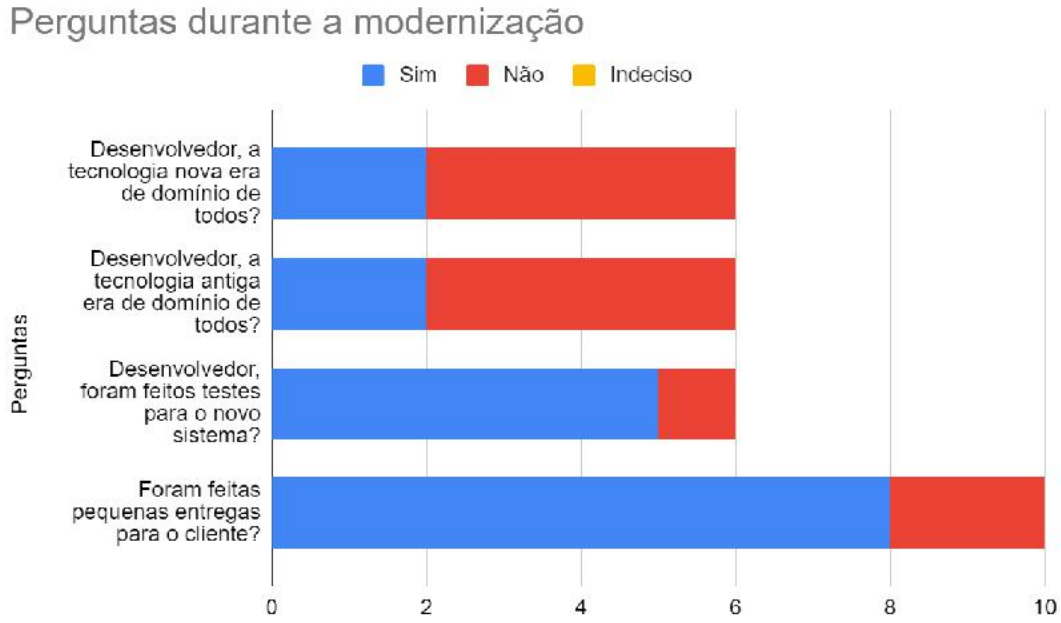
comentários foram referentes as equipes não precisam de papéis bem definidos e o gerente por exemplo pôde fazer mais de uma função quando foi necessário.

4.3.3 Durante a modernização

Foi questionado o quão igual era a versão nova comparada com a versão anterior, para os entrevistados darem uma nota de um até dez e os resultados foram de sete (uma ocorrência), oito (seis ocorrências) e nove (três ocorrências).

Em caso afirmativo de haver testes, foi questionado qual tipo de teste foi feito, as respostas foram: testes unitários, de integração e de segurança. Demais perguntas desta sessão de respostas curtas estão na imagem a seguir:

Figura 9 - Perguntas durante a modernização



Fonte: desenvolvido pelo autor, 2021

4.3.4 A qualquer momento

No cenário onde o entrevistado respondeu que houveram documentações produzidas, foi questionado quais os tipos de documentação que foram produzidas, as respostas foram: documento de arquitetura, especificação de requisitos, histórias de usuários, diagrama dos componentes e informação de gerenciamento de conteúdo.

Foi perguntado se algum tipo de ferramenta foi utilizada durante o andamento do projeto, os participantes responderam: ferramenta de análise de código,

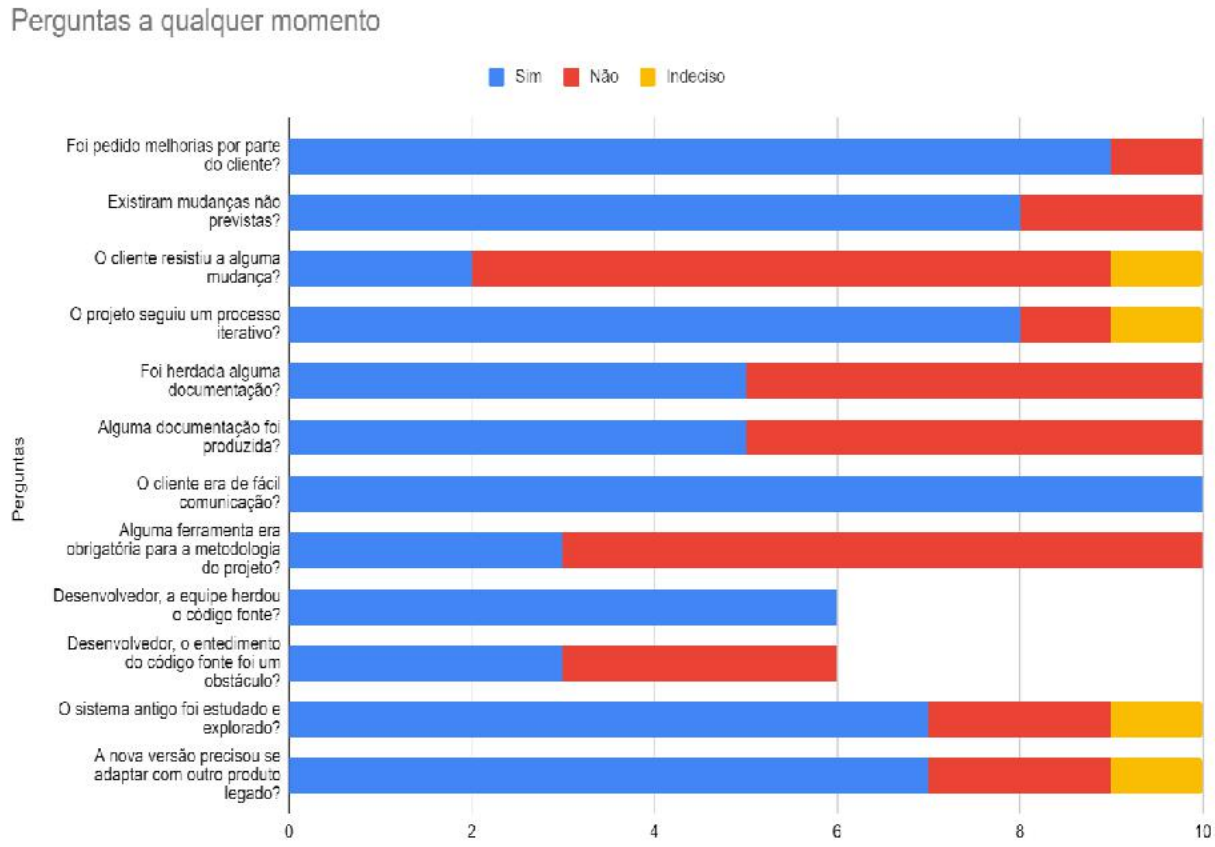
gerenciamento de atividades, guia de reuniões, de testes automáticos, de comunicação e devops.

Os desenvolvedores foram questionados para dar uma nota de um até dez sobre o quanto conheciam do produto em sua versão anterior, as notas foram distribuídas entre um (uma ocorrência), dois (uma ocorrência), três (duas ocorrências), nove (uma ocorrência) e dez (uma ocorrência), mas para os analistas foram seis (uma ocorrência), oito (uma ocorrência) e nove (duas ocorrências). Para a pergunta “Algum componente do sistema antigo foi reutilizado?”, os desenvolvedores responderam que trechos de código, bibliotecas específicas e regras de negócio foram reaproveitadas.

Ao fim da sessão existe um campo de texto aberto para adicionar algum comentário, e um único comentário afirmava que o maior desafio da migração foi extrair regras de negócio direto do código legado.

Demais respostas curtas dessa sessão estão no gráfico a seguir:

Figura 10 - Perguntas a qualquer momento



Fonte: desenvolvido pelo autor, 2021

4.3.5 Final

Essa seção possui um campo aberto para uma observação geral sobre o formulário como um todo, caso o participante queira destacar alguma dificuldade ou acontecimento. As respostam foram: tecnologia do projeto novo era nova para todos e o time precisou aprender; por falta de documentação houveram momentos de inspeção do código fonte para extrair regras de negócio e também momentos de retrabalho; após a migração concluída as manutenções ficaram mais rápidas; o sistema legado continuou com novas funcionalidades e precisou ser acordado com o cliente o momento de parar de dar suporte a versão que iria ser substituída.

4.4 Discussão

Como apresentado anteriormente na seção 4.3, pode-se afirmar que os métodos ágeis foram empregados em ambos os projetos, e segundo Stavru, Krasteva e Ilieva (2013) são esses os mais indicados. Porém, quando questionados sobre a devida aplicação das obrigações do framework, as respostas (principalmente por parte dos desenvolvedores) entregaram que não foram fielmente seguidas.

Embora o termo “Migração” seja por vezes relacionado com sistemas legados, foi explicado na seção 2.3 que isso não é uma exclusividade desse tipo de sistema. Observando a figura 7 nota-se uma tendência por parte dos desenvolvedores em responder que a migração não era de um sistema legado, enquanto analistas e gerentes afirmam se tratar de um. Portanto, de acordo com o estudo de Chervenski, Bordin e Dos Santos (2017), são várias as definições para sistema legado, o que pode ter sido a causa dessa discordância.

Em relação à preocupação com os stakeholders, a maioria dos analistas e gerentes respondeu de forma negativa. Desse modo, não estão alinhados com o SMART e com o Processo de Manutenção de Software, que julgam ser importante analisar as pessoas envolvidas.

As respostas para a relação da equipe de migração com a equipe do sistema mais antigo sugerem diferentes cenários, equipe com acesso às pessoas que participaram da versão anterior e equipe que não teve esse acesso. Quando a metodologia de Processo de Manutenção cita o importante envolvimento das

pessoas envolvidas, a metodologia também inclui as pessoas que participaram do desenvolvimento anterior.

Todos os entrevistados souberam responder quais os objetivos do projeto, os principais foram: melhorar a performance, abaixar custos, atualização tecnológica e diminuir complexidade da manutenção. O objetivo de diminuir a complexidade com a nova implementação pode ser entendido como uma maneira de mitigar o problema da degradação do código, como apresentado na seção 2.1.

Sobre a equipe, há comentários sobre um dos projetos ser mais "simples" e não precisar de papéis bem definidos e sobre a capacidade do gerente também assumir papel de analista quando necessário, desse modo, pode ser identificada a característica de equipe multifuncional, algo considerado pela abordagem ágil.

Desenvolvedores foram questionados sobre suas habilidades e experiências com as tecnologias dos sistemas. A maioria das respostas foi negando que todos da equipe dominavam as tecnologias tanto envolvidas no projeto novo quanto do projeto antigo e afirmaram ainda haver uma pessoa mais especializada no uso dessas tecnologias como também para coordenar a metodologia.

E de acordo com a tabela 2 o desafio de "Superar tecnologias obsoletas" entra na lista dos maiores problemas técnicos, enquanto que também "Aumento de risco devido a nova tecnologia" está presente na tabela 1.

Em conformidade com as metodologias XIRUP, Processo de Manutenção de Software e Perspectiva Ágil, que julgam a atividade de teste como importante para validação do software, as respostas relataram que essa prática foi seguida (figura 9).

Respostas de analistas/gerentes e de desenvolvedores afirmaram que pequenas entregas para validação do cliente e processos iterativos estavam no planejamento do projeto. Esses comportamentos já eram esperados quando no início os entrevistados afirmaram seguir métodos ágeis e seguem concordando com a Perspectiva Ágil.

Com a totalidade dos desenvolvedores e alguns votos dos analistas/gerentes o cliente pediu por melhorias ao produto durante o projeto, concordando com o ponto de vista da maioria dos próprios entrevistados que esse seria um bom momento para incrementar as funcionalidades (apresentado na figura 10). Vale ressaltar que em ambos os projetos o objetivo era manter os fluxos e layout o mais parecido possível (como mencionado na seção 4.1) e "Garantir equivalência comportamental entre sistemas" está na tabela 2 dos desafios.

Mudanças não previstas também aconteceram, o que de certa forma é considerado por todas as metodologias apresentadas no capítulo 3. O cliente pode ser um risco de resistência a essas alterações, mas a maioria dos entrevistados disseram que isso não aconteceu, caracterizando assim um desafio a menos, considerando a tabela 1 de desafios organizacionais (falta de suporte).

De maneira dividida, os entrevistados responderam que nenhum tipo de documento foi herdado (figura 10), e responderam igual para produção de documentação para o sistema novo (figura 10), assim pode-se perceber que a definição de documentação é diferente entre os perfis.

Com a totalidade das respostas positivas, o cliente era de fácil acesso para comunicação (figura 10). Um cenário ideal para facilitar o andamento do projeto, o que supera o desafio de "Falta de suporte" da tabela 1 e traz o cliente para dentro do projeto, algo também observado em todas as metodologias apresentadas no capítulo 3 (Relacionamento explícito com clientes).

As ferramentas utilizadas no projeto foram ferramentas de devops, de comunicação, de testes, de gerenciamento de atividades, guia de reunião e de análise de código. Quanto às ferramentas de análise de código e guia de reunião, há uma chance de o participante não saber do que se tratava, pois não são comuns no mercado.

A opção de ferramenta de análise de código foi colocada para referenciar às ferramentas automáticas e semiautomáticas do XIRUP e do Processo de Manutenção de Software. Já o guia de reuniões foi para fazer referência ao SMIG utilizado pelo SMART.

A maioria respondeu que nenhuma dessas ferramentas eram obrigatórias na metodologia do projeto. Nessa questão também há chances de as pessoas terem interpretado incorretamente, pois também não é algo comum encontrado no mercado. A pergunta faz jus ao XIRUP suite, um conjunto de ferramentas criado especificamente para ser usado com uma metodologia e que não pode existir o XIRUP sem seu ferramental.

Nota-se que os desenvolvedores tiveram acesso ao código fonte e metade das respostas dizem que isso representou um obstáculo para o desenvolvimento (figura 10). Tal característica de "Extrair conhecimentos comercial e técnico do sistema legado" foi listada como um dos desafios na tabela 2.

Na questão de estudo da versão anterior do sistema (Explícita análise do sistema legado) e entender o contexto da aplicação (Gerenciamento do conhecimento do negócio), percebe-se que o conhecimento estava mais concentrado com o perfil de analistas/gerentes. Quase que todas as metodologias do capítulo 3 possuem essas características e segundo Fuentes-Fernández, Pavón e Garijo (2012) existir um sistema já pronto que pode ser estudado é uma das particularidades nesse tipo de projeto.

Trechos de código e uso das mesmas bibliotecas externas foram toda a reutilização do código herdado segundo os desenvolvedores. As metodologias XIRUP, SMART e na Perspectiva Ágil o reuso é considerado, mas de forma limitada e em cenários específicos.

Com a maioria das respostas positivas, o sistema teve que se adaptar para integrar com serviços que já faziam parte do ambiente da aplicação (figura 10). Uma característica específica do cenário de migração já citada por Fuentes-Fernández, Pavón e Garijo (2012) e corresponde ao desafio de "Coexistência de sistemas" da tabela 2.

Com os comentários finais que fazem referência a todo o formulário foi possível destacar os maiores desafios relatados:

- Adquirir conhecimento da nova tecnologia para desenvolver a nova versão, algo mencionado pela tabela 1 em "Aquisição de competências e conhecimentos";
- A necessidade de ler e entender o antigo código fonte, característica presente em todas as metodologias do capítulo 3 e presente na tabela 2;
- Dificuldade em manter duas versões em paralelo. Considerando o cenário que a mesma empresa fazia a manutenção e o desenvolvimento da nova versão do produto. Precisou ser planejado com o cliente o momento de "congelar" as atualizações na versão que seria substituída.

5 CONCLUSÃO

5.1 Contribuições do trabalho

O presente trabalho visou responder a seguinte pergunta: “Quais as semelhanças entre teoria e prática em projetos de evolução de software?”. De acordo com os resultados obtidos pode-se concluir que embora a teoria exponha que esse tipo de projeto possui suas particularidades e para isso também existem metodologias mais focadas, ao mesmo tempo foi capaz de classificar os métodos ágeis como recomendados para tal cenário.

Na prática, apesar das equipes não estarem fielmente alinhadas com nenhuma metodologia, os métodos ágeis já parecem ser bem aceitos e difundidos no mercado, influenciando as equipes com algumas de suas práticas.

Teoria e prática estavam mais alinhadas a respeito dos desafios que tiveram de ser superados: o desafio em aprender uma nova tecnologia, falta de documentação, atividade de adaptar à nova versão para o ambiente antigo da aplicação e garantir equivalência comportamental entre as versões.

Além disso, na literatura foram relatadas especificidades que os entrevistados também mencionaram: o código da versão a ser substituída foi herdado, a aplicação em funcionamento e a equipe da versão mais antiga estavam disponíveis para consulta e o reuso foi considerado na medida do possível.

A fim de responder a seguinte pergunta: “Quais são as características de um projeto de evolução entre versões de software na prática?”, foi gerada a tabela 4 com base na análise da seção 4.4.

Tabela 4 - Relação entre as características e os resultados

	Característica	**Resultados**
	Antes da modernização	
1	Gerenciamento do conhecimento do negócio	Parcial
2	Análise dos stakeholders	Parcial
3	Estudo de viabilidade	Parcial
4	Requer especialista	X
5	Equipes multifuncionais	X
	Durante a modernização	

6	Geração de código (manual ou automatizado)	X
7	Teste de software	X
8	Pequenas entregas	X
	A qualquer momento	
9	Adaptação a mudanças	X
10	Processo iterativo	X
11	Criação de artefatos	Parcial
12	Relacionamento explícito com clientes (Participação do cliente)	X
13	Conjunto de ferramentas	Parcial
14	Inspeção do código fonte	X
15	Explícita análise do sistema legado	Parcial
16	Reuso	X

Fonte: desenvolvido pelo autor, 2021.

Gerenciamento de conhecimento do negócio (1) foi avaliado como parcial, pois fica claro com o questionário que os analistas tinham mais conhecimento em comparação com os desenvolvedores. Análise dos stakeholders (2), embora não considerado pelos entrevistados, precisou ser levado em consideração que a equipe nova tinha acesso a equipe da versão mais antiga. Estudo de viabilidade (3) foi parcial, pois as respostas dos analistas ficaram divididas.

A participação de especialistas (4) foi considerada, pois se pode observar que havia alguém para coordenar a equipe e alguém de referência para auxiliar no uso das tecnologias. A equipe multifuncional (5) também foi considerada, havia a possibilidade de uma mesma pessoa ocupar diferentes funções quando necessário.

Geração de código (6) foi feita, o projeto entregou o software em funcionamento. Os testes (7) foram considerados, e pequenas entregas (8) faziam parte do cronograma.

Adaptação a mudanças (9) foi necessário, pois além das melhorias pedidas pelo cliente, as respostas também deram positivo para mudanças não antes previstas. Segundo as respostas um processo iterativo (10) foi adotado. Para criação dos artefatos (11), apenas os analistas, em maioria, afirmaram que tinham produzido.

A participação do cliente (12) foi fundamental para o projeto. De forma unânime todos concordaram que o cliente era de fácil acesso. O conjunto de ferramentas (13) utilizado foi dos mais variados, porém nada específico que fosse

obrigatório usar por conta das práticas ágeis. A inspeção do código fonte (14) esteve presente, pois os desenvolvedores o herdaram e sua interpretação até representou um desafio a ser superado.

Explícita análise do sistema legado (15) foi parcial pois pode-se perceber que os analistas estavam com maior entendimento do sistema em comparação com os desenvolvedores. E o reuso (16) estava presente, pois os desenvolvedores consideraram aproveitar trechos de código e procuraram manter as mesmas bibliotecas.

Podemos perceber a partir da observação da tabela 4, que proporcionalmente a maioria das características com alguma carência estão na sessão de "Antes da modernização", sugerindo que a maioria dos desafios encontrados poderiam ser suavizados pelas equipes caso se investisse mais esforço em análise, comunicação e estudo de uma forma mais cuidadosa e completa antes da modernização começar.

O presente trabalho foi capaz de trazer a literatura para conhecimento da empresa em questão, como também expor as características de experiências passadas para que os próximos projetos de evolução sejam cada vez melhor gerenciados e com riscos mais controlados.

No entanto, não podemos generalizar os resultados devido a um pequeno número de metodologias analisadas, limitado número de projetos, apenas uma empresa considerada e um restrito grupo de participantes. Participantes que por sua vez deram suas próprias opiniões sobre o assunto, ou seja, os resultados a respeito dos projetos reais dependeram de suas memórias e experiências. Embora isso seja uma ameaça a validade, podemos entender como um forte indício de uma prática da empresa para projetos dessa natureza.

5.2 Trabalhos futuros

Como trabalho futuro, novas metodologias de "Migração", "Modernização", "Evolução" e "Manutenção" presentes na literatura podem ser consideradas a fim de validar ou incrementar a lista de características da seção 3.2. O questionário, se aplicado a outros projetos dessa mesma natureza, trará mais confiabilidade no momento de gerar a tabela de resultados.

REFERÊNCIAS

CHERVENSKI, Alex Severo; BORDIN, Andrea Sabedra; DOS SANTOS, Daniele Martins. MAPEAMENTO DE CARACTERÍSTICAS DE SISTEMAS LEGADOS. **Anais do Salão Internacional de Ensino, Pesquisa e Extensão**, v. 8, n. 2, 2017.

DURELLI, Rafael S. et al. A mapping study on architecture-driven modernization. In: **Proceedings of the 2014 IEEE 15th international conference on information reuse and integration (IEEE IRI 2014)**. IEEE, 2014. p. 577-584.

FUENTES-FERNÁNDEZ, Rubén; PAVÓN, Juan; GARIJO, Francisco. A model-driven process for the modernization of component-based systems. **Science of Computer Programming**, v. 77, n. 3, p. 247-269, 2012.

HASSAN, Ahmed E.; HOLT, Richard C. A lightweight approach for migrating Web frameworks. **Information and Software Technology**, v. 47, n. 8, p. 521-532, 2005.

KHAN, Khaled et al. Tasks and Methods for Software Maintenance: a process oriented framework. **Australasian Journal of Information Systems**, v. 9, n. 1, 2001.

LEHMAN, Meir M.; PERRY, Dewayne E.; RAMIL, Juan F. On evidence supporting the feast hypothesis and the laws of software evolution. In: **Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No. 98TB100262)**. IEEE, 1998. p. 84-88.

LEWIS, Grace et al. Service-oriented migration and reuse technique (smart). In: **13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05)**. IEEE, 2005. p. 222-229.

RAJLICH, Václav T.; BENNETT, Keith H. A staged model for the software life cycle. **Computer**, v. 33, n. 7, p. 66-71, 2000.

SOMMERVILLE, Ian. **Software engineering**. 9. ed. Addison-Wesley/Pearson, 2011.

STAVRU, Stavros; KRASTEVA, Iva; ILIEVA, Sylvia. Challenges of Model-driven Modernization-An Agile Perspective. In: **MODELSWARD**. 2013. p. 219-230.

SHULL, Forrest; SINGER, Janice; SJOBERG, Dag I. K. Guide to Advanced Empirical Software Engineering. **Springer**, 2008. p. 63-92

APÊNDICE A: Questionário completo sem distinção entre os perfis de desenvolvedores e gerente/analistas.

Qual metodologia de desenvolvimento foi utilizada no seu projeto?

- ◯ SCRUM
- ◯ XP
- ◯ KANBAN
- ◯ Processo Unificado (RUP)
- ◯ Outros...

Essa metodologia foi seguida com todas as obrigações que ela propõe?

	1	2	3	4	5	6	7	8	9	10	
Nada foi seguido	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Tudo foi seguido

Você considera que o sistema migrado é um sistema legado? Por que?

Texto de resposta longa

Você entende que em um projeto de migração deve-se construir um sistema exatamente igual ao antigo? Sem nada a mais e nem nada a menos.

Texto de resposta curta

Antes de iniciar a migração foi entendido o contexto que a aplicação estava inserida?

- ◯ Sim
- ◯ Não

Foram realizadas análises para saber as necessidades e dificuldades dos stakeholders que utilizam o sistema?

- ◯ Sim
- ◯ Não

Houveram mudanças que surgiram a partir dos próprios usuários finais?

- ◯ Sim
- ◯ Não

A equipe responsável pela migração tinha acesso a equipe que foi responsável pelo desenvolvimento? Ou é a mesma equipe?

- ◯ Sim
- ◯ Não
- ◯ Mesma equipe

Para um processo de migração acontecer é necessário entender os objetivos do cliente com esse projeto. Quais eram os principais objetivos do cliente com a migração?

Texto de resposta longa

Esses objetivos foram todos atendidos?

- ◯ Sim
- ◯ Não

Foi considerada modernização através de manutenção para resolver o problema do cliente antes mesmo de pensar em fazer a migração do sistema?

- ◯ Sim
- ◯ Não

Qual foi o critério para escolher as tecnologias envolvidas no projeto?

Texto de resposta curta

Existia alguém para coordenar/manter a metodologia de desenvolvimento em pleno funcionamento?

- ◯ Sim
- ◯ Não

Existia alguém no projeto que era referência com conhecimentos da tecnologia empregada no sistema antigo?

- ◯ Sim
- ◯ Não

Existia alguém no projeto que era referência com conhecimentos da tecnologia empregada no sistema novo?

- ◯ Sim
- ◯ Não

Existia uma equipe multifuncional no projeto? Ou seja, cada membro tinha suas diferentes responsabilidades e papéis bem definidos?

- ◯ Sim
- ◯ Não

O quão igual ao sistema antigo é o sistema novo? (interface, navegação, funcionalidades)

	1	2	3	4	5	6	7	8	9	10	
Nada igual	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Completamente igual

A tecnologia empregada no sistema novo era de domínio de todos da equipe?

- ◯ Sim
- ◯ Não

A tecnologia empregada no sistema antigo era de domínio de todos da equipe?

- ◯ Sim
- ◯ Não

Foram feitos testes para a implementação do novo sistema?

- ◯ Sim
- ◯ Não

Quais tipos de teste foram feitos?

- ☐ Testes unitários
- ☐ Teste de integração
- ☐ Testes manuais
- ☐ Teste de usabilidade
- ☐ Teste de segurança
- ☐ Teste de estresse
- ☐ Outros...

Algum teste ou fluxo de teste pôde ser aproveitado do sistema antigo?

- ◯ Sim
- ◯ Não

Foram feitas pequenas entregas para validação do cliente antes da entrega do produto final?

- ◯ Sim
- ◯ Não

Foi pedido que a migração adicionasse melhorias ao produto final?

- ◯ Sim
- ◯ Não

Existiram mudanças necessárias não antes previstas, por questões de arquitetura, tecnologia ou paradigma?

- ◯ Sim
- ◯ Não

O cliente resistiu a alguma mudança no sistema?

- ◯ Sim
- ◯ Não

Foi utilizado uma metodologia de desenvolvimento com processos iterativos para fazer a migração?

- ◯ Sim
- ◯ Não

Foi herdada algum tipo de documentação do sistema antigo?

- ◯ Sim
- ◯ Não

Foi produzida algum tipo de documentação para o sistema novo?

- ◯ Sim
- ◯ Não

Quais foram as documentações produzidas?

Texto de resposta curta

O cliente era de fácil acesso para comunicação?

- ◯ Sim
- ◯ Não

Foi utilizada alguma ferramenta para facilitar o andamento do projeto?

- ☐ Ferramenta para análise de código (automática ou semi-automática)
- ☐ Ferramenta para criação de código (automática ou semi-automática)
- ☐ Ferramenta para gerenciar atividades
- ☐ Guia de reuniões
- ☐ Ferramenta de testes automatizados
- ☐ Ferramenta de comunicação
- ☐ Ferramenta de devops
- ☐ Outros...

Alguma ferramenta era específica para a metodologia de desenvolvimento escolhida para o projeto? (A metodologia não pode acontecer sem o uso da ferramenta)

- ◯ Sim
- ◯ Não

A equipe tinha acesso ao código fonte?

- ◯ Sim
- ◯ Não

O entendimento do código antigo representou um obstáculo para a migração?

- ◯ Sim
- ◯ Não

O sistema antigo foi explorado e estudado para fazer a migração?

- ◯ Sim
- ◯ Não

O quanto você conhecia das funcionalidades do sistema antigo?

	1	2	3	4	5	6	7	8	9	10	
0%	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	100%

Algum componente do sistema antigo foi reutilizado?

- ☐ Trecho de código
- ☐ Fluxo de teste
- ☐ Arquitetura
- ☐ Bibliotecas específicas
- ☐ Outros...

O sistema desenvolvido teve que se adaptar para se comunicar com algum serviço que já fazia parte do ambiente da aplicação?

- ◯ Sim
- ◯ Não

Alguma dificuldade/acontecimento percebido por você que queira destacar?

Texto de resposta longa

APÊNDICE B: Relação das características com as perguntas do questionário.

Gerenciamento do conhecimento do negócio	• Antes de iniciar a migração foi entendido o contexto que a aplicação estava inserida?
Análise dos stakeholders	• Foram realizadas análises para saber as necessidades e dificuldades dos stakeholders que utilizam o sistema? • Houveram mudanças que surgiram a partir dos próprios usuários finais? • A equipe responsável pela migração tinha acesso à equipe que foi responsável pelo desenvolvimento? Ou é a mesma equipe?
Estudo de viabilidade	• Para um processo de migração acontecer é necessário entender os objetivos do cliente com esse projeto. Quais eram os principais objetivos do cliente com a migração? • Esses objetivos foram todos atendidos? • Foi considerada modernização através de manutenção para resolver o problema do cliente antes mesmo de pensar em fazer a migração do sistema? • Qual foi o critério para escolher as tecnologias envolvidas no projeto?
Requer especialista	• Existia alguém para coordenar/manter a metodologia de desenvolvimento em pleno funcionamento? • Existia alguém no projeto que era referência com conhecimentos da tecnologia empregada no sistema antigo? • Existia alguém no projeto que era referência com conhecimentos da tecnologia empregada no sistema novo?
Equipes multifuncionais	• Existe uma equipe multifuncional no projeto? Ou seja, cada membro tinha suas diferentes responsabilidades e papéis bem definidos?
Geração de código	• O quão igual ao sistema antigo é o sistema novo? (interface, navegação, funcionalidades) • A tecnologia empregada no sistema novo era de domínio de todos da equipe?
Teste de software	• Foram feitos testes para a implementação do novo sistema? • Quais tipos de testes foram feitos? (seleção e outros) • Algum teste ou fluxo de teste pôde ser aproveitado do sistema antigo?
Pequenas entregas	• Foram feitas pequenas entregas para validação do cliente antes da entrega do produto final?
Adaptação a mudanças	• Foi pedido que a migração adicionasse melhorias ao produto final? • Existiram mudanças necessárias, não antes previstas, por questões de arquitetura, tecnologia ou paradigma? • O cliente resistiu a alguma mudança no sistema?
Processo iterativo	• Foi utilizado uma metodologia de desenvolvimento com processos iterativos para fazer a migração?
Criação de artefatos	• Foi herdada algum tipo de documentação do sistema antigo?

	• Foi produzido algum tipo de documentação para o sistema novo? • Quais foram as documentações produzidas?
Relacionamento explícito com o cliente	• O cliente era de fácil acesso para comunicação?
Conjunto de ferramentas	• Foi utilizada alguma ferramenta para facilitar o andamento do projeto? • Alguma ferramenta era específica para a metodologia de desenvolvimento escolhida para o projeto?
Inspeção do código fonte	• A equipe tinha acesso ao código fonte? • O entendimento do código antigo representou um obstáculo para a migração?
Explícita análise do sistema legado	• O sistema antigo foi explorado e estudado para fazer a migração? • O quanto você conhecia das funcionalidades do sistema antigo?
Reuso	• Algum componente do sistema antigo foi reutilizado? • O sistema desenvolvido teve que se adaptar para se comunicar com algum serviço que já fazia parte do ambiente da aplicação?