Technical Debt Management in the Context of Agile Methods: A Systematic Literature Review.

Abstract

The adoption of agile methods has been growing at a fast pace. Its values and principles speak about software artifacts and ways to write good quality code and, thus, minimize the introduction of technical debts. This term is an analogy to the concept of financial debt, related to immature, incomplete or inadequate artifacts in the software development cycle, which cause low quality and increase the cost of maintaining systems. However, the adoption of the agile methodology alone does not guarantee debt-free software, so that to maintain a sustainable pace, technical debt needs to be managed efficiently. In addition, interest on technical debt is difficult to assess. The severity of its negative effects can depend on the context of the organization and the estimates can be subjective. This research aims to identify in the literature methods and techniques for making these measurements more systematic, less subjective and consuming less time and cost.

Keywords: Technical Debt; Agile Methods.

Gerenciamento da Dívida Técnica no Contexto de Métodos Ágeis: Uma Revisão Sistemática da Literatura.

Resumo

A adoção de métodos ágeis vem crescendo em ritmo acelerado. Seus valores e princípios falam sobre artefatos de *software* e maneiras de escrever código de boa qualidade e, assim, minimizar a introdução de dívidas técnicas. Este termo é uma analogia ao conceito de dívida financeira, relacionado a artefatos imaturos, incompletos ou inadequados no ciclo de desenvolvimento de *software*, que causam baixa qualidade e aumentam o custo de manutenção dos sistemas. No entanto, só a adoção da metodologia ágil não garante *software* livre de dívidas, de modo que para manter um ritmo sustentável, a dívida técnica precisa ser gerenciada com eficiência. Além disso, os juros da dívida técnica são de difícil avaliação. A severidade de seus efeitos negativos podem depender do contexto da organização e as estimativas podem ser subjetivas. Esta pesquisa tem como objetivo identificar na literatura métodos e técnicas de tornar estas medições mais sistematizadas, menos subjetivas e consumindo menor tempo e custo.

Palavras-Chave: Dívida Técnica; Métodos Ágeis.

1. INTRODUÇÃO

Nesta seção são descritos o problema e sua contextualização, os objetivos a serem alcançados, a questão de pesquisa, a metodologia adotada, as contribuições à comunidade científica e da indústria de software e a organização das demais seções deste trabalho.

O termo Dívida Técnica (DT) não é uma teoria ou conceito cientifico, é uma metáfora que foi apresentado pela primeira vez em 1992 na conferência Object-Oriented Programming, Systems, Languages & Applications (OOPSLA) por Ward Cunningham, como uma alusão à dívida financeira.

Segundo o autor, a implantação de artefatos de software imaturos, incompletos ou inadequados, se assemelha a entrar em uma dívida, que pode ser acrescida de juros na forma de esforço, custo e tempo extra na realização de manutenções futuras relacionadas a estes módulos.

Para Cunningham, uma pequena dívida acelera o desenvolvimento, desde que seja paga prontamente com uma reescrita. Benefícios de curto prazo, como tempo ou esforço de desenvolvimento reduzido, tornam o custo dessa transação tolerável. O perigo ocorre quando a dívida não é paga, cada minuto gasto em um código com baixa qualidade conta como juros sobre essa dívida.

Muitas pesquisas tem sido realizadas para encontrar formas que possam auxiliar a medição e o gerenciamento da DT. O *framework* proposto por Seaman e Guo (2011), que propõe um método para identificar, medir e gerir a DT, é o mais encontrado na revisão sistemática efetuada neste trabalho, e também teve 163 citações no *Google Scholar* até 13 de junho de 2020.

O trabalho Oliveira (2015) utiliza o framework de Seaman e Guo (2011), mas aponta duas dificuldades. A primeira está relacionada ao uso de três varáveis de natureza probabilística, o que dificulta sua estimativa e enfraquece sua credibilidade. A segunda está relacionada à falta de ferramentas que gerem gráficos para comparação entre os juros e o principal associados à DT.

Para os autores de Griffith et al. (2014), a DT é um fenômeno bem compreendido, porém existe uma lacuna no que se refere à verificação e validação dos métodos propostos para o gerenciamento da DT no contexto do desenvolvimento ágil. Na prática, essas avaliações são caras ou consomem muito tempo para serem realizadas usando os métodos empíricos tradicionais.

Segundo os autores, o acumulo da DT diminui a produtividade do time, pois se o custo da mudança aumenta devido a DT, enquanto o número de engenheiros de *software* permanece constante, o impacto é que a produtividade diminui. Desta forma, entendem ser necessário mais estudos empíricos que auxiliem na diminuição dos custos para fazer a gestão da dívida técnica.

Para Martini et al.(2017), os juros da DT são difíceis de calcular. As abordagens atuais são baseadas na avaliação de questões refinadas, mas não há evidências sobre como a DT é avaliada no nível do projeto. Algumas ferramentas usam uma função de agregação que soma os problemas de DT no nível do projeto. No entanto, isso não ajuda a avaliar o impacto da DT numa visão ampla.

Para Martini e Bosch (2017), os juros da DT são de difícil avaliação. A gravidade dos efeitos negativos da dívida técnica podem depender do contexto da organização e as

estimativas podem ser subjetivas e de alto custo. Desta forma, torna-se necessário avaliar os juros da DT de maneira mais sistemática.

Martini e Bosch (2017), com base nos resultados de pesquisas anteriores, propõe o uso de uma ferramenta chamada AnaConDebt (Analysis of Contagious Debt) desenvolvidas por eles, para avaliar a gravidade dos juros de itens de DT. A avaliação sistemática e semi-automática de sete fatores e seu crescimento foi comparada às estimativas intuitivas das partes interessadas. Os resultados mostram que o resultado da ferramenta está muito próximo da estimativa dada pelas partes interessadas.

Para os autores, as implicações são que, se dados adicionais corroborarem a hipótese, a gravidade dos juros poderá ser sistematicamente avaliada pelas partes interessadas, estimando apenas sete fatores de maneira econômica com resultados aceitáveis.

Pelo estudo publicado pelo Gartner (2016), sistemas de software são executados frequentemente com eficiência abaixo do ideal, devido à dívida técnica que se acumula inevitavelmente em softwares grandes e complexos. Relaciona diversas queixas causadas por dívida técnica, como por exemplo: infraestrutura e operações com alto custo operacional; times de negócios frustrados com o custo e o ritmo lento de entrega das mudanças nos softwares e times de segurança e gerenciamento de riscos de TI descobrindo que o aplicativo está falhando em acompanhar os padrões de segurança.

No estudo Besker et al. (2019) foi constatado que quase um quarto do tempo de trabalho de todos os desenvolvedores, objetos do estudo, é relatado como desperdiçado devido a dívida técnica.

Nas metodologias ágeis de desenvolvimento de software as funcionalidades priorizadas são àquelas que entregam mais valor do ponto de vista do negócio, colocando muitas vezes a qualidade do sistema em segundo plano, devido a restrições orçamentárias ou de tempo, aumentando assim o acumulo de DT ao longo do tempo. O time técnico tem dificuldade para justificar e defender a priorização de entregas técnicas que resolvam exclusivamente a DT.

A adoção de métodos e práticas ágeis vem crescendo no Brasil e no mundo. Segundo previsões do Gartner publicada em 2018, até 2021 90% das organizações que produzem software terão adotado metodologias ágeis em 60% dos seus projetos. Também preveem que até 2023, 90% de toda a dívida técnica existente hoje ainda existirá e continuará a estrangular as inovações de negócios.

Empresas tradicionais, mais antigas no mercado, possuem sistemas legados com tecnologias obsoletas e arquiteturas complexas. Enfrentam o desafio de responder rapidamente às necessidades de mercado, sejam elas novas funcionalidades, lançamento de novos produtos ou mudanças regulatórias. Estes sistemas carregam dívidas técnicas que funcionam como ancoras que seguram e atrasam sua evolução.

Em função do que foi descrito nesta seção é possível verificar a necessidade de buscar formas mais econômicas e sistematizadas para auxiliar no gerenciamento da dívida técnica.

O objetivo principal deste trabalho é identificar na literatura métodos e técnicas, no contexto de desenvolvimento ágil de software, que tragam mais visibilidade aos itens de dívida técnica e como estes impactam negativamente às organizações e seus clientes, assim como buscar formas de aprimorar as medições de custo e juros, tornando-as mais sistematizadas, menos subjetivas, e consequentemente com menor tempo e custo de execução.

De modo a alcançar este objetivo, esta pesquisa procura responder à seguinte questão:

- Como as abordagens, métodos ou técnicas ágeis existentes contribuem para tornar as medições e priorização da dívida técnica mais sistematizadas e menos subjetivas?

Espera-se que o conhecimento gerado por esta pesquisa contribua no aumento da qualidade e redução de custos no desenvolvimento e manutenção de *software*, a fim de alavancar a inovação tecnológica de empresas que utilizam métodos ágeis em seu desenvolvimento de *software*.

O artigo está dividido em quatro seções. A segunda traz a fundamentação teórica com uma visão geral sobre dívida técnica e métodos ágeis. A terceira apresenta o método de revisão sistemática da literatura adotado neste trabalho. A quarta seção fornece a análise e os resultados obtidos na revisão sistemática e a quinta seção traz as conclusões, limitações e sugestões para futuros trabalhos de pesquisa e aplicação prática dos conceitos nas organizações.

2. FUNDAMENTAÇÃO TEÓRICA

Nesta seção será apresentada uma visão geral sobre dívida técnica, métodos e algumas práticas ágeis.

2.1 Dívida Técnica

Nesta subseção é apresentada uma visão geral do termo DT em sistemas de *software*, iniciando com definições na literatura trazendo conceitos gerais sobre o tema, seguido do que não é considerado dívida técnica, acompanhado de suas propriedades, classificação e categorias, finalizando com as estratégias de gestão para seu pagamento.

2.1.1 Conceitos Gerais

O termo Dívida Técnica (DT) não é uma teoria ou conceito cientifico, é uma metáfora que foi apresentada pela primeira vez em 1992 na conferência *Object-Oriented Programming, Systems, Languages & Applications* (OOPSLA) por Ward Cunningham, como uma alusão à dívida financeira.

Segundo Cunningham (1992), a implantação de artefatos de software imaturos, incompletos ou inadequados no ciclo de desenvolvimento de *software* assemelha-se a entrar em uma dívida, que pode ser acrescida de juros na forma de esforço, custo e tempo extra na realização de manutenções futuras nestes artefatos.

Para McConnell (2008), o time do projeto toma decisão de seguir por uma abordagem de projeto ou construção que é conveniente no curto prazo, mas que aumenta a complexidade e é mais caro no longo prazo.

Seaman e Guo (2011) concordam com esta consequência da DT, ao mencionar que artefatos imaturos, incompletos ou inadequados no ciclo de desenvolvimento de *software*, causam baixa qualidade e aumentam o custo de manutenção dos sistemas.

Outra maneira de explicar, segundo Brown et al. (2010), a DT é a diferença entre o estado atual de um software e hipóteses de estado “ideal”, este último quando o *software* estiver otimizado em um ambiente particular.

Para Bob (2009), as empresas fazem esse tipo de troca o tempo todo; e não há nada de imprudente nisso. Se o lançamento antecipado da versão 1.0 com DT, impulsionar o negócio que paga pelo desenvolvimento da versão 2.0, o negócio tem ganho em adotar esta

prática. Portanto, esse tipo de DT fundamentada pode realmente ser uma estratégia apropriada.

Da mesma forma que ocorre na dívida financeira, a DT implica em pagamento de juros, que é um esforço extra para se realizar um desenvolvimento devido a uma escolha original não apropriada. Pode-se optar por continuar pagando os juros ou pagar a dívida por meio da refatoração para melhorar aspectos da construção ou do projeto (FOWLER, 2003).

Para Fowler et al. (2018), refatoração é o processo de mudar um *software* a fim de melhorar sua estrutura interna e reusabilidade sem alterar seu comportamento funcional original.

O uso da metáfora da DT auxilia na explicação de aspectos técnicos críticos aos *stakeholders* que não fazem parte da equipe técnica, conforme destacado por Fowler (2009). Brown et al. (2010) também compartilham da mesma opinião quando mencionam que a metáfora está cada vez mais reconhecida como de grande utilidade tanto para a comunicação técnica quanto para a comunicação entre engenheiros e executivos.

A mudança do diálogo a partir de um vocabulário técnico para um vocabulário financeiro viabiliza um cenário mais claro e compreensível para essas discussões (MCCONNELL, 2008).

2.1.2 Mal Entendido sobre Dívida Técnica

Segundo Bob (2009), um *software* desestruturado não é uma DT, é apenas *software* desestruturado. As decisões técnicas de DT são tomadas com base em restrições reais do projeto, podendo ser arriscadas, porém devem trazer algum tipo de benefício. A decisão de fazer um *software* desestruturado nunca é racional, sempre se baseia na falta de conhecimento técnico e de profissionalismo, e não tem chance de ser paga no futuro. Um software desestruturado é sempre uma perda e não traz nenhum benefício.

Para McConnell (2008), nem todo trabalho incompleto é DT, por exemplo *backlog* de produto, funcionalidades adiadas, funcionalidades excluídas e assim por diante não são dívidas técnicas, a não ser que gerem pagamento de juros no futuro.

2.1.3 Framework de Gerenciamento da Dívida Técnica

Carolyn Seaman e Yuepu Guo propõem um *framework* de gerenciamento da DT (SEAMAN E GUO 2011), representado na figura 1, composto por 3 etapas: identificação da DT, estimativa da DT e tomada de decisão, que giram em torno de uma lista de itens de DT. Cada item de DT representa uma tarefa que não foi concluída, mas que corre o risco de causar problemas no futuro.

Figura 1 – Framework de Gerenciamento da DT
Fonte: Adaptado de Seaman e Guo (2011)

Identificação da DT

Nesta etapa, o item da DT é identificado, classificado e registrado no formulário de DT apresentado no quadro 1**Error! Reference source not found.**.

Exemplos de itens de DT técnica podem ser módulos que precisam de refatoração, testes que precisam ser executados, inspeções ou revisões de código que precisam ser feitas, documentação que precisa ser escrita ou atualizada, problemas de conformidade arquitetural, defeitos latentes conhecidos que precisam ser removidos e assim por diante. Cada item inclui uma descrição de onde está o item da DT no sistema e por que ela precisa ser feita.

Li et al. (2015) e Alves et al. (2016) classificam a DT de acordo com a sua causa, baseada em sua origem.

A seguir estão listados os 15 tipos de DT mapeados em Alves et al. (2016), ordenados pelos que ocorreram com maior frequência em seu estudo. A lista contém o tipo de DT, sua descrição e suas referências.

1) DT de Projeto: refere-se a dívidas que podem ser descobertas analisando o código-fonte e identificando violações dos princípios do bom design orientado a objetos, como por exemplo, classes muito grandes ou fortemente acopladas. Referências: Guo e Seaman (2011); Izurieta et al.(2012);

2) DT de Arquitetura: refere-se a problemas encontrados na arquitetura do produto, por exemplo, violação da modularidade, que pode afetar os requisitos arquiteturais como desempenho, robustez, entre outros. Referências: Brown et al. (2010); Kruchten et al. (2012);

3) DT de Documentação: refere-se a problemas encontrados na documentação do projeto de software, e pode ser identificada procurando documentação faltante, inadequada ou incompleta. Referência: Guo e Seaman (2011);

4) DT de Teste: refere-se a problemas encontrados nas atividades de teste que podem afetar a qualidade do software. Exemplos desse tipo de DT são testes planejados que não foram executados ou deficiências conhecidas no conjunto de testes, por exemplo, baixa cobertura de código. Referência: Guo e Seaman (2011);

5) DT de Código: refere-se a problemas encontrados no código-fonte que podem afetar negativamente a legibilidade do código, tornando-o mais difícil de manter. Geralmente, essa DT pode ser identificada examinando o código-fonte em busca de problemas relacionados a práticas inadequadas de codificação. Referência: Bohnet e Dollner (2011);

6) DT de Defeito: refere-se a defeitos conhecidos, geralmente identificados por atividades de teste ou pelo usuário e relatados em sistemas de rastreamento de defeitos (*bugs*), que o time responsável pela liberação da versão concorda que devem ser corrigidos, mas devido a prioridades concorrentes e recursos limitados, devem ser adiados para mais tarde. As decisões tomadas para adiar a correção de defeitos podem acumular uma quantidade significativa de DT para o produto, dificultando sua correção posteriormente. Referência: Snipes et al. (2012);

7) DT de Requisitos: refere-se a trocas feitas com relação a quais requisitos a equipe de desenvolvimento precisa implementar ou como implementá-los. Engloba requisitos funcionais e não funcionais que se não implementados tem o risco de trazer problemas de qualidade no futuro. Alguns exemplos desse tipo de dívida são: requisitos que são implementados apenas parcialmente, ou de uma maneira que não atenda a todos os requisitos

não funcionais, por exemplo, segurança, desempenho e etc.. Referência: Kruchten et al. (2012);

8) DT de Infraestrutura: refere-se a problemas de infraestrutura que, se presentes no ciclo de vida do *software*, podem atrasar ou dificultar algumas atividades de desenvolvimento. Alguns exemplos desse tipo de DT são versões obsoletas de sistema operacional que atrasam uma atualização ou correção de infra-estrutura. Referência: Seaman e Spínola (2013);

9) DT de Pessoas: refere-se a problemas relacionados a pessoas que podem atrasar ou dificultar algumas atividades de desenvolvimento. Um exemplo desse tipo de dívida é a especialização concentrada em poucas pessoas, como efeito de atraso no treinamento ou contratação. Outro exemplo é a motivação do time, que se estiver em baixa pode impactar a produtividade. Referência: Seaman e Spínola (2013);

10) DT de Automação de Testes: refere-se ao trabalho envolvido nos testes de automação de funcionalidades desenvolvidas anteriormente para oferecer suporte à integração contínua e a ciclos de desenvolvimento mais rápidos. Essa dívida pode ser considerada um subtipo de dívida de teste. Referência: Codabux e Williams (2013);

11) DT de Processo: refere-se a processos que mudaram e o sistema projetado deixou de atender à nova necessidade. Referência: Codabux e Williams (2013);

12) DT de Compilação: refere-se a problemas que dificultam a tarefa de compilação e/ou geração do executável e consomem tempo desnecessariamente. O processo de compilação pode envolver código que não contribui para agregar valor ao cliente. Além disso, se o processo de compilação precisar executar dependências mal definidas, o processo se torna desnecessariamente lento e complexo. Referência: Morgenthaler et al. (2012);

13) DT de Serviço: refere-se à seleção e substituição inadequadas de serviços da Web que levam à incompatibilidade entre os recursos de serviço e os requisitos dos aplicativos. Além disso, esse tipo de dívida também leva à subutilização ou superutilização do sistema, integrando um serviço que não utiliza os recursos do sistema da maneira esperada, por exemplo, falta de memória devido a um serviço que não segue o processo de processamento de dados esperado, ou falta de desempenho devido a um serviço que não usa a memória disponível para a tarefa. Esse tipo de dívida é relevante para sistemas com arquiteturas orientadas a serviços. Referência: Alzaghoul e Bahsoon (2013);

14) DT de Usabilidade: refere-se a decisões inadequadas de usabilidade que precisarão ser ajustadas posteriormente. Exemplos dessa dívida são a falta de padrão de usabilidade e a inconsistência entre os aspectos de navegação do *software*. Referências: Zazworka et al. (2013); Potdar e Shihab (2014);

15) DT de Versão: refere-se a problemas de controle de versão de código-fonte. Referência: Greening (2013).

Estimativa da DT

Nesta etapa são estimados o custo e os juros da DT. Assim como a dívida financeira, o termo principal refere-se ao custo associado ao esforço necessário para se pagar a dívida no momento atual.

Os juros são composto de duas partes: valor dos juros, que é uma estimativa da quantidade de trabalho extra que será necessária para se pagar o item da dívida no futuro; e probabilidade dos juros, que é a probabilidade de que a dívida, senão for paga, tornará outros trabalhos mais caros durante um determinado período de tempo ou um release.

Os juros, resultado da multiplicação do valor do juros pela probabilidade dos juros também é chamado de benefício, uma vez que está associado ao valor que deixará de ser gasto caso a DT seja paga no momento atual, evitando assim o custo extra de deixá-la para ser paga depois.

O custo e os juros são registradas para completar o formulário da DT, conforme mostra o quadro 1. A lista de itens de DT deve ser revisada e atualizada após cada release, quando itens devem ser adicionados e/ou removidos.

Identificador	Código que identifica a DT.
Data	Data em que a DT foi identificada.
Responsável	Pessoa responsável pela identificação da DT.
Classificação	Representa o tipo da DT dentre os quinze a seguir: requisitos, arquitetura, design, código, teste, compilação, documentação, infra, controle de versão, defeito, pessoas, automação de testes, processo, serviço, usabilidade.
Descrição	Descrição sucinta da DT.
Localização	Descrição de onde se encontra a DT, por exemplo método X no módulo Y, do sistema Z.
Custo ou Principal	Estimativa do nível de esforço necessário para modificação do sistema a fim de pagar a DT: baixo, médio ou alto.
Valor de Juros	Estimativa do nível de esforço extra necessário para modificar o sistema no futuro, a fim de pagar a DT: baixo, médio ou alto.
Probabilidade dos Juros	Estimativa da probabilidade dos juros acontecer no futuro, possibilidade de aumento na quantidade de esforço, caso a modificação seja realizada no futuro: baixa, média ou alta.
Juros ou Beneficio	Multiplicação do "valor de juros" pela "probabilidade dos juros".

Quadro 1 – Formulário de Dívida Técnica
Fonte: Adaptado de Seaman e Guo (2011)

Tomada de Decisão

Em relação à tomada de decisão para pagamento ou não da DT, Seaman et al. (2012) propõem quatro abordagens distintas para priorizar os itens de DT. São elas:

- Abordagem AHP (Analytic Hierarchical Process): neste modelo são atribuídos pesos e escalas para critérios que são usados para medir a DT. Em seguida, são realizadas comparações em pares entre as alternativas a fim de obter uma ordem priorizada de itens da DT;
- Abordagem Portfólio: neste modelo, o objetivo é selecionar um conjunto de ativos, ou itens de DT, que maximize o retorno do investimento e minimize o risco para então decidir a ordem de prioridade entre os itens de DT;
- Abordagem Opções: neste modelo dá-se prioridade aos itens de DT que tem potencial para facilitar mudanças futuras;
- Análise de Custo-Benefício: abordagem utilizada no framework de Seaman e Guo (2011), onde são atribuídas escalas ordinais de medida como "baixo", "médio" ou

"alto" para o Principal, Juros e Probabilidade de Juros para cada item da DT. Esta abordagem será adaptada na pesquisa-ação deste estudo.

Nesta abordagem, segundo Seaman et al. (2012), é sugerida a utilização de uma matriz de custo versus impacto, onde o eixo X representa o custo de pagar a DT no momento atual e o eixo Y representa o impacto que o item da DT está causando, ou pode vir a causar, caso seu pagamento seja postergado. Os pontos IDT1, IDT2, até IDT9 da

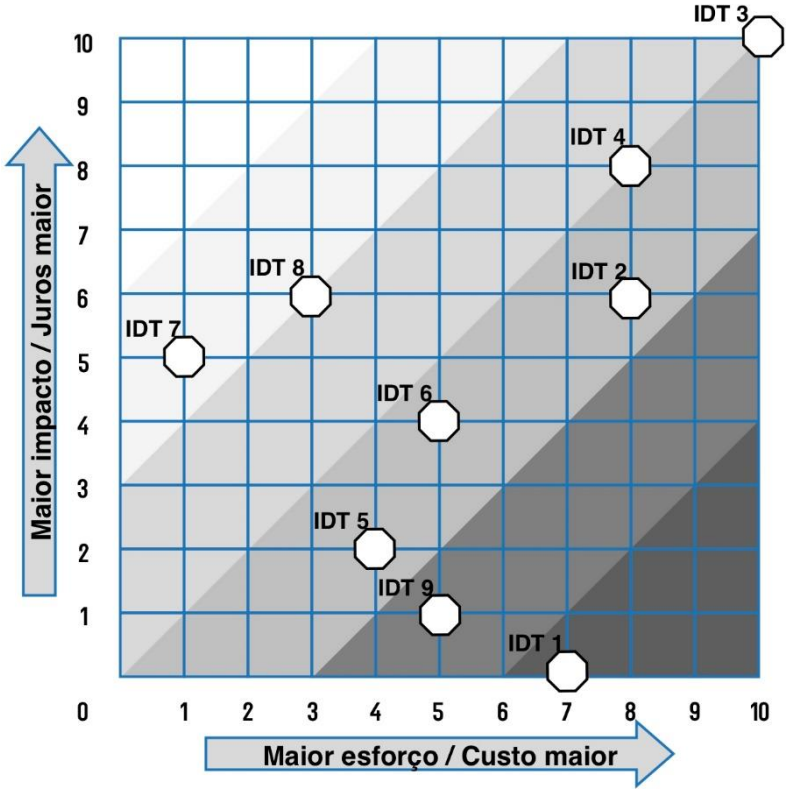

Figura representam os custos (eixo X) e impactos negativos (eixo Y) de um conjunto de itens de DT.

Segundo os autores, uma estratégia simples de priorização é primeiramente priorizar os itens começando com o canto superior esquerdo do gráfico, área branca, e, em seguida, movendo-se para baixo e para a direita no sentido das áreas em tonalidades de cinza. Esta classificação prioriza os itens de DT com maior impacto negativo e menor custo de pagamento, conforme mostra figura 2.

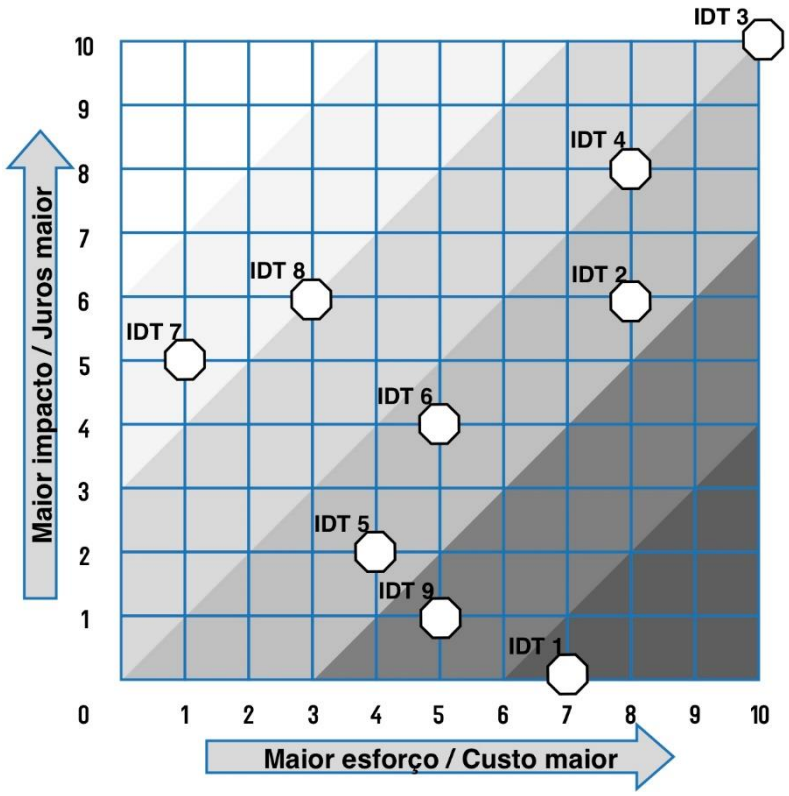

Figura 2 - Matriz de Custo versus Impacto
Fonte: Adaptado de Seaman et al. 2012

Segundo Seaman e Guo (2011), durante o planejamento do ciclo da *sprint*, os itens do *backlog* de DT devem ser revisados e analisados, a fim de verificar se há sinergias de tais itens com os do *Backlog* do produto.

2.2 Métodos Ágeis

Os modelos de desenvolvimento de software ágil surgiram a partir da necessidade de desenvolver aplicativos de software que pudessem acomodar a rápida entrega de resultados ao cliente, e consistem em uma série de ciclos ou iterações curtas de desenvolvimento com um alto grau de colaboração entre os envolvidos.

Cada um destes ciclos representa miniprojetos dentro do projeto original com todas as fases características de um projeto, resultando ao final de cada ciclo a implantação de uma parte da funcionalidade requisitada funcionado. Ao final de todos os ciclos tem se o sistema proposto funcionando por completo.

Em 2001, surgiu o manifesto para o desenvolvimento ágil de software (AGILE MANIFESTO, 2001), que valoriza:

- **Indivíduos e interações** mais que processos e ferramentas;

- **Software em funcionamento** mais que documentação abrangente;

- **Colaboração com o cliente** mais que negociação de contratos;

- **Responder a mudanças** mais que seguir um plano;

Ou seja, mesmo havendo valor nos itens à direita (sem negrito), valoriza-se mais os itens à esquerda (em negrito).

O Manifesto Ágil também apresentou 12 princípios representados na figura 3 descritos a seguir:

1. A maior prioridade é a satisfação do cliente por meio da entrega rápida e contínua de software que traga valor;

2. Mudanças nos requisitos são aceitas, mesmo em estágios avançados de desenvolvimento. Processos ágeis aceitam mudanças que trarão vantagem competitiva ao cliente;

3. Software que funciona é entregue frequentemente, em períodos que variam de semanas a meses, quanto menor o tempo entre uma entrega e outra melhor;

4. As pessoas relacionadas ao negócio e os desenvolvedores devem trabalhar juntos no dia a dia do projeto;

5. Construa projetos formados por indivíduos motivados, fornecendo o ambiente e o suporte necessário e confiando que realizarão o trabalho;

6. O modo mais eficiente e eficaz de transmitir informações dentro e fora do time de desenvolvimento é a comunicação face a face;

7. A principal medida de progresso é o software funcionando;

8. Processos ágeis promovem o desenvolvimento em um ritmo sustentável. Os investidores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante;

9. Cuidar continuamente da excelência técnica e do bom design ajuda a aprimorar a agilidade;

10. Simplicidade é essencial;

11. Os melhores requisitos, arquiteturas e design surgem de equipes auto gerenciadas e

12. Em intervalos regulares, o time reflete sobre como se tornar mais eficiente, refinando e ajustando seu comportamento apropriadamente.

Figura 3 - Os 12 Princípios Ágeis
Fonte: Agileschool.com.br

Kruchten et al. (2012) relatam que algumas equipes de desenvolvimento ágil acreditam estar completamente imunes à DT pelo fato de usarem um processo de desenvolvimento iterativo. Mas, segundo os autores, com as entregas muito rápidas, o pouco

tempo para desenvolver o projeto ou a falta de rigor em testes automatizados, levam rapidamente alguns softwares a um nível considerável de DT.

2.3 Práticas Ágeis

Existem diversos *frameworks* de gerenciamento ágil de desenvolvimento de projetos de *software*. Segundo Azizyan et al. (2011) os mais utilizados são o *Scrum* e o *eXtreme Programming* (XP).

Porém, segundo previsões do Gartner divulgada em 2018, até 2022, metade dos *frameworks* ágeis adotados atualmente nas empresas terão sido abandonados, recomendando assim que as empresas adaptem os *frameworks* às características e restrições organizacionais, respeitando os princípios ágeis.

Desta forma, nesta seção serão descritas as práticas ágeis utilizadas neste estudo, sem se prender a um *framework* especifico.

2.3.1 Planning Poker

O *planning poker* é uma técnica de estimativa ágil baseada em consenso onde se utiliza cartões de *planning poker* que normalmente são constituídos pela sequência 1, 3, 5, 8, 13, 21, e assim por diante. O *planning poke*r envolve uma comparação relativa, em vez de atribuir um valor absoluto ao item avaliado.

Segundo Goldstein (2013), a técnica chamada *Planning Poker* foi criada por James W. Grenning, um dos signatários do manifesto ágil, publicada em abril de 2002 no artigo "*Planning Poker or How to avoid analysis paralysis while release planning*", e popularizada em Cohn (2005).

Segundo Grenning (2002), a técnica do *planning poker* envolve uma mecânica simples de ser utilizada que traz os seguintes benefícios: diminuição no tempo gasto para elaboração de estimativas e envolvimento de todo time do projeto.

Cohn (2005) sugere que antes do início do jogo, o time selecione de forma unânime um item do backlog, cuja estimativa seja igual a um *story point*. Em seguida, seleciona outro item, cuja estimativa seja cinco *stories points*, também de forma unânime. Esta atividade é utilizada para encontrar uma base para ser usada na comparação

Em seguida o *Product Owner* (PO) lê uma história. Há uma discussão esclarecendo a história conforme necessário. Cada desenvolvedor anota sua estimativa em um cartão de nota sem discutir sua estimativa. Depois que todos os programadores tiverem escrito sua estimativa, viram todas os cartões ao mesmo tempo. Se houver concordância, nenhuma discussão será necessária, registra-se a estimativa e passam para a próxima história.

Se os valores das cartas não forem iguais, os jogadores com valores nos extremos justificam suas escolhas, as cartas serão escolhidas e reveladas novamente, até chegar ao consenso.

Passo a Passo do Planning Poker:

- Cada integrante do time de desenvolvimento pode utilizar cartões de papel em branco para anotar as estimativas conforme necessário, ou usar um baralho de *Planning Poker*;
- As estimativas são baseadas em uma séria de Fibonacci modificada, que normalmente é constituída de 1, 3, 5, 8, 13, 21 e assim por diante;

- O representante da área usuária ou *Product Owner* (PO) lê e apresenta a história de usuário, ou o requisito de usuário;
- O time de desenvolvimento discute a história de usuário com o objetivo de entender o requisito;
- Cada integrante do time escolhe uma estimativa, colocando a carta virada para baixo, escondendo o número da estimativa. Em seguida, a cada história de usuário analisada, cada membro da equipe joga uma carta com a face para baixo sobre a mesa, nela estará contido o valor numérico de pontos que o mesmo considera justo para que a história seja concluída;
- Quando todos integrantes do time colocarem suas estimativas na mesa, viram-se as cartas apresentando as estimativas no mesmo instante;
- Caso a carta de todos os participantes tenha o mesmo valor, a estimativa está pronta. Vá para a próxima história de usuário, se houver. Se não houver, a sessão de *Planning Poker* finaliza e
- Caso não sejam iguais, os membros que jogaram as cartas de maior e menor valor explicarão seus pontos de vista e justificativas para seleção. Após a discussão, o processo de *poker* se repete, com cada integrante do time jogando uma carta para o valor de estimativa. Este passo se repete até que um consenso seja encontrado e uma estimativa seja definida.

A figura 4 resume o passo a passo da técnica.

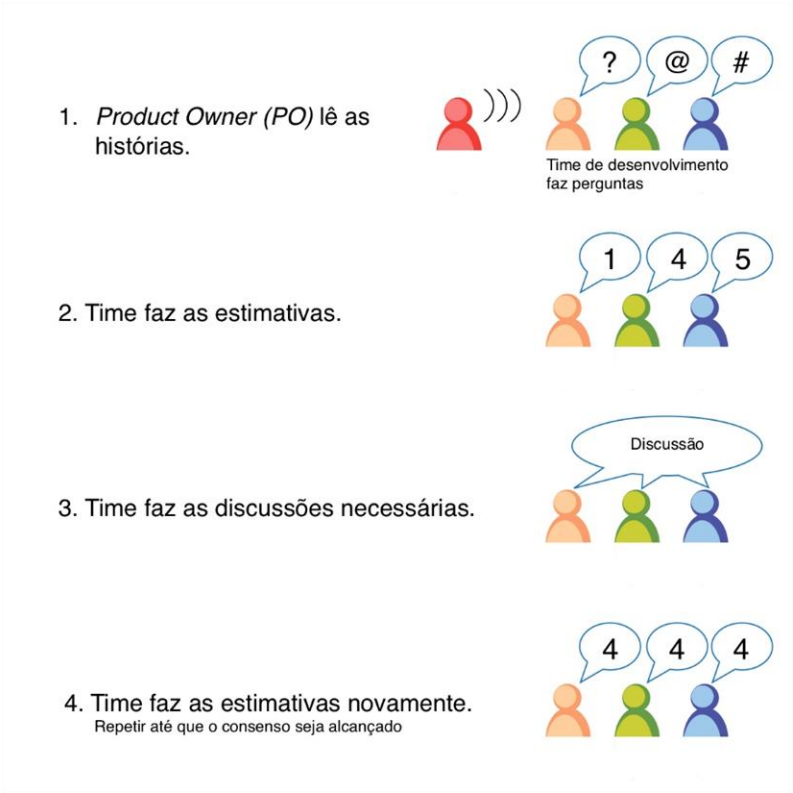

Figura 4 - Passo a Passo Resumido do *Planning Poker*
Fonte: adaptado de blog.acelerato.com

Na opinião de Cohn (2005), *Planning Poker* é a melhor maneira de se fazer estimativas no contexto ágil. A técnica combina opiniões de especialistas, comparações por analogias e negociações até que o consenso seja atingido em uma abordagem lúdica de estimar que resulta em estimativas rápidas, e ao mesmo tempo confiáveis.

2.3.2 Planejamento da Release e da Iteração

Segundo Rubin (2012), o planejamento da release faz parte das cerimonias ágeis, cujo principal objetivo é elaborar um plano em alto nível contendo as *features* que farão parte das iterações da release a ser planejada. O *Backlog* do Produto (do inglês *Product Backlog*) é a lista ordenada de *features* por prioridade contendo todos os itens que devem ser desenvolvidos na release que acabou de ser planejada.

Para o autor, o dono do produto (PO) (do inglês *Product Owner*) representa o cliente e é o responsável pelo conteúdo, assim como pela priorização do *Backlog* do produto.

Uma release é composta por um conjunto de iterações com tempo definido entre duas a quatro semanas, cada uma, durante o qual uma versão incremental potencialmente utilizável do produto é desenvolvida pelo time de projeto. No framework *Scrum*, as iterações são chamadas de *Sprints*.

Segundo Rubin (2012), o planejamento da *Sprint* também faz parte das cerimonias ágeis. Seu principal objetivo é determinar quais itens do *Backlog* do produto farão parte da *Sprint*. Estes itens são transformados em histórias de usuário que contemplem pelo menos uma entrega de valor dentro da *Sprint*. O conjunto de histórias de usuário que serão entregues na iteração é denominado *Backlog* da *Sprint*.

2.4 Requisitos Funcionais e Não Funcionais

No guia BABOK IIBA (2015), requisito é descrito como uma representação documentada de uma condição ou capacidade necessária para resolver um problema ou atingir um objetivo. Os requisitos não funcionais (RNFs) são classificados em categorias que descrevem o grau em que uma solução sistêmica, ou parte dessa solução, chamadas de funções, podem ser medidas e consideradas unidades de medição de qualidade ou restrição imposta aos requisitos funcionais (RFs).

O quadro 2 descreve as categorias segundo BABOK GUIDE 3.0 (2015)

Categorias	Descrição
Disponibilidade	Grau em que a solução é operável e acessível quando necessária para uso.
Compatibilidade	Grau em que a solução opera efetivamente com outros componentes em seu ambiente.
Funcionalidade	Grau em que as funções da solução atendem às necessidades do usuário, incluindo aspectos de adequação, precisão e interoperabilidade.
Manutenção	Grau de facilidade com a qual uma solução ou componente pode ser modificado para corrigir falhas, melhorar o desempenho ou outros atributos, ou adaptar-se a um ambiente alterado.
Eficiência de desempenho	Grau em que uma solução ou componente executa suas funções designadas com consumo mínimo de recursos.
Portabilidade	Grau de facilidade com que uma solução ou componente pode ser transferido de um ambiente para outro.
Confiabilidade	Capacidade da solução ou componente em executar suas funções sob condições estabelecidas por um período de tempo especificado.
Escalabilidade	Grau com o qual uma solução é capaz de crescer ou evoluir para lidar com quantidades maiores de trabalho.
Segurança	Aspectos que protegem uma solução ou componente de acesso acidental ou mal-intencionado.
Usabilidade	Facilidade com que um usuário pode aprender a usar a solução.
Certificação	Restrições sobre a solução que são necessárias para atender a certas normas ou convenções da indústria.
Conformidade	Restrições regulamentares, financeiras ou legais que podem variar com base no contexto ou jurisdição.
Localização	Requisitos que lidam com idiomas locais, leis, moedas, culturas, grafias e outras características dos usuários, ou que requer atenção ao contexto.
Acordos de Nível de Serviço	Restrições da organização sendo atendida pela solução que é formalmente acordada tanto pelo provedor quanto pelo usuário da solução.
Extensibilidade	A capacidade de uma solução incorporar novas funcionalidades.

Quadro 2 – Categorias de Requisitos Não Funcionais
Fonte: Adaptado de BABOK GUIDE 3.0

De acordo com Sachdeva (2018), existem várias técnicas de classificação de requisitos na literatura, a mais amplamente utilizada divide os requisitos em duas categorias: requisitos funcionais (RFs) e requisitos não funcionais (RNFs).

Para Cysneiros e Leite (1997), RNFs, ao contrário dos RFs, não expressam nenhuma função a ser realizada pelo software, e sim comportamentos e restrições que este software deve satisfazer.

Uma das definições mais difundidas atualmente para RNFs está na norma ISO/IEC 9126-1:2001, atualizada e substituída pela ISO/IEC 25010:2011. Esta norma é um conjunto de normas técnicas que estabelece padrões para qualidade, e que inclui gerência, modelo, medição, requisitos e avaliação de qualidade para qualquer produto de software.

A dimensão qualidade do produto inclui atributos de qualidade, ou requisitos não funcionais do software decorrentes de seu processo de implementação, revisão e testes (qualidade interna) e atributos do software associados à sua execução (qualidade externa). A dimensão do modelo de qualidade do produto da ISO/IEC 25010: 2011 especializa-se em um conjunto de características e subcaracterísticas de qualidade, descritas no quadro 3 e quadro 4.

Caracteristica	Descrição da Caracteristica
Suportabilidade Funcional	Capacidade do produto de software em prover funções para atender a necessidades explícitas e implícitas para as quais foi concebido.
Eficiencia no Desempenho	Capacidade do produto de software de manter um nível de desempenho apropriado, quando usado em condições especificadas.
Compatibilidade	Capacidade do produto de software possibilitar a troca de informações com outras aplicações e/ou compartilhar o mesmo ambiente de hardware ou software.
Usabilidade	Capacidade do produto de software, uma vez possuindo efetividade e eficiência, de ser compreendido, aprendido, operado e atraente ao usuário, quando usado sob condições especificadas.
Confiabilidade	Capacidade do produto de software de executar suas funções de modo contínuo.
Segurança	Capacidade do produto de software de proteger informações e dados: pessoas ou sistemas não autorizados não podem lê-los nem modificá-los e o acesso às pessoas ou sistemas não autorizados é negado.
Manutenibilidade	Capacidade do produto de software de ser modificado. As modificações podem incluir correções, melhorias ou adaptações do software devido a mudanças no ambiente e em seus requisitos ou especificações funcionais.
Portabilidade	Capacidade do produto de software de ser transferido de um ambiente para outro.

Quadro 3 – Características de Qualidade do Produto ISO/IEC 25010
Fonte: Adaptado de Morais e Costa (2014)

Caracteristica	Subcaracterística	Descrição da Subcaracterística
Suportabilidade funcional	Completude	Capacidade do produto de software de prover um conjunto apropriado de funções para tarefas e objetivos do usuário especificados.
	Corretude	Capacidade do produto de software de prover, com o grau de precisão necessário, resultados ou efeitos corretos ou conforme acordados.
	Adequação	Capacidade do produto de software em facilitar a realização das tarefas e objetivos do usuário.
Eficiencia no Desempenho	Comportamento em Relação ao Tempo	Capacidade do produto de software de fornecer tempos de resposta e de processamento apropriados, quando o software executa suas funções, sob condições estabelecidas.
	Utilização de Recursos	Capacidade do produto de software de usar tipos e quantidades apropriados de recursos, quando executa suas funções sob condições estabelecidas.
	Capacidade	Limites máximos de parâmetros do sistema (itens que podem ser armazenados, número de usuários concorrentes, largura de banda, velocidade de transações, tamanho da base de dados etc.) que atendem aos seus requisitos.
Compatibilidade	Coexistência	Capacidade do produto de software de coexistir com outros produtos de software independentes, em um ambiente comum, compartilhando recursos comuns.
	Interopetabilidade	Capacidade do produto de software de interagir com um ou mais sistemas especificados, pela troca de informações e do uso de informações que são trocadas.
Usabilidade	Inteligibilidade	Capacidade do produto de software de possibilitar ao usuário compreender se o software é apropriado e como ele pode ser usado para tarefas e condições de uso específicas. Depende da documentação do software.
	Aprensibilidade	Capacidade do produto de software de possibilitar ao usuário aprender seu uso. Depende da documentação do software.
	Operabilidade	Capacidade do produto de software de possibilitar facilidade ao usuário para operá-lo e controlá-lo.
	Proteção ao Erro do Usuário	Capacidade do produto de software em proteger o usuário de erros.
	Estética da interface com o usuário	Capacidade do produto de software de ser atraente ao usuário, ao oferecer uma interface com interação agradável.
	Acessibilidade	Capacidade do produto de software ser utilizado por um amplo espectro de pessoas, que inclui portadores de necessidades especiais e com limitações associadas à idade.
Confiabilidade	Maturidade	Capacidade do produto de software de evitar falhas decorrentes de defeitos no software, mantendo sua operação normal.
	Disponibilidade	Capacidade do produto de software em ser operacional e acessível quando seu uso for requerido.
	Tolerância a Falhas	Capacidade do produto de software de operar em um nível de desempenho especificado em casos de defeitos no software ou no hardware.
	Recuperabilidade	Capacidade de o produto de software restabelecer seu nível de desempenho especificado e recuperar os dados diretamente afetados no caso de uma falha.
Segurança	Confidencialidade	Capacidade do produto de software de garantir que os dados serão acessíveis apenas por pessoas que possuem acesso a eles.
	Integridade	Capacidade do produto de software de evitar o acesso não autorizado para acesso ou modificação de programas ou dados
	Não Questionamento	Capacidade do produto de software em garantir que a ocorrência de ações ou eventos possam ser provados, evitando-se questionamentos futuros.
	Responsabilização	Capacidade do sistema em auditar a rastreabilidade de acesso a operações.
	Autenticação	Capacidade do sistema em validar a identidade de um usuário.
Manutenibilidade	Modularidade	Capacidade de o sistema possuir componentes discretos de modo que uma modificação em um componente tenha impacto mínimo em outros componentes.
	Reusabilidade	Capacidade de os componentes do software serem utilizados em outro software ou na construção de outros componentes ou sistemas.
	Analisibilidade	Capacidade do produto de software de permitir o diagnóstico de deficiências ou causas de falhas, ou a identificação de partes a serem modificadas.
	Modificabilidade	Capacidade do produto de software de permitir que uma modificação especificada seja implementada.
	Testabilidade	Capacidade do produto de software de permitir que o mesmo, quando modificado, seja validado.
Portabilidade	Adaptabilidade	Capacidade do produto de software de ser adaptado para diferentes ambientes especificados, sem necessidade de aplicação de outras ações ou meios além daqueles fornecidos para essa finalidade pelo software considerado.
	Capacidade de Ser Instalado	Capacidade do produto de software para ser instalado em um ambiente especificado.
	Capacidade para Substituir	Capacidade do produto de software de ser usado em substituição a outro produto de software especificado, com o mesmo propósito e no mesmo ambiente.

Quadro 4 – Características de Qualidade do Produto ISO/IEC 25010
Fonte: Adaptado de Morais e Costa (2014)

3. MÉTODO

Com objetivo de construir uma visão ampla sobre o tema, incluindo resultados relevantes e com um caráter imparcial em relação às referências encontradas, optou-se por realizar uma revisão sistemática da literatura para identificar e analisar os métodos, processos e técnicas existentes na literatura, que identifiquem, medem e/ou gerenciem dívida técnica em contexto de métodos ágeis.

De acordo com Nakagawa, et al. (2017), a pesquisa exploratória é considerada uma pesquisa não-sistemática, não possui controle organizado (protocolo) que proporcione a possibilidade de conferência ou reprodução posterior. É realizada conforme a experiência do pesquisador e com base em termos ou expressões que compõem o problema ou a questão de pesquisa. Cria oportunidade para definir a abrangência e delimitar a amplitude da revisão sistemática, assim como para identificar as palavras chaves que serão utilizadas.

Já a revisão sistemática, segundo Biolchini et al. (2005) e Kitchenham (2004), é apoiada numa metodologia controlada, por meio de protocolo de pesquisa, com o objetivo de identificar, analisar e interpretar evidências literárias disponíveis sobre um tópico específico, questão de pesquisa ou fenômeno de interesse. De acordo com Nakagawa, et al. (2017), a possibilidade de conferir, reproduzir e auditar a revisão sistemática, por meio de seu protocolo, é a principal diferença entre um pesquisa sistematizada e a não-sistemática.

A principal função da revisão sistemática na Engenharia de *Software*, segundo Biolchini et al. (2005) e Kitchenham (2004), é melhorar cientificamente a validade de afirmações que podem ser feitas no campo e, consequentemente, o grau de confiabilidade dos métodos que são empregados para desenvolver tecnologias e suportar processos de software. Estes autores dividem o processo de revisão sistemática em 3 fases principais:

1) Planejamento: Fase composta pelas etapas de identificação da necessidade da revisão, desenvolvimento de uma proposta de revisão e elaboração de um protocolo de pesquisa;
2) Execução: Fase responsável pela identificação das fontes de pesquisa, extração dos dados, seleção dos estudos primários, avaliação da qualidade e relevância dos trabalhos selecionados no contexto de interessa;
3) Síntese: Fase responsável pela seleção final dos trabalhos, análise e síntese dos resultados, elaboração de conclusão e recomendações, e apresentação dos resultados.

As fases de planejamento com o protocolo da revisão e síntese com a análise dos resultados e as considerações finais da revisão sistemática estão presentes nas seções seguintes. A fase de execução se encontra no Apêndice A.

3.1. Planejamento da Revisão Sistemática da Literatura

Esta etapa tem como objetivo encontrar e analisar métodos, processos ou técnicas existentes na literatura que identifiquem, medem, priorizem e gerenciem dívidas técnicas (DTs) de *software* no contexto de métodos ágeis.

A revisão sistemática procura identificar estudos que possam auxiliar na resposta à questão da pesquisa:

- Como as abordagens, métodos ou técnicas ágeis existentes contribuem para tornar as medições e priorização da dívida técnica mais sistematizadas e menos subjetivas?

A questão de pesquisa será decomposta e organizada utilizando a estratégia PICO de Petticrew e Roberts (2005). PICO representa um acrônimo para *Population*, *Intervention*, *Comparison* e *Outcome*.

População: trabalhos que utilizam metodologias ágeis para desenvolvimento de software e que fazem gestão de suas DT.

Intervenção: abordagens, métodos ou técnicas de medição, priorização e gestão de DT em contextos de métodos ágeis.

Controle: Os estudos base de apoio à revisão sistemática tem origem na pesquisa exploratória, realizada por meio de consulta bibliográfica de artigos publicados em conferências, periódicos e dissertações de mestrado, obtidas por consultas em base de dados eletrônica. A lista de referências utilizadas para definição da abrangência do tema, dívida técnica em contextos de métodos ágeis, pesquisado e que serviram de base para escolha das palavras chaves, fontes e períodos de busca foram:

- ✓ Alves (2017);
- ✓ Bomfim Junior (2016);
- ✓ Oliveira (2015);
- ✓ Pires (2014);
- ✓ Seaman e Guo (2011);
- ✓ Seaman et al. (2012).

O quadro 5 apresenta os artigos que serviram de base para a revisão sistemática.

#	Ano	Título	Autores	Base Eletrônica
1	2017	Uma Proposta para Identificar, Medir e Gerenciar a Dívida Técnica em Requisitos de Software http://cassiopea.ipt.br/teses/2017_EC_Mirla_Nunes.pdf	ALVES, Mirla Nunes	Repositório IPT
2	2016	Estratégias para redução de dívidas técnicas em equipes ágeis http://cassiopea.ipt.br/teses/2016_EC_Marcelo_Mazzini.pdf	BOMFIM JUNIOR, Marcelo Mazzini	Repositório IPT
3	2015	Gerenciamento da dívida técnica em projetos de software utilizando Scrum: uma pesquisa-ação http://cassiopea.ipt.br/teses/2015_EC_Frederico_Sousa.pdf	OLIVEIRA, Frederico Sousa	Repositório IPT
4	2014	Um estudo sistemático sobre identificação e gerenciamento de dívida técnica em uma empresa de tecnologia com desenvolvimento baseado em Scrum http://cassiopea.ipt.br/teses/2014_EC_ROGERIO_PIRES.pdf	PIRES, Rogério Chaves	Repositório IPT
5	2011	Measuring and monitoring technical debt. Advances in Computers, v. 82, no 25-46, p. 44, 2011. https://www.sciencedirect.com/science/article/pii/B9780123855121000025	SEAMAN, C.; GUO, Y.	ACM Digital Library
6	2012	Using Technical Debt Data in Decision Making: Potential Decision Approaches	SEAMAN, C. et al.	IEEE Xplore

Quadro 5 - Artigos Base para a Revisão Sistemática
Fonte: Elaborado pelos autores

Resultados: visão abrangente dos estudos propostos na literatura que tratam de forma sistemática a medição e priorização da dívida técnica de *software* no contexto de metodologias ágeis.

Critérios de Seleção das Bases de Busca:

A escolha das fontes de busca foi feita utilizando os seguintes critérios:

a) Bibliografias da análise exploratória;

b) Anais de eventos das áreas;

c) Revisões Sistemáticas anteriores;

d) Bases bibliográficas indexadas na internet;

e) Motores de busca que indexam diversas bases bibliográficas;

f) Fontes híbridas que indexam artigos próprios e artigos de outras bases bibliográficas;

g) Teses e dissertações disponíveis em bases indexadas ou acessíveis via portal web de suas instituições de ensino;

h) A base de busca deve conceder acesso ao texto completo e na íntegra dos artigos.

Lista das Bases de Busca:

Com base nos critérios definidos, foram escolhidas as fontes do quadro 6, por serem reconhecidas mundialmente pela produção literária de alta qualidade e por incluírem as principais revistas literárias e eventos científicos no campo da Engenharia de *Software*:

Fonte de Busca	Endereço Online
ACM *Digital Library*	https://dl.acm.org/
IEEE *Xplore*	https://ieeexplore.ieee.org/xplore/
Scopus *Digital Library*	http://www.scopus.com/

Quadro 6 – Fontes de Busca da Revisão Sistemática
Fonte: Elaborado pelos autores

Idioma dos Trabalhos:

Com base nas fontes escolhidas, o idioma inglês será o utilizado pois trata-se daquele com maior aceitação internacional para os trabalhos científicos da área.

Palavras-chave

Foram estabelecidas as seguintes palavras-chave:

Em língua Portuguesa: dívida técnica; métodos ágeis.

Em língua Inglesa: technical debt; agile.

Recorreu-se aos operadores lógicos “AND” e “OR” para combinação das palavras chave:

Strings de Busca

Em termos gerais foi aplicada a seguinte string de busca, que posteriormente foi adaptada à fonte de busca:

Língua Inglesa: ((agile OR SCRUM OR XP or kanban) AND debt*)

Estratégia de busca para a identificação dos trabalhos:

Busca manual

A busca manual será realizada por meio da aplicação da técnica de *Pearl Growing* (NAKAGAWA et al., 2017), que utiliza a lista de citações e referências dos estudos aceitos após a segunda ou terceira avaliação, com o objetivo de identificar novos estudos.

Buscas automática

A busca automática será realizada por meio da aplicação da *string* de busca e suas variações nas bases de busca selecionadas, no período entre 2015 e 2020.

Critérios de Inclusão e Exclusão dos Trabalhos:

Na revisão sistemática, os critérios de inclusão e exclusão, limitam a seleção dos trabalhos conforme sua relevância para o objetivo da pesquisa. Portanto, com base em avaliações qualitativas foram definidos os seguintes critérios:

Critérios de inclusão:

a) Estudos qualitativos que abordam medição, e/ou priorização e/ou gestão da DT de *software* no contexto de métodos ágeis;

b) Estudos que apresentam alguma proposta, experimento ou aplicação de um método ou técnica sobre medição, e/ou priorização e/ou gestão da DT de *software* no contexto de métodos ágeis.

Critérios de exclusão:

a) Estudos que não estejam relacionados a DT de software;

b) Estudos que não ocorrem no contexto de métodos ágeis;

c) Estudos que não abordam métodos ou técnicas para a medição, ou priorização ou gestão da DT de software;

d) Estudos que apresentam avaliações sem apresentar o método utilizado;

e) Estudo repetido, quando comparado com o resultado das demais buscas;

f) Estudos com texto, conteúdo e/ou resultados incompletos.

g) Trabalhos publicados como artigos curtos, pôsteres ou tutoriais.

Cabe ressaltar que alguns artigos levantados na pesquisa exploratória, foram considerados relevantes para o trabalho, portanto, compõem o processo proposto aplicado na pesquisa-ação desta dissertação, somente não pertencendo à lista de resultados da pesquisa sistemática se não estiverem nas fontes selecionadas.

Estratégia para seleção dos trabalhos:

Todos os trabalhos retornados pelas buscas serão devidamente documentados em tabelas específicas para cada fonte de busca. Após essa documentação, será realizada a primeira avaliação destes trabalhos por meio da leitura de seus títulos, resumos e palavras-

chave, e considerando os critérios de inclusão e exclusão definidos no protocolo. Na sequência, e também considerando os mesmos critérios, será realizada uma segunda avaliação por meio da leitura da introdução e conclusão dos estudos que passaram pela primeira avaliação. Por fim, e também considerando os mesmos critérios, será realizada uma terceira avaliação por meio da leitura completa dos estudos que passaram pela segunda avaliação.

O quadro para documentar as informações, também registrará quais critérios de inclusão e/ou exclusão serão indicados para cada um dos trabalhos avaliados, sejam eles incluídos ou não na síntese geral.

4. RESULTADOS

4.1 Resumo da Condução da Revisão Sistemática

A revisão sistemática foi conduzida durante os meses de março a maio de 2020. No total foram encontrados e avaliados 145 trabalhos. A primeira avaliação destes estudos foi realizada por meio da leitura de seus títulos e resumos e então foram aplicados os critérios de inclusão e exclusão, resultando em 18 trabalhos considerados elegíveis para a segunda avaliação.

Na sequência, e também considerando os mesmos critérios, foi realizada uma segunda avaliação acrescentando a leitura da introdução e conclusão dos 18 estudos que passaram pela primeira avaliação, resultando em 7 trabalhos. Estes tiveram a leitura completa e analisada, onde foram selecionados 4 artigos que assim pudessem compor a síntese da pesquisa.

Além destes 4 trabalhos escolhidos, foram acrescentados 3 trabalhos encontrados e selecionados por meio da técnica manual *Pearl Growing*, sendo que um deles fazia parte da seleção das bases, porém havia sido rejeitado na primeira avaliação, e acabou retornando após segunda avaliação por meio da sua referência numa das citações.

Segundo Nakagawa et. al (2017), a técnica denominada *Pearl Growing* consiste em investigar as referências utilizadas pelos artigos mais relevantes e dos principais autores da área. Desta forma, alguns estudos que faziam parte da lista de referências de trabalhos selecionados como relevantes das fontes anteriores foram incluídos nesta etapa da revisão sistemática.

A figura 5 representa o resumo da revisão sistemática descrita anteriormente. São apontadas as bases de busca com a amplitude da busca, período e quantidade de artigos encontrados.

Figura 5 – Trabalhos Selecionados na Síntese da Revisão Sistemática
Fonte: Elaborado pelos autores

A partir da revisão sistemática realizada, pode-se afirmar que a etapa de estimativa e priorização da DT ainda não é um tópico de pesquisa maduro, apresentando notória escassez literária relacionada a trabalhos empíricos. O que é corroborado pelos autores Seaman et al. (2012) e Avgeriou et al. (2016), quando sugerem que sejam incorporados outros critérios relevantes e objetivos no processo decisório para diminuir custos nas etapas de identificação e estimativa da DT.

4.2 Análise dos Resultados

Com os resultados obtidos, nesta seção foi redigida uma síntese geral que descreve sinteticamente as análises críticas elaboradas pelos autores. Análises qualitativas com relação aos trabalhos pesquisados e algumas considerações sobre os resultados observados nos trabalhos selecionados.

O trabalho realizado por Griffith et al. (2014) descreve um conjunto de simulações baseadas em modelos de processo de desenvolvimento ágil, Scrum, e a integração do gerenciamento da DT. O objetivo deste estudo foi identificar qual estratégia é superior e fornecer evidências empíricas para apoiar as alegações existentes.

Os quatro modelos e estratégias de gestão da DT comparados foram:

1) Base: não considera o gerenciamento da DT;

2) Lista de DTs: mantém uma lista separada de dívida técnica e separa parte da *sprint* para redução de DT;

3) Lista de DTs com gerenciamento da DT ativo: mantém uma lista separada de dívida técnica e separa uma das *sprints* para redução da DT e

4) Limiar da DT: utiliza duas abordagens baseadas em limites: Limite superior: inicia a remoção da DT quando o nível atual atinge um limite superior; Limiar superior e inferior: utiliza um limite superior e um limite inferior para iniciar e interromper a fase de remoção da DT.

O trabalho conclui que os modelos 2 e 3 são mais eficientes, porém o 3 pode não ser factível, pois na simulação aplicada não foi levado em conta a moral do time e o fator *time to market.*

Os resultados das simulações fornecem evidências convincentes para as atuais estratégias técnicas de gerenciamento de dívida propostas na literatura que podem ser aplicadas imediatamente pelos profissionais.

No trabalho de Guo et al.(2016), cujo estudo de caso constatou que as etapas com maior custo no gerenciamento da DT são as de identificação da DT e de estimativa da DT, propõe as seguintes formas para diminuir e/ou otimizar estes custos:

1) Limitar o foco dos esforços iniciais de gerenciamento de DT: selecionar itens de DT que estão causando algum tipo de "dor". Isso justificaria o custo inicial do desenvolvimento de uma lista de itens de DT, que realmente estão causando algum tipo impacto. Desta forma, o pagamento destes itens trarão sucessos iniciais por estarem resolvendo problemas reais da organização;
2) Integrar o gerenciamento de DT ao processo de gerenciamento de projetos existente, tanto quanto possível, por exemplo incorporando itens de DT no *backlog* da *Sprint* e
3) Comece com noções simples de principal e juros que se concentram no esforço e incorpore gradualmente outros critérios de decisão usando-os para ajustar valores de principal e juros.

Os autores sugerem que sejam incorporados outros critérios relevantes e objetivos no processo de medição e decisório a fim de diminuir custos nas etapas e estimativa e priorização da DT, e na tomada de decisão.

No trabalho de Martini et al.(2017), por meio de um estudo de caso múltiplo em quatro projetos, foi constatado que a função de mapear o juros de DT para um projeto em geral não é a soma dos juros dos itens da DT no nível do problema.

Os autores identificaram que os engenheiros e o gerente de produto estimam o impacto da TD no projeto usando as informações fornecidas pelos itens individuais, mas não como uma soma. Em vez disso, intuitivamente calculam uma espécie de média dos itens, e os pesam de acordo com outros fatores, como por exemplo a parte do código afetada pela DT, tamanho e *roadmap* do projeto, o impacto positivo de adquirir a DT, a existência de uma alternativa paliativa e os fatores culturais como atitude da equipe em relação a contrair a DT.

Este estudo representou um primeiro passo para avaliar o juros da DT, levando em consideração informações qualitativas de seu impacto.

Martini e Bosch (2017) visando a analisar os efeitos negativos e econômicos do juros relacionados aos itens de dívida técnica, criaram uma abordagem de priorização das dívidas a partir da estimativa de sete fatores de gravidade. Para isso propões o uso de uma ferramenta chamada AnaConDebt (Analysis of Contagious Debt), desenhada por eles.

A avaliação sistemática do juros da DT se dá pela comunicação entre desenvolvedores, arquitetos e gerentes, pela análise dos sete fatores, e os cálculos com apoio da AnaConDebt, seguindo algumas etapas, são elas:

1) Na primeira etapa, os desenvolvedores de forma intuitiva repassam à ferramenta uma avaliação quantitativa entre 1 a 10, em relação ao impacto negativo de cada dívida técnica;
2) A segunda etapa, consiste na avaliação dos sete fatores relacionados ao juros das dívidas, ou seja, os desenvolvedores informavam à ferramenta um valor de 0 (sem gravidade) a 5 (alta gravidade) a cada fator.
3) Em seguida, a ferramenta retornava um valor de 1 a 10, representando a gravidade do item de dívida técnica analisado.
4) E como última etapa, as pessoas interessadas podem fazer uma avaliação entre os valores obtidos pela ferramenta e a análise intuitiva dos desenvolvedores, permitindo a priorização de itens considerados mais graves em relação aos aspectos desejados.

Os resultados mostraram que o resultado da ferramenta está muito próximo da estimativa dada pelas partes interessadas. As implicações são que, se dados adicionais corroborarem a hipótese, a gravidade dos juros poderá ser sistematicamente avaliada pelas partes interessadas, estimando apenas sete fatores de maneira econômica com resultados aceitáveis.

No trabalho de pesquisa-ação de Oliveira et al. (2015), os times envolvidos reconheceram que a abordagem proposta por Seaman e Guo (2011) é viável para ser considerada no processo de desenvolvimento de software, mas sugerem que sejam feitas algumas modificações para diminuir o tempo e complexidade de uso do modelo, especialmente em relação a variável probabilidade de juros.

Desta forma, propõem a redução de três para duas métricas: principal e a quantidade atual de juros. Em consequência, a tomada de decisão foi beneficiada pela consideração antecipada das dívidas que realmente já tem juros conhecidos. Em vez de usar probabilidades para calcular os juros, as métricas são registradas toda vez que ocorre uma dívida técnica.

Em Oliveira et al. (2015), propõe a utilização de um Backlog separado com itens da DT e sugere a incorporação de itens de DT no Backlog da Sprint.

Em Oliveira (2015), o autor sugere como trabalho futuro a utilização da técnica *planning poker* para estimar o principal, ou seja, o custo de pagar a DT no momento atual.

Em Osses et al. (2017), os autores propõe uma adaptação da técnica de *planning poker* para seleção de táticas de segurança relacionadas a RNFs.

Segundo Sachdeva (2018), as metodologias ágeis estão ganhando popularidade em ritmo acelerado e proporcionaram à indústria de software uma maneira de entregar produtos de forma incremental em um ritmo acelerado. Neste contexto, a priorização dos itens do *backlog* da *sprint* se torna vital para o sucesso do produto e, portanto, da organização.

Para o autor, os valores e princípios ágeis falam sobre o desenvolvimento de software e maneiras de escrever código de boa qualidade e, assim, minimizar a introdução de qualquer nova dívida técnica. No entanto, nenhuma solução é descrita sobre como lidar com a dívida técnica presente nos sistemas existentes.

Sachdeva (2018) apresenta uma abordagem para estimar a prioridade de *business features* utilizando a técnica ágil *planning poker*, e fornece uma solução multifásica para criar um *backlog* de produtos buscando o retorno sobro o investimento (ROI).

O autor também propõe um método para lidar e priorizar a DT, onde a técnica de *planning poker* também é utilizada na estimativa do custo da DT, o impacto de requisitos não funcionais envolvidos na DT são levados em conta em sua priorização num backlog separado com itens da dívida técnica. Por fim, o backlog da sprint é composto de histórias de usuários, do *backlog* do produto, e itens de DT, do *backlog* de DT, priorizados em ordem decrescente do ROI.

4.3 Síntese da Revisão Sistemática

Após a leitura minuciosa e completa dos trabalhos incluídos após a terceira avaliação, foi elaborado um resumo comparativo entre eles, destacando os métodos ou técnicas utilizadas, mostrando como os mesmos se inter-relacionam e também como respondem, ou ajudam a responder, a pergunta de pesquisa, conforme mostra o quadro 7.

#	Ano	Título	Autor (es)	Fonte de Busca	Abordagens Recomendadas
1	2015	A simulation study of practical methods for technical debt management in agile software development	Griffith, I. and Taffahi, H. and Izurieta, C. and Claudio, D.	Scopus Digital Library	Uso de uma lista separada de DT e Incorporação de itens de DT no Backlog da Sprint.
2	2016	Costs and obstacles encountered in technical debt management – A case study	Guo, Yuepu and Seaman, Carolyn and Silva, Fabio Q. B.	Citação: Science Direct	1) Seleção de itens de DT que causm impacto; 2) Utilização de um Backlog separado com itens da DT; 3) Incorporação de itens de DT no *Backlog* da *Sprint* e 4) Incorporar outros critérios relevantes e objetivos no processo de medição e decisório a fim de diminuir custos nas etapas e estimativa e tomada de decisão.
3	2017	Technical Debt Interest Assessment: from Issues to Project	Martini, A., Vajda, S., Vasa, R., Jones, A., Abdelrazek, M., Grundy, J., & Bosch, J.	Citação: ACM Digital Library	Avaliação dos juros da DT levando em consideração informações qualitativas de seu impacto.
4	2017	The Magnificent Seven: Towards a Systematic Estimation of Technical Debt Interest	Martini, Antonio and Bosch, Jan	ACM Digital Library e Scopus Digital Library	Priorização dos itens de DT a partir da estimativa de sete fatores de gravidade.
5	2015	Managing Technical Debt in Software Projects Using Scrum: An Action Research	Oliveira, Frederico and Goldman, Alfredo and Santos, Viviane	IEEE Xplore e Scopus Digital Library	1) Utilização de um *Backlog* separado com itens da DT e 2) Incorporar itens de DT no *Backlog* da *Sprint*.
6	2017	Towards the Selection of Security Tactics based on Non-Functional Requirements: Security Tactic Planning Poker	Osses, Felipe and Márquez, Gastón and Orellana, Cristian and Astudillo, Hernán	Citação: IEEE Xplore	Adaptação da técnica de *planning poker* para seleção de táticas de segurança relacionadas a RNFs.
7	2018	Requirements Prioritization in Agile: Use of Planning Poker for Maximizing Return on Investment	Sachdeva, Vaibhav	Scopus Digital Library	1) Utiliza o *planning poker* na estimativa do custo da DT; 2) Considera o impacto de requisitos não funcionais envolvidos na DT; 3) Utilização de um *Backlog* separado com itens da DT e 4) Incorpora itens de DT no *Backlog* da *Sprint*

Quadro 7 – Trabalhos Selecionados na Síntese da Revisão Sistemática
Fonte: Elaborado pelos autores

Com base nas abordagens mencionas na síntese da revisão sistemática, é possível propor adaptações ao método de gerenciamento da DT de Seaman e Guo (2011):

1) Uso de *backlog* separado para itens de DT, conforme proposto em Griffit et al (2015);

2) Utilização da técnica de *planning poker* em estimativas, conforme proposto em Oliveira (2015), Sachdeva (2018) e Osses et al. (2017);

3) Aplicação de um questionário para avaliação da gravidade do impacto dos itens da DT, segundo proposto em Martini e Bosch (2017) e Martini et al. (2017)

4) Avaliação do relacionamento de requisitos não funcionais na estimativa da DT, conforme proposto em Sachdeva (2018) e Osses (2017).

5) Incorporação de itens de DT no *Backlog* da iteração ou *Sprint*, segundo proposto em Griffit et al (2015), Guo, Seaman e Silva (2016), Oliveira, Goldman e Santos (2015) e Sachdeva (2018).

A figura 6 representa o processo baseado nas adaptações ao framework de gerenciamento de dívida técnica proposto por Semana e Guo (2011).

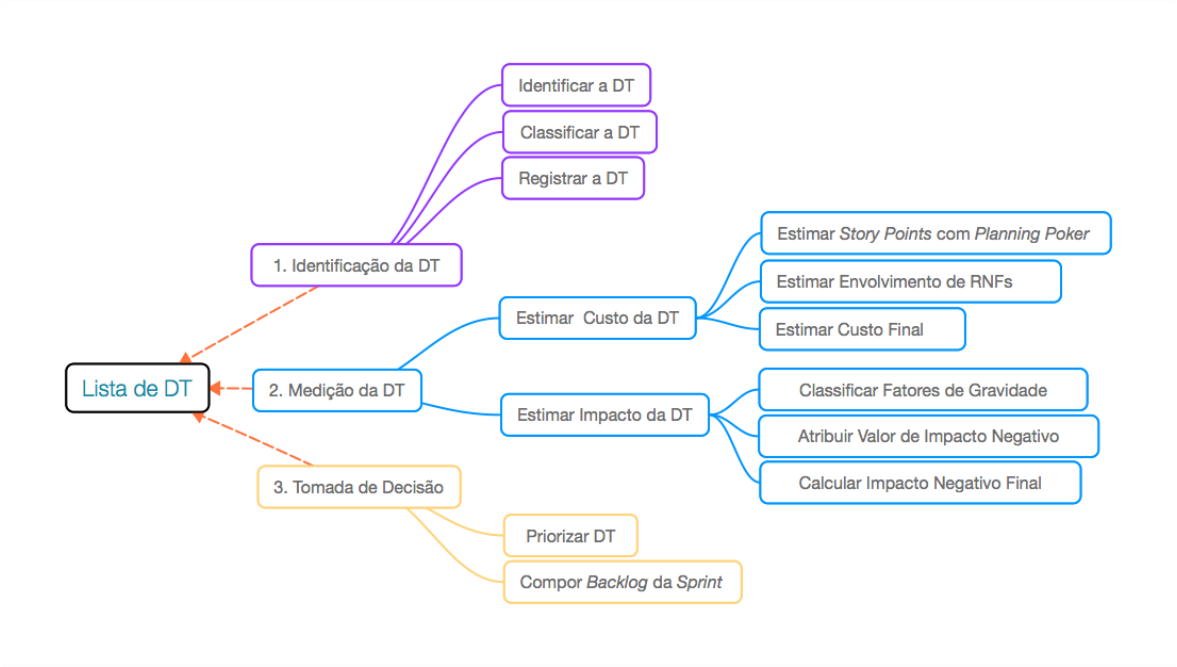

Figura 6 – Framework de Gerenciamento de Dívida Técnica
Fonte: Elaborado pelos autores

5. CONCLUSÕES

No início deste trabalho foram apresentados os conceitos e teorias dos principais temas que permeiam a questão de pesquisa e que foram utilizados como base para a estruturação da revisão sistemática da literatura conduzida nesta pesquisa.

Em seguida, foram apresentados os principais conceitos, objetivos e importância da revisão sistemática da literatura em um projeto de pesquisa. Após conceituada a revisão sistemática, foram apresentadas a etapa de planejamento com o protocolo da pesquisa, a síntese final com os artigos encontrados e selecionados bem como uma análise geral dos resultados obtidos na execução.

A revisão sistemática evidenciou o esforço da comunidade científica em propor soluções e alternativas para auxiliar na gestão da dívida técnica, especialmente nas etapas de estimativa e priorização da dívida técnica. Entretanto, ainda há um longo caminho a ser percorrido para deixar estas etapas mais sistemáticas e fáceis de serem executadas.

A partir dos elementos obtidos na revisão sistemática da literatura deste trabalho, foi obtido uma adaptação ao framework de gerenciamento da dívida técnica de Seaman e Guo (2011), incluindo a utilização de *planning poker* e envolvimento de requisitos não funcionais em sua estimativa de custo, além de medir o impacto negativo da dívida, com uma forma de

valorar a prioridade de itens da dívida técnica a fim de facilitar a tomada de decisão de pagá-la ou não.

Espera-se com estas adaptações trazer mais visibilidade aos itens de dívida técnica e como estes impactam negativamente às organizações e seus clientes, assim como aprimorar o processo de sua estimativa de custo e juros, tornando-o mais sistematizado, menos subjetivo, e consequentemente com menor tempo e custo de execução.

Para trabalhos futuros é sugerido detalhar o processo e fazer estudos empíricos a fim de medir sua efetividade.

REFERÊNCIA BIBLIOGRÁFICA

Manifesto, A. (2001). Manifesto for agile software development. Disponível em: <http://agilemanifesto.org/>. Acesso em: 11 set. 2019.

Alves, M. N. (2017). Uma proposta para identificar, medir e gerenciar a dívida técnica em requisitos de software. 142 f. Dissertação (Mestrado Profissional em Engenharia de Computação) - Coordenadoria de Ensino Tecnológico, Instituto de Pesquisas Tecnológicas do Estado de São Paulo, São Paulo, 2017.

Alves, N. S., Mendes, T. S., de Mendonça, M. G., Spínola, R. O., Shull, F., & Seaman, C. (2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, *70*, 100-121.

Alzaghoul, E., & Bahsoon, R. (2013, December). Economics-driven approach for managing technical debt in cloud-based architectures. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing* (pp. 239-242). IEEE.

Avgeriou, P., Kruchten, P., Ozkaya, I., & Seaman, C. (2016). Managing technical debt in software engineering (dagstuhl seminar 16162). In *Dagstuhl Reports* (Vol. 6, No. 4). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Azizyan, G., Magarian, M. K., & Kajko-Matsson, M. (2011, August). Survey of agile tool usage and needs. In *2011 agile conference* (pp. 29-38). IEEE.

Besker, T., Martini, A., & Bosch, J. (2019). Software developer productivity loss due to technical debt—A replication and extension study examining developers' development work. Journal of Systems and Software, 156, 41-61.

Bob, U. A Mess is not a Technical Debt. 2009. Disponível em: <https://sites.google.com/site/unclebobconsultingllc/a-mess-is-not-a-technical-debt>. Acesso em: 18 dez. 2019.

Bohnet, J., & Döllner, J. (2011, May). Monitoring code quality and development activity by software maps. In *Proceedings of the 2nd Workshop on Managing Technical Debt* (pp. 9-16).

Bomfim Junior, M. M. (2016). Estratégias para redução de dívidas técnicas em equipes ágeis. 143 f. Dissertação (Mestrado Profissional em Engenharia de Computação) - Coordenadoria de Ensino Tecnológico, Instituto de Pesquisas Tecnológicas do Estado de São Paulo, 2016.

Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P. & Sangwan, R. (2010, November). Managing technical debt in software-reliant systems. In Proceedings of the FSE/SDP workshop on Future of software engineering research (pp. 47-52).

Cysneiros, L. M., & Leite, J. C. S. P. (1997). Definindo requisitos não funcionais. *XI Simpósio Brasileiro de Engenharia de Software. Fortaleza, CE*, 33.

Codabux, Z., & Williams, B. (2013, May). Managing technical debt: An industrial case study. In *2013 4th International Workshop on Managing Technical Debt (MTD)* (pp. 8-15). IEEE.

Cohn, M. (2005). *Agile estimating and planning*. Pearson Education.

Cunningham, W. (1992). The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger*, *4*(2), 29-30.

Fowler, M. Technical Debt. 2003. Disponível em: <http://www.martinfowler.com/bliki/TechnicalDebt.html/>. Acesso em: 19 dez. 2019.

Fowler, M. Technical Debt Quadrant. 2009. Disponível em: <http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html/>. Acesso em: 19 dez. 2019.

Fowler, M. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

Gartner. A Primer on Technical Debt (2016). Disponível em: <https://www.gartner.com/en/documents/3467217>. Acesso em: 31 out. 2019.

Gartner. Predicts 2019: Governing Application and Product Portfolios (2018). Disponível em: <https://www.gartner.com/en/documents/3894845>. Acesso em: 1 nov. 2019.

Greening, D. R. (2013, January). Release duration and enterprise agility. In *2013 46th Hawaii International Conference on System Sciences* (pp. 4835-4841). IEEE.

Grenning, J. (2002). Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, *3*, 22-23.

Griffith, I., Taffahi, H., Izurieta, C., & Claudio, D. (2014, December). A simulation study of practical methods for technical debt management in agile software development. In *Proceedings of the Winter Simulation Conference 2014* (pp. 1014-1025). IEEE.

Goldstein, I. (2013). *Scrum shortcuts without cutting corners: agile tactics, tools, & tips*. Addison-Wesley (p. 69).

Guo, Y., & Seaman, C. (2011, May). A portfolio approach to technical debt management. In *Proceedings of the 2nd Workshop on Managing Technical Debt* (pp. 31-34).

Guo, Y., Spínola, R. O., & Seaman, C. (2016). Exploring the costs of technical debt management–a case study. *Empirical Software Engineering*, *21*(1), 159-182.

International Institute of Business Analysis. (2015). *A Guide to the Business Analysis Body of Knowledge (BABOK Guide), Version 3.0*. International Institute of Business Analysis.

ISO/IEC. (2011). ISO/IEC 25010: 2011 Systems and software engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)-System and software quality models.

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Ieee software*, *29*(6), 18-21.

Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, *101*, 193-220.

Martini, A., & Bosch, J. (2017, May). The magnificent seven: towards a systematic estimation of technical debt interest. In *Proceedings of the XP2017 Scientific Workshops* (pp. 1-5).

Martini, A., Vajda, S., Vasa, R., Jones, A., Abdelrazek, M., Grundy, J., & Bosch, J. (2017, May). Technical debt interest assessment: from issues to project. In *Proceedings of the XP2017 Scientific Workshops* (pp. 1-6).

McConnell, S. (2008). Managing technical debt. *Construx Software Builders, Inc*, 1-14.

Morais, R. M. D., & Costa, A. L. (2014). Um modelo para avaliação de sistemas de informação do SUS de abrangência nacional: o processo de seleção e estruturação de indicadores. *Revista de Administração Pública*, *48*(3), 767-793.

Morgenthaler, J. D., Gridnev, M., Sauciuc, R., & Bhansali, S. (2012, June). Searching for build debt: Experiences managing technical debt at Google. In *2012 Third International Workshop on Managing Technical Debt (MTD)* (pp. 1-6). IEEE.

Nakagawa, E. Y., Scannavino, K. R. F., Fabbri, S. C. P. F., & Ferrari, F. C. (2017). *Revisão sistemática da literatura em engenharia de software: teoria e prática*. Elsevier Brasil.

Oliveira, F. S. (2015). Gerenciamento da Dívida Técnica em projetos de software utilizando Scrum: uma pesquisa-ação. 92 f. Dissertação (Mestrado Profissional em Engenharia de Computação) - Coordenadoria de Ensino Tecnológico, Instituto de Pesquisas Tecnológicas do Estado de São Paulo, São Paulo, 2015.

Oliveira, F., Goldman, A., & Santos, V. (2015, August). Managing technical debt in software projects using scrum: An action research. In *2015 Agile Conference* (pp. 50-59). IEEE.

Osses, F., Márquez, G., Orellana, C., & Astudillo, H. (2017, October). Towards the selection of security tactics based on non-functional requirements: Security tactic planning poker. In *2017 36th International Conference of the Chilean Computer Science Society (SCCC)* (pp. 1-8). IEEE.

Pires, R. C. (2014). Um estudo sistemático sobre identificação e gerenciamento de Dívida Técnica em uma Empresa de Tecnologia com Desenvolvimento baseado em Scrum. 68 f. Dissertação (Mestrado Profissional em Engenharia de Computação) - Coordenadoria de Ensino Tecnológico, Instituto de Pesquisas Tecnológicas do Estado de São Paulo, São Paulo, 2014.

Potdar, A., & Shihab, E. (2014, September). An exploratory study on self-admitted technical debt. In *2014 IEEE International Conference on Software Maintenance and Evolution* (pp. 91-100). IEEE.

Rubin, K. S. (2012). *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley (p. 452).

Sachdeva, V. (2018). Requirements prioritization in agile: use of planning poker for maximizing return on investment. In *Information Technology-New Generations* (pp. 403-409). Springer, Cham.

Seaman, C., Guo, Y., Zazworka, N., Shull, F., Izurieta, C., Cai, Y., & Vetrò, A. (2012, June). Using technical debt data in decision making: Potential decision approaches. In *2012 Third International Workshop on Managing Technical Debt (MTD)* (pp. 45-48). IEEE.

Seaman, C., & Guo, Y. (2011). Measuring and monitoring technical debt. In *Advances in Computers* (Vol. 82, pp. 25-46). Elsevier.

Seaman, C., & Spínola, R. (2013). Managing technical debt In:(Short Course) XVII Brazilian Symposium on Software Quality. *SBC, Salvador, Brazil*.

Zazworka, N., Spínola, R. O., Vetro', A., Shull, F., & Seaman, C. (2013, April). A case study on effectively identifying technical debt. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering* (pp. 42-47).