PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM ESTUDO EMPÍRICO SOBRE A GERÊNCIA DE DÍVIDA TÉCNICA EM PROJETOS DE DESENVOLVIMENTO DE SOFTWARE QUE UTILIZAM SCRUM

CIRO GOULART DOS SANTOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Rafael Prikladnicki

Porto Alegre
2015

Dados Internacionais de Catalogação na Publicação (CIP)

S237e Santos, Ciro Goulart dos

Um estudo empírico sobre a gerência de dívida técnica em projetos de desenvolvimento de software que utilizam Scrum / Ciro Goulart dos Santos. – 2016.
128 p.

Diss. (Mestrado) – Faculdade de Informática, PUCRS. Orientador: Prof. Dr. Rafael Prikladnicki

1. Engenharia de Software. 2. Scrum (Desenvolvimento de Software). 3. Informática. I. Prikladnicki, Rafael. II. Título.

CDD 23 ed. 005.1

Salete Maria Sartori CRB 10/1363

Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Um Estudo Empírico sobre a Gerência de Dívida Técnica em Projetos de Desenvolvimento de Software que Utilizam Scrum" apresentada por Ciro Goulart dos Santos como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 17/03/2015 pela Comissão Examinadora:

Prof. Dr. Rafael Prikladnicki – Orientador — PPGCC/PUCRS

Prof. Dr. Duncan Dubugras – PPGCC/PUCRS

Prof. Dr. Rodrigo Penteado Ribeiro de Toledo – UFRJ

Homologada em 05/05/2016, conforme Ata No. 09 pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

DEDICATÓRIA

Aos meus pais, Carmo e Iara, e à minha esposa Dayane

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus pelo dom da vida e por me dar saúde, permitindo assim realizar este projeto até o fim. Também pela saúde de meus familiares que me deram suporte nestes dois anos.

À minha esposa, Dayane, sem a qual a caminhada seria muito mais árdua. Seu incentivo, companheirismo e compreensão nos momentos mais difíceis foram fundamentais. Obrigado pelo carinho, amor e por acreditar sempre que eu venceria todas as dificuldades.

À minha família e principalmente aos meus pais, que me proporcionaram uma educação sólida e por me transmitirem seus valores que moldaram meu caráter. Obrigado por sempre me incentivarem a progredir de forma honesta. Também pelo apoio e torcida o tempo todo, mesmo que às vezes de longe, e pelas palavras de estímulo.

Ao meu orientador, Prof. Dr. Rafael Prikladnicki, por ter aceitado meu pedido de ser seu orientando. Obrigado pela disponibilidade e por todo apoio que tive através de críticas construtivas, pelos “empurrões” nos momentos certos e pelas inúmeras oportunidades de aprendizado proporcionadas durante o decorrer do curso. Muito obrigado!

Ao Prof. Dr. Duncan Ruiz por acompanhar o meu trabalho dando importantes *feedbacks* durante as avaliações intermediárias.

Aos meus colegas de mestrado, principalmente os que me acompanharam mais de perto como o Bernardo Estácio, o Isaque Vacari, o Tiago Duarte e a Silvia Nunes. Obrigado pela amizade, ajuda mútua e momentos de descontração.

À Hewlett-Packard, onde trabalho, por me proporcionar esta oportunidade através do apoio com os custos e pela flexibilidade necessária para que eu pudesse conduzir este trabalho.

À PUCRS, universidade na qual orgulhosamente me graduei e passei boa parte de minha vida. Obrigado aos professores e funcionários do PPGCC e pela excelente infraestrutura proporcionada para que eu pudesse ter um ótimo aprendizado e experiência acadêmica.

Aos entrevistados do estudo de campo, pela disponibilidade de tempo e interesse demonstrados. A todos que direta ou indiretamente contribuíram, muito obrigado!

UM ESTUDO EMPÍRICO SOBRE A GERÊNCIA DE DÍVIDA TÉCNICA EM PROJETOS DE DESENVOLVIMENTO DE SOFTWARE QUE UTILIZAM SCRUM

RESUMO

Em um mundo de recursos finitos onde priorização e escolhas são constantemente necessárias, é inevitável que de alguma maneira o processo de desenvolvimento de software seja comprometido gerando custos ao longo do tempo, fenômeno que Ward Cunningham chamou de “dívida técnica” em alusão à dívida financeira. Cunningham afirma que “entregar código imaturo é como entrar em dívida. Um pouco de dívida agiliza o desenvolvimento contanto que ela seja paga de volta prontamente com reescrita”. Esta pesquisa de mestrado tem como objetivo aprofundar e entender os limites dessa metáfora bem como propor uma forma de integrar o gerenciamento dessa dívida em um projeto de desenvolvimento de software. Para tanto, conduziu- se um estudo utilizando métodos secundários (revisão de literatura) e primários (estudo de campo) de pesquisa. O estudo contribui no sentido de propor uma abordagem preliminar para gerenciamento da dívida técnica em projetos de desenvolvimento de software que utilizam Scrum.

Palavras-chave: Dívida Técnica, Engenharia de Software, *Refactoring*.

AN EMPIRICAL STUDY ABOUT TECHNICAL DEBT MANAGEMENT IN SOFTWARE DEVELOPMENT PROJECTS USING SCRUM

ABSTRACT

In a world of limited resources where prioritization and trade-offs are constantly needed it's inevitable that the software development process is somehow impacted, thus increasing costs over time, phenomenon which Ward Cunningham called "technical debt" as a metaphor to financial debt. Cunningham states, "Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite". The goal of this dissertation is to deepen and understand the limits of this metaphor as well as to propose a way to integrate the technical debt management into a software development project. For such, we have used both secondary (literature review) and primary (field study) research methods. This research contributes in a way that it proposes a preliminary approach to technical debt management in software development projects that use Scrum.

Keywords: Technical Debt, Software Engineering, Refactoring.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

COTS	*Commercial off-the-shelf*
DSDM	*Dynamic Systems Development Method*
DT	Dívida Técnica
IDE	*Integrated Development Environment*
PO	*Product Owner*
ROI	*Return on Investment*
RSL	Revisão Sistemática da Literatura
TI	Tecnologia da Informação
XP	*eXtreme Programming*

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais, indivíduos e nações dependem do avanço de sistemas de software no controle de sua infraestrutura, produção industrial, sistema financeiro e entretenimento [SOM11]. Portanto, a Engenharia de Software tem papel fundamental no funcionamento das sociedades modernas [SOM11]. Softwares são abstratos e intangíveis, tornando-se complexos, difíceis de entender e caros de se manter [SOM11]. Aliado a isso, com o crescimento da indústria de software o desenvolvimento de software se tornou, por necessidade, cada vez menos uma atividade solitária e cada vez mais um esforço coletivo [BAE98]. Sendo assim, o principal desafio é que o conhecimento originalmente disperso entre várias pessoas precisa ser reunido e incorporado no software [BAE98].

Pensar o software como um “capital” [BAE98] resultante desse processo traz a noção do valor que os sistemas de software geram para as organizações e a importância de se ter um processo eficiente para produzi-lo. Porém, em um mundo de recursos finitos onde priorização e escolhas são constantemente necessárias, é inevitável que de alguma maneira o processo de desenvolvimento de software seja comprometido gerando custos ao longo do tempo [TOM13], fenômeno o qual Ward Cunningham [CUN92] chamou de “dívida técnica” em alusão à dívida financeira.

De acordo com um estudo realizado pelo instituto de pesquisas Gartner [THI10], a “Dívida de Tecnologia da Informação (TI)” global em 2010 era estimada em US$ 500 bilhões, com potencial de chegar a US$ 1 trilhão até 2015. O Gartner define a dívida de TI como o custo de limpar o *backlog* de manutenção necessário para manter as aplicações do portfolio de TI rodando na versão mais atual, trazendo a organização para um ambiente com suporte total.

Recentemente a dívida técnica, ou no termo em inglês *“technical debt”*, é uma metáfora que vem sendo bastante discutida pela comunidade de desenvolvimento de software na internet em *blogs* e fóruns de discussão [BRO10] [TOM13]. O termo tem ganhado um espaço significativo na comunidade de desenvolvimento ágil de software como uma forma de entender e comunicar questões como efeitos em longo prazo de decisões tomadas no presente [BRO10]. Porém, estudos realizados em 2012 mostravam uma falta de definição abrangente ou modelo conceitual sobre dívida técnica na literatura acadêmica [TOM13].

Frequentemente usada para referir-se aos custos de se entregar código imaturo em detrimento de prazos apertados [TOM13], a dívida técnica tem despertado bastante interesse tanto em profissionais que atuam na indústria do desenvolvimento de software quanto na comunidade científica. Contudo, pouco se tem aprofundado o tema [SEA11a], sendo necessários estudos empíricos que tratem das seguintes questões:

- **Visualização e comunicação da dívida técnica:** Como comunicar a dívida técnica, formas de visualização e análise, como dar a visibilidade adequada, torná-la explícita, rastreá-la e os benefícios de tornar seu gerenciamento explícito.
- **Suporte à tomada de decisão:** Como a dívida técnica pode dar apoio à tomada de decisão, proporcionar *feedback* sobre decisões de aceitar ou pagar a dívida técnica, análise de riscos de se assumir certos tipos de dívida, técnicas de mensuração para cada tipo de dívida técnica combinadas com uma forma útil para a tomada de decisão, entender como a dívida técnica é inserida, visualizada e resolvida.
- **Gerenciamento e monitoração da dívida técnica:** Rastrear decisões não relacionadas a código ao longo do ciclo de desenvolvimento de um projeto, particularmente artefatos de *design*, teste e requisitos, monitorar e gerenciar a dívida técnica arquitetural [TOM13], conhecer as diferentes fontes de dívida técnica, coletar informações sobre dívida técnica empiricamente.
- **Ferramental da dívida técnica:** Buscar formas de identificar automaticamente a dívida técnica em projetos de desenvolvimento de software, monitorar o nível de dívida técnica com o passar do tempo, reconhecer tendências e disparar avisos nos momentos adequados integrando tudo isso em um ambiente de desenvolvimento e gerenciamento através de ferramentas adequadas.

Nesse sentido, fazem-se necessárias diretrizes rigorosamente avaliadas e técnicas que abordem a dívida técnica no intuito de identificá-la, estimá-la, gerenciá-la e utilizá-la na tomada de decisão em ambientes corporativos [COD13]. É nesse contexto que surge esta dissertação, cujos objetivos serão detalhados a seguir.

1.1 Objetivo

Esta dissertação tem como objetivo geral propor uma abordagem para gerenciamento da dívida técnica em projetos de desenvolvimento de software que utilizam Scrum. Para tanto, foi realizada uma análise dos principais conceitos relacionados à dívida técnica em engenharia de software em busca de oportunidades de pesquisa e um estudo empírico sobre a gerência da dívida técnica em projetos de desenvolvimento de software, com foco no gerenciamento da dívida técnica.

De forma a complementar o objetivo geral proposto, os seguintes objetivos específicos foram definidos:

- Revisar e apresentar os principais conceitos de Engenharia de Software e Processo de Desenvolvimento de Software
- Pesquisar os conceitos em busca de um entendimento abrangente da dívida técnica. Definição de dívida técnica, os tipos, propriedades, características, como medir, precedentes e consequências através de uma revisão de literatura.
- Identificar oportunidades de pesquisa relacionadas à dívida técnica em projetos de desenvolvimento de software
- Planejar e realizar um estudo de campo de forma a compreender como ocorre o gerenciamento da dívida técnica em um projeto real de desenvolvimento de software
- Propor uma extensão do Scrum para gerenciamento da dívida técnica

1.2 Justificativa

Existe uma oportunidade de estudos e avaliações empíricas que ajudem a mostrar como evidenciar e tornar a dívida técnica transparente para áreas de negócio. A dívida técnica representa um custo de juros relacionado à perda dos benefícios que de outro modo poderiam ter sido alcançados pelo negócio [CUR12]. Ainda, por se tratar de uma metáfora, há uma necessidade de tornar a dívida técnica em fatos literais que possam ser utilizados no dia-a-dia para gerenciá-la no domínio do desenvolvimento de software [TOM13].

A prática atual na indústria é o gerenciamento implícito da dívida técnica [GUO11]. O time de desenvolvimento sabe (ou aprende) que está entregando software com defeitos estruturais que precisam ser corrigidos sob pena de o custo e risco da aplicação crescer inaceitavelmente [CUR12]. Sendo assim, tendo a consciência que ela existe e tornando seu

gerenciamento explícito, acredita-se que isso possa fazer a diferença no desempenho de projetos de desenvolvimento.

Neste sentido, considerando os aspectos supramencionados, este estudo busca responder a seguinte questão de pesquisa: como uma equipe de desenvolvimento de software que utiliza o método ágil Scrum pode integrar o gerenciamento da dívida técnica em seu processo?

1.3 Organização do trabalho

Visando uma melhor compreensão dos objetivos descritos, esta dissertação está organizada em 6 capítulos, de forma a possuir a seguinte estrutura:

O capítulo 2 trata dos conceitos envolvidos em Processo de Desenvolvimento do Software, abordando o surgimento da Engenharia de Software e os principais modelos de processo de software e projeto de software. Além disso, o capítulo 2 apresenta os conceitos de dívida técnica, tais como definições, taxonomia, dimensões e atributos. Também são abordados os precedentes e consequências da dívida técnica além de como ela é quantificada, gerenciada e comunicada em um ambiente de desenvolvimento de software. Ainda neste capítulo são apresentados os principais desafios e oportunidades de pesquisa em dívida técnica, baseado na análise de publicações de impacto na comunidade científica.

O capítulo 3 detalha o método de pesquisa utilizado, descrevendo cada uma das etapas do estudo bem como a justificativa das escolhas e uso dos métodos.

No capítulo 4 é apresentado o estudo de campo, descrevendo a metodologia, as fases e caracterização dos respondentes. Ainda nesse capítulo é feita uma análise do Scrum sobre a perspectiva da dívida técnica mapeada no estudo de campo.

A proposta de extensão do Scrum é apresentada no capítulo 5, como resultado da análise feita no capítulo 4 resultante do estudo de campo.

Por fim, o capítulo 6 traz as considerações finais sobre o trabalho, limitações e trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1 Processo de Desenvolvimento de Software

Processo de desenvolvimento de software, ou simplesmente processo de software, é um conjunto de atividades relacionadas que conduzem à produção de um produto de software [SOM11]. A maior parte do desenvolvimento de software existente é profissional, ou seja, grandes equipes de profissionais são pagas para desenvolver software para que outros o utilizem para fins de negócio. Esse tipo de desenvolvimento exige a utilização de técnicas que apoiem a especificação, projeto e manutenção de programas, as quais normalmente não possuem a mesma relevância no desenvolvimento para fins pessoais [SOM11]. A disciplina que visa dar suporte ao desenvolvimento profissional de software é a Engenharia de Software. Sommerville [SOM11] lista os seguintes atributos como responsáveis pelas características essenciais de um sistema de software profissional:

Tabela 1 - Atributos essenciais em um software profissional.

Características do produto	Descrição
Manutebilidade	O software deve ser escrito de maneira a evoluir para atender às necessidades do cliente.
Confiabilidade e segurança	Inclui características tais como confiabilidade, segurança e proteção.
Eficiência	O software não deve desperdiçar recursos do sistema como memória e ciclos de processador.
Aceitabilidade	O software deve ser aceitável para o usuário para o qual foi projetado.

2.1.1 Origem da Engenharia de Software

O termo “Engenharia de Software” foi originalmente proposto em 1968 em uma conferência organizada pelo Comitê de Ciência da Organização do Tratado do Atlântico Norte (OTAN) para discutir a então chamada “Crise do Software” [NAU69]. O nome “Crise do Software” foi atribuído devido às dificuldades em se desenvolver sistemas grandes e

complexos nos anos 60. A adoção de uma abordagem como a engenharia reduziria custos de desenvolvimento e produziria um software mais confiável [SOM11].

Embora existam várias definições para o termo, a proposta por Fritz Bauer [NAU69] em 1969 ainda serve como base para a discussão:

> “Engenharia de Software é o estabelecimento e uso de princípios sadios de engenharia no intuito de obter economicamente um software confiável e que funciona eficientemente em máquinas reais.”

Porém essa definição por si só não contempla aspectos importantes tais como qualidade, satisfação do cliente, métricas ou entrega no prazo [PRE11]. Uma definição mais abrangente foi desenvolvida pela IEEE [IEE90] ao afirmar:

> “Engenharia de Software: (1) A aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de software; isto é, a aplicação de engenharia ao software. (2) O estudo das abordagens como em (1).”

Posteriormente, durante os anos 70 e 80 foram desenvolvidas várias técnicas e métodos, tais como programação estruturada, abstração e desenvolvimento orientado a objetos. Ferramentas como *Computer-Aided Software Engineering* (CASE) e *Integrated Development Environment* (IDE), bem como notações padrão, foram desenvolvidas e são utilizadas até hoje [SOM11].

2.1.2 Modelos Prescritivos de Processo de Software

Um modelo de processo de software é uma representação simplificada de um processo de software. Independentemente do processo, as atividades a seguir são fundamentais para a engenharia de software [SOM11]:

- **Especificação do software:** Definição de funcionalidades e restrições.
- **Projeto e implementação do software:** Produção do software atendendo aos requisitos.
- **Validação do software:** Verifica se o software atende o que o cliente deseja.
- **Evolução do software:** O software deve evoluir de modo a atender às mudanças solicitadas pelo cliente.

As atividades são organizadas diferentemente em cada processo de desenvolvimento. Como essas atividades são conduzidas depende do tipo de software, pessoas e estruturas organizacionais envolvidas [SOM11].

Modelos de processos prescritivos, ou tradicionais, possuem em sua estrutura uma ordem formal de elementos do processo. Tais elementos incluem atividades, ações, tarefas, produtos de trabalho, mecanismos de garantia de qualidade e controle de modificações para cada projeto [PRE11]. As subseções seguintes abordarão os principais modelos prescritivos.

2.1.2.1 Modelo *waterfall* (Cascata)

Proposto em 1970 por [ROY70], foi o primeiro modelo publicado, derivado de processos de engenharia de sistemas mais genéricos. Os principais estágios do modelo *waterfall* refletem as atividades fundamentais de desenvolvimento [SOM11]:

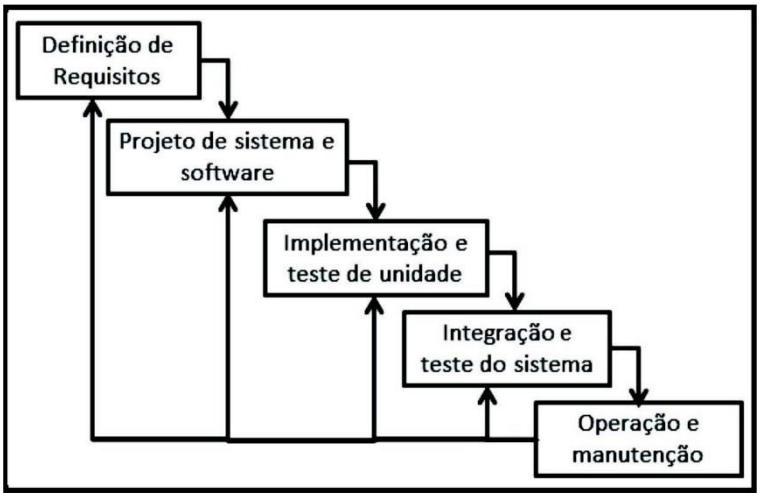


Figura 1 - Modelo *waterfall* [SOU15]

As cinco fases do modelo *waterfall* possuem as seguintes características [SOM11]:

1. Análise e definição de requisitos: os serviços, restrições e objetivos do sistema são estabelecidos através da consulta aos usuários. Estes são detalhados e servem como especificação do sistema.
2. Projeto do sistema e do software: os processos de projeto do sistema incluem requisitos de hardware ou sistemas de software, estabelecendo uma arquitetura geral. O projeto do software descreve as abstrações e relacionamentos fundamentais do sistema de software.

3. Implementação e teste do sistema: o projeto de software é realizado através de um conjunto de programas e unidades de programa. Testes unitários são realizados para garantir que cada unidade atende ao que foi especificado.
4. Testes de integração e de sistema: as unidades de programa são integradas testado como um sistema completo para garantir o atendimento aos requisitos. Após os testes, o sistema é liberado para o cliente.
5. Operação e manutenção: é a fase mais longa do ciclo de vida. O sistema é instalado e utilizado. A manutenção envolve corrigir erros não detectados previamente, melhorar a implementação das unidades do sistema e implementar novas funcionalidades.

2.1.2.2 Modelo incremental

A ideia básica deste modelo é desenvolver uma implementação inicial, expô-la ao usuário e evolui-la através de várias versões até que um sistema adequado seja desenvolvido. As principais vantagens em relação ao modelo *waterfall* são [SOM11]:

1. O custo de mudanças nos requisitos é reduzido.
2. É mais fácil de obter *feedback* do cliente, pois ele pode comentar em demonstrações e ver o quanto já foi implementado.
3. Entregas rápidas são possibilitadas, e clientes podem usar e obter retorno mais cedo do que no *waterfall*.

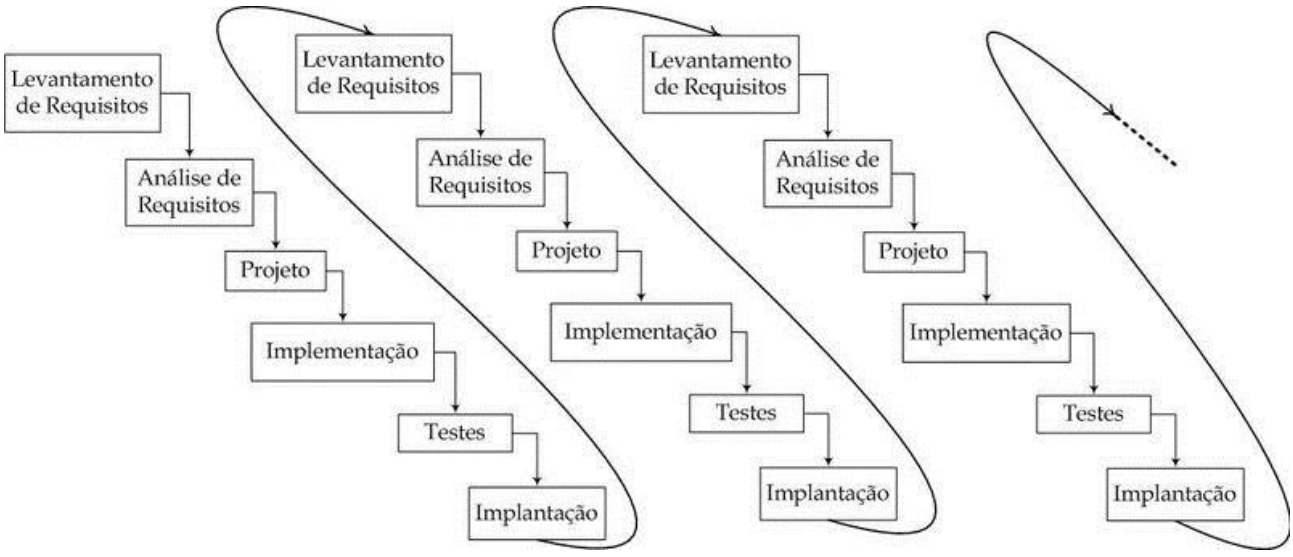


Figura 2 - Modelo Incremental [IFS15]

Sommerville [SOM11] lista ainda algumas desvantagens a partir das perspectivas gerenciais e de adoção desse modelo:

1. O processo não é visível, portanto gerentes necessitam de entregas regulares para medir o progresso.
2. A estrutura do sistema tende a se degradar durante o acréscimo de novas funcionalidades, caso não seja investido tempo e dinheiro em *refactoring* [FOW99].
3. Grandes empresas possuem processos burocráticos, cujos procedimentos podem não corresponder a um processo iterativo.
4. A validação pode se tornar difícil pela ausência de documentação de especificação do sistema, pois esta é intercalada com o desenvolvimento.

2.1.2.3 Prototipação

Um protótipo é uma versão inicial de um software que é utilizado para demonstrar conceitos, experimentar opções de design e entender melhor sobre o problema e suas principais soluções [SOM11]. A prototipação, ou prototipagem, é um modelo idealmente utilizado como um mecanismo de identificação de requisitos, auxiliando cliente e desenvolvedores a entenderem melhor o precisa se construído quando os requisitos estão confusos [PRE11].

O processo consiste em estabelecer os objetivos do protótipo, definir suas funcionalidades, desenvolvê-lo e avaliá-lo. Os objetivos já devem estar explícitos a partir do início do processo. Eles podem ser variados como desenvolver um protótipo de interface com usuário, desenvolver um sistema para validar os requisitos ou para demonstrar a viabilidade de uma aplicação para os gerentes [SOM11].

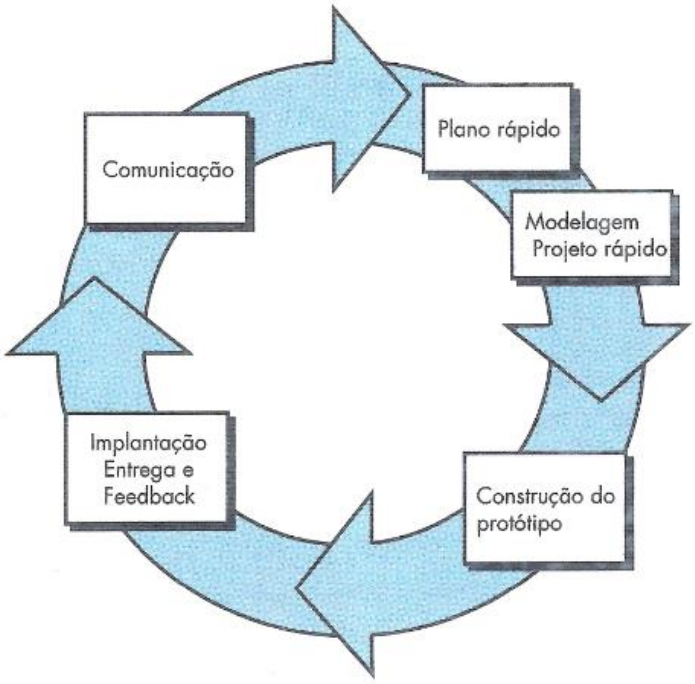


Figura 3 – Prototipação [GAL15]

Um problema geral com a prototipação é que o protótipo pode não necessariamente ser utilizado da mesma maneira que um sistema final, pois o usuário que testa o protótipo pode não ser um típico usuário do sistema e os avaliadores podem ajustar seu modo de trabalhar de acordo com o tempo de resposta do protótipo [SOM11]. Outro ponto fraco é que às vezes os desenvolvedores são pressionados pelos gerentes a entregar um protótipo, especialmente quando há atrasos na entrega da versão final. Isso tem consequências negativas em requisitos não funcionais como performance, segurança, robustez e confiabilidade, além de falta de documentação e dificuldade de manutenção [SOM11].

2.1.2.4 RAD

Proposto em 1991 por James Martin, o modelo *Rapid Application Development* (RAD) enfatiza iterações curtas (três meses ou menos), causando rápida sucessão de entregas [PRE11]. É uma adaptação do modelo cascata, no qual o desenvolvimento rápido é conseguido com o uso de uma abordagem de construção baseada em componentes, sendo eles: comunicação, planejamento, modelagem, construção e implantação. Pode haver múltiplos times, cada um trabalhando em um componente, sendo todos integrados no final [PRE11].

Assim como outros modelos, o RAD também possui algumas desvantagens [PRE11]:

1. Pode requerer um grande número de pessoas para formar todos os times.
2. Cliente e desenvolvedores devem estar comprometidos para ações rápidas.
3. O sistema deve ser passível de ser quebrado em componentes.
4. O sistema resultante pode não possuir uma performance adequada
5. Pode não ser adaptável para utilização de novas tecnologias, pois requer uma tecnologia bem conhecida.

2.1.2.5 Modelo Espiral

O modelo de processo de software direcionado a risco (modelo espiral) foi proposto por Boehm em 1988 [SOM11]. Neste modelo, o processo de software é representado como uma espiral ao invés de uma sequência de atividades, conforme ilustrado na figura abaixo [SOM11]:

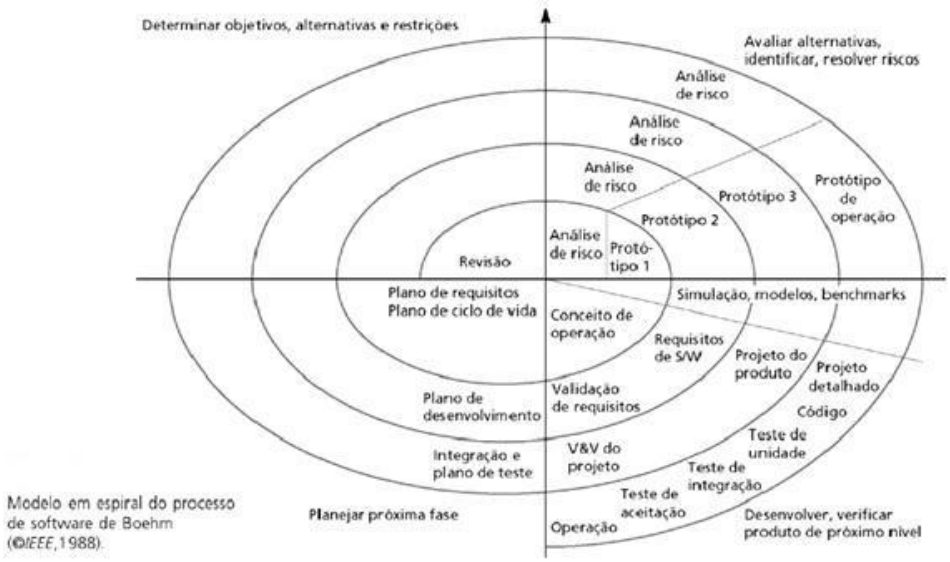


Figura 4 - Modelo espiral

Cada volta na espiral é dividida em quatro setores [SOM11]:

1. Definição do objetivo: são definidos objetivos específicos para aquela fase do projeto.
2. Análise e redução de riscos: para cada risco identificado no projeto, uma análise detalhada é conduzida.
3. Desenvolvimento e validação: um modelo para o sistema é escolhido dependendo dos riscos envolvidos. Por exemplo, um protótipo de interface com o usuário pode ser escolhido caso riscos relacionados à interface com o usuário sejam dominantes.

Se o principal risco identificado for uma integração de subsistemas, *waterfall* pode ser a melhor escolha.

4. Planejamento: o projeto é revisado e então é decidido se deve ser dada mais uma volta na espiral. Caso afirmativo, planos são definidos para a próxima fase.

A principal diferença entre o modelo espiral e os outros processos é o reconhecimento explícito do risco. Riscos levam a mudanças propostas no software e problemas de projeto tais como prazos e custos, portanto a mitigação dos riscos é uma atividade muito importante no gerenciamento de projetos [SOM11].

2.1.3 RUP

O *Rational Unified Process* (RUP) é um exemplo de modelo de processo moderno derivado do trabalho no UML e o relativo *Unified Software Development Process*. Ele une elementos de todos os processos genéricos, estabelece boas práticas em especificação e design e suporta prototipação e entregas incrementais [SOM11].

RUP é normalmente descrito a partir de três perspectivas:

1. Uma perspectiva dinâmica, mostra as fases do modelo ao longo do tempo;
2. Uma perspectiva estática, mostra as atividades do processo que são ordenadas;
3. Uma perspectiva prática, sugere boas práticas a serem usadas durante o processo.

O RUP é um modelo em fases que identifica quatro fases discretas no processo de desenvolvimento. No entanto, diferentemente do *waterfall* onde as fases são equivalentes às atividades do processo, as fases no RUP são mais intimamente ligadas ao negócio ao invés de aspectos técnicos. As quatro fases são descritas abaixo [SOM11], podendo ser observadas na figura 5:

1. Concepção ou Iniciação: tem por objetivo estabelecer uma visão de negócios para o sistema. São identificadas entidades externas (pessoas e sistemas) que irão interagir com o sistema e como irão interagir. Após são avaliadas as contribuições que o sistema dará para o negócio.
2. Elaboração: seus objetivos são desenvolver um entendimento do domínio do problema, estabelecer um *framework* arquitetural, o plano de projeto e identificar riscos-chave de projeto.
3. Construção: Envolve o *design*, programação e teste do sistema, onde partes dele são desenvolvidas paralelamente e integradas no final.

4. Transição: o sistema é movido dos desenvolvedores para os usuários finais e colocado em produção.

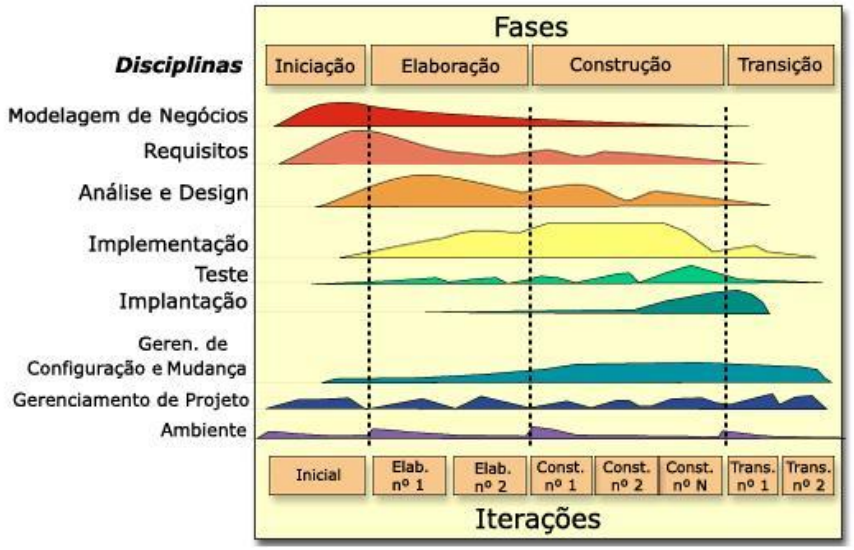


Figura 5 - Modelo RUP [SOM11]

O modelo RUP ainda prescreve boas práticas de engenharia de software, recomendadas para utilização no desenvolvimento de sistemas. São elas [SOM11]:

1. Desenvolver software iterativamente: incrementos devem ser baseados nas prioridades do cliente, sendo as de mais alta prioridade desenvolvidas mais cedo no processo de desenvolvimento.
2. Gerenciar requisitos: documentar explicitamente os requisitos do cliente e acompanhar as mudanças nesses requisitos, analisando seu impacto antes de aceitá-las.
3. Utilizar arquiteturas baseadas em componentes: estruturar a arquitetura do sistema em componentes.
4. Modelar o sistema visualmente: utilizar modelos gráficos em UML para apresentar as visões estática e dinâmica do software.
5. Verificar a qualidade do software: garantir que o software atinja os padrões de qualidade organizacionais.
6. Controlar mudanças no software: gerenciar as mudanças utilizando um sistema de apoio, bem como procedimentos e ferramentas de gerência de configuração.

2.1.4 Modelos Adaptativos de Processo de Software

De acordo com Keen [KEE80], uma abordagem adaptativa possui os seguintes componentes: usuário, analista e o próprio sistema. Os três interagem entre si durante o processo de desenvolvimento. A noção de adaptabilidade é influenciada pelos princípios da melhoria contínua, muito utilizado na indústria automotiva.

Modelos adaptativos são processos mais flexíveis, priorizam o produto e as pessoas envolvidas em seu desenvolvimento e são abertos à mudança [PRE11]. Tais modelos funcionam bem em ambientes que possuem requisitos mais voláteis. Os métodos ágeis possuem características dos modelos adaptativos, possuindo poucas regras bem definidas e com princípios e práticas que favorecem a adaptabilidade no processo de desenvolvimento de software.

2.1.5 Métodos Ágeis de Desenvolvimento de Software

Em um ambiente cada vez mais dinâmico onde os negócios operam globalmente e necessitam de uma resposta rápida, torna-se muitas vezes impraticável obter-se de antemão um conjunto completo de requisitos estáveis de software [SOM11]. A partir do início dos anos 80, com a introdução do desenvolvimento incremental pela IBM [MIL80], passou-se a reconhecer a necessidade de um desenvolvimento rápido de software capaz de lidar com a necessidade de mudanças nos requisitos [SOM11]. Porém, somente na década de 90 com propostas ágeis como o *Dynamic Systems Development Method* (DSDM) [STA97], *Scrum* [SCW01] e *eXtreme Programming* (XP) [BEC00] foi que então a ideia de “métodos ágeis” realmente se estabeleceu [SOM11].

O descontentamento por parte dos desenvolvedores de software com processos pesados de engenharia de software orientados a planejamento que frequentemente sobrecarregavam o processo de desenvolvimento foi o que motivou esse “movimento ágil” na indústria durante os anos 90 [SOM11]. Foi então que em 2001 um grupo de dezessete desenvolvedores e consultores se reuniu e publicou o chamado “Manifesto Ágil”, cuja filosofia visa valorizar [BEC13]:

- **Indivíduos e interações** mais do que processos e ferramentas.
- **Software funcional** mais do que documentação compreensível.
- **Colaboração do cliente** mais do que negociação de contratos.
- **Resposta às mudanças** mais do que seguir um plano.

O manifesto ainda adiciona: “Enquanto houver valor nos itens à direita, nós valorizamos mais os itens à esquerda”.

O objetivo principal dos métodos ágeis consiste em fazer com que a equipe de desenvolvimento foque apenas no software em si ao invés de em seu projeto e documentação [SOM11]. Alguns tipos de desenvolvimento de sistema são mais adequados aos métodos ágeis, tais como [SOM11]:

1. Desenvolvimento de produtos onde empresas de software desenvolvem pequenos e médios produtos para serem vendidos.
2. Desenvolvimento de sistemas customizados dentro das organizações, onde há um envolvimento do cliente durante o processo desde que não haja muitas regras externas afetando o software.

A seguir são apresentadas as características do Scrum, o qual se insere no contexto desta pesquisa no capítulo 5 deste trabalho.

2.1.5.1 Scrum

O Scrum é um *framework* utilizado a partir do início da década de 90 no qual pessoas podem endereçar problemas adaptativos complexos, ao mesmo tempo entregando produtos do mais alto valor possível de forma produtiva e criativa [SCW13]. O Scrum possui as seguintes características:

- Leve
- Simples de entender
- Difícil de dominar

O fundamento do Scrum é a teoria de controle de processo empírico, ou empiricismo. Nesta teoria, o conhecimento vem da experiência e a tomada de decisão é baseado no que é conhecido. O Scrum adota uma estratégia iterativa de modo a estimular a adaptabilidade e gestão de riscos [SCW13].

Três pilares compõem o Scrum: transparência, inspeção e adaptação. A transparência requer um entendimento comum e visível dos aspectos significativos do processo por parte dos responsáveis pelo resultado [SCW13]. Quanto à inspeção e adaptação, o Scrum prescreve quatro eventos formais contidos no Sprint, que é um período fechado de um mês ou menos no qual um incremento de produto “pronto”, utilizável e potencialmente “entregável” é criado. São eles [SCW13]:

- Sprint Planning: é o planejamento do Sprint, criado colaborativamente por todo o time de Scrum.
- Daily Scrum: é um período fechado de 15 minutos para o time de desenvolvimento sincronizar atividades e criar um plano para as próximas 24 horas.
- Sprint Review: é realizada ao final do Sprint para inspecionar o incremento e adaptar o Product Backlog, se necessário.
- Sprint Retrospective: é uma oportunidade que o time de Scrum tem para se auto inspecionar e criar um plano de melhoria para ser executado no próximo Sprint.

O time de Scrum consiste de um Product Owner, o Time de Desenvolvimento e o Scrum Master. Times de Scrum são auto organizados e multidisciplinares. Times multidisciplinares decidem a melhor maneira de realizar seu trabalho, ao invés de serem direcionados por outros de fora do time. Além disso, eles possuem as competências necessárias para concluir seu trabalho sem depender de outros que não fazem parte do time [SCW13].

O **Product Owner** é responsável por maximizar o valor do produto e do time de desenvolvimento [SCW13]. Ele é o responsável por gerenciar o Product Backlog, o que inclui [SCW13]:

- Expressar claramente os itens do Product Backlog
- Ordenar os itens do Product Backlog para atingir os objetivos e missões da melhor forma
- Otimizar o valor do trabalho que o Time de Desenvolvimento executa
- Garantir que o Product Backlog seja visível, transparente e claro para todos, e mostrar no que o Time de Desenvolvimento irá trabalhar em seguida
- Garantir que o Time de Desenvolvimento entenda os itens no Product Backlog no nível necessário

O Product Owner pode realizar todo o trabalho acima ou delegar para o Time de Desenvolvimento. Porém, ele permanece sendo o responsável. Para o Product Owner obter sucesso, toda a organização precisa respeitar suas decisões. Ninguém é autorizado a dizer ao Time de Desenvolvimento para trabalhar em um conjunto diferente de requisitos, nem o Time de Desenvolvimento é autorizado a agir no que quer que outros digam [SCW13].

O **Time de Desenvolvimento** executa o trabalho de entregar um incremento ao final de cada Sprint. Ele possui as seguintes características [SCW13]:

- Eles são auto-organizados
- Multidisciplinares, com as habilidades necessárias para criar um incremento de produto
- O Scrum não reconhece nenhum título para membros do Time de Desenvolvimento além de Desenvolvedor, não importa o trabalho que ele execute; não há exceções a esta regra
- O Scrum não reconhece sub-times no Time de Desenvolvimento, quaisquer que sejam os domínios a serem endereçados como teste ou análise de negócios; não há exceções a esta regra
- Membros do Time de Desenvolvimento possuem habilidades especializadas e áreas de foco, porém a responsabilidade pertence ao Time de Desenvolvimento como um todo

O tamanho do Time de Desenvolvimento deve ser pequeno o suficiente para permanecer ágil e grande o suficiente para completar um trabalho significativo dentro de um Sprint. Com menos de três membros diminui a interação e resultados em ganhos menores de produtividade. Mais de nove membros requer muita coordenação, pois geram muita complexidade para um processo empírico gerenciar. Os papéis de Product Owner e Scrum Master não entram nesta contagem a menos que estejam executando o trabalho do Sprint Backlog [SCW13].

O **Scrum Master** é responsável por garantir que o Scrum seja entendido e praticado [SCW13], garantindo a aderência à teoria, prática e regras do Scrum. Ele também é um líder-servidor para o Time de Scrum, ajudando os que estão fora do time a entender quais interações com o time são úteis e quais não.

O Scrum possui ainda os seguintes artefatos: Product Backlog, Sprint Backlog e o incremento. O Product Backlog é uma lista ordenada de tudo que é necessário no produto e única fonte de requisitos. O Sprint Backlog é um conjunto do Product Backlog selecionado para o Sprint e o incremento é a soma de todos os itens do Product Backlog completado num Sprint mais o valor dos incrementos de todos os Sprints anteriores [SCW13]. A figura a seguir ilustra o processo do Scrum:

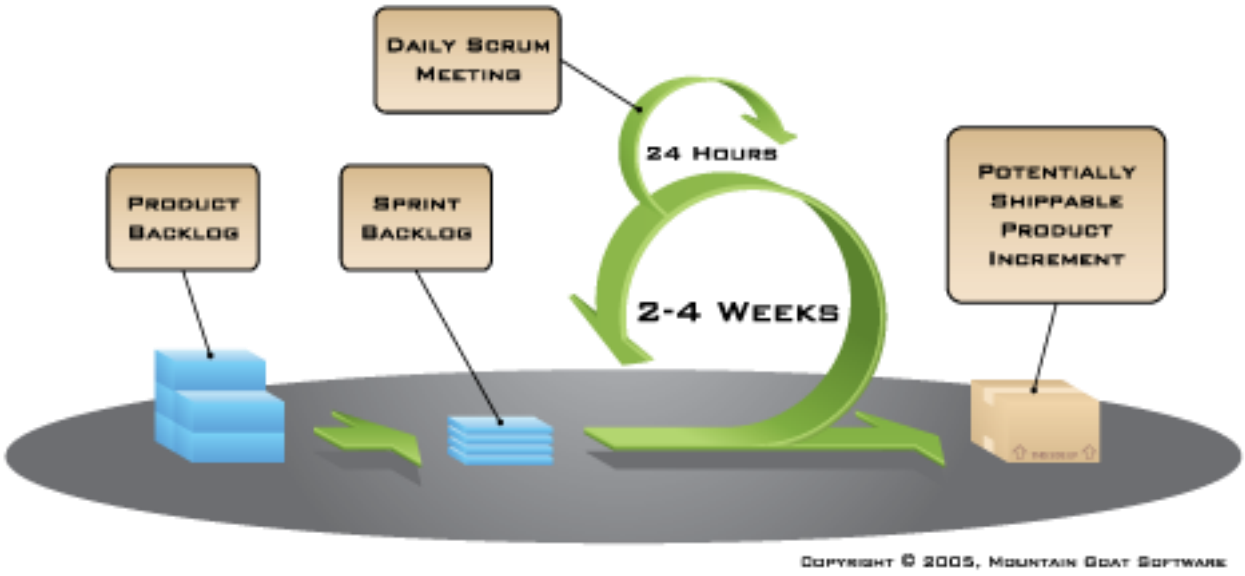


Figura 6 - O processo Scrum [MOU15]

O Scrum baseia-se na transparência. O Scrum Master precisa trabalhar com o Product Owner, o Time de Desenvolvimento e outras partes envolvidas para entender se os artefatos são completamente transparentes e aumentar a transparência caso necessário. Isso geralmente envolve aprendizado, convencimento e mudança [SCW13].

Quando um item do Product Backlog ou um incremento é descrito como "*Done*" (ou pronto), todos precisam entender o que "*Done*" significa. Embora isso varie significativamente por Time de Desenvolvimento, os membros precisam ter um entendimento compartilhado do que significa um item estar completo, de forma a garantir a transparência. Esta é a definição de "*Done*" para o Time de Scrum e é utilizada para avaliar quando o trabalho está concluído no Incremento do produto.

2.1.6 Atividades do Processo de Desenvolvimento de Software

As quatro atividades básicas que constituem um processo de software são: especificação, desenvolvimento, validação e evolução [SOM11].

2.1.6.1 Especificação de Software

Especificação de software ou engenharia de requisitos é o processo de entender e definir quais serviços são requeridos para o sistema e identificar restrições no desenvolvimento e operação do sistema [SOM11].

Requisitos normalmente são apresentados em dois níveis de detalhes: para usuários finais e desenvolvedores de sistema. Usuários finais precisam de um nível mais alto de detalhe, enquanto desenvolvedores necessitam uma especificação mais detalhada. Existem quatro atividades básicas no processo de engenharia de requisitos [SOM11]:

1. Análise de viabilidade
2. Elicitação e análise de requisitos
3. Especificação de requisitos
4. Validação de requisitos

Obviamente, nem sempre essas atividades são executadas em sequência. Por exemplo, em métodos ágeis, os requisitos são desenvolvidos incrementalmente de acordo com a prioridade dos usuários e a elicitação dos requisitos vem de usuários que fazem parte do time de desenvolvimento [SOM11].

2.1.6.2 Projeto e Implementação de Software

Projeto ou "*design"* e implementação de software é o estágio do processo de engenharia de software em que o sistema de software executável é desenvolvido [SOM11].

Sendo o centro da engenharia de software, o projeto de software é a primeira das atividades técnicas envolvidas na construção e verificação do software, a saber: projeto, codificação e teste [PRE11]. Em sistemas pequenos, projeto e implementação são a própria engenharia de software, tendo todas as outras atividades diluídas nesse processo. Em grandes sistemas, porém, eles são apenas um dos processos da engenharia de software tais como engenharia de requisitos, verificação e validação, etc. [SOM11]. O produto final de todas essas etapas desde a especificação do sistema implica em um software computacional validado [PRE11].

Segundo Pressman [PRE11], projeto de software é onde a qualidade é fomentada na engenharia de software, por possuir representações do software que podem ser avaliados em relação à qualidade. Por estar em constante evolução, o projeto de software não possui metodologias com profundidade, flexibilidade e natureza quantitativa como em outras disciplinas clássicas de projeto de engenharia. Porém, existem métodos e critérios de qualidade de projeto, além de notações que podem ser aplicadas [PRE11].

2.1.6.3 Verificação e Validação de Software

De acordo com Sommerville [SOM11], a qualidade do software está atrelada não apenas ao que o software faz, mas também ao comportamento dele enquanto é executado bem como à estrutura e organização dos programas e documentação associada.

A validação, mais genericamente, verificação e validação (V&V) pretende mostrar que um sistema está em conformidade com sua especificação e atinge as expectativas do cliente. Dito de outra forma, podemos diferenciar a verificação da validação através das seguintes perguntas [PRE11]:

- *Verificação*: “Estamos construindo o produto corretamente?”
- *Validação*: “Estamos construindo o produto certo?”

Verificação e validação englobam uma gama de atividades que são conhecidas como *software quality assurance* (SQA), as quais incluem revisões técnicas formais, auditoria de qualidade e configuração, monitoramento de *performance*, simulação, testes de desenvolvimento, testes de qualificação e testes de instalação, etc. [PRE11].

2.1.6.4 Evolução de Software

Historicamente existe uma divisão entre o processo de desenvolvimento de software e processo de evolução de software (manutenção de software). Apesar dos custos de manutenção serem bem maiores que os custos iniciais de desenvolvimento, os processos de manutenção são considerados menos interessantes e menos desafiadores que os de desenvolvimento. Porém cada vez mais essa distinção tem se tornado irrelevante [SOM11]. Ao invés disso, faz mais sentido pensar a engenharia de software como um processo evolutivo contínuo, onde o software é mudado ao longo do seu tempo de vida em resposta a mudanças de requisitos e necessidades de clientes [SOM11].

2.2 Dívida Técnica em Desenvolvimento de Software

Até então, a dívida técnica tem sido usada como uma metáfora e dispositivo retórico dentro da comunidade de desenvolvimento ágil com utilidade reconhecida para comunicação técnica e comunicação entre engenheiros de software e executivos [BRO10]. O conceito de dívida técnica tem ganhado força como uma forma de focar no gerenciamento de longo prazo de complexidades acidentais causadas por compromissos de curto prazo [BRO10]. Esta seção apresenta a metodologia utilizada na pesquisa,

conceitos de métodos ágeis, definições e implicações da dívida técnica no desenvolvimento de software bem como desafios e oportunidades de pesquisa.

2.2.1 “Dívida” ou “Débito”?

O termo original em inglês cunhado por Cunningham, “*Technical Debt*”, pode gerar dúvidas ao ser traduzido para o português. A palavra “*debt*” é comumente traduzida para “débito” erroneamente, quando o correto é “dívida”, sendo “*debit*” a palavra correspondente a “débito”.

De acordo com o dicionário Aurélio, porém, tanto “dívida” quanto “débito” possuem como definição “aquilo que se deve”. A palavra “débito” ainda cita “dívida” como sinônimo”. Para este trabalho, no entanto, foi adotada a palavra “dívida” por ser a tradução correta de “*debt*”.

2.2.2 Metodologia de pesquisa

De forma a elaborar uma revisão de literatura para ser utilizada como base teórica para este trabalho, foi escolhida a técnica “*backward snowballing”* [JAL12]. O conjunto de trabalhos inicial foi obtido através da utilização da ferramenta de busca Scopus. O motivo da escolha deu-se por ser essa ferramenta considerada a maior base de dados de *abstract*s e citações de pesquisa de literatura e fontes de qualidade na web [ELS13], além de agregar várias bases de dados importantes tais como ACM, IEEE, Elsevier, Springer, etc.

O critério de pesquisa utilizado foi a busca pelos termos em inglês “*technical debt*” e “*software debt*”, sendo que o último trouxe praticamente a metade dos resultados do primeiro termo além de resultados repetidos, mostrando ser um termo pouco utilizado em relação ao primeiro. Sendo assim, apenas os resultados obtidos com o termo “*technical debt*” foram considerados para esta pesquisa. A utilização do termo em inglês se dá devido ao termo ter sido utilizado pela primeira vez em inglês e por não ter sido encontrado nenhum resultado com os termos traduzidos para o português como “dívida técnica” ou “débito técnico”, sendo portanto o termo em inglês mais adequado para a busca.

Os filtros utilizados por áreas de assunto foram *Physical Sciences*, por conter títulos de fonte em Ciência da Computação, e *Social Sciences & Humanities*, por conter títulos de fontes em áreas relacionadas à Engenharia de Software tais como *Business, Management and Accounting*, *Social Sciences* e *Multidisciplinary*. O critério de ordenação foi por relevância, por ser mais fiel ao termo de busca.

Foram ainda consideradas publicações de 2013 para trás sem limite mínimo de ano de publicação, por ser a dívida técnica um assunto relativamente recente na comunidade de pesquisa. Quanto ao tipo de documento, não foi aplicado nenhum filtro, pois o termo de busca é bastante específico, sendo que este filtro poderia restringir demais os trabalhos pesquisados.

Com base nos parâmetros informados foram encontrados 342 resultados, a partir dos quais foram selecionados os cinco primeiros trabalhos em ordem de relevância utilizando-se a ordenação da própria ferramenta, conforme os filtros mencionados. Estes artigos foram considerados como sendo o conjunto de artigos de partida para a condução da revisão de literatura [JAL12].

A análise dos resultados obtida através da ferramenta evidencia o crescimento recente em atenção que a dívida técnica tem despertado na comunidade científica:

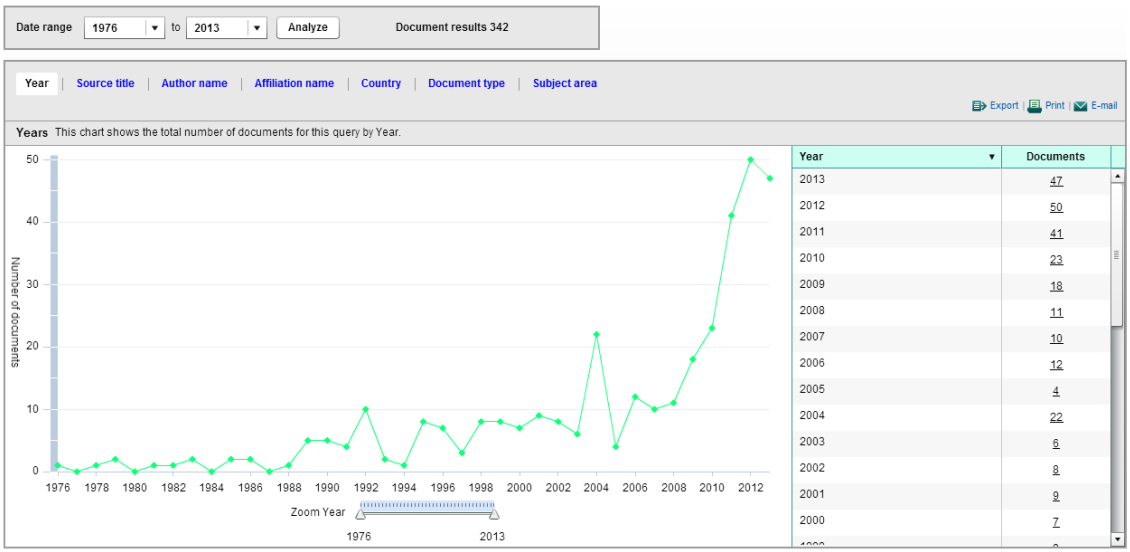


Figura 7 - Quantidade de publicações por ano

Uma vez selecionados os artigos de partida, foi realizada a busca para trás (*backward snowballing*) nas listas de referências desses artigos, gerando assim uma nova lista com 98 referências. Dessa lista foram excluídas referências duplicadas e não acessíveis, tais como links para sites ou blogs inexistentes e artigos não disponíveis na ferramenta de busca. A partir disso, foram analisados todos os títulos e selecionados os que tinham maior relevância e afinidade com o tema da pesquisa. Após a leitura do título, se ainda houvesse dúvidas sobre a relevância do trabalho, lia-se o *abstract* para então decidir se o mesmo deveria ser incluído para posterior análise. Referências de artigos vindos de sites ou blogs cujos links eram acessíveis tiveram que ser lidos por completo por

não haver *abstract*. Feita essa análise, das 98 referências resultaram 52 durante a primeira iteração dentre as quais 32 eram artigos científicos, 15 links para sites ou blogs especializados e 5 links para sites de ferramentas de suporte ao gerenciamento da dívida técnica. Este passo foi repetido e uma nova lista foi gerada com 414 referências, das quais apenas 10 trabalhos foram aproveitados após análise similar à feita na primeira iteração. Destes 10, um era referência a um vídeo e outro a uma ferramenta.

Na segunda etapa da revisão foi efetuada a busca para frente (*forward snowballing*), procurando por citações dos artigos de partida na ferramenta de busca. Nessa etapa foram encontrados 3 artigos, sendo 1 relevante à pesquisa. Na segunda iteração, 12 artigos foram selecionados sendo 10 relevantes.

Na tabela abaixo encontra-se um resumo dos resultados obtidos através da utilização da técnica *snowballing* para revisão de literatura utilizada como fundamentação teórica sobre a dívida técnica neste trabalho.

Tabela 2 - Resumo dos resultados obtidos através do *snowballing*

Fase/Iteração	**Lista inicial**	**Trabalhos Relevantes**	**Referências não acessíveis**
Busca inicial	342	5	0
Backward/1	98	52	22
Backward/2	414	10	4
Forward/1	3	1	0
Forward/2	12	10	0
Total	**869**	**78**	**26**

No Apêndice D desta dissertação encontra-se a lista dos trabalhos relevantes. Para referenciá-los utilizou-se uma numeração para diferenciá-los das outras referências desta dissertação. O gráfico a seguir mostra a divisão das referências relevantes pesquisadas por tipo de fonte.

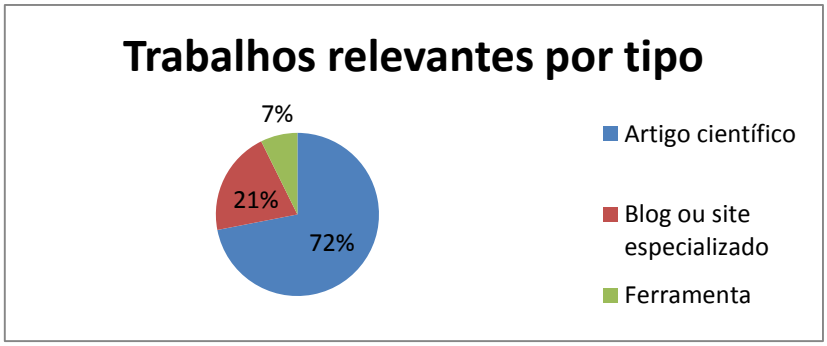


Figura 8 - Trabalhos relevantes por tipo

Podemos observar pelo gráfico que há uma variedade de fontes sobre o assunto e que há bastante referências para blogs e sites especializados mesmo em artigos científicos. Também já é possível identificar ferramentas de mercado que buscam tratar da dívida técnica.

2.2.3 Definições e conceitos de dívida técnica

Dívida técnica (também conhecida como débito técnico ou "*technical debt*") é uma metáfora que se refere às consequências do desenvolvimento de software deficiente. O termo foi introduzido em 1992 por Ward Cunningham [CUN92], onde o autor afirma que:

"Entregar código imaturo é como entrar em dívida. Um pouco de dívida agiliza o desenvolvimento contanto que ela seja paga de volta prontamente com reescrita".

Outas definições incluem "lacuna entre o estado atual do software e algum estado hipotético 'ideal' no qual o sistema é otimamente bem sucedido em um ambiente em particular" [BRO10], "o grau de incompletude" [KLE05], "um *backlog* de problemas técnicos adiados" [TOR11] e "qualquer parte do sistema atual que é considerado sub ótimo de uma perspectiva técnica" [KTA10, TOM13].

Uma revisão de literatura realizada por [TOM11] em 2011, que seguiu as práticas propostas por [KIT04] revelava um "entendimento aparentemente fragmentado por parte de diversos temas díspares e anedóticos ou por outro lado definições em alto nível que fracassaram em descrever o fenômeno exaustivamente ou em detalhe" [TOM13], por se tratar de um conceito relativamente novo e ainda sendo explorado. Em uma pesquisa realizada em 2013, [TOM13] faz uma revisão de literatura "multivocal" [OGA91] seguida de entrevistas com profissionais da área e acadêmicos, buscando definir um framework teórico

que visasse endereçar a natureza da dívida técnica propriamente dita, bem como precedentes e resultados.

2.2.3.1 Taxonomia da dívida técnica

McConnell [MCC07] classifica a dívida técnica em dois grandes grupos:

a) **Não Intencional:** Causada por trabalho de má qualidade, por exemplo, um *design* que se torna suscetível a erro ou um código escrito por um desenvolvedor júnior que escreve código ruim.
b) **Intencional:** Causada por uma decisão consciente feita pela organização com o intuito de otimizar para o presente ao invés do futuro. É subdividida em dois tipos:
    a. **Curto Prazo:** Em geral ocorre reativamente, por questões táticas, por exemplo: atalhos identificáveis individualmente (como financiar um carro) ou vários pequenos atalhos (como dívida de cartão de crédito).
    b. **Longo Prazo:** Em geral ocorre pró ativamente, por motivos estratégicos.

Fowler define um quadrante que consiste em duas dimensões: imprudente/prudente e consciente/inconsciente, conforme ilustra a figura abaixo [FOW09].

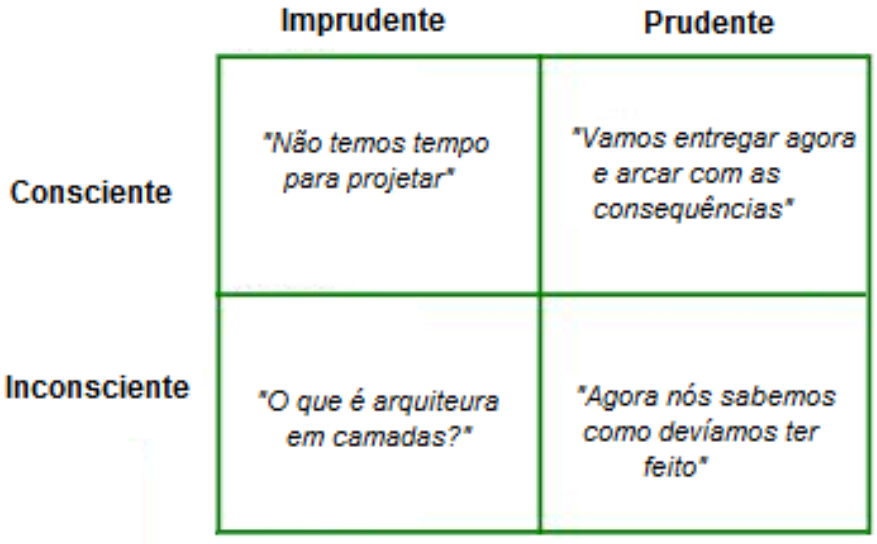


Figura 9 - Quadrante da dívida técnica

Dentro de cada célula há um exemplo do tipo de dívida associada. O quadrante separa problemas que surgem de imprudência das decisões que são feitas estrategicamente.

2.2.3.2 Dimensões da dívida técnica

Em seu trabalho, [TOM13] sintetiza algumas dimensões da dívida técnica, apresentadas a seguir.

- **Dívida de código:** A dívida técnica é frequentemente manifestada através de código mal escrito – “qualquer improviso, contorno, trecho de código ruim constitui dívida técnica” [STO10].
- **Dívida de projeto (ou *design*) e arquitetural:** Atalhos arquiteturais e de *design* constituem outra forma de dívida técnica. Tanto na forma de “*upfront design”*, subestimando qualidades como manutebilidade, adaptabilidade, etc. quanto subsequentes *designs* com a ausência de *refactoring* [SHR10].
- **Dívida de ambiente:** A dívida técnica também se manifesta no ambiente de uma aplicação, o qual inclui processos relacionados a desenvolvimento bem como *hardware*, infraestrutura e aplicações de apoio. Processos manuais, componentes defasados também contribuem para a dívida técnica.
- **Dívida de distribuição de conhecimento e documentação:** A falta de compartilhamento de conhecimento e documentação bem escrita são outro aspecto da dívida técnica. Slinker [SLI08] cita um exemplo: “Um sistema grande com milhões de linhas de código pode ser mantido sem custo. Um fator que mantém o custo baixo é que os desenvolvedores originais permaneçam trabalhando no sistema. Eles sabem por que e como as coisas foram feitas. Portanto a dívida de código é afetada pelos membros do time. Suponhamos que de alguma maneira os membros do time sejam insultados e todos saiam. Repentinamente a [dívida técnica] mudou de pequena para extremamente alta!”.
- **Dívida de teste:** Dívida técnica pode existir na da falta de scripts de testes ou testes de cobertura insuficientes, independentemente se os testes são automatizados ou rodados manualmente. A dívida técnica pode gerar custos associados à imagem quando defeitos críticos afetam os clientes, necessitando implementar correções em ambientes de produção, a necessidade de diagnosticar e corrigir regressões e executar testes manuais quando testes automatizados não estão disponíveis.

2.2.3.3 Atributos da dívida técnica

Quanto aos atributos, [TOM13] os descreve refletindo analogias obtidas entre o fenômeno da dívida técnica e facetas de dívida financeira:

- **Custo monetário:** Dívida técnica não é apenas uma “metáfora” financeira, mas está de fato associada a custo monetário real. O tempo gasto por um desenvolvedor é caro e utilizado ineficientemente quando a velocidade de desenvolvimento é reduzida e desenvolvedores são forçados a corrigir regressões ao invés de desenvolver novas funcionalidades.
- **Anistia:** A anistia da dívida acontece quando a dívida técnica acumulada pode ser pensada como liquidada e não precisa ser reembolsada. Por exemplo, quando um protótipo é desenvolvido ou uma funcionalidade ou produto é julgado não mais necessário (por razões que não sejam dívida técnica, como perda do interesse ou valor pelo cliente) [RIE09].
- **Falência:** Ocorre quando há uma situação em que o juro da dívida é tão esmagador que é necessário parar e reescrever completamente o código. Muitas vezes a sensação é de que o ponto de falência ocorre ainda antes de a reescrita ser necessária, por exemplo, pelo fato de o código ser tão ruim que o desenvolvedor não tem mais interesse em trabalhar nele.
- **Juros e capital:** Uma noção fundamental na dívida técnica é o pagamento de juros. No desenvolvimento de software, pagar o capital é uma questão de completar uma tarefa adiada ou implementar uma substituição correta para um atalho. Para Cunningham [CUN12], *refactoring* é uma forma de quitar o capital emprestado e atrasar o desenvolvimento devido à complexidade é como pagar juros. Mais amplamente, pagamentos de juros de dívida técnica estão associados a seus impactos na moral, produtividade e qualidade.
- **Alavancagem:** A dívida técnica é frequentemente usada para alavancagem e pode aumentar a produtividade no curto prazo. Isso permite às equipes criar atalhos ou adiar trabalho em troca de “tempo emprestado”.
- **Reembolso e saque:** Aspectos de reembolso e saque da dívida técnica podem ser caracterizados por “centenas de milhares” de pequenos saques, quando comparado a compras feitas com cartão de crédito [MCC07]. Exemplos deste tipo de dívida são variáveis com nomes genéricos, comentários escassos, não seguir convenções de código, etc. [MCC07].

2.2.4 Precedentes da dívida técnica

Os principais fatores que contribuem para o aumento da dívida técnica podem ser categorizados em inúmeros precedentes mais abrangentes, tais como [TOM13]:

- **Pragmatismo:** Muitas vezes é necessário abdicar de princípios puros de projeto e simplesmente “cuspir” código em detrimento de necessidades de negócio, o que tem como consequência a necessidade de se refatorar inúmeras instâncias de código mal escrito.
- **Priorização:** Ocorre quando os desenvolvedores e times decidem priorizar suas tarefas, fazendo com que funções críticas sejam geralmente priorizadas em relação à qualidade geral. Os motivos principais são tempo, orçamento e restrições de recursos.
- **Processos:** Neste precedente, a dívida técnica é gerada tanto no caso de processos que possuem pouca comunicação e colaboração quanto na falta de processos que desencorajariam os desenvolvedores a criá-la. A revisão de código é um exemplo de processo que garante que a dívida técnica seja identificada e tratada com antecedência.
- **Atitudes:** Algumas atitudes comuns como hesitar melhorar o código por medo de introduzir novos problemas em geral tendem a contribuir para a dívida técnica e podem ser descritos como uma apatia em relação à dívida técnica. Além disso, assumir riscos pode impactar tanto positivamente quanto negativamente no curto prazo. Sendo assim, baixo apetite para riscos pode influenciar decisões que criam dívida técnica para reduzir o risco de entrega do projeto, enquanto que alto risco pode influenciar decisões que criam dívida técnica sem considerar o aumento do risco do projeto.
- **Ignorância e descuido:** Apesar de o descuido ser uma forma de ignorância, no qual os desenvolvedores não sabem do erro que está criando a dívida técnica, a ignorância aqui se refere a como evitar a dívida técnica, ao invés da ignorância da sua presença. A ignorância pode referir-se à inabilidade dos desenvolvedores em escrever um código de qualidade, enquanto que o descuido pode originar-se ao fazer algo sem considerar o impacto da mudança ou o que será construído com base nela.

2.2.5 Consequências da dívida técnica

De acordo com os dados coletados em sua pesquisa [TOM13], o acréscimo da dívida técnica tem quatro consequências principais: moral do time, produtividade, qualidade do produto e risco do projeto.

- **Impacto na moral:** A dívida técnica, além de seu impacto econômico, impacta também psicologicamente afetando a moral do time. Permitir que a dívida técnica se acumule é benéfico à moral do time no curto prazo, pois tarefas que previnem ou pagam a dívida técnica são por vezes consideradas “não muito divertidas”. Já no longo prazo tem um forte impacto negativo quando os desenvolvedores são forçados a pagar essa dívida.
- **Impacto na produtividade:** O principal benefício da dívida técnica no curto prazo é um aumento na velocidade, permitindo que os desenvolvedores possam usufruir de atalhos, adiamentos e outras escolhas. Porém, se não for pago em tempo hábil, o pagamento de juros requeridos da dívida técnica diminui a velocidade de desenvolvimento no longo prazo de várias maneiras, tais como: a) dificultando a manutenção da base de código devido ao impacto na qualidade estrutural, e b) desenvolvedores gastando tempo gerenciando e trabalhando em problemas decorrentes da dívida técnica.
- **Impacto na qualidade:** A dívida técnica tem impacto negativo na qualidade do produto em termos de defeitos e outros problemas de qualidade estrutural. Código mal escrito ou decisões de projeto sub-ótimas que não consideram todas as classes facilmente geram defeitos que são ainda mais difíceis de serem encontrados quando há falta de testes ou cobertura de teste. Atributos de qualidade afetados por decisões que introduzem dívida técnica mais frequentemente comprometidos são os atributos estruturais de extensibilidade, escalabilidade, manutebilidade, adaptabilidade, performance e usabilidade.
- **Impacto no risco:** Dívida técnica inadvertida, ou seja, não reconhecida ou não visível, bem como dívida incremental são difíceis de controlar, sendo difíceis de estimar até para os desenvolvedores mais experientes, tendo como consequência atrasos nas entregas.

A figura a seguir resume o framework proposto por [TOM13]:

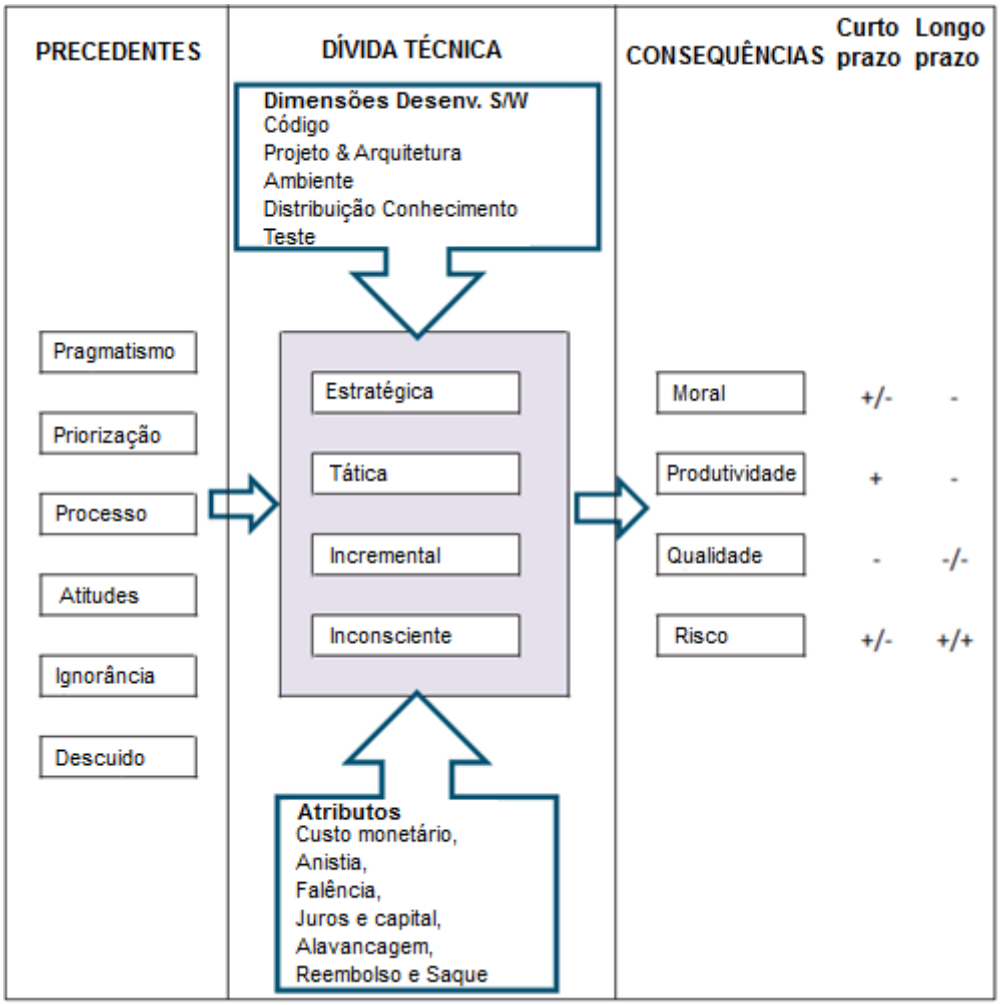


Figura 10 - Framework teórico da dívida técnica

O estudo ainda revelou diferentes tipos de dívida técnica baseado nos dados coletados, que foram classificados nos seguintes grupos: estratégica, tática, incremental e inconsciente.

2.2.6 Quantificando a dívida técnica

Assim como a dívida financeira, a dívida técnica incorre em pagamento de juros. Muito tem se falado na comunidade ágil que a dívida técnica deve ser paga o quanto antes, mas times de desenvolvimento falham em estimar o custo monetário da dívida, o qual pode dar informações valiosas [HAZ10].

Uma vez tendo o valor monetário associado à dívida técnica, é possível responder várias questões complexas sobre o software. Existem inúmeros benefícios em se quantificar monetariamente a dívida técnica, dentre eles [HAZ10]:

1. Diz aos times quando é hora de parar de adicionar funcionalidades para começar a refatorar.
2. Os clientes tem uma ideia do risco associado ao software.
3. Facilita decisões de investidores de risco.
4. Determina o quão acessível é desenvolver e manter o software à medida que ele evolui no seu ciclo de vida.
5. Ajuda na decisão entre refatorar e reescrever.
6. Ajuda a definir limites de crédito, ajudando os envolvidos a tomarem decisões informadas.

[GOF10] tenta quantificar a dívida técnica através de um exemplo simples:

"Assumindo que X horas por semana são gastas em esforço de codificação que paga a dívida técnica existente. Fazendo uma analogia a cartões de crédito, assuma que metade do pagamento vai para o valor gasto e metade para os juros. Para simplificar, assuma uma taxa de juros de 10% e um programador que custa US$ 2000,00 por semana. A quantificação se dá da seguinte maneira: Supondo que X sejam oito horas, ou seja, o programador gasta um dia por semana refatorando código legado, pagando a dívida técnica. Metade disso são quatro horas, ou em torno de US$ 200,00/semana em juros. Sendo quatro semanas por mês isso dá US$ 800,00/mês em juros, ou em torno de US$ 10.000,00 de juros ao ano. Neste exemplo a taxa de juros é de 10%, o que implica em um capital de US$ 100.000,00. Isso significa que levaria um ano inteiro para pagar toda a dívida técnica. Isto é, um ano inteiro sem criar nenhuma nova funcionalidade." Se este modelo estiver correto, então a dívida técnica é um problema muito maior do que imaginamos [GOF10].

Outra maneira de calcular a dívida técnica é encontrada no *plugin* para o SonarQube [SON13a], que é uma plataforma de código aberto para gerenciamento de qualidade de software. O custo é calculado primeiro descobrindo-se o valor da dívida técnica, que consiste em [SON13b]:

- Dívida (em homens/dia) = custo_de_corrigir_duplicacoes + custo_de_corrigir_violacoes + custo_de_comentar_API_publica + custo_de_corrigir_complexidade_nao_coberta + custo_de_trazer_complexidade_abaixo_do_limite + custo_de_cortar_ciclos_em_nivel_de_pacote

Este cálculo leva em conta um custo de desenvolvedor por dia de US$ 500,00 trabalhando oito horas por dia. Partindo-se de um custo definido em horas para cada uma das violações, pode-se fazer uma análise monetária para cada uma delas para então obter a dívida técnica total [HAZ10].

2.2.7 Comunicando e gerenciando a dívida técnica

O software está em constante evolução, e mais do que isso, ele se deteriora ao longo do tempo [TAE13]. Jim Highsmith usa o gráfico a seguir para ilustrar o efeito da dívida técnica no custo da mudança e capacidade de resposta aos clientes.

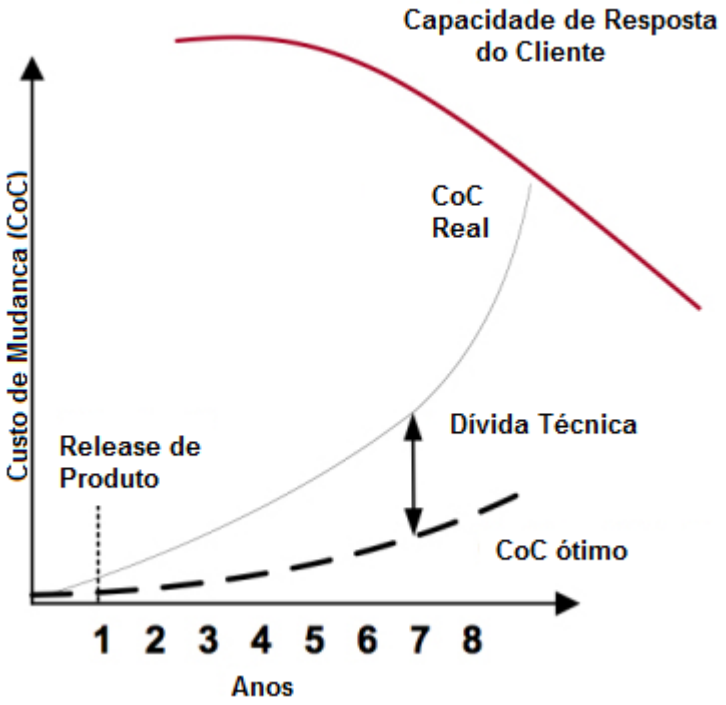


Figura 11 - Custo da mudança na evolução do software

O principal desafio para uma equipe em gerenciar a dívida técnica está em comunicá-la a equipes de negócio e ajudá-los a compreender as implicações da dívida previamente introduzida [MCC07]. A dívida técnica é uma metáfora importante na comunicação tanto entre a equipe técnica quanto entre desenvolvedores e executivos. Caso não seja devidamente gerenciada, a dívida traz problemas significantes no longo prazo, tais como custos de manutenção elevados [BRO10].

É necessário, para que se faça entender a dívida técnica por parte do negócio, estabelecer um vocabulário financeiro, de modo que se tenha uma comunicação mais

transparente por parte de equipes técnicas. Apesar de ainda não utilizada amplamente, a terminologia da dívida técnica soa imediatamente a um executivo [MCC07].

Algumas sugestões para comunicar sobre a dívida técnica para pessoal de negócio [MCC07]:

a) Usar o orçamento de manutenção de uma organização como um comparativo grosseiro para a carga de serviço de dívida técnica.
b) Discutir a dívida em termos de dinheiro ao invés de em termos de funcionalidades. Por exemplo, “40% de nosso orçamento de Pesquisa e Desenvolvimento vai para dar suporte a *releases* anteriores”.
c) Tenha certeza que está assumindo o tipo certo de dívida técnica. O problema está em diferenciar dívida resultante de boas decisões de negócio da dívida resultante de más práticas técnicas ou comunicação errônea sobre qual tipo de dívida o negócio pretende assumir.

Em projetos de desenvolvimento de software, normalmente a dívida técnica é gerenciada implicitamente. Com o objetivo de determinar quando a quantidade de juros evitado justifica o custo de pagar o capital da dívida, o próximo passo é gerenciá-la explicitamente. Isso possui os seguintes benefícios em potencial, especialmente para trabalho de manutenção [SEA11a]:

- **Custos de manutenção reduzidos:** Evita-se pagamento de juros e trabalho de “aperfeiçoamento” desnecessário.
- **Aumento na produtividade de manutenção:** Permite uma melhor priorização das tarefas em cada *release* e a manutenção é feita sempre em código fácil de trabalhar.
- **Evitar surpresas:** Menos componentes falham sem aviso, menos tarefas inesperadamente grandes de manutenção acima do orçamento e melhor estimativa de custos e riscos de se postergar tarefas de manutenção.

O framework a seguir resume os passos necessários para se gerenciar a dívida técnica [SEA11a].

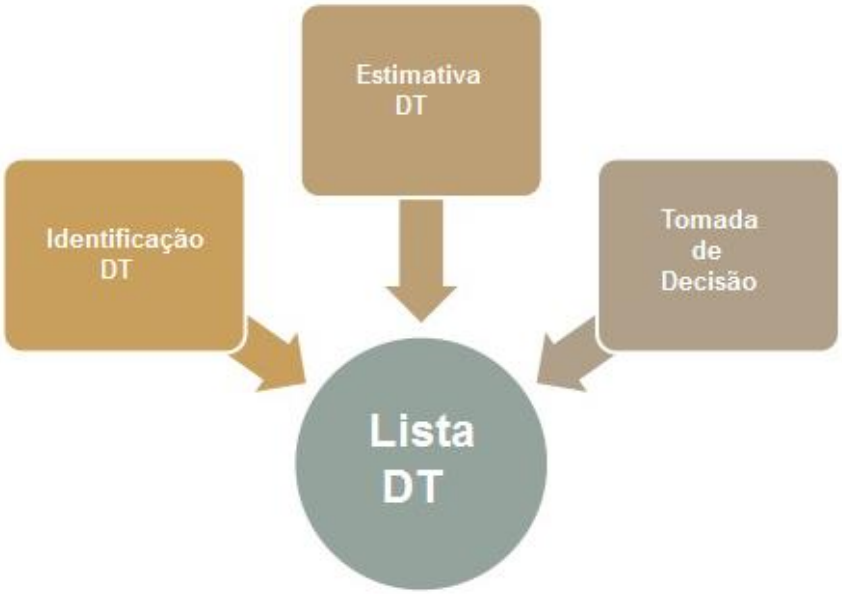


Figura 12 - Framework de gerenciamento da dívida técnica (DT)

Identificar e estimar a dívida técnica envolve listar tarefas relacionadas à dívida em um *backlog* comum durante a *release* planejamento da iteração, assim como outras "coisas a fazer" [KRU12]. Philippe Kruchten [KRU08] declara que é provável que haja quatro cores no *backlog* que correspondem a possíveis melhorias (Figura 10). As tarefas a serem executadas no futuro para agregar valor, como adicionar novas funcionalidades (em verde) ou investir em arquitetura (em amarelo), e reduzir os efeitos negativos dos defeitos (vermelho) e dívida técnica (preto) [KRU12].

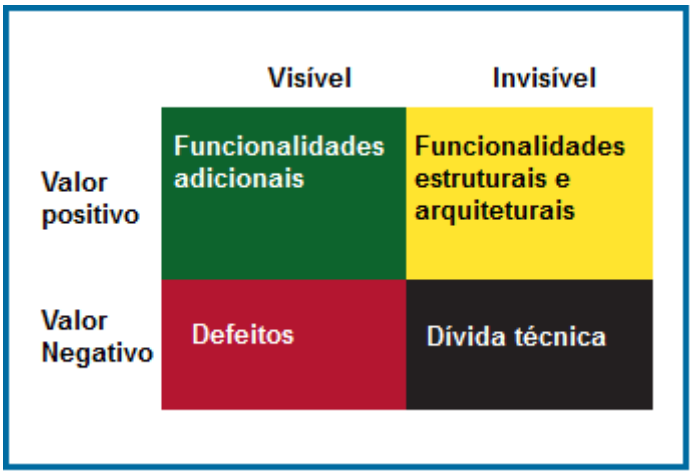


Figura 13 - Backlog técnico

A tabela a seguir (Tabela 2) ilustra as vantagens e desvantagens ao se gerenciar a dívida técnica através de um *backlog* técnico [ORH13]:

Tabela 3 - Vantagens e desvantagens do backlog técnico

Vantagens	Desvantagens
A dívida é visível e clara a todos	Vários tipos ou “cores” no *backlog,* difícil de priorizar
O custo por tarefa é rastreável	O cliente pode não entender os reais benefícios das tarefas técnicas
Separação entre tarefas técnicas e de funcionalidade	Mudanças que envolvem um alto custo devem sempre ter uma justificativa de negócio

Outras formas de se gerenciar o pagamento da dívida técnica são [ORH13]:

- **Pagamento de dívida em iteração mínima:** Por exemplo, usar meio dia de uma semana para pagar a dívida.
- **Tarefas de *buffer*:** Podem ser usadas para *refactoring*, por exemplo, durante 10% do tempo do tempo disponível.
- **Releases de limpeza:** *Releases* puramente técnicas utilizadas de tempos em tempos para melhorar a base de código.

2.3 Desafios e Oportunidades na Pesquisa de Dívida Técnica

Esta seção visa apresentar um resumo dos principais desafios e oportunidades que a dívida técnica traz para a pesquisa científica à medida que a mesma é um assunto de crescente atenção tanto na comunidade de desenvolvimento de software quanto na comunidade científica [BRO10]. Para tanto, foram analisados os principais trabalhos relacionados à dívida técnica, publicados em artigos, conferências e periódicos importantes na área de Engenharia de Software a fim de se ter uma visão de potenciais oportunidades de pesquisa a partir dos desafios constatados nos trabalhos.

2.3.1 Desafios de Pesquisa em Dívida Técnica

Os trabalhos utilizados como base para esta análise sugerem inúmeros desafios e oportunidades de pesquisa. Os temas foram mapeados e agrupados de acordo com a seguinte classificação obtida segundo os termos mencionados nos trabalhos, os quais serão detalhados posteriormente:

- Carência de estudos empíricos

    - Evidências empíricas
    - Verificações empíricas
- Visualização e comunicação
    - Comunicação da dívida técnica
    - Visualização e análise
    - Tornar a dívida técnica explícita (visibilidade)
    - Rastreamento e gerenciamento
    - Benefícios do gerenciamento explícito da dívida técnica
- Suporte à tomada de decisão
    - Prover *feedback* em decisões em se aceitar ou pagar a dívida técnica
    - Análise de riscos sobre se assumir alguns tipos de dívida
    - Técnicas de medição para cada tipo de dívida técnica, combinadas com uma forma útil para a tomada de decisão
    - Entender como a dívida é inserida, vista e resolvida
- Gerenciamento e monitoramento
    - Rastrear decisões não relacionadas a código durante o ciclo de vida do projeto
    - Artefatos não relacionados a código
    - Monitorar e gerenciar a dívida técnica na arquitetura
    - Conhecer as diferentes fontes da dívida
    - Coletar informações sobre a dívida técnica empiricamente
    - Evolução futura
- Ferramental de apoio
    - Identificar automaticamente a dívida em um projeto de desenvolvimento de software
    - Monitorar o nível da dívida técnica ao longo do tempo, reconhecer tendências e disseminar avisos em momentos apropriados
    - Integrar tudo isso no ambiente de desenvolvimento e gerenciamento através de ferramental adequado

2.3.2 Oportunidades de Pesquisa em Dívida Técnica

O trabalho *An Exploration of Technical Debt* [TOM13] aponta a necessidade dos seguintes estudos empíricos investigando a dívida técnica: Uma validação e teste mais rigorosos do framework proposto, qualificar as associações para diferentes formas de dívida técnica e procurar estabelecer métricas para quantificar a dívida técnica e seus impactos.

Também foram sugeridos estudos empíricos para validar heurísticas e técnicas que irão ajudar profissionais em seu gerenciamento da dívida técnica.

Em *An Enterprise Perspective on Technical Debt* [KLI11], os autores sugerem que o rastreamento de decisões arquiteturais e outras decisões é um passo necessário para poder avaliar a dívida técnica e traçar desde decisões de negócio até implicações arquiteturais para entender o impacto da mudança. Além disso, a falta de métodos comuns para quantificação e/ou monetização da dívida torna difícil garantir que todos os envolvidos entendam completamente as ramificações da decisão de adquirir a dívida. Para que as organizações consigam prosperar e gerenciar propriamente sua dívida técnica, é necessário não só uma solução comum para comunicar sobre a dívida técnica, mas também mecanismos para prover *feedback* sobre decisões de se aceitar ou pagar esta dívida.

O estudo ainda destaca quatro questões para futuros estudos. Primeiro, aplicar conceitos de alavancagem de investimento e/ou teoria das opções à análise da dívida técnica. Segundo, devido à diversidade dos *stakeholders* que necessitam estar envolvidos no gerenciamento da dívida técnica e à falta de suporte à decisão para esta atividade, há uma oportunidade de se aplicar conceitos da ciência de decisão ao processo de gerenciamento da dívida técnica. Terceiro, a dívida técnica é normalmente invisível ou impressionista, especialmente para pessoal não técnico. Encontrar maneiras de quantificar a dívida técnica de uma forma significativa para *stakeholders* específicos é essencial. E por último, foi destacada a importância da ocorrência do débito não intencional, surgida devido às forças dinâmicas que estão além do controle da equipe técnica e também da organização.

Em *Managing Technical Debt – An Industrial Case Study* [COD13], o autor aponta que há necessidade de trabalhos futuros que avaliem os riscos de se assumir certos tipos de dívida, tanto no curto quanto no longo prazo. Também é sugerida uma estratégia para gerenciar a dívida técnica apontada como sendo efetiva, onde a mesma consiste em se ter times dedicados cujo objetivo é a redução da dívida. Tais times ainda usariam em torno de 20% do *Potentially Shippable Increment* (PSI) para focar na redução da dívida. O autor ainda afirma que “enquanto a priorização da dívida técnica é em grande parte influenciada pelo seu impacto no cliente e severidade da dívida, estudos adicionais são necessários para se conseguir determinar quais dívidas são mais críticas no longo prazo”.

O artigo *Technical Debt in Test Automation* [WESL12] enfatiza a necessidade de estudos sobre a dívida técnica fazendo referência a [BRO10], o qual sugere estudos de

campo para determinar a relativa importância das diferentes fontes da dívida, quais têm maior impacto e como ela é tratada. Além disso, foi relatado que não foram encontradas diretrizes claras e detalhadas de como projetar, implementar e manter um sistema de execução automática de testes de forma a se manter a dívida técnica acumulada a um nível aceitável. Os autores concluíram, a partir dos estudos de caso citados no artigo na área de testes e automação de testes, que há uma real necessidade por parte dos profissionais de desenvolvimento de software e teste de software de tais diretrizes.

Os autores de *An Empirical Model of Technical Debt and Interest* [NUG11], o qual propõe quantificar a dívida e juros da dívida técnica baseado em um método empírico de avaliação desenvolvido pelo *Software Improvement Group* (SIG), afirmam que há necessidade de uma validação formal do modelo proposto no artigo utilizando dados empíricos. Tal validação pode ser feita para avaliar a precisão do modelo de estimativa de custos.

No trabalho *A Portfolio Approach to Technical Debt* [GUO11], não ficou evidente a performance da abordagem de gerenciamento da dívida técnica por portfolio. Portanto, existe a necessidade de testar esta abordagem em ambientes reais através de um estudo de caso em projetos em andamento. Neste estudo de caso seriam coletados dados usados como *baseline* que posteriormente poderiam ser comparados, tais como esforço, custo e produtividade do projeto. A abordagem proposta seria então aplicada e comparada em termos de custo-benefício em relação aos dados de *baseline*. Usando este tipo de avaliação seria possível identificar os benefícios do gerenciamento explícito da dívida técnica.

Em *Using Technical Debt Data in Decision Making* [SEA11b], os autores propõem estudos de caso profundos para cada modelo de decisão proposto no artigo, de forma a compreender os pontos fortes e fracos de cada um no contexto do gerenciamento da dívida técnica. Tais estudos de caso devem ser projetados amplamente de forma a estudar o gerenciamento da dívida técnica em geral, desde sua identificação, passando pela medição até a tomada de decisão. Como consequência, os estudos de caso não só levarão ao entendimento da adequação da estratégia de decisão utilizada, mas também compreender o que torna a tomada de decisão sobre a dívida técnica em particular diferente de outros tipos de tomada de decisão relacionadas à manutenção de software.

Após compreender o processo de tomada de decisão em detalhe, estudos poderiam ser feitos para comparar as estratégias de decisão lado a lado. Estudos comparativos industriais seriam mais difíceis, pois geralmente não é possível repetir um cenário de decisão de duas formas diferentes sem um efeito de aprendizado. No entanto, repetidos estudos de caso em contextos industriais seriam úteis para iterativamente refinar e

personalizar as abordagens, bem como entender qual delas funciona melhor em diferentes contextos.

Em *Prioritizing Design Debt Investment Opportunities* [ZAZ11a], é sugerido que pesquisas futuras avaliem apropriadamente a abordagem de utilizar a análise custo/benefício para priorização do trabalho de redução da dívida técnica ranqueando o valor e juros da dívida de *design* causada por "*god classes*". A abordagem de ranqueamento está sujeita a múltiplas extensões, tanto se pode estender a outros "*code smells*" quanto adicionar, substituir e pesar diferentes características de qualidade para o ranking de qualidade.

São necessárias avaliações empíricas futuras de forma a se ter evidências convincentes que *god classes* com piores métricas requerem um maior esforço de *refactoring*. É recomendado que estudos empíricos futuros investiguem se o ranking é consistente com o esforço medido real das atividades de *refactoring*. Ainda, é preciso validar a escolha das características de qualidade e seu impacto no processo de ranqueamento. Estudos empíricos precisam ser conduzidos para confirmar que o ranking de impacto previsto se correlaciona com a experiência de especialistas em software. Este estudo irá ajudar a evoluir a abordagem de ranqueamento para modelos de predição que ajudam a estimar o valor absoluto e custo de *refactoring*.

Por último, mais dados qualitativos precisam ser coletados, por exemplo, através de entrevistas com os desenvolvedores para compreender por que certas *god classes* têm um maior impacto na qualidade do que outras e quais ganhos de curto prazo, se é que existem, foram obtidos criando uma *god class* em primeiro lugar.

No artigo *Managing Technical Debt in Software Reliant Systems* [BRO10], os autores referem-se à falta de pesquisas que elucidem pontos fortes e fracos da metáfora da dívida técnica e que desenvolvam as definições necessárias para modelagem quantitativa e gerenciamento da dívida técnica.

Os autores relatam que o conceito conforme desenvolvido até o momento (da escrita do artigo) deixa muitas questões em aberto:

- A "dívida" é uma metáfora prudente para gerenciar investimentos oportunos e remediativos em projetos de software? Caso não seja, existe uma metáfora estreitamente relacionada que seja melhor?
- A metáfora da dívida pode levar a teorias prováveis sobre como utilizar, medir e pagar atalhos de software?
- Como se pode identificar a dívida em um projeto de desenvolvimento de software e produto, quem sabe, automaticamente?

- Quais são os tipos de dívida? Quais técnicas podem ajudar os projetos a elicitar, comunicar, analisar e gerenciar a dívida?
- Como a dívida técnica está relacionada com manutenção de software?
- Como informações sobre a dívida técnica podem ser coletadas empiricamente para o desenvolvimento de modelos conceituais?
- Como a dívida técnica pode ser visualizada e analisada?

No *workshop* em que o artigo foi baseado, diversos problemas de pesquisa foram levantados pelos participantes. Também os autores mencionam que há espaço para contribuição de pesquisadores com conhecimento em várias áreas da engenharia de software. Os problemas foram organizados por assunto:

Oportunidades de Refactoring: Há dificuldade em se avaliar quantitativamente o impacto de *refactorings* sugeridos a partir de análises de detecção de *code smells* que encontram sintomas de *design* ruim.

Problemas arquiteturais: Atualmente, a pesquisa foca apenas em como fazer análise de custo/benefício baseada em requisitos arquiteturais significativos. No entanto, uma vez que as decisões são feitas, normalmente elas não são monitoradas no decorrer da vida do projeto e não são relacionadas a artefatos de código, resultando em dívida técnica no nível arquitetural. Monitorar e gerenciar a dívida técnica na arquitetura proporcionaria análises antecipadamente no ciclo de desenvolvimento de forma a manter o projeto sob controle.

Identificando fontes dominantes de dívida técnica: Seria útil saber, em um determinado contexto, quais fontes de dívida técnica são mais numerosas, custosas ou úteis. Estudos de campo da prática de desenvolvimento e elicitação a partir de profissionais especialistas são direções promissoras de pesquisa para revelar a importância relativa das diferentes fontes. Uma questão relacionada a isso é entender a dinâmica de como a dívida é inserida, visualizada e resolvida. O papel do pesquisador está em desenvolver esta metáfora, baseado em exames empíricos minuciosos das práticas atuais de desenvolvimento.

Problemas de medição: A medição da dívida técnica é considerada uma tarefa difícil, portanto há várias áreas de pesquisa sobre este assunto. Por exemplo, a quantidade de códigos “clone”, o número de *To Be Defined* (TBD) nos documentos de requisitos, etc. Porém, essas medidas isoladas precisam ser combinadas com uma forma útil para a tomada de decisão.

Artefatos não relacionados a código: Existem questões de pesquisa relacionadas a artefatos que não código-fonte, em especial artefatos de *design*, teste e documento de requisitos. Pode haver fontes de dívida técnica em quaisquer desses artefatos. Pesquisa sobre a viabilidade de caracterização da dívida técnica, ou encontrar outras maneiras de caracterizá-la, por exemplo, utilizando outras partes da metáfora financeira, como estratégia de investimento no contexto de suporte à tomada de decisão é um problema de pesquisa.

Monitoramento: Há a necessidade de métodos para determinar o nível de dívida técnica ao longo do tempo, reconhecendo tendências, e disseminando avisos nos momentos apropriados. Tal monitoramento precisa ser integrado no ambiente de desenvolvimento e gerenciamento através de ferramental adequado.

Questões de processo: A dívida técnica possui custos econômicos significativos. Porém, atualmente faltam formas rigorosas de quantificar os valores presentes destes custos. O gerenciamento efetivo da dívida técnica demanda uma base racional que permita fazer decisões de investimento. São necessárias pesquisas para possibilitar a valoração do déficit criado pela dívida técnica e de projetos criados para remediá-lo. Tais técnicas, por sua vez, requerem maneiras de tornar a dívida técnica explícita e estão sujeitas a rastreamento e gerenciamento dentro de um dado processo de desenvolvimento.

Em *In Search of a Metric for Managing Architectural Technical Debt* [NOR12], os autores fazem uma revisão dos trabalhos relacionados baseados em três aspectos: (1) fundamentos da dívida técnica e abordagens para gerenciá-la, (2) métricas para guiar *refactoring* e processo de re-arquitetura, e (3) suporte ferramental que pode proporcionar um aumento na visibilidade, agilidade e informação oportuna para gerenciar a dívida técnica efetivamente.

Fundamentos da dívida técnica: Um experimento com arquitetos da IBM concluiu que a habilidade de avaliar a dívida técnica é importante, porém ainda existem lacunas importantes e significativas na demonstração dessa habilidade.

Métricas para guiar *refactoring* e processo de re-arquitetura: A análise de defeitos, medição da qualidade de código e trabalhos relacionados olham para a análise de artefatos de código após o fato, quando o sistema é entregue ou quando está perto de ser entregue. Enquanto esse tipo de análise provê dicas sobre redução de defeitos ao longo do tempo, ela não provê orientação para ajustar um curso de ação enquanto o sistema está sendo desenvolvido, ou para reconhecer a dívida enquanto ela acumula ao invés de ser

pego de surpresa mais tarde. Ainda, é difícil reivindicar legitimamente que o sistema possa ser refatorado para incluir uma solução chave de *design* após o fato, sem um *redesign* significativo.

Suporte ferramental: Análise estática de código e *plug-ins* para suporte à análise da dívida técnica começaram a ganhar atenção na área de ferramentas por sua promessa de prover auxílio nos princípios da dívida. Enquanto ferramentas como Sonar, Lattix e SonarGraph focam atualmente apenas em análise de código e fornecem uma análise geral uniforme da dívida, sua crescente atração é devido à promessa de automatizar o processo e melhorar a visibilidade do sistema durante o desenvolvimento.

A habilidade de elicitar e melhorar a visibilidade do estado do projeto é uma área de crescente pesquisa e interesse prático, tanto de uma perspectiva de gerenciamento de projeto quanto de qualidade.

O artigo *Investigating the Impact of Design Debt on Software Quality* [ZAZ11b] propõe que mais sintomas de dívida de *design* além dos mencionados no artigo sejam estudados, com o intuito de validar que estes tipos de indicadores são bons previsores de dívida técnica. Também, visto que os dados do estudo não indicaram uma correlação entre probabilidade de defeitos e tamanho de classe, futuros trabalhos devem considerar esta constatação e relatar sobre coeficientes de correlação antes de normalizar os dados.

Em *A Case study on Effectively Identifying Technical Debt* [ZAZ13], estudos futuros devem considerar incluir ou propor ferramentas para outros tipos de artefatos de desenvolvimento afetados pela dívida técnica.

Os autores pensam que a metáfora da dívida técnica tem potencial para ir além do mecanismo para comunicação, a ser traduzido em um conjunto de ferramentas para medir e gerenciar a dívida técnica. O estudo mostrou que diferentes *stakeholders* possuem um entendimento diferente de dívida no projeto, indicando que a dívida técnica deve incluir uma gama de membros de um time de projeto.

O estudo levantou mas não mostrou, segundo os autores, várias questões importantes e interessantes que são recomendadas a futuros pesquisadores:

- O quanto da dívida técnica potencial identificada por ferramentas e não por desenvolvedores é considerada dívida técnica “real”? Isto é, há valor em se utilizar ferramentas para identificar a dívida técnica que desenvolvedores não estão cientes?
- Como a identificação manual da dívida técnica pode ser melhor integrada no processo de desenvolvimento com o intuito de torná-lo mais eficiente e viável?

- Qual a maneira de identificação de dívida técnica mais provável de encontrar dívida técnica “real”, através de ferramentas ou de desenvolvedores?

O trabalho *Practical Considerations, Challenges and Requirements of Tool Support for Managing Technical Debt* [FAL13] descreve algumas dificuldades práticas que os autores encontraram em prover estimativas de problemas de dívida técnica. Estas foram classificadas como a seguir:

A. Capital
    1. Inadequação de valores únicos
B. Juros
    1. Distância do domínio econômico
    2. Efeito da interação
    3. Não linearidade e limites
    4. Juros múltiplos
    5. Indisponibilidade de dados históricos
    6. Probabilidade de ocorrência de juros como probabilidade de eventos
C. Não técnicos
    1. Rotatividade
    2. Restrições organizacionais
    3. Tradição
    4. Cultura de sentimentos pessoais
    5. Questões políticas
    6. Conflito de objetivos

Foram ainda elencados 10 requisitos em alto nível para uma ferramenta de gerenciamento da dívida técnica, sendo eles:

1. Gerenciar capital, juros e *time-to-market*
2. Traduzir decisões em consequências econômicas
3. Gerenciar incertezas de maneira rigorosa
4. Gerenciar a evolução de consequências econômicas
5. Balancear rigor e facilidade de uso através de escalabilidade
6. Completude e integração
7. Balancear opiniões de *experts* e estimativas automatizadas
8. Análise “e se” como interpretação de possíveis distribuições
9. Análise de sensitividade
10. Análise de cenário

Ainda é necessário muito esforço para que sejam proporcionadas ferramentas confiáveis para o gerenciamento da dívida técnica aos profissionais de desenvolvimento de software. Os requisitos listados podem ser utilizados como um início para esta agenda de pesquisa.

Os autores de *Investigating technical debt folklore: Shedding some light on technical debt opinion* [SPI13] constatam que há uma abundância de pronunciamentos sobre dívida técnica na internet que não possuem uma avaliação, as quais podem levar à emergência de um folclore. A popularidade da discussão sobre o assunto pode ser observada no gráfico a seguir reportado em trends.google.com. Entre 2007 e 2012 houve uma busca crescente pelo termo *"Technical Debt"* por parte dos usuários do Google.

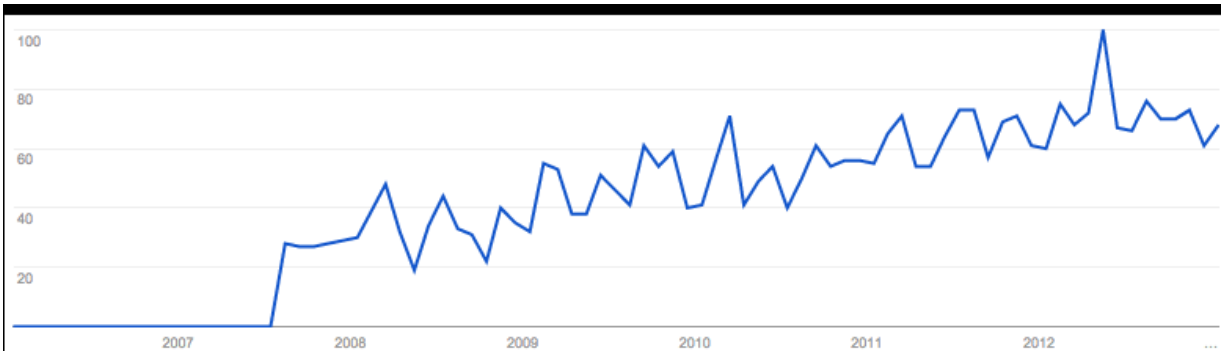


Figura 14 - Volume relativo de busca por "Technical Debt" de 2007 a 2012 em Google.com

No mesmo estudo, foram identificadas afirmações sobre folclore de dívida técnica e criada uma lista com estas afirmações que posteriormente foram avaliadas para saber se as opiniões convergem (concordando ou discordando) ou são mistas (o que significa que o termo ainda precisa ser refinado).

Os resultados apontam algumas evidências e motivação para explorar as seguintes questões na pesquisa em dívida técnica:

- Métodos e ferramentas para encontrar dívida técnica não intencional e, portanto escondida no código-fonte e em outros artefatos;
- Métodos e técnicas para gerenciar e rastrear a dívida técnica;
- Investigação do ponto ótimo entre um nível aceitável e saudável de dívida, e um nível que se aproxima do perigoso;
- A relação entre dívida técnica e a moral e motivação do time;
- Explorar a diferença em causa e efeito entre dívida técnica intencional e não intencional.

Em *On the limits of the technical debt metaphor: some guidance on going beyond* [SCH13a], é ressaltada a importância de focar em evolução futura, algo frequentemente negligenciado na pesquisa atual em dívida técnica. Além disso, é necessário diferenciar entre dívida técnica potencial e efetiva, aceitando que não existem sistemas livres de dívida.

Finalmente, é necessário tomar muito cuidado com abordagens de gerenciamento da dívida quanto à quantificação da dívida. O motivo é que o método de estimativa tem um grande impacto no resultado, podendo não representar a dívida em questão.

Os autores de *A Formal Approach to Technical Debt Decision Making* [SCH13b] veem potencial para trabalhos futuros baseados na abordagem apresentada. Em particular, estudos de caso onde diferentes aproximações poderiam ser avaliadas. Resultados interessantes poderiam ser obtidos em se estendendo as métricas para além do custo de desenvolvimento como benefício do cliente, custos de treinamento, *time-to-market* e critérios similares.

Eles afirmam que o gerenciamento da dívida técnica é muito difícil de se validar objetivamente pois seria necessário comparar o desenvolvimento com uma certa técnica de gerenciamento com uma situação sem a utilização da técnica.

Em *Towards a Model for Optimizing Technical Debt in Software Products* [RAM13], são observadas quatro linhas de pesquisa inter-relacionadas na literatura emergente de dívida técnica:

1. Taxonomia, *frameworks* e perspectivas em dívida técnica;
2. Identificação, medição e visualização da dívida técnica;
3. Medição e avaliação do impacto da dívida técnica no projeto e na *performance* da organização;
4. *Frameworks* de decisão para gerenciamento da dívida técnica.

O valor em se aceitar ou evitar a dívida técnica precisa ser cuidadosamente considerado em relação à vida útil do produto, oportunidades de inovações tecnológicas e habilidade de derivar benefícios de negócio a partir das funcionalidades do produto.

3 MÉTODO DE PESQUISA

Este capítulo apresenta o método de pesquisa utilizado neste estudo. A seção 3.1 apresenta o desenho de pesquisa e suas etapas. Na seção 3.2 são detalhados os aspectos metodológicos do estudo.

Esta pesquisa se caracteriza pela realização de um estudo empírico sobre o gerenciamento da dívida técnica em projetos de desenvolvimento de software. A abordagem da pesquisa foi do tipo exploratória envolvendo estudos primários e secundários. A natureza empírica do estudo é justificada pelo fato de a dívida técnica ser um fator presente apenas em contextos reais de práticas da indústria e com forte intervenção humana.

Os métodos de pesquisas utilizados foram: revisão de literatura utilizando a técnica *Snowballing* [JAL12] e estudo de campo. A revisão de literatura (estudo secundário) buscou identificar trabalhos relacionados e oportunidades de pesquisa, além de incluir um levantamento de ferramentas de apoio à identificação e gestão da dívida técnica. O estudo de campo (estudo primário) permitiu obter informações de como a dívida técnica é tratada em projetos reais por meio de entrevistas, buscando identificar necessidades, dificuldades e práticas existentes na realidade das equipes de desenvolvimento de software, de forma a complementar o estudo secundário.

3.1 Desenho de pesquisa

O conhecimento em Ciência da Computação pode ser obtido usando metodologias de pesquisa quantitativas e qualitativas, coletivamente chamadas de pesquisa empírica [WAI07]. Desta forma, para alcançar os objetivos gerais e específicos propostos anteriormente e de forma a se obter evidências empíricas que possam ser úteis na pesquisa em dívida técnica, este trabalho foi conduzido em duas fases, conforme detalhado a seguir e ilustrado na figura 15:

- **Fase 1 - Aprofundar o estudo do referencial teórico, análise de ferramentas e estudo de campo:** o referencial teórico serviu para aprofundar os conhecimentos em Engenharia de Software, mais especificamente em processos de desenvolvimento de software e qualidade estrutural de software. Além disso, foi feita uma revisão de literatura sobre a dívida técnica, utilizando a técnica *Backward Snowballing* [JAL12], de modo a compreender o significado dessa metáfora,

destacando desafios e oportunidades de pesquisa nessa área conforme descrito no capítulo 2. Ainda nesta fase, a partir da revisão de literatura sobre dívida técnica, foi possível mapear seis ferramentas focadas em análise geral da dívida técnica. Cinco dessas ferramentas são de uso comercial e uma desenvolvida como parte de um projeto acadêmico (DebtFlag), são elas:

- CAST Application Intelligence Platform (AIP)
- SonarQube
- Lattix
- IBM Rational Team Concert
- DebtFlag
- Sonargraph

No Apêndice C são detalhadas as características das ferramentas bem como pontos fortes e pontos fracos de cada uma, com base na documentação disponível *online*. Foram utilizados alguns critérios tais como tipo de dívida a qual a ferramenta se propõe a gerenciar, funcionalidades, pontos fortes e fracos, etc. Também foram levados em conta alguns aspectos que dizem respeito à identificação, visualização, quantificação e monitoramento da dívida. Um estudo de campo [MCG94] também foi planejado e executado nesta fase, gerando assim uma proposta de extensão do Scrum para gestão da dívida técnica, detalhada no capítulo 5.

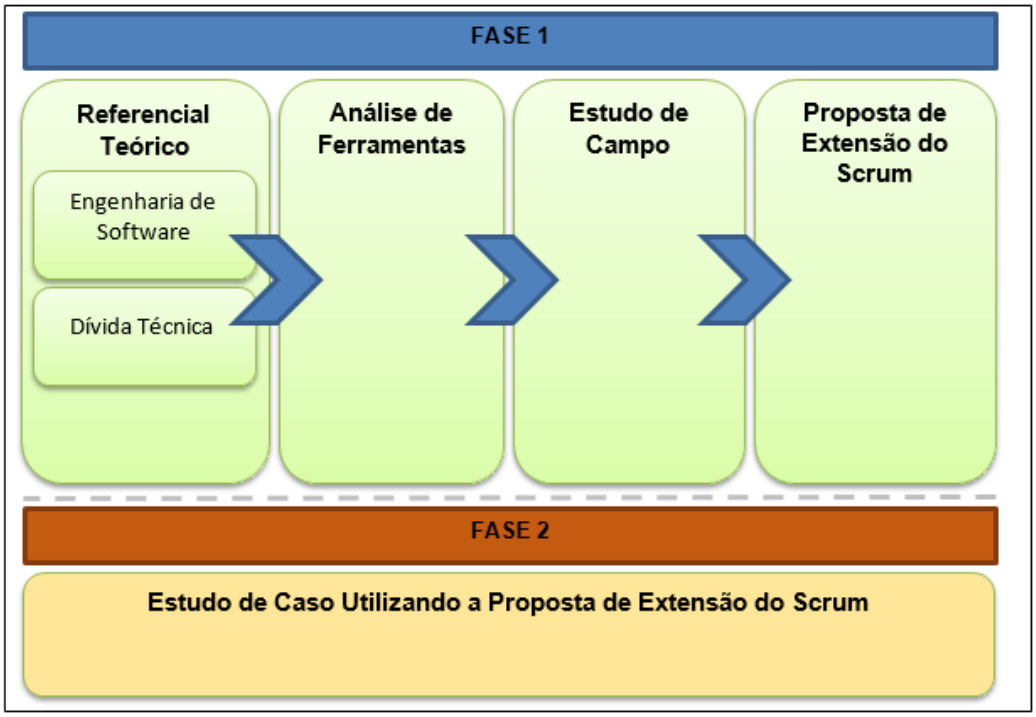


Figura 15 - Desenho de pesquisa

- **Fase 2 - Estudo de Caso:** O estudo de caso foi planejado para entender como uma equipe de desenvolvimento integra o gerenciamento da dívida técnica em um projeto real utilizando a extensão do Scrum proposta na primeira fase. Essa equipe seria treinada nos conceitos de dívida técnica e na proposta em si. Após, seria monitorada durante algumas iterações, onde alguns dados seriam coletados, tais como duração de atividades relacionadas ao gerenciamento da dívida técnica, ferramentas utilizadas, decisões tomadas e eventuais dificuldades encontradas. Além disso, entrevistas buscariam coletar dados relativos, mas não restritos a: problemas encontrados, sugestões de melhoria e verificar se tomadores de decisão gostariam de continuar gerenciando explicitamente a dívida técnica com a extensão proposta. Para tanto, seriam feitas entrevistas com os profissionais após a realização do estudo de caso.

Apesar de o estudo ter sido inicialmente planejado para ser executado incluindo o estudo de caso da fase 2, esta fase acabou sendo modificada. A razão principal é o fato de que o estudo de campo tomou mais tempo do que o planejado devido a vários fatores, dentre eles a disponibilidade dos entrevistados e o cronograma agressivo para o estudo como um todo. Neste sentido, a fase dois foi impactada devido à falta de tempo e recursos necessários para ser executada com qualidade, podendo servir de base para um estudo futuro. Entretanto, seguindo a sugestão indicada pela banca durante o Seminário de Andamento, caso houvesse de fato restrição de tempo para a condução do estudo de caso, conduziu-se uma avaliação da proposta de extensão do Scrum com cinco especialistas.

3.2 Aspectos metodológicos

Para assegurar a confiabilidade deste estudo, um processo de pesquisa foi planejado. Nele, a utilização de protocolos para desenvolvimento e formalização dos estudos executados tiveram por objetivo sistematizar as tarefas de análise, de forma a aumentar a confiabilidade da pesquisa. O estudo de campo, além de um protocolo formal, passou por uma validação de face e conteúdo e um pré-teste com o objetivo de garantir a integridade dos resultados. Os protocolos para o estudo de campo e avaliação da proposta de extensão do Scrum encontram-se nos Apêndices A e B desta dissertação, respectivamente.

O viés de todo o processo de pesquisa foi minimizado com a avaliação das etapas de pesquisa com um pesquisador experiente (o professor orientador). Na revisão de

literatura, a avaliação ocorreu principalmente na definição dos artigos de partida e na análise dos resultados. No estudo de campo, o protocolo da coleta dos dados e a análise dos resultados também contaram com esta avaliação.

4 ESTUDO DE CAMPO

Este capítulo apresenta o estudo de campo desenvolvido. Na seção 4.1 é apresentada a base metodológica do estudo de campo, na seção 4.2 são descritos os resultados do estudo e uma análise do Scrum sob a perspectiva da dívida técnica mapeada através do estudo de campo é mostrada na seção 4.3.

4.1 Base metodológica do estudo de campo

Em um estudo de campo o pesquisador se propõe a fazer observações diretas de sistemas “naturais” em curso, à medida que interfere ou perturba esses sistemas o mínimo possível [MCG94]. O critério para utilização de tal método é manter o grau de realismo do contexto ao máximo.

O estudo de campo foi desenvolvido com profissionais de diversas organizações que possuíam experiência com gerenciamento de projetos de desenvolvimento de software e Scrum. O estudo buscou estudar como a dívida técnica é percebida e gerenciada em projetos de desenvolvimento de software na indústria. A partir deste estudo também foi possível elaborar uma abordagem preliminar para gerenciamento da dívida técnica como uma extensão do *framework* Scrum.

4.1.1 Seleção das organizações e unidades de análise

A unidade de análise do estudo foi definida como sendo profissionais de gerenciamento de projetos de desenvolvimento de software em empresas de TI nacionais ou multinacionais sediadas no Brasil. Assim sendo, foram escolhidos, por conveniência, profissionais que já tiveram experiência ou lidam com aspectos da dívida técnica em seu dia-a-dia, e possuíam algum grau de conhecimento sobre o Scrum.

Os profissionais selecionados pertenciam a onze organizações diferentes as quais colaboraram no processo de coleta de dados através de entrevistas. A seção 4.3 apresenta detalhadamente os resultados obtidos neste estudo.

4.1.2 Fonte dos dados e seleção dos participantes

A coleta de dados foi constituída por fontes primárias. As fontes primárias foram constituídas de entrevistas, sendo realizadas quinze entrevistas semiestruturadas

individuais em profundidade. Partiu-se de um roteiro básico com questões formuladas aos entrevistados e adequadas conforme seu desenvolvimento.

O critério inicial para a definição dos entrevistados centrou-se na unidade de análise e nos objetivos do estudo. Neste sentido, a população envolvida constituía-se de profissionais que possuíam o perfil de gerentes de projetos de desenvolvimento de software ou que já haviam tido alguma experiência neste sentido. O instrumento de coleta de dados consistiu de um roteiro para entrevista semiestruturada (Apêndice A). As entrevistas foram organizadas de forma a estudar como a dívida técnica é percebida e gerenciada em projetos de desenvolvimento de software na indústria.

4.1.3 Análise de dados

A técnica utilizada na análise de dados foi a análise qualitativa [OAT06]. Desta forma, todas as entrevistas foram gravadas, transcritas e analisadas posteriormente. Com o objetivo de familiarizar o pesquisador com os dados coletados antes de iniciar a codificação dos mesmos, realizou-se uma leitura detalhada das transcrições. Para cada uma das questões foram identificadas categorias preliminares, para as quais os dados foram codificados. Este processo foi desenvolvido independentemente pelo pesquisador e após consolidado com o orientador, definindo um conjunto definitivo de categorias a serem consideradas.

4.1.4 Fases e operacionalização do estudo de campo

A pesquisa foi desenvolvida com quinze profissionais de onze diferentes organizações. Os profissionais foram indicados tanto pelo professor orientador quanto pelo próprio pesquisador através de contatos profissionais. Após entrar em contato com cada um dos indicados, todos os participantes dispuseram cerca de uma hora de tempo para as entrevistas que ocorreram tanto de forma presencial quanto remota (via S*kype* e *Hangouts*), de acordo com a localização e disponibilidade de horários dos participantes. Ao todo foram nove entrevistas presenciais, quatro através da ferramenta *Skype*, uma através da ferramenta *Hangouts* utilizando vídeo e uma via *Hangouts* utilizando *chat* pois era a forma que o entrevistado dispunha devido a restrições nos horários disponíveis em sua rotina.

Como instrumento de coleta foi utilizado um questionário semiestruturado. Este foi desenvolvido tomando-se por base um roteiro inicial de questões, a partir dos resultados da revisão de literatura executada e representada pelo protocolo de pesquisa desenvolvido para o estudo de campo (Apêndice A).

A validação de face e conteúdo do protocolo do estudo de campo foi realizada por um doutorando. A partir disso foi executado um pré-teste, com um mestrando da PUCRS. Após sua aplicação foi possível identificar melhorias e adaptações às perguntas do protocolo de modo a adequar-se ao objetivo da pesquisa. Foram realizadas algumas iterações até gerar uma versão estável do roteiro.

Após a execução das entrevistas foi feita uma análise qualitativa dos dados coletados no estudo de campo. A análise de conteúdo envolveu a preparação dos dados com a transcrição das entrevistas, a identificação e refinamento de categorias representando tópicos significativos em função das quais o conteúdo foi classificado e, por fim, a análise propriamente dita.

4.2 Caracterização dos respondentes

As entrevistas foram realizadas com quinze profissionais previamente selecionados de acordo com sua experiência com gerenciamento de projetos, independentemente do método de desenvolvimento. Não foi exigida experiência com gerenciamento da dívida técnica como pré-requisito, visto que é um conceito relativamente novo, porém presente em projetos de desenvolvimento de software.

Em relação à experiência em desenvolvimento de software, todos possuíam pelo menos 5 anos sendo o tempo médio de 14,8 anos. Quanto ao tempo de experiência com gerenciamento de projetos (papel de gerente de projetos ou Scrum Master), foi registrada uma média de 8,4 anos sendo 1,5 anos o menor tempo informado.

Sobre a presença das empresas em que os profissionais atuam, 60% é multinacional e 40% nacional. Quanto à característica dos projetos em que trabalham, 60% é de desenvolvimento, 20% de manutenção e os outros 20% de características diversas tais como pesquisa e desenvolvimento e implantação de métodos ágeis. O método de desenvolvimento mais utilizado pelos entrevistados é o ágil com 86,7%, sendo que destes, 69,2% utilizam Scrum.

4.3 Resultados do estudo de campo

A realização da análise qualitativa do estudo de campo permitiu traduzir a realidade estudada e seu impacto nos objetivos desta pesquisa. A seguir são apresentados os elementos analisados e as categorias obtidas.

4.3.1 Ferramentas

As questões desta categoria buscaram identificar as ferramentas utilizadas para a identificação e monitoramento da dívida técnica nos projetos. Foram citadas principalmente ferramentas de análise estática automatizada tais como Sonar, FindBugs, Checkstyle e Coverity. Outras formas de identificação também foram citadas, tais como a utilização de ferramentas colaborativas como wiki para compartilhamento de conhecimento e relatórios de ferramentas de integração contínua como o Jenkins. Também foi citada a ferramenta Fortify para identificação de brechas de segurança. Além das ferramentas, um dos entrevistados cita que além de análise estática, a identificação da dívida técnica se dá de forma não automatizada, ou seja, por interação humana:

> *"Detectado tanto por ferramentas como Sonar quanto pelos próprios desenvolvedores nas interações deles."*

Outro entrevistado cita a utilização de Kanban para expor a dívida técnica, além de ferramentas de análise estática e marcações no próprio código feitas na IDE Eclipse:

> *"Bom... nós mapeamos os riscos desde o desenho do roadmap de produto nas iterações, os* ***itens identificados como débito técnico são expostos no kanban*** *(tem uma classe de serviço específica para isso), entrando como item planejado na próxima iteração. Do ponto de vista técnico,* ***usamos integração contínua (Jenkins) e Sonar com o plugin de débito técnico, além de outras ferramentas que ajudam a identificar possíveis problemas (PMD, FindBugs, Checkstyle)****. E, sempre que identificado mesmo que em um peer review,* ***marcamos o código com FIXME no Eclipse****. No <empresa>, fazemos periodicamente um* ***workshop técnico*** *para discutir débito técnico e boas práticas, ontem mesmo fizemos um* ***dojo****. Sobre o PMD: é uma ferramenta de análise estática que permite identificar violações (más práticas) de programação, que pode derivar em débito técnico."*

Quanto a ferramentas de monitoramento, ou seja, que permitam acompanhar o nível de dívida técnica no projeto ao longo do tempo, grande parte dos entrevistados afirma utilizar os próprios dados de *issues* de análise estática como parâmetro do nível da dívida. Dois dos entrevistados citaram a palavra "imaturo" por não possuírem uma forma de fazer este tipo de acompanhamento em seus projetos:

> *"Nada específico, somos bem imaturos neste processo."*

Outro entrevistado cita:

*"... A gente **não tem um controle proativo** de saber que hoje nós tínhamos 20% do nosso backlog de dívida técnica no nosso backlog e agora estamos com 40% e está aumentando, nós não temos esse controle. **É bem imaturo** isso ainda."*

Um dos entrevistados menciona o *"feeling"* como a forma utilizada de monitorar a dívida técnica, por não haver uma ferramenta que concentre toda a informação sobre a dívida técnica do projeto:

*"A gente tem mais ou menos um **feeling** sobre isso. A gente não tem uma ferramenta onde a gente possa jogar todos nossos issues de análise estática + defeitos + enhancements +refactorings na mesma ferramenta e ver como a gente está indo."*

Há ainda um entrevistado que afirmou utilizar os dados de dívida técnica gerados pelo Sonar como métrica de projeto:

*"A gente utiliza **análise estática**, tem **build integrado**, roda o **Sonar e gera um cálculo de quanto tem de débito técnico**. A gente tem, digamos um débito técnico de 5000 no início do projeto. No final do projeto, a gente mede de novo e vai dar 5500, então aumentou. Então a gente tem uma métrica de projeto que é assim, tu podes aumentar o débito técnico até, digamos, 10 dias a cada 100 horas de projeto. Se tu aumentares mais do que isso tu estás prejudicando o projeto, aí desconta do resultado do teu projeto. **Não é só prazo, qualidade e bugs, mas também débito técnico**."*

Outras formas de controle da dívida mencionados incluem análise de defeitos como densidade de defeitos por componente e ferramentas para gerenciamento de riscos. As principais formas de identificação mapeadas através do estudo de campo podem ser agrupadas conforme descrito a seguir:

Tabela 4 - Formas de identificação da DT

Automática	Humana
Sonar	Interação entre desenvolvedores
Findbugs	Feeling
Coverity	Kanban
PMD	Marcações no código
Checkstyle	Análise de defeitos
Bullseye	Análise de riscos
Pylint	

Jenkins Fortify	Wiki Atas de reuniões “Xerife” do código

4.3.2 Qualidade

Esta categoria contém questões que visam identificar os processos utilizados para qualificação do código bem como rastreamento de decisões arquiteturais no projeto.

Revisão por pares, testes (unitários, funcionais, de regressão, de integração e automatizados), análise de segurança e análise de cobertura foram os principais processos ou técnicas utilizadas para qualificar o código. Também, dependendo da metodologia utilizada, são feitas revisões de documentos de requisitos e documentos técnicos. Um dos entrevistados, que trabalha com métodos ágeis, relatou:

> *“Seria o done. A gente tem **código desenvolvido em pares, code review** feito por uma pessoa que não a que desenvolveu, tem que estar **livre de defeitos críticos** (aquele que me impede de executar alguma funcionalidade ou algum trecho de código) e para ser considerado done ele **tem que necessariamente estar no nosso ambiente de CI (Continuous Integration) com build funcionando.**”*

Quanto ao rastreamento de decisões arquiteturais, grande parte dos entrevistados relatou não possuir uma forma de fazê-lo de maneira fácil. Alguns afirmaram possuir documentos técnicos, matriz de rastreabilidade e alguma documentação em wiki, porém esse conhecimento nem sempre é mantido atualizado e ainda fica concentrado em grande parte com as pessoas, conforme descrito abaixo:

> *“Não, não tem essa rastreabilidade. O que tem são os dois arquitetos que mais estão envolvidos, tudo acaba passando por eles, uma equipe de arquitetura, mas que **o conhecimento está centralizado em uma pessoa.**”*

Em outra entrevista, a mesma situação é apontada:

> *“Não que eu saiba. Eu diria não facilmente. **O conhecimento fica mais com determinadas pessoas.**”*

Também outro entrevistado expõe evidências da concentração de conhecimento nas pessoas e a importância de que o time possua um bom conhecimento do software no qual está trabalhando:

“Nos últimos projetos não houve nenhuma maneira formal de fazer isso aí. Isso é atingido de uma maneira informal através da ***discussão do design ou da arquitetura entre o time****. Aí tu contas com que o time tenha uma boa visão de como está estruturado o software, para garantir que quando tu mexes em algo tu sabes onde pode afetar.”*

Nesta entrevista também é evidente a comunicação informal:

“Tem uma equipe de arquitetura separada. Lá na fase de análise do projeto a arquitetura analisa e vê se tem que mexer algo na arquitetura. Se tiver eles já especificam como fazer, já vem decidido, por exemplo, vai ser um módulo do JBoss, etc. ***Mas a comunicação é só por e-mail, bem informal.****”*

A seguir são resumidos os principais processos de qualidade utilizados nos projetos dos entrevistados:

- Revisão de código
- Testes (unitários, funcionais, de regressão, de integração e automatizados)
- Análise de cobertura
- Revisão de documentos de requisitos
- Revisão de documentos técnicos

4.3.3 Custos

A categoria de custos visa entender como são estimados os custos monetários da dívida técnica potencial e efetiva.

Em geral, os entrevistados disseram não fazer este tipo de estimativa tanto para dívida em potencial quanto efetiva, seja por não ter acesso a informações financeiras do projeto ou por não conhecer técnicas ou ferramentas que possibilitem realizar este tipo de cálculo. Novamente a palavra “maturidade” foi utilizada de forma a justificar o motivo de não o fazerem. Abaixo é relatada esta situação, além disso o entrevistado questiona a validade de se utilizar esta estimativa:

“Não, nunca fiz isso. ***É um ROI que eu desconheço****, até deve ter, mas em todas as empresas que trabalhei nunca vi calcular. Envolve maturidade, tecnologia, diversos fatores que gastar energia para chegar nesse número... É um número inócuo. Tu chegaste nele e vais fazer o que com ele? A estrutura funcional, seja ela uma decisão*

da empresa para fazer um refatoramento para aumentar a manutebilidade são decisões muito subjetivas. Tentar transformar isso em números é muito questionável."

Um entrevistado que trabalha no governo afirma ser ainda mais difícil obter este tipo de informação:

"Não tem nada. Se colocar na área de governo isso ainda é quase uma utopia. A não ser ***subcontratações de fábrica de software****, por exemplo, onde aí fica mais evidente a questão do custo, mas* ***tirando isso o custo não é tratado****. É uma variável como outra qualquer. É hora e pronto. Em cronograma os caras usam dias e não horas."*

Já dois dos entrevistados atribuem à falta de ferramentas o fato de não haver uma forma de calcular o custo monetário da dívida. De acordo com o entrevistado 10:

"Nós ***não temos uma maneira fácil de calcular isso*** *porque a gente não usa uma ferramenta como falei anteriormente para monitorar tudo isso. Mas tem como estimar isso em pontos, enfim, todo o teu backlog de débito e tentar colocar isso no papel para ver quantos sprints vai dar, enfim, vezes o custo por FTE. Dá pra ter uma ideia de custo disso, mas não é fácil."*

O entrevistado 14 afirma:

"... Mas ***não tenho visto uma ferramenta ajudando nisso de uma maneira mais proativa****, de novo, de ter algo que tu consigas registrar de maneira que tu consigas ver de uma maneira mais clara ao término do projeto, e sim de uma maneira mais manual do gerente de projetos fazendo isso. Em resumo, a questão do custo geralmente é absorvido pelo projeto e aí vai gerar um prejuízo ou não, ou vou estar dentro da margem de contingência do projeto para cobrir isso. Não vejo o não pagamento da dívida, normalmente se faz mas se fica devendo no projeto ou diminuindo o lucro ou, se realmente já está dentro do fundo de contingência acaba sendo absorvido pelo projeto e talvez até já esteja previsto com orçamento. Mas te diria que* ***a gestão efetiva de custos eu não tenho visto acontecer.****"*

O entrevistado 2 traz um caso em que a dívida técnica gerou custos devido à dívida de código e de distribuição de conhecimento. Tais custos são tanto mensuráveis, como relativo a horas de desenvolvimento, como não mensuráveis, relacionados à imagem da organização e credibilidade do produto:

“Estamos vivenciando o problema no <aplicação>, existe um algoritmo que foi projetado para fazer o ranqueamento e ordenamento das tarefas, cálculo de burnout e este algoritmo ficou meio falho. Não foi feita uma escala para a quantidade de clientes atual, e este desenvolvedor saiu da empresa. Existe vários problemas neste algoritmo. O custo é uma equipe inteira voltada para resolver estes problemas de performance. O custo está em horas de desenvolvimento, suporte e prejuízo em relação à marca e à credibilidade do produto em si, também alguns contratos que não são cumpridos ou licenças que acabam gerando descontos no contrato e que portanto são facilmente mensuráveis. O contrato de X, acaba sendo dado desconto de 20%, 30% porque o produto acaba não entregando aquilo que era necessário. É bem mensurável pois estamos falando de licenças. É um case emblemático, acho que caracteriza bem isso.”

É difícil estimar o custo monetário da dívida, muitas vezes são feitas estimativas levando em consideração as horas de desenvolvimento. Itens possíveis de calcular o custo incluem:

- Horas de desenvolvimento
- Subcontratações
- Licenças do software

Existem custos não mensuráveis, tais como:

- Imagem da marca ou da organização
- Credibilidade do produto

Empecilhos ao cálculo do custo monetário da dívida:

- Dados financeiros sobre o projeto inacessíveis
- Falta de ferramentas adequadas
- Imaturidade da equipe ou dos processos da organização
- Subjetividade relacionada às atividades inerentes ao desenvolvimento de software
- Dívida técnica não é considerada na previsão de custos do projeto

4.3.4 Tempo

Esta categoria busca entender a forma como a dívida técnica é estimada e as estratégias utilizadas para o seu pagamento.

As respostas para esta categoria variaram bastante devido às peculiaridades de cada projeto, como tipo de organização e método de desenvolvimento utilizado. Os entrevistados 2 e 3 também relataram casos em que projetos novos foram criados inteiramente para pagamento da dívida. Outros entrevistados citaram a criação de defeitos, estórias de melhoria e subcontratação. O entrevistado 10 disse utilizar uma reserva de 10% a 20% do time para eventual pagamento de dívida:

> *"O ideal na minha opinião é **tentar evitar ao máximo que se crie débito técnico**. Porque, depende muito dos projetos, mas a realidade dos nossos projetos aqui com essa BU ela não é muito de priorizar o pagamento dos débitos técnicos. Então a gente tenta primeiro evitar ao máximo e se não der a gente tenta manter **em torno de 10 a 20% dos sprints focados** para que a gente não deixe crescer muito esse débito técnico. Para que a gente sempre tenha um débito digamos assim fair para se pagar naquele período de transição da estabilização do produto. Se a gente chegar cegado lá sem olhar para isso durante os Sprints de desenvolvimento a gente não vai pagar esse débito, historicamente."*

Em geral nos projetos que utilizam métodos ágeis a dívida técnica entra como itens normais de *backlog*, planejados e priorizados. Para projetos tradicionais, foi constatado que geralmente a dívida fica para ser paga na fase de estabilização na forma de *bug fixing*, às vezes por uma equipe de suporte, ou então através de um projeto de manutenção, conforme relatado pelo entrevistado 4:

> *"Vamos supor que entreguei algo agora, esperei um mês, tu dizes que não tem nada de errado, não surgiu mais nada. Aí eu fechei o projeto e não me envolvo mais com aquilo lá. **Vai cair para outra equipe, uma equipe de suporte por exemplo** para resolver. Se tiver muito defeito vai ter que ser aberto outro projeto. Algumas aplicações grandes tem todo ano um "release tático" ou um projeto tático que vai durar o ano inteiro para fazer isso. Pegar defeitos que surgiram ad-hoc e as pessoas vão corrigir. Mas seriam outro tipo de projeto, um projeto de manutenção. Mas os meus, depois que acabou é L3, suporte."*

Outros fatores observados que influenciam na estimativa da dívida foram o tamanho da dívida, a forma de contratação da equipe e negociação junto ao cliente. Isso é evidenciado pelo entrevistado 7:

*“São ligadas em **argumentação**. Se o time não tiver argumentação junto ao cliente quem é o que paga a conta, se isso não estiver no contrato, um percentual da carga horária, um contrato desse tipo nessa linha... assim como já tive equipes negociando, o que é muito comum, a introdução de estórias técnicas para refatoração e melhoria. Mas não existe mágica, **é argumentação, negociação**. A não ser que, também já tive, por contrato tu tens uma carga horária destinada à melhoria contínua do teu software. Isso é uma reserva que se usa para manter o software sob controle.”*

Há indícios de que os fatores abaixo influenciam na estimativa e pagamento da dívida:

1. Método de desenvolvimento
    a. Tradicional
    b. Ágil
2. Estrutura organizacional
    a. Equipe de TI interna
    b. Consultoria
    c. Fábrica de software
3. Forma de contratação
    a. Prevê melhoria contínua ou não
4. Tamanho da dívida
    a. Não pode exceder um determinado limiar (10 a 20% do time, por exemplo)
5. Tipo de cliente
    a. Interno
    b. Externo

4.3.5 Comunicações

Esta categoria busca verificar a utilização do termo “dívida técnica” para comunicação entre os *stakeholders*. Tanto comunicação interna entre o time quanto comunicação do time com o cliente, seja ele interno ou externo.

Praticamente metade dos entrevistados afirmou utilizar o termo ou variações do mesmo, como “débito técnico”. Dos que não utilizam o termo para se referir a situações em que existe dívida técnica, alguns afirmaram que o time sabe do que se trata mas fica implícito na comunicação. Três dos quinze entrevistados afirmam não utilizar o termo com

o cliente por temer certa resistência, ou uma espécie de preconceito pelo fato de ser um nome com conotação negativa. O entrevistado 12 relata esta impressão:

> *“Quando tu falas com um cara de business em débito, pra ele é coluna de despesa. Ao invés de estar ganhando eu estou perdendo.* ***É uma coisa ruim, definitivamente a mensagem que tu passas é que é uma coisa ruim****, que tu não querias ter que fazer. Ele vai querer que tu faças feature, mais funcionalidade, mais funcionalidade.”*

Em outro caso, o entrevistado expõe as diferentes situações em que a dívida pode transparecer para o cliente. Neste caso, a reputação da equipe é evidenciada, considerando que o caso se tratava de uma fábrica de software:

> *“Eu diria que depende da complexidade. Se for algo pequeno e que vá até talvez dar uma resposta de má reputação da fábrica de software para o cliente, o que eu tenho visto é a equipe de projeto absorvendo isso e tentando não transparecer. Agora, se é uma situação que de fato precise ser compartilhada com o cliente até porque pode ser que tenha parte da culpa dele ou até parte de um requisito priorizado sem uma análise de impacto feita corretamente em virtude de uma restrição de tempo por parte do cliente, aí sim eu vejo isso sendo comunicado.”*

4.3.6 Riscos

Esta categoria visa explorar os riscos envolvidos no impacto e tomada de decisão relacionados à dívida técnica. Foram exemplificados alguns tipos mais comuns de dívida, tais como dívida de design/arquitetural, dívida de ambiente, de distribuição de conhecimento/documentação e de teste e então perguntado quais tem mais risco se não gerenciados, na visão do entrevistado. Dívida de design/arquitetura e distribuição de conhecimento foram colocados como as mais críticas em relação ao risco. Outras dívidas como teste, segurança e requisitos também foram destacadas como importantes. O entrevistado 3 descreve uma situação em que a dívida de design resultou na reescrita da aplicação:

> *“Vou destacar duas: a* ***dívida de design/arquitetura e documentação/conhecimento****. Arquitetura porque nós vivenciamos claramente isso, as aplicações que foram construídas há 5, 6, 10 anos atrás foram criadas para resolver um problema específico e não pensaram no crescimento de número de usuários, não pensaram na performance da aplicação. Hoje o custo disso foi reescrever a aplicação. Então é muito alto o custo para pensar um pouco melhor no design lá atrás e daqui a pouco pensar numa*

arquitetura distribuída, alguma coisa do tipo. Então o time técnico, o IT como um todo, todo o meu time praticamente, eles vão levantar esse ponto como o principal da dívida técnica. Código, defeito de código, normalmente é resolvido dentro do projeto e eles descartam. É uma dívida que é consumida rápida. Ela é paga rápida. Design não, muitas vezes ele te força a reescrever uma aplicação do zero porque tu usaste uma arquitetura que já está defasada. E a outra que eu considero importante na minha opinião, não só minha mas do time como um todo é o conhecimento técnico. Como tu vais fazer o knowledge sharing dessa nova aplicação? Desses requisitos de negócio? O que mais pega na <empresa> não é o teu conhecimento em tecnologia, conhecer especificamente o Weblogic, o framework XYZ mas sim as regras de negócio do business. O cara que conhece as regras de negócio end-to-end agrega muito mais valor. E ele tem, claro uma bagagem de conhecimento técnico, ele conhece os frameworks, etc. Se ele sai, normalmente a gente trabalha com wiki pages dentro do nosso sharepoint, a gente usa solução Microsoft pra tudo. Essa é uma dívida bem cara, quando esse cara move ou sai da empresa."

Quanto à tomada de decisão, a grande maioria relatou que o time é quem decide se aceita incorrer dívida ou não por questões estratégicas. Dois dos entrevistados que atuam em projetos tradicionais afirmaram ser o gerente de projetos o tomador dessa decisão e que, dependendo do impacto pode-se envolver o cliente ou não. A resposta abaixo ilustra uma situação típica de um projeto ágil:

*"A gente toma as decisões **sempre como um time**, a gente não chega a envolver o cliente nesse tipo de tomada de decisão. A gente se reúne, vê quais as consequências isso pode dar e toma a decisão. A gente chega a comunicar o cliente, mas a gente sempre tenta apresentar algo de valor pra ele mas explica, agora a gente está tendo que refatorar algo, então a gente tem essa transparência com ele. Mas a decisão é nossa. Mas uma das estratégias que a gente usa é em todas as sprints pelo menos mostrar algo de valor, não que a gente deixe de resolver a dívida. A gente resolve mas tenta aliar com algo que dê o retorno pra ele também. Porque se a gente falar, ah, nessa Sprint a gente só vai fazer refactoring que a gente deveria ter feito, que é uma dívida, não vai ficar legal. Então a gente sempre tenta pegar uma estória que realmente vai agregar para ele e a gente vai entregar, e fazer junto com o refactoring da dívida que a gente deveria ter feito. Então a gente tenta sempre casar os dois."*

As dívidas consideradas mais importantes pelos entrevistados foram:

1. Design/arquitetura
2. Distribuição de conhecimento
3. Teste
4. Segurança
5. Requisitos

4.3.7 Comentários adicionais

Nesta seção foram feitas perguntas de opinião sobre a dívida técnica em geral, seus benefícios e desafios em gerenciá-la explicitamente, bem como formas de contornar suas dificuldades.

Em geral os entrevistados consideram o gerenciamento da dívida técnica um aspecto importante nos projetos. Muitos se interessam pelo assunto, porém consideram que atualmente não é feito, é feito implicitamente ou é negligenciado nos projetos, conforme citado abaixo:

Também alguns entrevistados consideram o gerenciamento da dívida um tipo de risco, como colocado pelo entrevistado 10:

> *"Acho que é essencial.* ***Manter o teu débito técnico sob controle faz parte de um bom gerenciamento de projeto****, faz parte de entender onde o teu time está errando, faz parte de, digamos assim, saber o quanto de investimento tu vais precisar para botar realmente esse produto no mercado. Então se tu vens controlando isso desde o início tu já sabes quanto tempo tu vais precisar realmente para estabilizar aquilo e dizer que o teu produto está pronto, é um tratamento de riscos, então eu vejo que é essencial tu teres o gerenciamento da dívida técnica."*

Outro entrevistado cita que há muita oportunidade nesta área para os times de projeto. Ele considera que a dívida técnica está relacionada a riscos e também compara a gestão da dívida nos projetos tradicionais e ágeis, como descrito abaixo em sua fala:

> *"Eu acho que* ***não é o principal foco dos times de projeto hoje****, acho tem muita oportunidade. Como eu te falei está muito vinculado a riscos pelo o que eu vejo, e muitas vezes, como tu falaste, se toma a decisão de ir por uma linha mais simplória para não explorar muitos recursos técnicos para não incorrer ou diminuir os riscos de diminuir os riscos de ter mais dívidas técnicas. Então eu diria que* ***o mercado ainda tem uma resistência para tomar algumas decisões quanto a isso para***

> ***minimizar esse risco**, e na minha opinião o mercado acaba não tentando explorar muito os recursos para justamente minimizar o risco. A forma que se minimiza o risco é não tentando fazer algo mais rebuscado. É não reinventar a roda, ir na linha tradicional que já se sabe que vai funcionar. Outra coisa, na minha opinião sobre a gestão em si, eu não vejo muito eficiente. Na metodologia ágil, aí sim em cima das reuniões diárias as pessoas acabam trazendo um pouco mais esses impedimentos, digamos assim, que estão vinculados à dívida técnica para se ter uma gestão mais apropriada. Mas **os projetos tradicionais e as ferramentas de mercado, pelo menos as que eu conheço, eu não vejo que ajudam de uma maneira mais proativa, ou de maneira mais efetiva, a gestão da dívida técnica.**"*

Dentre os benefícios em gerenciar a dívida explicitamente, os mais citados foram antecipar problemas, entender os riscos, ter recursos para tomada de decisão, ter uma gestão de projetos efetiva e transparência. O entrevistado 10 também ressalta o impacto da dívida na qualidade final do produto:

> *"É não varrer pra baixo do tapete a questão da qualidade. Eu vejo a dívida técnica muito relacionada à **qualidade**, às vezes é muito difícil até separar esses dois conceitos. Em algumas vezes já vi projetos serem liberados com a qualidade que eu não achava aceitável. E acho que tu trazendo isso desde o início do projeto, tu já crias uma expectativa de que tu vais precisar um investimento maior lá no final pra pagar essa dívida se tu não conseguires fazer isso durante o projeto como um todo. Então acho que tu consegues preparar melhor a cabeça das pessoas que decidem pelo projeto de que a gente precisa pagar esse débito e não sair pro mercado assim."*

Já o entrevistado 8 levanta uma questão sobre a diferença de valor do gerenciamento da dívida para o *"business"* e para o time. Ele acredita que o negócio não percebe tanto valor quanto o time percebe:

> *"Eu acho que, pensando como business, talvez isso não tenha tanto valor. Mas como equipe de execução de projeto sem dúvida nenhuma isso pode gerar valor. Com o business, eu não sei como colocaria isso para ele. Principalmente com esse nome, dívida técnica. Porque para eles, else vão dizer: isso é problema do IT. E assim, no final, na minha opinião é coisa que a equipe tem que resolver dentro dela. **Tem que cuidar muito a maneira que leva isso pro business** eu acho. Pensando em todos que já trabalhei até hoje. De repente, conversando com eles podem dizer algo*

diferente. Se eu testar com unit test ou não testar, problema teu. É isso que eles vão te dizer."

Em relação às dificuldades em se gerenciar explicitamente a dívida técnica, surgiram vários aspectos interessantes como confiança, mostrar a importância para os *stakeholders*, o time dar visibilidade, pré-conceitos, julgamento, prioridades, pressão por prazos e entrega, mudança de cultura/*mindset*, *overhead* no gerenciamento, ferramentas e custo. O entrevistado 1 mostra a questão da confiança do cliente:

"Não vejo dificuldade na nossa situação, eu acho que em alguns projetos ***as pessoas podem não querer deixar explícito talvez por perder a confiança do próprio cliente****, dependendo do nível de parceria e comprometimento dos dois. No nosso caso é tranquilo então é bem transparente. Ferramentas eu acho que tem que sentir a necessidade de ter, ter um Excel e ficar sempre de olho nele já está bom.* ***Pagar logo a dívida também é importante.****"*

O entrevistado 7 acha que a questão da pressão pela entrega e a distorção do ROI é um empecilho:

"É o tempo, a ***pressão do tempo****. A pressão por entrega não interessa como. Os projetos dão errado, e eu tenho vários em 28 anos de carreira, são porque o cliente não está preocupado com o que vai custar depois. Ele está preocupado em atingir uma data. Ele tem que entregar tudo do jeito que for sem abrir mão. Entrega a qualquer custo gera um custo posterior gigantesco que é exatamento o ROI brasileiro. O cálculo do ROI não leva em consideração o tamanho do abacaxi que tu deixas pra depois."*

Já o entrevistado 9 considera que haja um pré-conceito que acaba expondo o time. Ele acredita que quando se trata de vários times pode haver uma competição que pode distorcer o conceito:

"Na verdade o grande problema, ***como existem alguns pré-conceitos tu acabas expondo o time****. Os times que tem dívida técnica são expostos mas tu acabas expondo só o lado ruim. Por exemplo, até o que era previsto para um ano nós fizemos em seis meses. Tu estás dizendo o seguinte, estrategicamente a gente criou esse débito mas se tu falas olha só, esses caras tem um débito tal, gigante... então tu foste para o lado ruim né, não vê o lado bom. Então acho que tem um limite que* ***tem que mudar a cultura primeiro****. Um problema que acontece bastante é quando tu*

tens dentro do mesmo time ou no mesmo programa, beleza, dívida técnica, está homogêneo o conhecimento. Mas quando eu abro e digo o time B tem uma dívida técnica tal, e falo isso para o time A que está fora do contexto deles, existe um julgamento. Isso tem que ter cuidado para não virar uma guerra, os caras são muito ruins, isso vai ser sempre assim, etc. E isso não tem nada a ver. A questão é se foi uma ***dívida consciente ou inconsciente****. Uma coisa é pegar dinheiro da tua conta conta-corrente achando que está gastando dinheiro teu. Outra coisa é ficar dando OK num terminal lá de saque eletrônico, quando vê baixou 10 mil lá e tu vais pagar juros."*

Outro entrevistado expõe a priorização como dificuldade. Ele também coloca o efeito contrário de se ter o controle da dívida:

"A dificuldade é a ***oposição de alguns PO's que querem maximizar a bando alocada do time*** *e expandir, alguns gerentes de produto sofrem do mesmo mal, eles acham que essas coisas não são legais e sempre que tu tens que priorizar essa coisas perdem espaço no jogo de priorização. Por outro lado tu tens que fazer o máximo para que o time produza coisas com mais qualidade, que preciso de menos retrabalho no futuro. Se tu tornas muito fácil tu crias outro mindset no cara que é ruim, eu faço mal feito porque eu sei que posso voltar depois."*

A questão do custo também é relatada como dificuldade. O entrevistado 15 afirma não possuir muita margem para lidar com a dívida após o início do projeto:

"A dificuldade é que gera ***custo de projeto****, e quando envolve custos a decisão é mais no nível estratégico da empresa, direção, gerente de área, até chegar a mim. E quando chega em mim eu já não tenho margem para manobra, na verdade tem zero. Então a regra é: vai ficando, a gente não consegue vender a ideia de que é um software mais caro e tem que tratar isso. Só consegue vender que olha, vai lançar um equipamento novo e que tem que suportar. Alguém lá na direção ou gestor da área que reporta direto para a direção bater o pé, olha, nessa versão não vai sair tal coisa porque a gente vai tratar tal coisa. Isso não acontece."*

Algumas formas relatadas de se contornar essas dificuldades incluem *refactoring*, embutir o custo no projeto, comunicação diária, transparência, registro, manter a dívida documentada, envolver as pessoas, compartilhar que é importante e aprender a conviver

com a dívida. O entrevistado 14 sugere usar exemplos estatísticos de dívidas comuns e custo relativo como base para outros projetos:

> *"[...] acho que **é feito um contorno tardio**, mas no momento que isso acontece vira prioridade máxima, mas aí o retrabalho já existe. E aí tu vais mais apagar incêndio do que fazer algo que estaria previsto, ou enfim, que com uma gestão efetiva tu conseguirias talvez até evitar. Compartilhar que isso é importante, usar exemplos até estatísticos de dívidas técnicas muito comuns em projetos e o custo atrelado, acho que dá para fazer uma boa base quanto a isso em muitas empresas que tem esse tipo de situação. Um levantamento das principais dívidas técnicas e o custo atrelado a elas, mostrar para os executivos que isso é importante e impacta muito. E aí acho que isso, **quando pesa no bolso, as empresas ficam um pouquinho mais sensíveis a priorizar esse tipo de assunto.**"*

Já para o entrevistado 7, que é de métodos ágeis, não existe nem um tratamento especial para lidar com a dívida:

> *"A dívida técnica, para quem é de métodos ágeis, **é commodity, é uma discussão**. Então eu não tenho nenhum tratamento especial, ela faz parte do desenvolvimento de software."*

O entrevistado 2 acredita que o tempo investido na especificação de uma determinada arquitetura pode ser desperdiçado devido aos *refactorings* feitos à medida que o software evolui. Ele também afirma que o pagamento da dívida acaba sendo feito como um esforço extra nos projetos:

> *"Acho que o tempo que tu farias bem feito a arquitetura identificando a dívida técnica e mitigando isso, acaba matando no refactoring. O contornar acaba sendo contra algumas premissas da gestão de projetos. Tu vais **ter que ter uma "puxada extra" do time para pagar essa dívida embutindo esse custo no projeto.**"*

Para o entrevistado 15, a dívida que é ignorada gera bugs que são corrigidos através de *"workarounds"* que vão para o cliente. Ele afirma ter que conviver com isso:

> *"**A gente simplesmente ignora** e isso gera bugs, principalmente que não tem correção, limitação de arquitetura mesmo. Aí **a gente faz um workaround, vai para o cliente o problema**, tem que arrumar a base de dados do cliente, mas a gente convive com isso."*

4.3.8 Análise do Scrum sob a perspectiva da Divida Técnica

Por ser o método ágil mais amplamente utilizado na indústria, e também pelos participantes do estudo de campo, optou-se por fazer uma análise do Scrum com base nos resultados previamente mencionados. Buscou-se fazer um mapeamento dos itens identificados no estudo com as práticas do Scrum de modo a verificar sua aderência. Para cada item, respectivamente, foi criada uma tabela de aderência onde são atribuídos um dos três valores possíveis: “Não Atendido”, “Parcialmente atendido” e “Atendido”. Ao final é apresentada uma tabela consolidada.

4.3.8.1 Identificação e monitoramento

O Scrum define o artefato Product Backlog, onde são listados todos os itens necessários no produto sendo a única fonte de requisitos para qualquer mudança feita nele. O Scrum também define que itens do Product Backlog incluem características, funções, requisitos, melhorias e correções. Além disso, o Product Owner é o responsável por seu conteúdo, disponibilidade e ordenação.

No *framework* Scrum não são feitas referências à identificação e monitoramento da dívida técnica. Não há, portanto, uma diferenciação de itens de produto e itens técnicos relacionados à dívida técnica que vai se acumulando ao longo dos Sprints. Desta forma, faz-se a pergunta: *Existe algum artefato no Scrum no qual os itens identificados como dívida técnica podem ser incorporados ao produto?* Não fica claro que os itens do Product Backlog sejam itens identificados como dívida técnica. Já a dívida técnica normalmente é identificada pelo time de desenvolvimento através de ferramentas e das próprias interações entre eles, sendo, portanto, fora do domínio do Product Owner.

A seguir são apresentados os itens relacionados à dimensão de identificação e monitoramento da dívida técnica. Correspondente a eles, são determinadas as aderências ao Scrum conforme a escala previamente definida.

Tabela 5 – Identificação, monitoramento da DT e aderência ao Scrum

Aspectos mapeados no estudo de campo	Não atendido	Parcialmente atendido	Atendido
Identificação da dívida	X		
Monitoramento da dívida	X		

4.3.8.2 Qualidade e rastreamento de decisões arquiteturais

O Scrum possui a definição de "*done*" para um item do Product Backlog ou um Incremento, e todos devem entender seu significado no time. Essa definição deve ser transparente e varia entre equipes de Scrum, porém seus membros devem ter um entendimento comum do que significa um trabalho completo. A definição de *"done"* serve para avaliar quando um Incremento do produto está com seu trabalho completo. Essa definição, porém, não especifica os processos utilizados para qualificar o código para que ele possa ser considerado pronto. Porém, como se trata de um *framework* no qual podem ser aplicados técnicas e processos, podem-se definir quais processos de qualidade mais adequados ao tipo de software desenvolvido pelo time devem ser utilizados.

Quanto ao rastreamento de decisões arquiteturais, não há nenhuma definição no Scrum que especifique onde nem como tais decisões devem ser registradas. O que ocorre normalmente é a comunicação informal e essas decisões acabam ficando com as próprias pessoas. À medida que ocorre a rotatividade no time, caso não seja devidamente gerenciado isso tende a se agravar gerando uma dívida de distribuição de conhecimento no time [TOM13].

O rastreamento de decisões arquiteturais não é feito facilmente, porém o conhecimento pode ser encontrado das seguintes formas:

- Documentos técnicos
- Matriz de rastreabilidade
- Wiki
- Nas próprias pessoas
- Comunicação por e-mail

A tabela abaixo apresenta os itens relacionados aos processos de qualidade e rastreamento de decisões arquiteturais. Correspondente a eles, são determinadas as aderências ao Scrum conforme a escala previamente definida.

Tabela 6 - Processos de qualidade, rastreamento de decisões arquiteturais e aderência ao Scrum

Aspectos mapeados no estudo de campo	Não atendido	Parcialmente atendido	Atendido
Processos de qualidade			X
Rastreamento de decisões arquiteturais	X		

4.3.8.3 Custos monetários

O Scrum não possui nenhum recurso para cálculo de custos monetários relacionados à dívida técnica. A única estimativa utilizada é a das estórias de usuário, para se ter uma previsão do que o time acredita que poderá ser feito no Sprint seguinte. O trabalho planejado para os primeiros dias do Sprint é decomposto após o Sprint Planning, geralmente em unidades de um dia ou menos.

Conforme os resultados do estudo de campo, no que diz respeito à estimativa do custo monetário da dívida, muitas vezes são feitas estimativas levando em consideração apenas as horas de desenvolvimento. Estes custos, no entanto, são considerados difíceis de quantificar monetariamente e não foi captada no estudo de campo nenhuma forma que pudesse ser útil para estimar os mesmos antes ou após o fato [NOR12]. Além disso, no estudo de campo foi questionada a utilidade de se atribuir um custo monetário à dívida técnica.

A tabela a seguir apresenta os itens relacionados aos custos monetários da dívida técnica. Correspondente a eles, são determinadas as aderências ao Scrum conforme a escala previamente definida.

Tabela 7 - Custos da DT potencial, efetiva e aderência ao Scrum

Aspectos mapeados no estudo de campo	Não atendido	Parcialmente atendido	Atendido
Custos de dívida potencial	X		
Custos de dívida efetiva	X		

4.3.8.4 Estimativa e estratégia de pagamento

Após o objetivo do Sprint ser definido, o time de desenvolvimento seleciona as User Stories que estão no Product Backlog para serem trabalhadas no Sprint. Elas são então planejadas, estimadas, priorizadas e adicionadas ao Sprint Backlog a cada iteração. No entanto, como não há explicitamente o conceito de User Stories de dívida técnica, não existe uma definição quanto à proporção entre estórias relativas a requisitos e estórias puramente técnicas de pagamento da dívida. Conforme os resultados do estudo de campo quanto às estratégias de pagamento da dívida, ela em geral segue uma das seguintes abordagens:

- Não pode exceder um determinado limiar (10% a 20% do time, por exemplo);
- É absorvida pelo time nas tarefas das User Stories;
- Criação de defeitos
- Criação de estórias de usuário priorizadas junto ao PO
- Negociar introdução de estórias técnicas para *refactoring* e melhoria
- Horas destinadas à melhoria contínua por contrato

O Scrum é um *framework* independente de estrutura organizacional, tipo de cliente ou forma de contratação. O tamanho da dívida técnica e objetivo de pagamento normalmente não estão presentes explicitamente no Sprint Backlog, porém ela pode ser estimada pelo time assim como as User Stories normais.

Sob o ponto de vista de manter a dívida visível e gerenciável, é interessante que se tenha um discernimento entre o trabalho que irá gerar valor para o cliente (visível) e o de manutenção estrutural do software (invisível) de forma a poder monitorar o nível de dívida acumulado ao longo do tempo para que se possam tomar as ações necessárias.

Os objetivos do pagamento da dívida mapeados no estudo de campo incluem, por exemplo, a reescrita de um software, *bug fixing* e melhorias ou *refactoring*. Para os entrevistados os quais trabalhavam em projetos que utilizam Scrum, em geral a técnica de estimativa das User Stories é o Planning Poker.

A tabela abaixo apresenta os itens relacionados à estimativa e estratégias de pagamento da dívida técnica. Correspondente a eles, são determinadas as aderências ao Scrum conforme a escala previamente definida.

Tabela 8 - Estimativa, estratégia de pagamento da DT e aderência ao Scrum

Aspectos mapeados no estudo de campo	Não atendido	Parcialmente atendido	Atendido
Estimativa da dívida		X	
Estratégia de pagamento da dívida		X	

4.3.8.5 Comunicação

O Scrum possui a noção de incremento, que é a soma de todos os itens do Product Backlog completados durante o Sprint. O Incremento pode conter trabalho de pagamento de dívida técnica, porém a comunicação sobre a dívida técnica fica muitas vezes implícita.

Os termos “dívida técnica”, “débito técnico”, *“technical debt”* ou quaisquer variações são inexistentes no glossário do Scrum.

A dívida pode ser implicitamente ou explicitamente comunicada. Caso seja exposta ao cliente ou equipes de negócio, pode ocorrer as seguintes situações conforme relatado por alguns entrevistados:

- Preconceito ou resistência em se aceitar o “débito”
- Reputação da equipe pode ser afetada

A dívida também é referenciada através dos seguintes termos: “débito técnico”, *“improvement”*, *”enhancement”*, “melhoria e “retrabalho”. Também pode ser individualizado, na forma de *issues* de análise estática, defeitos ou testes não executados, por exemplo.

A tabela abaixo apresenta os itens relacionados à comunicação sobre a dívida técnica. Correspondente a eles, são determinadas as aderências ao Scrum conforme a escala previamente definida.

Tabela 9 - Comunicação entre o time, com o negócio e aderência ao Scrum

Aspectos mapeados no estudo de campo	Não atendido	Parcialmente atendido	Atendido
Comunicação entre o time	X		
Comunicação com o negócio	X		

4.3.8.6 Riscos e tomada de decisão

O Scrum evidencia uma abordagem iterativa e incremental, cujo intuito é de aperfeiçoar a previsibilidade e controle de riscos. Portanto, para a dívida técnica vale o mesmo princípio. No que diz respeito à tomada de decisão, o Product Owner é o responsável por gerenciar o Product Backlog e priorizar seus itens. Quanto à priorização de itens relativos à dívida técnica, não são especificados os papéis dos responsáveis envolvidos.

Os tipos de dívida técnica considerados mais relevantes pelos entrevistados do estudo de campo foram, em ordem de importância, classificados como:

1. Design/arquitetura
2. Distribuição de conhecimento
3. Teste

4. Segurança
5. Requisitos
6. Performance

A tomada de decisão ocorre de forma diferente para projetos tradicionais e ágeis. No ágil o time geralmente faz a tomada de decisão sobre se aceitar ou não a dívida junto com o PO, enquanto que no tradicional o gerente de projetos é quem dá a última palavra.

Alguns exemplos de papéis envolvidos na tomada de decisão citados pelos entrevistados são o Scrum Master, o Product Owner, o time de desenvolvimento, o cliente, o gerente de projetos ou de programa, o gerente de contas, o time de arquitetura, o líder de desenvolvimento e as equipes de negócio.

A tabela abaixo apresenta os itens relacionados aos riscos e tomada de decisão sobre a dívida técnica. Correspondente a eles, são determinadas as aderências ao Scrum conforme a escala previamente definida.

Tabela 10 - Riscos, tomada de decisão sobre a DT e aderência ao Scrum

Aspectos mapeados no estudo de campo	Não atendido	Parcialmente atendido	Atendido
Riscos da dívida			X
Tomada de decisão sobre a dívida		X	

4.3.9 Consolidação da análise dos resultados

A tabela a seguir compreende cada uma das dimensões do estudo de campo, onde cada uma é analisada e então verificado se o Scrum atende às necessidades de gerenciamento da dívida técnica.

Tabela 11 - Consolidação da análise de aderência ao Scrum

Aspectos mapeados no estudo de campo	Não atendido	Parcialmente atendido	Atendido
Ferramental			
Identificação da dívida	X		
Monitoramento da dívida	X		
Qualidade			
Processos de qualidade			X

Rastreamento de decisões arquiteturais	X		
Custo			
Custos de dívida potencial	X		
Custos de dívida efetiva	X		
Tempo			
Estimativa da dívida		X	
Estratégia de pagamento da dívida		X	
Comunicações			
Comunicação entre o time	X		
Comunicação com o negócio	X		
Riscos			
Riscos da dívida			X
Tomada de decisão sobre a dívida		X	

A partir desta análise, elaborou-se uma proposta para incorporar no Scrum alguns aspectos da DT identificados através dos resultados obtidos neste estudo. Tal proposta será detalhada no capítulo seguinte.

5 PROPOSTA DE EXTENSÃO DO SCRUM PARA GERENCIAMENTO DA DÍVIDA TÉCNICA

Este capítulo apresenta uma proposta de extensão do Scrum para gerenciamento da dívida técnica. Durante o estudo de campo foram apontadas algumas dificuldades em se gerenciar explicitamente a dívida técnica, tais como confiança do cliente, exposição do time, mostrar a importância para os *stakeholders*, preconceitos ou julgamentos, prioridades, pressão por prazos e entrega, mudança cultural (*mindset*), *overhead* no gerenciamento, ferramentas, custos entre outros. A proposta a seguir pretende, com base na análise feita no capítulo anterior, fazer com que o próprio Scrum incorpore os aspectos referentes ao gerenciamento da dívida técnica de modo a atenuar essas dificuldades. Por outro lado, pretende-se utilizar de algumas formas citadas pelos respondentes para contornar essas dificuldades como *refactoring*, comunicação diária, transparência e manter a dívida documentada para que sejam consideradas nesta extensão. Dessa forma, pretende-se atingir os benefícios levantados pelos entrevistados como antecipar problemas, entender os riscos, prover recursos para a tomada de decisão, gestão de projetos mais efetiva, transparência e qualidade final do produto.

O critério utilizado para as extensões é a análise feita no capítulo anterior, onde foram selecionadas algumas áreas em que foram atribuídos os conceitos “Não atendido” ou “Parcialmente atendido” em relação ao estudo de campo sobre gerenciamento da dívida técnica. As áreas abaixo foram escolhidas para a proposta de extensão por terem permitido uma avaliação mais objetiva durante o estudo de campo:

- Identificação da dívida
- Monitoramento da dívida
- Rastreamento de decisões arquiteturais
- Estimativa da dívida
- Estratégia de pagamento da dívida
- Comunicação entre o time
- Comunicação com o negócio
- Tomada de decisão sobre a dívida

O estudo de campo buscou identificar práticas e ferramentas de gerenciamento da dívida técnica utilizadas na indústria. Porém as respostas para algumas perguntas não

foram suficientes para identificar tais práticas de modo que pudessem ser incorporadas nesta extensão do Scrum. Como oportunidades de estudos futuros estão as seguintes áreas:

- Custos de dívida potencial
- Custos de dívida efetiva

A seguir são detalhados os complementos propostos ao *framework* Scrum de forma que ele satisfaça às necessidades de gerenciamento da dívida técnica em projetos que utilizam este método.

5.1 Identificação e monitoramento da dívida técnica

Para que a dívida seja efetivamente identificada e monitorada no Scrum, é proposta a introdução de um artefato contendo itens de dívida técnica a serem pagos durante os sprints. Este “Technical Backlog” segue o mesmo princípio do Product Backlog, com a diferença de ser uma lista de tudo que precisa ser feito para que a dívida técnica se mantenha sob controle. Isso significa a inclusão de itens detalhados não limitados a melhorias de design/arquitetura, distribuição de conhecimento, teste, segurança, requisitos e performance. Cada item representa uma tarefa que foi deixada para trás, mas que traz um risco de causar problemas caso não seja feita. O time de desenvolvimento será responsável por manter e priorizar esta lista, que em acordo com o PO, será adicionado aos sprints conforme necessidade.

Os itens do Technical Backlog são identificados de forma automática ou humana. Abaixo é apresentado um modelo de um item do Technical Backlog com seus respectivos atributos. Este modelo é uma adaptação de Seaman et al, 2013 (A Case Study on Effectively Identifying Technical Debt).

Tabela 12 - Item do Technical Backlog

ID	*Número de identificação da Dívida Técnica*
Responsável	*Pessoa ou papel que deverá resolver este item*
Tipo	*design, documentação, defeito, teste ou outro tipo de dívida*
Localização	*Lista de todos os arquivos/classes /métodos ou documentos/páginas envolvidos*
Descrição	*Descreve a anomalia e possíveis impactos na manutenção futura*

Capital estimado	*Quantidade de trabalho necessário para pagar este item técnico em uma escala de três pontos:* ***Alto/Médio/Baixo***
Valor do juro estimado	*Quantidade de esforço extra a ser empregado no futuro caso este item técnico não seja pago imediatamente em uma escala de três pontos:* ***Alto/Médio/Baixo***
Probabilidade de juros estimado	*Probabilidade de este item, caso não seja pago, requerer trabalho extra no futuro em uma escala de três pontos:* ***Alto/Médio/Baixo***
Intencional?	***Sim/Não/Não Sei***

O capital estimado refere-se ao custo de se eliminar completamente a dívida, isto é, resolver completamente a imperfeição técnica. Isso inclui diferentes tipos de atividades como adicionar documentação faltante, fazer *refactoring* em um código difícil de manter ou manter um conjunto de testes de regressão para se adequar ao código e aos requisitos. Dados históricos podem ser utilizados de forma a se obter estimativas mais precisas e confiáveis além de Alto/Médio/Baixo.

O segundo componente da dívida, o juro, é composto de duas partes: (1) a probabilidade de juros é a probabilidade da dívida que, caso não seja paga, fará com que outro trabalho seja mais custoso em dado um período de tempo ou um *release*; (2) o valor do juro é uma estimativa da quantidade de trabalho extra necessário caso este item de dívida não seja pago de volta. A probabilidade de juros pode ser estimada com dados históricos, por exemplo, e dados de defeitos. Além disso deve-se considerar a variável tempo, pois a probabilidade varia ao longo do tempo. Por exemplo, a probabilidade de um módulo que necessite *refactoring* causar problemas na próxima *release* (porque modificações serão feitas nele) pode ser muito baixa, porém ela aumenta se considerarmos períodos maiores, por exemplo, ao longo do próximo ano ou 5 anos.

Além das propriedades financeiras da dívida técnica, algumas propriedades que auxiliam na decisão de pagamento são capturadas neste modelo:

1) O tipo de dívida ajuda a refinar o pagamento da dívida com características de qualidade de interesse. Por exemplo, dívidas de defeito (defeitos conhecidos

ainda não consertados) podem ser diferentemente percebidas em sistemas críticos.

2) A decisão original de se aceitar a dívida foi feita intencionalmente ou não intencionalmente? Esta informação ajuda a entender como decisões relacionadas à dívida técnica são gerenciadas no projeto.
3) O responsável por consertar ou pagar a dívida técnica é importante por motivos administrativos e provê uma base para avaliar o capital e juros.
4) A localização da dívida é importante para entender o impacto no produto, relacionamentos entre os itens e efeitos colaterais no código quando a dívida for paga.

O processo de identificação e monitoramento começa com a detecção dos itens de dívida e construção desta lista pelo time de desenvolvimento. O próximo passo é o time medir os itens estimando o capital, valor do juro e probabilidade de juros. A dívida é então monitorada ao longo dos sprints pelo Scrum Master e decisões podem ser feitas em relação a quais itens da dívida devem ser pagos ou postergados, seguindo a priorização do Product Owner. Abaixo é ilustrado o Scrum estendido utilizando o Technical Backlog.

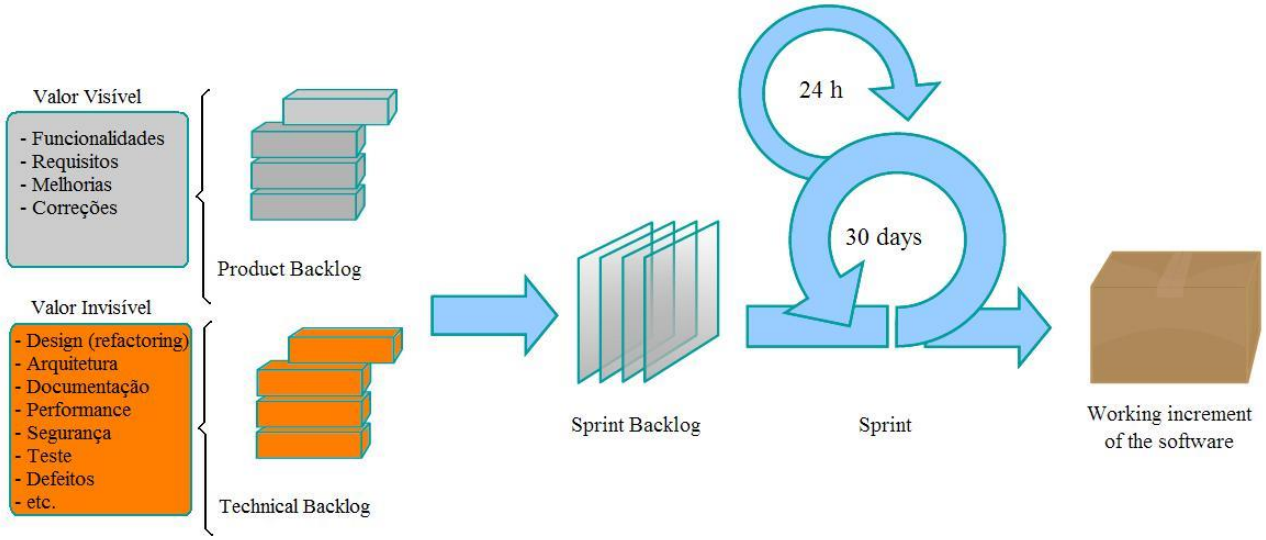


Figura 16 - Technical Backlog

Para ajudar no monitoramento da dívida, é proposta a utilização de um mecanismo semelhante ao “Burndown chart”. Esta ferramenta de planejamento e monitoramento ajudará a saber qual é o nível de dívida técnica existente no projeto ao longo do tempo.

A criação do “Technical Burndown” se dá após a quebra das tarefas de itens do Technical Backlog pelo time de desenvolvimento, geralmente durante o Sprint Planning. Uma vez quebradas as tarefas, um gráfico é gerado tendo no eixo X as datas do Sprint enquanto que o esforço restante, em ordem decrescente, no eixo Y.

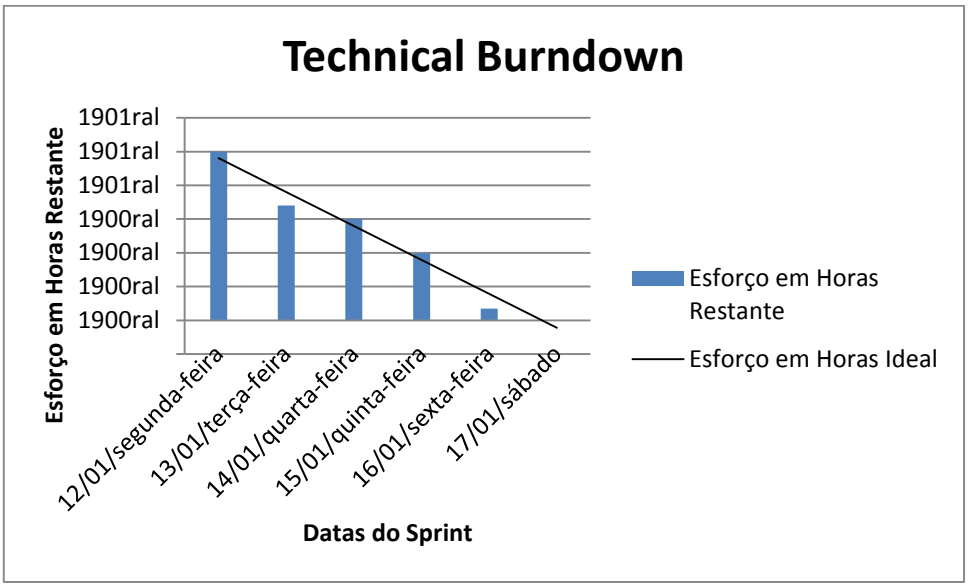


Figura 17 - Exemplo de um Technical Burndown

O Technical Burndown busca dar visibilidade ao progresso das tarefas do Technical Backlog. Desta forma, pode-se diferenciar o progresso do pagamento da dívida com tarefas de adição de funcionalidades ao produto, permitindo assim um melhor monitoramento e visibilidade da dívida técnica.

5.2 Rastreamento de decisões arquiteturais

> *“If we recognise that software aging is inevitable and expensive, that the first or next release of the program is not the end of it’s development, that the long-term costs are going to exceed the short term profit, we will start taking documentation more seriously.”*
>
> *Parnas - Software Aging*

Grady Booch forneceu um conjunto de diretrizes para a “arquitetura ágil” [BAS12]. Ele afirma que um processo ágil efetivo irá permitir que a arquitetura cresça incrementalmente à medida que o sistema é desenvolvido e amadurece. Ele comenta ainda que para ser ágil, a arquitetura deve ser visível e autoexplicativa no código. Isso significa

tornar padrões de projeto, preocupações transversais e outras decisões importantes óbvias, bem comunicadas e defendidas. Isso, por sua vez, requer documentação. A documentação da arquitetura para esta extensão do Scrum deve seguir o princípio ágil do *You ain't gonna need it* (YAGNI) [BEC00], ou seja, somente deve ser documentado quando necessário. Isto é, porções de arquitetura necessárias para ensinar novos integrantes do time, que incorporem riscos potenciais significativos se não gerenciadas devidamente e que precisam ser mudadas frequentemente [BAS12].

Para que decisões arquiteturais sejam gerenciadas e efetivamente documentadas durante os sprints é proposto um modelo para capturar as mudanças feitas na arquitetura do software. Este modelo é baseado no conceito de "*work item*" da ferramenta IBM Rational Team Concert [IBM15].

Tabela 13 - Modelo de captura de decisões arquiteturais

ID	*Número de identificação da mudança*
Sprint	*Iteração correspondente à mudança (por exemplo, 63)*
Feature	*Nome da feature afetada*
Data da mudança	*Data em que a funcionalidade ou mudança foi implementada*
Responsáveis	*Pessoas ou papéis envolvidos na decisão da mudança*
Artefatos afetados	*Conjunto de artefatos de código que foram modificados como parte desta mudança. Deve incluir nome, tipo e localização*
Motivo da mudança	*Motivo pelo qual se decidiu fazer a mudança*
Descrição da mudança	*Descrição do que foi mudado, por exemplo, introdução de um padrão de projeto ou substituição de uma biblioteca obsoleta. Também podem ser descritas graficamente na forma de diagramas, esquemáticos ou outros recursos que auxiliem na compreensão*

O processo de registro deve ser feito a qualquer momento durante as iterações no Sprint, sendo o time de desenvolvimento responsável pelo mesmo. No entanto, mudanças

pontuais que não impactam a arquitetura do software como um todo provavelmente não terão muito benefício em ter este tipo de documentação.

O Sprint significa o número da iteração em que foi feita a mudança. A *feature* é o nome que foi dado à funcionalidade em que está sendo feita a mudança. A data tem por objetivo gerar um histórico ordenado cronologicamente para entender como se chegou ao estado atual. Os responsáveis podem ser uma ou mais pessoas ou papéis envolvidos na decisão da mudança arquitetural. Os artefatos afetados são indicações de nome, tipo e localização do artefato pra que se possa rastreá-lo. O motivo e descrição da mudança devem ser claros e objetivos sobre a deliberação da mudança.

5.3 Estimativa e estratégia de pagamento da dívida técnica

A estimativa para itens de dívida técnica proposta será conforme a técnica utilizada no projeto, caso a equipe utilize alguma. Se for Planning Poker, por exemplo, o time irá estimar os itens utilizando a mesma técnica. Para a estratégia de pagamento, deve-se utilizar um percentual que pode ser definido pelo Scrum Master, sendo que no estudo de campo há um caso de utilização entre 10% e 20% dos Sprints focados em pagamento da dívida técnica. Deve-se, porém, respeitar o conceito do Sprint que é o de produzir, ao final de um mês ou menos, um Incremento “pronto”, utilizável e potencialmente “entregável”.

5.4 Comunicação e tomada de decisão sobre a dívida técnica

A comunicação da dívida pode ser dada através do gerenciamento dos itens do Technical Backlog. Deve-se utilizar o termo que mais convir para o time (“débito técnico” ou “dívida técnica”) tendo em vista suas preferências. Porém o entendimento do conceito de dívida técnica deve ser o mesmo por todos os membros do time e pelo negócio.

Os responsáveis principais pelo levantamento de itens do Technical Backlog são os membros do time de desenvolvimento. Tudo que for considerado importante para o gerenciamento da dívida técnica deve ser deixado explícito e ser dado conhecimento ao time para posterior resolução. O PO terá a responsabilidade de acatar as decisões que forem consideradas pelo time de desenvolvimento fundamentais para o gerenciamento da dívida e evolução do software, tendo a consciência de que estes terão impacto maior no futuro caso sejam negligenciados e também levando em conta o espaço que deve ser deixado nos sprints caso haja itens no Backlog Técnico.

Abaixo é apresentada uma tabela com as responsabilidades propostas de cada papel na tomada de decisão sobre a dívida técnica. Essas responsabilidades são complementares às responsabilidades normais de cada papel no Scrum tradicional.

Tabela 14 – Papéis do Scrum e responsabilidades relacionadas à DT

Papel do Scrum	**Responsabilidades relacionadas à DT**
Time de Desenvolvimento	Identificar, estimar, priorizar e comunicar a dívida técnica; Participar na tomada de decisão em se aceitar ou pagar a dívida técnica
Product Owner	Acatar as decisões de pagamento da dívida pelo time, permitir a introdução de user stories do Backlog Técnico levantadas pelo time no Sprint de forma a evitar que a dívida se acumule ao longo do tempo; Participar na tomada de decisão em se aceitar ou pagar a dívida técnica
Scrum Master	Facilitar a comunicação entre a equipe e o PO, monitorar o nível de dívida técnica; Participar na tomada de decisão em se aceitar ou pagar a dívida técnica

5.5 Avaliação da proposta por especialistas

A proposta de extensão do Scrum foi submetida a uma avaliação através de entrevistas com um subconjunto de cinco profissionais participantes no estudo de campo. Todos os cinco possuem experiência profissional com Scrum, sendo que dois deles possuem certificação *Certified Scrum Master* (CSM). Com o intuito de obter um parecer desses especialistas sobre a proposta, foram coletadas opiniões abertas e aplicado um questionário contendo um *checklist* utilizando-se a Escala Likert, cujo protocolo está definido no Apêndice B desta dissertação. O resultado desta avaliação é descrito a seguir e servirá de base para melhorias na proposta atual em estudos futuros.

Há indícios de que a avaliação geral da proposta pelos entrevistados seja favorável, considerada importante e que atende todos os aspectos mencionados da dívida técnica. Alguns pontos fortes citados foram a definição dos papéis, forçar um percentual para

introdução de estórias de pagamento da dívida, processo de identificação e monitoramento da dívida e controle mais preciso da dívida técnica formalizando esse item no planejamento do projeto. Segundo um dos entrevistados:

> *"... Fazer com que este gerenciamento faça parte do processo é bom, pode fazer com que no futuro diminua a manutenção do sistema. Pode ser que tu comeces até a entregar mais porque foram feitas mudanças estruturais na arquitetura que irão permitir com que as coisas sejam feitas mais facilmente depois."*

Outro entrevistado afirma:

> *"... A gente sabe que defeitos são só a ponta do iceberg. Então, ter uma metodologia que ajude o time a identificar os nossos débitos técnicos, que ajude a liderança do projeto a gerenciar isso de uma maneira que dê visibilidade do que a gente está fazendo, é importantíssimo."*

Como pontos fracos foi mencionado o possível *overhead* gerado em relação à abordagem Scrum, se é necessária uma nova cerimônia, como isso pode impactar. Outro ponto levantado foi que o Technical Burndown pode nunca chegar a zero. A priorização foi outro ponto a se tomar um pouco de cuidado pois pode haver um conflito entre *features* e dívida técnica a ser considerado pelo responsável por decidir. Para isso deve haver uma revisão dos papéis, pois entende-se que o Product Owner é o mais indicado a fazer a priorização do Technical Backlog e não o time apesar de poderem ser consultados, pois ele é responsável pelo negócio e possui um melhor entendimento do ROI que a dívida traria. Também foi mencionado que a proposta é limitada apenas a projetos que utilizam o Scrum, considerando que ainda há uma grande parcela de organizações que utilizam métodos tradicionais.

Sobre a viabilidade de utilização desta extensão em projetos reais, foi dado como viável porém com um treinamento prévio do time. Foi sugerido avaliar ferramentas que irão ajudar na medição. Um entrevistado incluiu uma ressalva

> *"...Porém, é necessário uma contextualização bastante apurada para sponsors de projeto para que a proposta seja adotada."*

Quanto ao comentário, entende-se que a contextualização mencionada se refere à preparação tanto do time quanto de quem patrocina o projeto, havendo assim um entendimento comum sobre o que é a dívida técnica e como gerenciá-la. Talvez a pessoa

mais indicada para fazer esse alinhamento seja o Scrum Master, dada sua responsabilidade em garantir a aderência ao processo.

As seguintes sugestões foram citadas: avaliar o custo/benefício de se adicionar um item no Technical Backlog, ao invés de simplesmente colocar todas as sugestões possíveis. Além disso, o Technical Burndown poderia ser medido por Sprints e não dias. Também deve ficar claro quando irá ocorrer a tomada de decisão, se em cerimônias existentes ou não. Finalmente, o Scrum Master deve prover o apoio para guiar o time em relação à dívida técnica:

- Tipos de dívida
- Como identificar
- Formas/estratégias de pagamento
- *Workshop* de dívida técnica no início do projeto ou Sprint Planning. Poderá rotacionar entre o time a facilitação desta cerimônia.

Também como sugestão foi identificada a extração de métricas a partir das definições do capital estimado, valor do juro estimado e probabilidade de juros estimado, conforme o entrevistado sugere:

> *“Definição de métricas para apoiar a gestão do produto, onde seja possível avaliar a evolução ou não dos débitos técnicos do produto, o tamanho do débito técnico, o tempo previsto para acabar com o estoque de débito técnico, entre outras possibilidades que se abrem com essa extensão. ”*

As métricas obtidas podem ajudar no acompanhamento do projeto, dando assim subsídios para decidir quando é hora de parar para de desenvolver novas funcionalidades para pagar dívida. Pode ser interessante definir um limiar de dívida para o projeto, baseado em projetos passados ou Sprints anteriores.

Além das questões anteriores, solicitou-se que os entrevistados respondessem a um *checklist* com dez questões sobre as categorias da proposta, contendo quatro opções da Escala Likert variando de “Concordo totalmente” a “Discordo totalmente”. O resultado é apresentado a seguir:

- Questão 1: A identificação da dívida técnica pode ser realizada através da criação de um Technical Backlog:

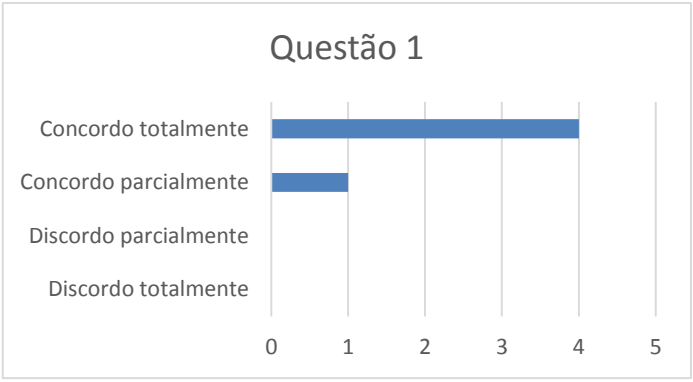


Figura 18 - Resultado da questão 1 do checklist

- Questão 2: O monitoramento da dívida técnica pode ser realizado através da criação de um Technical Burndown Chart:

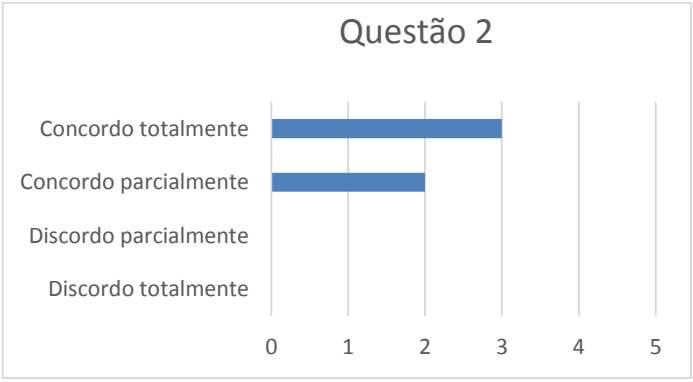


Figura 19 - Resultado da questão 2 do checklist

- Questão 3: O rastreamento de decisões arquiteturais pode ser realizado através da criação de um documento de captura de mudanças arquiteturais:

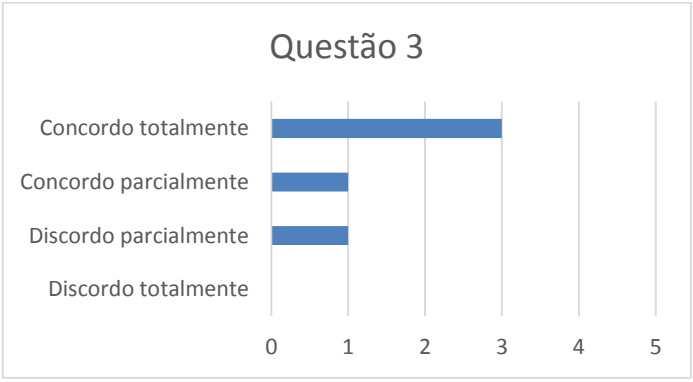


Figura 20 - Resultado da questão 3 do *checklist*

- Questão 4: Os itens do Technical Backlog podem ser estimados utilizando-se técnicas como o Planning Poker:

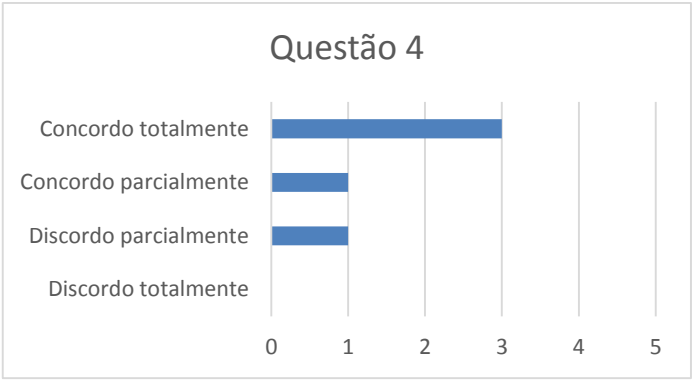


Figura 21 - Resultado da questão 4 do *checklist*

- Questão 5: Pode-se utilizar de 10% a 20% dos Sprints focados em pagamento da dívida técnica:

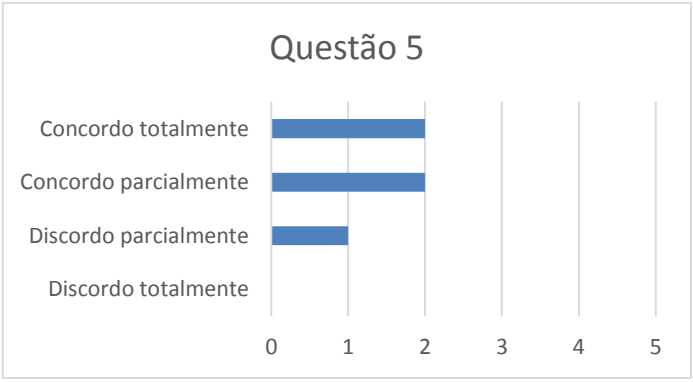


Figura 22 - Resultado da questão 5 do *checklist*

- Questão 6: O termo “dívida técnica” ou “débito técnico” pode ser utilizado para promover a comunicação entre os membros do time:

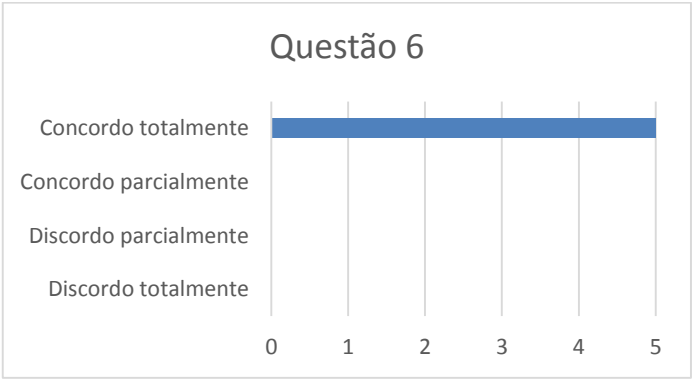


Figura 23 - Resultado da questão 6 do *checklist*

- Questão 7: O termo “dívida técnica” ou “débito técnico” pode ser utilizado para promover a comunicação com equipes de negócio ou clientes:

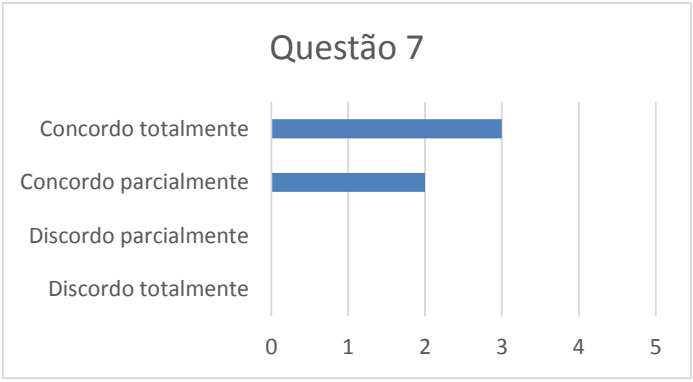


Figura 24 - Resultado da questão 7 do *checklist*

- Questão 8: O time de desenvolvimento pode ser responsável por identificar, estimar, priorizar e comunicar a dívida técnica e participar na tomada de decisão em se aceitar ou pagar a dívida técnica:

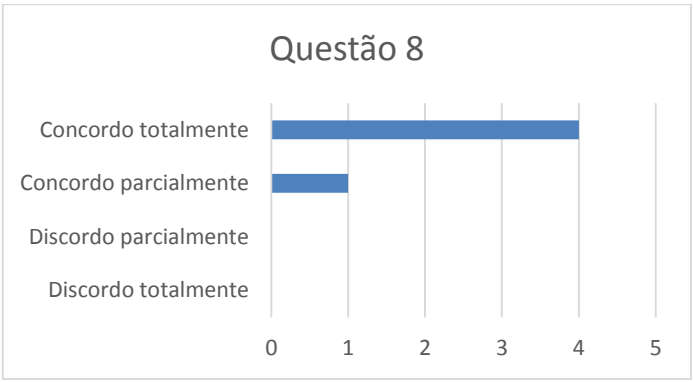


Figura 25 - Resultado da questão 8 do *checklist*

- Questão 9: O Product Owner pode ser responsável por acatar as decisões de pagamento da dívida pelo time, permitir a introdução de user stories do Technical Backlog levantadas pelo time no Sprint de forma a evitar que a dívida se acumule ao longo do tempo e participar na tomada de decisão em se aceitar ou pagar a dívida técnica:

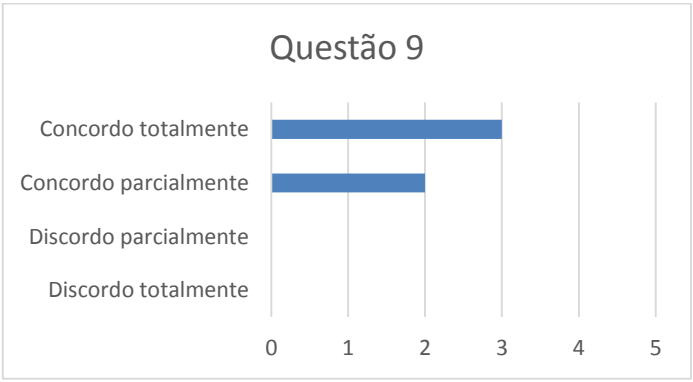


Figura 26 - Resultado da questão 9 do *checklist*

- Questão 10: O Scrum Master pode ser responsável por facilitar a comunicação entre a equipe e o PO, monitorar o nível de dívida técnica e participar na tomada de decisão em se aceitar ou pagar a dívida técnica:

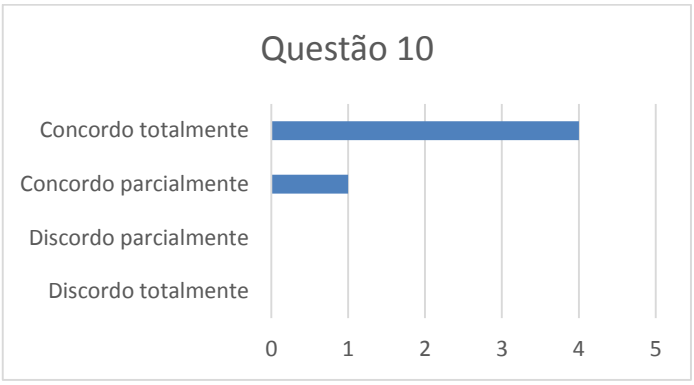


Figura 27 - Resultado da questão 10 do *checklist*

Com exceção da questão 5, todas as outras tiveram concordância total pela maioria. Entretanto, o pequeno número de respondentes não nos permite formar conclusões mais fortes do que apresentar indícios positivos de que a proposta de extensão do Scrum pode ser útil para apoiar a gestão da dívida técnica em projetos de desenvolvimento de software que utilizam o Scrum.

Pode-se observar que há um certo otimismo em relação à aplicação da proposta, devendo, no entanto, ser avaliada em contextos reais. Desta forma poderá ser feita uma comparação entre as respostas anteriores e posteriores à aplicação da extensão proposta, levantando alguns aspectos que possam ser incorporados à proposta bem como itens a serem melhorados.

6 CONCLUSÃO

A dívida técnica é um tema que tem recebido bastante atenção recentemente por tratar de um problema importante na engenharia de software: o foco excessivo na entrega de funcionalidades relevantes para o cliente pode levar a uma situação onde pode se tornar impraticável alcançar exatamente esse objetivo, à medida que crescem os problemas de implementação [SCH13a]. Por se tratar de uma metáfora, muitos conceitos são de fácil compreensão e fáceis de explicar. Por esse motivo, há uma vasta discussão sobre o assunto na internet, muitas vezes apenas refletindo opiniões ou palpites que não são avaliados antes de serem publicados [SPI13]. Vários estudos científicos, no entanto, buscam esclarecer os conceitos [TOM13] e definir os limites da dívida técnica [SCH13a], bem como definir formalismos para aplicar em projetos de desenvolvimento de software iterativos, incluindo abordagens ágeis [SCH13b].

O objetivo desta dissertação, conforme apresentado na seção 1.1 deste trabalho foi alcançado através do planejamento e execução de um estudo seguindo o método de pesquisa e passos propostos. O presente trabalho apresentou inicialmente um levantamento de desafios e oportunidades de pesquisa em dívida técnica apontados por trabalhos relevantes nessa área, ao mesmo tempo visando proporcionar um embasamento teórico para futuras pesquisas. Os estudos forneceram também conteúdo para uma melhor compreensão sobre o tema, tais como definições e conceitos, precedentes e consequências, quantificação, comunicação e gerenciamento da dívida técnica, bem como ferramental disponível. Logo a seguir, o estudo de campo foi fundamental para entender como a dívida técnica é gerenciada em projetos de desenvolvimento, servindo assim como instrumento para a proposta de extensão do Scrum de forma a contemplar aspectos que permitam manter a dívida técnica visível e, portanto, gerenciável.

Com base nos resultados obtidos a partir da análise dos trabalhos, existe ainda um grande espaço a ser explorado na pesquisa em dívida técnica, onde grande parte dos trabalhos revela uma necessidade de evidências empíricas que apoiem as abordagens propostas. Aliado a isso, está a necessidade de ferramentas confiáveis que apoiem os principais envolvidos na tarefa de controlar a dívida técnica e estimar possíveis consequências de decisões em termos de dívida técnica e *time-to-market* [FAL13].

6.1 Contribuições da pesquisa

Ao longo do processo de elaboração da proposta foram descritas as características da dívida técnica, os desafios e oportunidades na área (Capítulo 2) e como a dívida técnica é gerenciada em projetos de desenvolvimento de software reais através da análise dos dados coletados no estudo de campo (Capítulo 4). Adicionalmente, foi elaborada uma proposta de extensão do Scrum para apoiar a gestão da dívida técnica, bem como uma avaliação inicial da proposta de extensão do Scrum (Capítulo 5).

A presente pesquisa tem como principal contribuição a proposta de uma extensão do Scrum para incluir o gerenciamento da dívida técnica. Esta proposta foi desenvolvida a partir das evidências da literatura e da experiência dos profissionais que lidam com gerenciamento de projetos na indústria. Para a academia, a proposta agrega conhecimentos obtidos através de estudos empíricos na área de Engenharia de Software com foco no gerenciamento da dívida técnica, tema de atual interesse na comunidade científica. A extensão proposta também serve como um referencial teórico-prático para a indústria buscando um gerenciamento de projetos mais proativo e eficaz no que diz respeito à gestão da dívida técnica.

6.2 Limitações da pesquisa

Esta pesquisa possui algumas limitações relacionadas à sua metodologia. Tais limitações serão detalhadas a seguir de acordo com a etapa de pesquisa a qual pertencem.

Primeiramente, a revisão de literatura sobre dívida técnica foi baseada na técnica *snowballing*. No entanto, esta revisão possui algumas limitações. A primeira é devido à busca por artigos se dar somente na base de dados Scopus. É possível que outras bases de dados não utilizadas neste trabalho contenham alguns dos artigos não acessíveis mencionados na seção 4.1. Portanto, pode haver uma perda de informação sobre o assunto devido a essa limitação. Outra limitação está relacionada à relativa novidade em se utilizar a técnica *snowballing* como método principal de se encontrar literatura relevante em uma determinada área [JAL12]. Apesar disso, foi aplicado tanto a abordagem de *backward snowballing* quanto *forward snowballing* como sugerido em [JAL12].

Ainda sobre a revisão de literatura, existe a limitação dada pelo viés do pesquisador durante a seleção de artigos relevantes. Além disso, a revisão foi efetuada por apenas um

pesquisador, sendo que a ausência de um segundo revisor pode ocasionar equívocos na classificação e seleção de artigos.

Quanto à análise de ferramentas, foram consideradas apenas as ferramentas mapeadas através da revisão de literatura, sendo que podem existir ferramentas que visam analisar a dívida técnica porém não foram levadas em conta. A análise foi feita a partir da documentação disponível *online*, a qual pode mascarar algumas limitações dado o viés comercial das mesmas. Também não foram feitos testes com a utilização delas, o que se mostrou inviável pois grande parte delas possuía licença paga para utilização total de suas funcionalidades.

O estudo de campo possui algumas limitações relacionadas às restrições da metodologia adotada, sendo mitigadas através da validação de face e conteúdo do protocolo. A primeira é em relação à própria natureza no estudo de campo, a qual tem prejudicada a habilidade em se fazer interpretações consistentes dos dados resultantes [MCG94]. A análise qualitativa adotada possui algumas desvantagens como a grande quantidade de dados gerados até mesmo de poucas entrevistas, o que dificulta a identificação de temas e padrões [OAT06]. Além disso, a interpretação dos dados está sujeita ao viés do pesquisador (sua identidade, experiências, premissas e crenças) [OAT06].

A proposta de extensão do Scrum não contempla todos os elementos utilizados no estudo de campo, sendo os custos monetários da dívida técnica um exemplo. No entanto, a maioria dos elementos pôde ser explorada. Ainda, por tratar-se de uma proposta inicial, não contém todos os itens identificados na avaliação como pontos de melhoria. Também não foram feitos estudos empíricos utilizando a proposta com equipes de desenvolvimento reais, o que foi deixado como oportunidade de estudo futuro. Por fim, a avaliação da proposta ficou sujeita a um número reduzido de entrevistados. Portanto é possível que existam outros pontos de vista não explorados nessa avaliação.

6.3 Trabalhos futuros

Através dos estudos analisados e apresentados nesta dissertação, foi permitido identificar inúmeras oportunidades de pesquisa. Além das oportunidades descritas no capítulo 2, outras oportunidades poderão ser exploradas futuramente através de novos estudos empíricos. A dívida técnica é uma área em crescente interesse tanto pela

comunidade científica quanto pela indústria, tendo um grande potencial de trabalhos futuros por meio da cooperação entre os parceiros.

A partir do estudo de campo foram identificadas algumas oportunidades de estudos futuros com relação a quantificar custos econômicos da dívida técnica. Tais estudos são necessários para verificar a real necessidade e utilidade de se atribuir valores monetários à dívida técnica.

Assim como custos de projeto, as atividades de gerenciamento e pagamento da dívida técnica irão inevitavelmente gerar custos para o projeto. Porém, dada a natureza evolutiva do software, é possível questionar se estes valores, considerando a dívida devidamente gerenciada no projeto, representam um real prejuízo ou simplesmente fazem parte do ciclo de vida normal de desenvolvimento do software e portanto devem ser considerados como horas normais de desenvolvimento ou entrar em um fundo de contingência. Há também alguns desafios identificados para esta área no estudo como a falta de acesso a dados financeiros dos projetos, falta de ferramentas adequadas para obter este tipo de informação e subjetividade das atividades de desenvolvimento de software. A dívida técnica também não é considerada na previsão de custos de projeto. Existem ainda custos não mensuráveis relacionados à imagem da marca ou da organização e a credibilidade do produto.

A proposta de extensão do Scrum apresentada no capítulo 5 precisa ser avaliada de forma mais consistente através de um estudo empírico utilizando equipes de desenvolvimento reais. Dessa forma, novas evidências e melhorias podem ser incorporadas à proposta, gerando assim uma abordagem mais abrangente para o gerenciamento da dívida técnica.

REFERÊNCIAS BIBLIOGRÁFICAS

[BAE98] Baetjer, Jr., H. “Software as Capital: An Economic Perspective on Software Engineering”. IEEE Computer Society Press, 1998, 206p.

[BAS12] Bass, L. et al. “Software Architecture in Practice”. Addison-Wesley, 2012, 640p.

[BEC00] Beck, K. “Extreme Programming Explained: Embrace Change”. Addison-Wesley, 2000, 190p.

[BEC13] Beck, K. et al. “The Manifesto for Agile Software Development”. Capturado em: http://www.agilealliance.org/the-alliance/the-agile-manifesto/, Setembro 2013.

[BRO10] Brown et al. “Managing technical debt in software-reliant systems”. In: Proceedings of the 18th FSE/SDP Workshop on Future of Software Engineering Research, 2010, pp. 47-52.

[COD13] Codabux, Z.; Williams, B. “Managing technical debt: An industrial case study”. In: 4th International Workshop on Managing Technical Debt, MTD 2013 – Proceedings, 2013, 4p.

[CUN12] Cunningham, W. “Complexity as Debt”. Capturado em: http://www.c2.com/cgi/wiki?ComplexityAsDebt, Outubro 2013.

[CUN92] Cunningham, W. “The WyCash Portfolio Management System”. Capturado em: http://c2.com/doc/oopsla92.html, Outubro 2013.

[CUR12] Curtis, B.; Sappidi, J.; Szynkarski, A. "Estimating the Size, Cost, and Types of Technical Debt". In: Managing Technical Debt (MTD) - Third International Workshop on TD, 2012, pp. 49-53.

[ELS13] Elsevier B.V. “Scopus”. Capturado em: http://www.scopus.com/, Outubro 2013.

[FAL13] Falessi, D.; Shaw, M.A.; Shull, F.; Mullen, K.; Keymind, M.S. “Practical Considerations, Challenges and Requirements of Tool Support for Managing Technical Debt”. In: 4th International Workshop on Managing Technical Debt, MTD 2013 – Proceedings 6608673, 2013, pp. 16-19.

[FOW09] Fowler, M. “Technical Debt Quadrant”. Capturado em: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html, Outubro 2013.

[FOW99] Fowler, M. “Refactoring: Improving the Design of Existing Code”. Addison-Wesley, 1999, 455p.

[GAL15] Galeote, Sidney. “Modelos prescritivos para o desenvolvimento de software”. Capturado em: http://www.galeote.com.br/blog/2012/06/modelos-prescritivos-para-o-desenvolvimento-de-software/, Março 2015.

[GOF10] Goff, Allan. “Quantify Technical Debt”. Capturado em: http://www.c2.com/cgi/wiki?QuantifyTechnicalDebt, Outubro 2013.

[GUO11] Guo, Y.; Seaman, C. “A portfolio approach to technical debt”. In: 2nd International Workshop on Managing Technical Debt, 2011, pp. 31-34.

[HAZ10] Hazrati, Vikas. “Monetizing the Technical Debt”. Capturado em: http://www.infoq.com/news/2010/03/monetizing-technical-debt, Outubro 2013.

[IBM15] IBM. “Rational Team Concert”. Capturado em: http://www-03.ibm.com/software/products/pt/rtc, Fevereiro 2015.

[IEE90] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Standard 610.12-1990, 1990.

[IFS15] IF-SC. “Ciclo de Vida Iterativo e Incremental”. Capturado em: http://wiki.sj.ifsc.edu.br/wiki/index.php/Ciclo_de_Vida_Iterativo_e_Incremental, Setembro de 2015.

[JAL12] Jalali, S.; Wohlin, C. "Systematic Literature Studies: Database Searches vs. Backward Snowballing". In: Proceedings of the ACM-IEEE International symposium on Empirical software engineering and measurement, 2012, pp. 29-38.

[KEE80] Keen, P.G.W. “Decision support systems: A research perspective”. Center for Information Systems Research, Sloan School of Management, Massachusetts Inst. of Technol., 1980, 54p.

[KIT04] Kitchenham, B. “Procedures for Performing Systematic Reviews”. Technical Report, Keele University, 2004, 33p.

[KLE05] Klein, J. “How does the architect’s role change as the software ages?”. In: 5th Working IEEE/IFIP Conference on Software Architecture, 2005, p. 141.

[KLI11] Klinger, T.; Tarr, P.; Wagstrom, P.; Williams, C. “An enterprise perspective on technical debt”. In: Proceedings - International Conference on Software Engineering, 2011, 4p.

[KRU08] Kruchten, P. “What Colour Is Your Backlog?”. Capturado em http://philippe.kruchten.com/talks, Outubro 2013.

[KRU12] Kruchten, P.; Nord, R.L.; Ozkaya, I. “Technical Debt: from metaphor to theory and practice”. IEEE Software 29 (6), art. no. 6336722 , 2012, pp. 18-21.

[KTA10] Ktata, O.; Lévesque, G. “Designing and implementing a measurement program for scrum teams: what do agile developers really need and want?”. In: 3rd Conference on Computer Science and Software Engineering, 2010, pp. 101-107.

[MCC07] McConnell, Steve. “Technical Debt”. Capturado em: http://www.construx.com/10x_Software_Development/Technical_Debt/, Outubro 2013.

[MCG94] McGrath, J. E. “Methodology Matters: Doing Research in the Behavioral and Social Sciences”. Morgan Kaufmann Publishers, 1994, pp. 152-169.

[MCI68] Mcilroy, M. D. “Mass-produced software components”. In: Proc. NATO Conf. on Software Engineering, Garmisch, Germany: Springer-Verlag, 1968, pp. 138-155.

[MIL80] Mills et al. “The Management of Software Engineering”, *IBM Systems Journal*, vol. 19-4, Dec 1980, p. 414.

[MOU15] Mountain Goat Software. “Scrum Overview for Agile Software Development”. Capturado em: https://www.mountaingoatsoftware.com/agile/scrum/overview, Setembro 2015.

[NAU69] Naur, P.; Randell, B. “Software Engineering”. In: Report on a Conference Sponsored by the NATO Science Committee, 1969, p. 120.

[NOR12] Nord, R.; Ozakaya, I.; Kruchten, P.; Gonzalez, M. “In Search of a Metric for Managing Architectural Technical Debt”. In: Submitted to WICSA1ECSA 2012 conference, 2012, 100p.

[NUG11] Nugroho, A.; Visser, J.; Kuipers, T. "An Empirical Model of Technical Debt and Interest". In: Proceedings of the 2nd Workshop on Managing Technical Debt, Waikiki, Honolulu, HI, USA, 2011, pp. 1-8.

[OAT06] Oates, Briony J. “Researching Information Systems and Computing”. Sage, 2006, 341p.

[OGA91] Ogawa, R.T.; Malen, B. “Towards rigor in reviews of multivocal literatures: applying the exploratory case study method”. *Review of Educational Research*, vol. 61-3, Autumn 1991, 265–286.

[ORH13] Orhan, Lemi. “Technical Debt – Do Not Underestimate the Danger”. Capturado em: http://www.ontechnicaldebt.com/blog/technical-debt-do-not-underestimate-the-danger/, Outubro 2013.

[PAN13] Pant, R. “Organizing a Digital Technology Department of Medium Size in a Media Company”. Capturado em: http://www.rajiv.com/blog/2009/03/17/technology-department/, Dezembro 2013.

[PRE11] Pressman, R. S. “Software engineering: a practitioner's approach”. New York: McGraw-Hill, 2011, 7ª ed., 926p.

[RAM13] Ramasubbu, N.; Kemerer, C.F. “Towards a Model for Optimizing Technical Debt in Software Products”. In: 4th International Workshop on Managing Technical Debt, MTD 2013 – Proceedings 6608679, 2013, pp. 51-54.

[RIE09] Ries, E. “Embrace technical debt”. Capturado em: http://www.startuplessonslearned.com/2009/07/embrace-technical-debt.html, Outubro 2013

[ROY70] Royce, W. W. “Managing the Development of Large Software Systems: Concepts and Techniques”. In: Technical Papers of Western Electronic Show and Convention (WesCon), 1970, pp. 328-338.

[SCH13a] Schmid, K. “On the limits of the technical debt metaphor some guidance on going beyond”. In: 4th International Workshop on Managing Technical Debt, MTD 2013 – Proceedings 6608681, 2013, pp. 63-66

[SCH13b] Schmid, K. “A Formal Approach to Technical Debt Decision Making”. In: QoSA 2013 - Proceedings of the 9th International ACM Sigsoft Conference on the Quality of Software Architectures, 2013, pp. 153-162.

[SCW01] Schwaber, K.; Beedle, M. “Agile Software Development with Scrum”. Englewood Cliffs, NJ: Prentice Hall, 2001, 158p.

[SCW13] Schwaber, K.; Sutherland, J. “The Scrum Guide”. Capturado em: http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf, Fevereiro 2015.

[SEA11a] Seaman, C.; Zazworka, N. “Identifying and Managing Technical Debt”. Capturado em: http://www.slideshare.net/zazworka/identifying-and-managing-technical-debt, Novembro 2013.

[SEA11b] Seaman, C.; Guo, Y.; Izurieta, C.; Cai, Y.; Zazwirja, N.; Shull, F.; Vetro, A. “Using technical debt data in decision making: potential decision approaches”. In: Second International Workshop on Managing Technical Debt, ICSE 2011, Waikiki, Honolulu, Hawaii, USA, 2011, 48p.

[SHR10] Shriver, R. “Seven Strategies for Technical Debt”. Capturado em: http://www.theagileengineer.com/public/Home/Home files/TechnicalDebt_published.pdf, Novembro 2013.

[SLI08] Slinker, G. “More on code debt”. Capturado em: http://digerati-illuminatus.blogspot.com/2008/03/more-on-code-debt.html, Setembro 2013.

[SOM11] Sommerville, I. “Software Engineering”. Boston: Pearson, 2011, 773p.

[SON13a] SonarSource S.A. “SonarQube”. Capturado em: http://www.sonarqube.org/, Outubro 2013.

[SON13b] SonarSource S.A. “Technical Debt Calculation”. Capturado em: http://docs.codehaus.org/display/SONAR/Technical+Debt+Calculation, Outubro 2013.

[SOU15] Souza, Hugo Vieira. “Engenharia de Software”. Capturado em: http://hugovlsouza.com/unipe/~DET679/aulas/03%20-%20Processos%20de%20Software_ciclo%20de%20vida%20de%20software%20e%20modelos%20de%20processos.pdf, Setembro 2015.

[SPI13] Spínola, R.O.; Vetrò, A.; Zazworka, N.; Seaman, C.; Shull, F. “Investigating technical debt folklore: Shedding some light on technical debt opinion”. In: 2013 4th International Workshop on Managing Technical Debt, MTD 2013 – Proceedings 6608671, 2013, pp. 1-7.

[STA97] Stapleton, J. “DSDM Dynamic Systems Development Method”. Addison-Wesley, 1997, 163p.

[STO10] Stopford, A. “Technical Debt”. Capturado em: http://www.weblogs.asp.net/astopford/archive/2010/07/19/technical-debt.asp, Agosto 2011.

[TAE13] The Agile Executive. “Can you afford the software you are developing?”. Capturado em: http://theagileexecutive.com/2009/02/01/can-you-afford-the-software-you-are-developing/, Outubro 2013.

[THI10] Thibodeau, Patrick. “The new push to measure software's true cost”. Capturado em: http://www.computerworld.com/s/article/9190198/The_new_push_to_measure_software_s_true_cost?pageNumber=1, Outubro 2013.

[TOM11] Tom, E. “A Consolidated Understanding of Technical Debt”. Honours Thesis, School of Information Systems, Technology and Management, University of New South Wales, Sydney, Australia, 2011, 13p.

[TOM13] Tom et al. “An exploration of technical debt”, *Journal of Systems and Software*, vol. 86-6, 2013, pp. 1498-1516.

[TOR11] Torkar, R.; Minoves, P.; Garrigós, J. “Adopting free/libre/open source software practices, techniques and methods for industrial use”, *Journal of the Association for Information Systems*, vol 12, 2011, pp. 88–122.

[WAI07] Wainer, J. “Métodos de pesquisa quantitativa e qualitativa para a Ciência da Computação”. In: JAI 2007 - Jornada de Atualização em Informática, Anais do XXVII Congresso da Sociedade Brasileira de Computação, 2007, 42p.

[WESL12] Wiklund, K.; Eldh, S.; Sundmark, D.; Lundqvist, K. “Technical debt in test automation”. In: Third International Workshop on Managing Technical Debt, ICSE 2012, Zurich, Switzerland, 2012, 6p.

[ZAZ11a] Zazworka, N. “Prioritizing design debt: investment opportunities”. In: 2nd International Workshop on Managing Technical Debt, ICSE 2011, Waikiki, Honolulu, Hawaii, USA, 23 May, 2011, pp. 39-42.

[ZAZ11b] Zazworka, N.; Shaw, M. A.; Shull, F.; Seaman, C. “Investigating the impact of design debt on software quality”. In: Proceedings - International Conference on Software Engineering, 2011, pp. 17-23.

[ZAZ13] Zazworka, N.; Spínola, R.O.; Vetro, A.; Shull, F.; Seaman, C. “A case study on effectively identifying technical debt”. In: ACM International Conference Proceeding Series, 2013, pp. 42-47.

APÊNDICE A

PROTOCOLO PARA ESTUDO DE CAMPO

Protocolo de Estudo de Campo: Dívida Técnica no Desenvolvimento de Software: Um Estudo Empírico

Objetivo

Identificar como a dívida técnica é percebida e se é gerenciada ou não em projetos de desenvolvimento de software na indústria. A partir deste estudo também se pretende elaborar uma abordagem preliminar para gerenciamento da dívida técnica em projetos de desenvolvimento de software.

Característica-chave do método de pesquisa

Este é um roteiro para um estudo de campo cujo método de coleta de dados é a entrevista semiestruturada com questões abertas. O objetivo é fazer uma análise qualitativa através de técnicas de análise de conteúdo, buscando identificar práticas, tipos de dívida técnica e ferramentas que de alguma forma visam tratar determinados aspectos da dívida técnica na indústria.

Unidades de análise

Profissionais de gerenciamento de projetos de desenvolvimento de software em empresas de TI nacionais ou multinacionais sediadas no Brasil.

Organização desse Protocolo

O protocolo será organizado conforme segue:

1. **Procedimentos**

a. Levantamento das questões e estruturação do guia para a entrevista	
Participantes	Ciro Santos
Data	Agosto de 2014
Local	FACIN PUCRS – Faculdade de Informática da PUCRS

b. Revisão do guia para a entrevista	
Participantes	Prof. Dr. Rafael Prikladnicki
Data	Setembro de 2014
Local	TECNOPUC - Parque Científico e Tecnológico da PUCRS

c. Autorização das empresas participantes	
Participantes	
Data	
Local	

d. Validação de face e conteúdo	
Participantes	Bernardo Estácio
Data	Outubro de 2014
Local	FACIN PUCRS

e. Pré-teste	
Participantes	João Henrique Pinto
Data	Outubro de 2014

Local	FACIN PUCRS

f. Aplicação das entrevistas – Questões referentes aos projetos
Participantes
Data
Local

2. Escolha das pessoas entrevistadas

Respondentes:

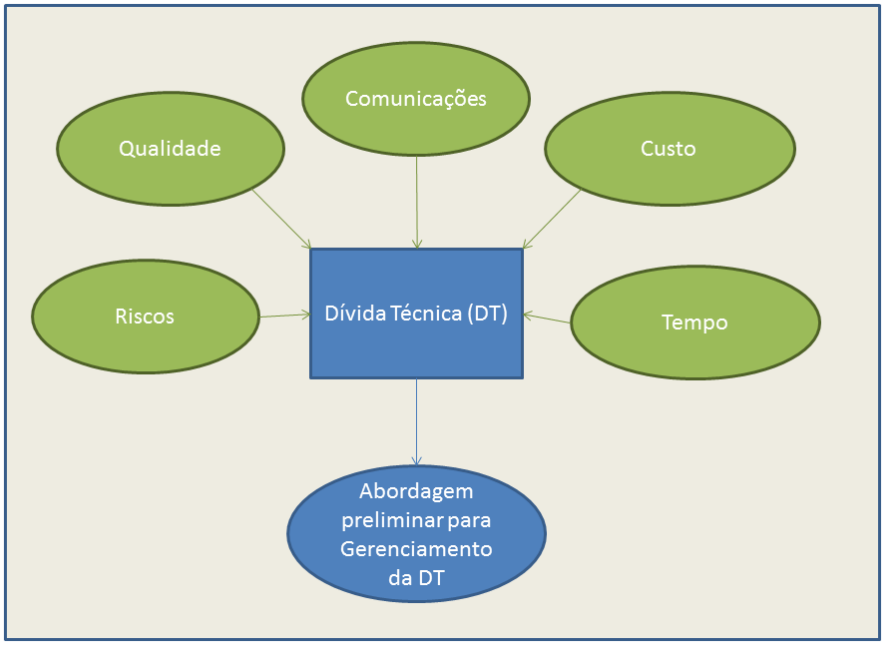
a. Gerentes de projetos de desenvolvimento de software e *Scrum Masters*.

3. Outros recursos utilizados

a. Recursos tecnológicos
   i. Microsoft Excel e Word.
b. Recursos materiais
   i. Uma sala de reuniões
   ii. Um gravador para gravar as entrevistas
   iii. Papel e caneta

4. Modelo de estudo e dimensões da pesquisa

A seguir são apresentados graficamente os principais aspectos envolvidos no desenvolvimento deste trabalho. O esquema baseia-se na perspectiva de algumas áreas de conhecimento do *Project Management Body of Knowledge* (PMBOK) que são afetadas pela dívida técnica.



- **Tempo**: Esta área descreve os processos relativos ao término do projeto no prazo definido no cronograma de atividades.

- **Qualidade**: Esta área descreve os processos envolvidos na garantia de que o projeto irá satisfazer os objetivos para os quais foi realizado.
- **Comunicações**: Esta área descreve os processos relativos à geração, coleta, disseminação, armazenamento e destinação final das informações do projeto de forma oportuna e adequada.
- **Custo**: Esta área descreve os processos envolvidos em planejamento, estimativa, orçamentação e controle de custos, de modo que o projeto termine dentro do orçamento aprovado.
- **Riscos**: Esta área descreve os processos relativos à realização do gerenciamento de riscos em um projeto.

Coleta de dados

Entrevistas semiestruturadas com questões abertas.

5. Análise de dados

Após a transcrição das entrevistas será realizada uma análise dos dados coletados.

6. Dimensões e questões do guia para entrevista semiestruturada

- Dados demográficos dos respondentes:

	Questões
Dados Demográficos	Nome:
	Organização:
	País de origem da organização:
	Porte da organização (nro. aprox. de funcionários):
	Presença da organização: [ ] Nacional [ ] Multinacional
	Tempo de experiência com desenvolvimento de software:
	Tempo de experiência no gerenciamento de projetos:
	Característica do projeto atual: [ ] Desenvolvimento [ ] Manutenção [ ] Outros _____
	Processo de desenvolvimento utilizado: [ ] Tradicional _____ [ ] Adaptativo _____

- Questões referentes a projetos:

	Questões
Ferramental	1. Quais ferramentas você utiliza para identificar a dívida técnica em potencial, ou seja, aquela a qual se não for devidamente gerenciada existe um risco de se tornar um problema em seu projeto à medida que o software evolui?
	2. Quais ferramentas você utiliza para monitorar o nível da dívida técnica em seu projeto?

	Questões
Qualidade	3. Quais processos utilizados para qualificar o código até que ele seja dado como aceito para entrar em produção?
	4. Como é feito o rastreamento de decisões arquiteturais no projeto?

	Questões
Custo	5. Como você estima o custo monetário da dívida técnica potencial, ou seja, aquela que poderá gerar algum retrabalho no futuro se não tratada adequadamente?
	6. Como você estima o custo monetário da dívida técnica efetiva, isto é, aquela que já se tornou um problema e precisa ser paga?

	Questões
Tempo	7. Como você estima as atividades de pagamento da divida técnica, ou seja, que não agregam valor diretamente ao negócio, mas que são necessárias para a qualidade estrutural do software e visam proporcionar melhorias de design ou arquiteturais para suportar o desenvolvimento de novas funcionalidades?
	8. Quais são as estratégias utilizadas para o pagamento da dívida técnica no seu projeto (por ex.: iterações mínimas, tarefas de buffer, releases de limpeza, etc.).

	Questões
Comunicações	9. De que forma a equipe utiliza a metáfora da dívida técnica para promover a comunicação entre os seus membros?
	10. De que forma a equipe utiliza a metáfora da dívida técnica para promover a comunicação com equipes de negócio ou clientes?

	Questões
Riscos	11. Quais “tipos de dívida” considera ter maior impacto se não gerenciada? Por exemplo: Dívida de Código, Dívida de Design/arquitetural, Dívida de Ambiente, Dívida de Distribuição de conhecimento/documentação, Dívida de Teste.
	12. Como ocorre a tomada de decisão em se assumir certos tipos de dívida, como por exemplo, abdicar de princípios puros de design por motivos estratégicos para reduzir o risco de entrega do projeto?

	Questões
Opinião	13. Qual a sua opinião sobre o gerenciamento da divida técnica?
	14. Em sua opinião, quais os benefícios em tornar a dívida técnica visível?
	15. Quais as dificuldades em se gerenciar explicitamente a dívida técnica?
	16. Como são contornadas as dificuldades em se gerenciar a dívida técnica no projeto?

APÊNDICE B

PROTOCOLO PARA REVISÃO DA PROPOSTA DE EXTENSÃO DO SCRUM PARA DÍVIDA TÉCNICA

Dívida Técnica no Desenvolvimento de Software: Um Estudo Empírico

Objetivo

Revisar a proposta de extensão do Scrum para dívida técnica resultante do estudo de campo previamente realizado, cujo objetivo foi identificar como a dívida técnica é percebida e gerenciada em projetos de desenvolvimento de software na indústria. A proposta de extensão visa uma abordagem preliminar para gerenciamento da dívida técnica em projetos de desenvolvimento de software.

Unidades de análise

Uma amostra dos profissionais participantes do estudo de campo realizado previamente como parte desta pesquisa.

Organização desse Protocolo

O protocolo será organizado conforme segue:

7. Procedimentos

g. Levantamento das questões e estruturação do guia para a entrevista	
Participantes	Ciro Santos
Data	Fevereiro de 2015
Local	FACIN PUCRS – Faculdade de Informática da PUCRS

h. Revisão do guia para a entrevista	
Participantes	Prof. Dr. Rafael Prikladnicki
Data	
Local	TECNOPUC - Parque Científico e Tecnológico da PUCRS

i. Validação de face e conteúdo
Participantes
Data
Local

j. Pré-teste
Participantes
Data
Local

k. Aplicação das entrevistas – Questões referentes à proposta de extensão do Scrum
Participantes
Data
Local

8. Escolha das pessoas entrevistadas

Respondentes:

b. Gerentes de projetos de desenvolvimento de software e *Scrum Masters* que participaram do estudo de campo.

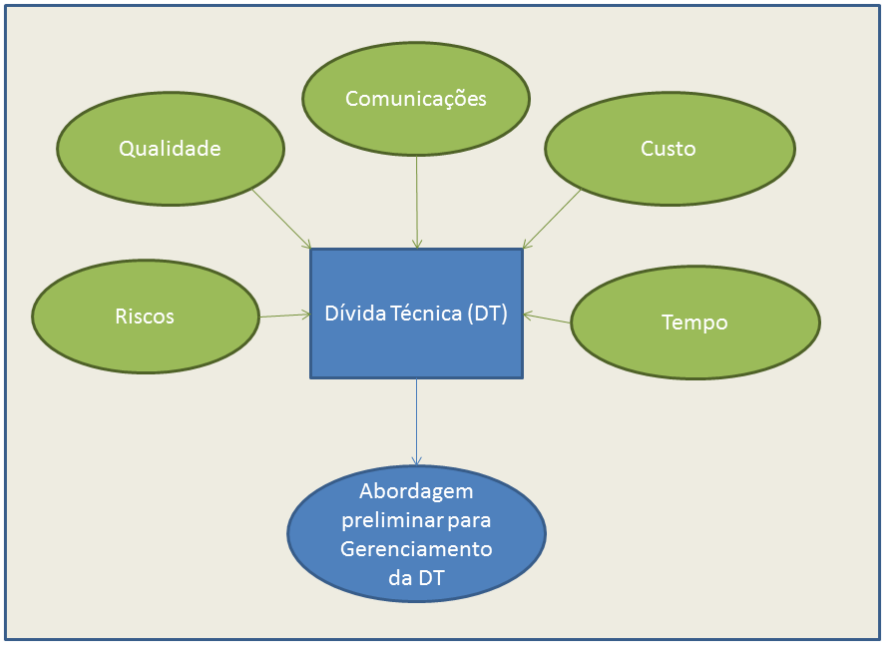
9. Outros recursos utilizados

a. Recursos tecnológicos
   i. Microsoft Excel e Word.

b. Recursos materiais
   i. Uma sala de reuniões
   ii. Um gravador para gravar as entrevistas
   iii. Papel e caneta

10. Modelo de estudo e dimensões da pesquisa

A seguir são apresentados graficamente os principais aspectos envolvidos no desenvolvimento deste trabalho. O esquema baseia-se na perspectiva de algumas áreas de conhecimento do *Project Management Body of Knowledge* (PMBOK) que são afetadas pela dívida técnica.



- **Tempo**: Esta área descreve os processos relativos ao término do projeto no prazo definido no cronograma de atividades.
- **Qualidade**: Esta área descreve os processos envolvidos na garantia de que o projeto irá satisfazer os objetivos para os quais foi realizado.
- **Comunicações**: Esta área descreve os processos relativos à geração, coleta, disseminação, armazenamento e destinação final das informações do projeto de forma oportuna e adequada.
- **Custo**: Esta área descreve os processos envolvidos em planejamento, estimativa, orçamentação e controle de custos, de modo que o projeto termine dentro do orçamento aprovado.

- **Riscos**: Esta área descreve os processos relativos à realização do gerenciamento de riscos em um projeto.

Coleta de dados

Entrevistas semiestruturadas com questões abertas.

11. Análise de dados

Após a transcrição das entrevistas será realizada uma análise qualitativa dos dados coletados.

12. Dimensões e questões do guia para entrevista semiestruturada

- Dados dos respondentes:

	Questões
Dados Demográficos	Nome:
	Organização:

- Questões referentes à proposta de extensão do Scrum:

	Questões
Opinião	17. Qual a sua opinião geral sobre a extensão proposta?
	18. Quais os pontos fortes da proposta?
	19. Quais os pontos fracos da proposta?
	20. Quais pontos da proposta não ficaram bem claros?
	21. Como você avalia a viabilidade de utilização desta extensão do Scrum em sua organização ou em projetos reais?
	22. Quais sugestões de melhoria você dá para a proposta?

13. Checklist

- A identificação da dívida técnica pode ser realizada através da criação de um *Technical Backlog*:

( ) Concordo totalmente ( ) Concordo parcialmente ( ) Discordo parcialmente ( ) Discordo totalmente

- O monitoramento da dívida técnica pode ser realizado através da criação de um *Technical Burndown Chart*:

( ) Concordo totalmente ( ) Concordo parcialmente ( ) Discordo parcialmente ( ) Discordo totalmente

- O rastreamento de decisões arquiteturais pode ser realizado através da criação de um documento de captura de mudanças arquiteturais:

  ( ) Concordo totalmente ( ) Concordo parcialmente ( ) Discordo parcialmente ( ) Discordo totalmente

- Os itens do *Technical Backlog* podem ser estimados utilizando-se técnicas como o *Planning Poker:*

  ( ) Concordo totalmente ( ) Concordo parcialmente ( ) Discordo parcialmente ( ) Discordo totalmente

- Pode-se utilizar de 10% a 20% dos Sprints focados em pagamento da dívida técnica:

  ( ) Concordo totalmente ( ) Concordo parcialmente ( ) Discordo parcialmente ( ) Discordo totalmente

- O termo “dívida técnica” ou “débito técnico” pode ser utilizado para promover a comunicação entre os membros do time:

  ( ) Concordo totalmente ( ) Concordo parcialmente ( ) Discordo parcialmente ( ) Discordo totalmente

- O termo “dívida técnica” ou “débito técnico” pode ser utilizado para promover a comunicação com equipes de negócio ou clientes:

  ( ) Concordo totalmente ( ) Concordo parcialmente ( ) Discordo parcialmente ( ) Discordo totalmente

- O time de desenvolvimento pode ser responsável por identificar, estimar, priorizar e comunicar a dívida técnica e participar na tomada de decisão em se aceitar ou pagar a dívida técnica:

  ( ) Concordo totalmente ( ) Concordo parcialmente ( ) Discordo parcialmente ( ) Discordo totalmente

- O *Product Owner* pode ser responsável por acatar as decisões de pagamento da dívida pelo time, permitir a introdução de *user stories* do *Technical Backlog* levantadas pelo time no Sprint de forma a evitar que a dívida se acumule ao longo do tempo e participar na tomada de decisão em se aceitar ou pagar a dívida técnica:

  ( ) Concordo totalmente ( ) Concordo parcialmente ( ) Discordo parcialmente ( ) Discordo totalmente

- O *Scrum Master* pode ser responsável por facilitar a comunicação entre a equipe e o PO, monitorar o nível de dívida técnica e participar na tomada de decisão em se aceitar ou pagar a dívida técnica:

( ) Concordo totalmente ( ) Concordo parcialmente ( ) Discordo parcialmente ( ) Discordo totalmente

APÊNDICE C

ANÁLISE DE FERRAMENTAS MAPEADAS NA REVISÃO DE LITERATURA

Ferramenta	**CAST Application Intelligence Platform (AIP)**
Tipo de Dívida	Dívida de código
Licença	Pago (requer contato para obter informações de licenciamento e preços)
Variantes	Não possui
Funcionalidades	-Identificação -Visualização -Quantificação -Monitoramento
Vantagens	-Tempo de *set up* mínimo -Diferentes *dashboards* para diferentes tipos de audiência -Multi-plataforma, multi-camada e multi-linguagem -*Benchmarking* – grande repositório de aplicações analisadas
Desvantagens	-Não foi mencionado cobertura de testes automatizados

Ferramenta	**SonarQube**
Tipo de Dívida	Dívida de código
Licença	GNU LGPL v.3 para Community Edition, o restante das edições é pago
Variantes	Community, Professional, Enterprise, Ultimate
Funcionalidades	-Identificação -Visualização -Quantificação -Monitoramento
Vantagens	-Vários dashboards disponíveis "*out of the box*" -Mais de 20 linguagens suportadas -Pre-commit checks -Extensibilidade, + de 60 plugins disponíveis -Integração com componentes de ALM padrão
Desvantagens	-Plugin de Dívida Técnica disponível somente nas edições pagas

Ferramenta	Lattix
Tipo de Dívida	Dívida de código Dívida arquitetural
Licença	Pago (licenças de avaliação por tempo limitado para uso não-comercial)
Variantes	Lattix Architect, Lattix Analyst, Lattix Web
Funcionalidades	-Identificação -Visualização -Monitoramento
Vantagens	-Permite criar modelos de dependência dos sistemas -Possui 3 diferentes soluções direcionadas a desenvolvedores, arquitetos e gerentes -Permite fazer re-engenharia do sistema e gerar lista de tarefas
Desvantagens	- Não foi mencionado cobertura de testes automatizados -Requer uso de uma API para integração com outras ferramentas

Ferramenta	IBM Rational Team Concert
Tipo de Dívida	Dívida de código Dívida arquitetural
Licença	Pago (trial 90 dias)
Variantes	Não possui
Funcionalidades	-Planejamento de Pagamento da DT
Vantagens	-Templates e ferramentas de planejamento -Features integradas de colaboração -Oferecido também na cloud como SaaS
Desvantagens	-Não faz análise estática, apenas gerencia o ciclo de vida do projeto

Ferramenta	DebtFlag
Tipo de Dívida	Dívida arquitetural
Licença	Não comercializado
Variantes	Não possui
Funcionalidades	-Identificação -Visualização

Vantagens	-Captura observações feitas por humanos -Integração com IDE
Desvantagens	-Pode tornar o acúmulo de dívida aceitável, pois a facilidade de documentá-la pode ser menos custosa que implementar de maneira ótima -Não possui integração com ferramentas de ALM nem análise estática

Ferramenta	**Sonargraph**
Tipo de Dívida	Dívida de código Dívida arquitetural
Licença	Pago (licença de avaliação e licenças personalizadas com desconto para programadores independentes)
Variantes	Sonargraph-Architect, Sonargraph-Developer, Sonargraph-Architect Build
Funcionalidades	-Identificação -Visualização -Quantificação -Monitoramento
Vantagens	-Modelo “in-memory”, analisa grandes sistemas rapidamente -Monitoramento permanente de regras de arquitetura e estrutura de dependência no processo de desenvolvimento -Refactorings planejados em cooperação com a IDE -Integra checagem de regras de arquitetura e métricas no build do Sonar -Detecta códigos duplicados
Desvantagens	-Necessita integração com Sonar para obter métricas mais sofisticadas

APÊNDICE D

REFERÊNCIAS DOS ESTUDOS SELECIONADOS NA RSL

E01	Adolph, S., Kruchten, P., and Hall, W. "Reconciling Perspectives: A Grounded Theory of How People Manage the Process of Software Development," Journal of Software and Systems, 2012 (in press). {doi: 10.1016/j.jss.2012.01.059}
E02	Allman, E., 2012. “Managing technical debt”. ACM 5 (May (5)), 50–55.
E03	Atwood, J., 2009. “Paying down your technical debt”. In: Coding Horror, Available from: http://www.codinghorror.com/blog/2009/02/paying-down-your-technicaldebt.html (accessed 24.08.11). (Online).
E04	Black, S., Boca, P.P., Bowen, J.P., Gorman, J., Hincley, M., 2009. “Formal versus agile: survival of the fittest”. Computer 42, 37–45, *A8.
E05	Boehm, B. W. "Software Risk Management: Principles and Practices," IEEE Software, vol. 8, pp. 32, 1991.
E06	Brazier, T., 2007. Managing technical debt. ACCU, Available from: http://accu.org/index.php/journals/1301 (accessed 24.08.11). (Online).
E07	Brown et al., 2010. “Managing technical debt in software-reliant systems”. FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2013, Santa Fe, NM, United States, November 7, 2010 – November 11, 2010. Association for Computing Machinery, 47-51, *A3.
E08	Cast, 2011. “Reduce Technical Debt”, Available from: http://www.castsoftware.com/solutions/reduce-technical-debt (accessed 19.10.11). (Online).
E09	Codabux, Z., Williams, B., “Managing technical debt: An industrial case study”. In: 4th International Workshop on Managing Technical Debt, MTD 2013 – Proceedings, 2013, 4p.
E10	Cunningham, W., 1992. “The WyCash portfolio management system”. Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum), Vancouver, British Columbia, Canada. ACM, *A19.
E11	Curtis, B., Sappidi, J., Szynkarski, A., 2012 "Estimating the Size, Cost, and Types of Technical Debt", Managing Technical Debt (MTD) - Third International Workshop on TD, 2012, pp. 49-53.
E12	Curtis, Bill, Jay Sappidi, and Alexandra Szynkarski. "Estimating the Principal of an Application's Technical Debt." *IEEE software* 6 (2012): 34-42.
E13	Darcy, D.P., Kemerer, C.F., Slaughter, S.A., and Tomayko, J.E. “The Structural Complexity of Software: An Experimental Test,” IEEE Trans. Software Eng., vol. 31, no. 11, pp. 982-995, Nov. 2005.
E14	Davis, J.D., Andersen, T.J., 2009. “Surviving the economic downturn”. 2009 Agile Conference, AGILE 2009, Chicago, IL, United states, August 24, 2009–August 28, 2009. IEEE Computer Society, 245–250.
E15	Debois, P., 2008. “Agile infrastructure and operations: how infra-gile are you?“. In: Agile, 2008. AGILE ‘08. Conference, 4–8 August 2008, pp. 202–207, *A11.
E16	Eick, S.G., Graves, T.L., Karr, A.F., Marron, J.S., Mockus, A., 2001. “Does Code Decay?” Assessing the Evidence from Change Management Data. IEEE Transactions on Software Engineering 27 (1), 1–12.
E17	ELM, J., 2009. “Design Debt Economics: A Vocabulary for Describing the Causes, Costs and Cures for Software Maintainability Problems”. IBM, Available: http://www.ibm.com/developerworks/rational/library/edge/09/jun09/designdebteconomics/ (accessed 24.08.11). (Online).

E18	Falessi, D., Shaw, M.A., Shull, F., Mullen, K., Keymind, M.S. “Practical Considerations, Challenges and Requirements of Tool Support for Managing Technical Debt”. In: 2013 4th International Workshop on Managing Technical Debt, MTD 2013 – Proceedings 6608673, pp. 16-19
E19	Fowler, M., 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley. ISBN 0-201-48567-2.
E20	Fowler, M., 2009a. “Technical Debt”, Available from: http://www.martinfowler.com/bliki/TechnicalDebt.html (accessed 28.04.11)
E21	Fowler, M., 2009b. “Technical Debt Quadrant”, Available from: http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html (accessed 28.04.11).
E22	Freeman, S. “Bad code isn’t technical debt, it’s an unhedged call option”. Higher-order Logic; http://www.higherorderlogic.com/2010/07/bad-code-isnt-technical-debt-its-an-unhedged-call-option/.
E23	Guo, Y., Seaman, C., 2011. “A portfolio approach to technical debt”. In: Second International Workshop on Managing Technical Debt, ICSE 2011, Waikiki, Honolulu, Hawaii, USA, 23 May 2011.
E24	6.3.1.1.1.1Guo, Y., Seaman, C., Gomes, R., Cavalcanti, A., Tonin, G., Da Silva, F. Q. B., Santos, A. L. M., and Siebra, C. "Tracking Technical Debt – An Exploratory Case Study," in 27th IEEE International Conference on Software Maintenance (ICSM’ 11), Williamsburg, VA, USA, 2011, pp. 528-531.
E25	Haack, P., 2005. “Going Into Design Debt”, Available from: http://haacked.com/archive/2005/09/24/GoingIntoDesignDebt.aspx (accessed 24.08.11).
E26	Heidenberg, J., Porres, I., 2010. “Metrics functions for Kanban guards”. In: 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS), 2010, 22–26 March 2010, pp. 306–310, *A4.
E27	Holvitie, Johannes, and Ville Leppänen. "DebtFlag: technical debt management with a development environment integrated tool." *Proceedings of the 4th International Workshop on Managing Technical Debt*. IEEE Press, 2013.
E28	Hilton, R., 2011. “When to work on technical debt”. In: Absolutely No Machete Juggling, Available from: http://www.nomachetejuggling.com/2011/07/22/when-to-work-on-technical-debt/ (accessed 24.08.11). (Online).
E29	Hira, A. “CodeCount”, Center for System and Software Engineering, available at http://sunset.usc.edu/research/CODECOUNT/, 2010
E30	ISO/IEC 9126:2004 Software engineering -- Product Quality. Geneva: ISO, 2004.
E31	Izurieta, Clemente, et al. "On the uncertainty of technical debt measurements." *Information Science and Applications (ICISA), 2013 International Conference on*. IEEE, 2013.
E32	Klinger, T., Tarr, P., Wagstrom, P., Williams, C., “An enterprise perspective on technical debt”. In: Proceedings - International Conference on Software Engineering, 2011, 4p.
E33	Kitchenham, B.A. et al. “Preliminary Guidelines for Empirical Research in Software Engineering,” IEEE Trans. Software Eng., vol. 28, no. 8, pp. 721-734, Aug. 2002.
E34	Klein, J., 2005. “How does the architect’s role change as the software ages?”. In: 5th Working IEEE/IFIP Conference on Software Architecture, WICSA 2005, p. 141, *A17.
E35	Kruchten, P. “The Rational Unified Process - An introduction”, 3rd ed, Boston, MA: Addison-Wesley-Longman, 2003. {ISBN:0321197704}
E36	Kruchten, P. "Strategic Management of Technical Debt: Tutorial Synopsis". Quality Software (QSIC), 2012 12th International Conference on. IEEE, 2012.

E37	Laribee, D., 2009. “Using agile techniques to pay back technical debt”. MSDN Magazine, Available from: http://msdn.microsoft.com/en-us/magazine/ee819135.aspx (accessed 24.08.11). (Online).
E38	Lattix. Available on April 9, 2012 from http://www.lattix.com
E39	Letouzey, J.-L. “The SQALE Method for Evaluating Technical Debt,” Proc. 3rd Int’l Workshop Managing Technical Debt, IEEE CS, 2012, pp. 31–36.
E40	Letouzey, Jean-Louis, and Michel Ilkiewicz. "Managing technical debt with the sqale method." *IEEE software* 6 (2012): 44-51.
E41	Lim, E., Taksande, N., Seaman, C., 2012. “A balancing act: what software practitioners have to say about technical debt”. IEEE Software, 22–28.
E42	Lindgren, M., Land, R., Norstrom, C., Wall, A., 2008a. “Key aspects of software release planning in industry”. In: 19th Australian Conference on Software Engineering, 2008. ASWEC 2008, 26–28 March 2008, pp. 320–329, *A13.
E43	Lindgren, M., Wall, A., Land, R., Norstrom, C., 2008b. “A method for balancing short- and long-term investments: quality vs. features”. In: 34th Euromicro Conference on Software Engineering and Advanced Applications, 2008. SEAA’08, 3–5 September 2008, pp. 175–182, *A12.
E44	Mantyla, M.V., Lassenius, C., 2009. “What types of defects are really discovered in code reviews?”. IEEE Transactions on Software Engineering 35, 430–448, *A9.
E45	McConnell, 2007. “Technical debt”. In: 10x Software Development, Available from: http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx (accessed 28.04.11). (Online).
E46	McConnell S. “Managing Technical Debt [Webinar]”, Sep. 2011. Available: http://www.youtube.com/watch?v=lEKvzEyNtbk (Accessed: Mar. 2012).
E47	Muller, M. “Interview with Ipek Ozkaya (Sei) on Technical Debt”, Agile and Architecture, 2012.
E48	Myers, R., 2009. “Ward Cunningham’s debt metaphor isn’t a metaphor”. In: Powers of Two, Available from: http://powersoftwo.agileinstitute.com/2009/03/ward-cunninghams-debt-metaphor-isnt.html (accessed 16.05.11). (Online).
E49	Neill, C.J., Laplante, P.A., 2006. “Paying down design debt with strategic refactoring”. Computer 39, 131–134, *A16.
E50	Nielsen, E., 2011. “Know Your Technical Debt Burden”, Available from: http://www.anticlue.net/archives/IT-Leadership/TechnicalDebtBurden.htm (accessed 24.08.11). (Online).
E51	Nord, R., Ozakaya, I., Kruchten, P., Gonzalez, M., “In Search of a Metric for Managing Architectural Technical Debt,” Submitted to WICSA1ECSA 2012 conference, April 2012.
E52	Nugroho, A., Visser, J., and Kuipers, T., "An Empirical Model of Technical Debt and Interest," Proceedings of the 2nd Workshop on Managing Technical Debt, Waikiki, Honolulu, HI, USA, 2011, ACM, pp. 1-8.
E53	Parnas, D. L. "Software aging," Proc. of the 16th international conference on Software engineering (ICSE 1994), 1994, IEEE Computer Society, pp. 279-287. {ISBN:0-8186-5855-X }
E54	Penchikala, S. “Architecture analysis tool SonarJ 6.0 supports structural debt index and quality model,” InfoQ, Aug. 2010. Available: http://www.infoq.com/news/2010/08/sonarj-6.0 (Accessed: Mar. 2012).
E55	Ramasubbu, N., Kemerer, C.F. “Towards a Model for Optimizing Technical Debt in Software Products”. In: 2013 4th International Workshop on Managing Technical Debt, MTD 2013 – Proceedings 6608679, pp. 51-54
E56	Ries, E., 2009. “Embrace technical debt”. In: Lessons Learned, Available from: http://www.startuplessonslearned.com/2009/07/embrace-technical-debt.html

	(accessed 24.08.11). (Online).
E57	Seaman, C.B. “Qualitative Methods in Empirical Studies of Software Engineering,” IEEE Trans. Software Eng., vol. 25, no. 4, pp. 557-572, July/Aug. 1999.
E58	Seaman, C., Guo, Y., Izurieta, C., Cai, Y., Zazwirja, N., Shull, F., Vetro, A., 2012. “Using technical debt data in decision making: potential decision approaches”. In: Second International Workshop on Managing Technical Debt, ICSE 2011, Waikiki, Honolulu, Hawaii, USA, 23 May 2011.
E59	Seaman, C. and Zazworka, N. “Identifying and Managing Technical Debt”. 2011; Available from: http://www.slideshare.net/zazworka/identifying-and-managing-technical-debt.
E60	Schmid, K. “A Formal Approach to Technical Debt Decision Making”. In: QoSA 2013 - Proceedings of the 9th International ACM Sigsoft Conference on the Quality of Software Architectures pp. 153-162
E61	Schmid, K. “On the limits of the technical debt metaphor some guidance on going beyond”. In: 2013 4th International Workshop on Managing Technical Debt, MTD 2013 – Proceedings 6608681, pp. 63-66
E62	Shalloway, A., 2011. “Why Technical Debt Matters”, Available from: http://www.netobjectives.com/blogs/why-techncial-debt-matters (accessed 24.0811).
E63	Shull, F., 2011. “Perfectionists in a world of finite resources”. IEEE Software 28, 4–6, *A1.
E64	Sindhgatta, R., Narendra, N. C., and Sengupta, B. “Software evolution in agile development: A case study,” Companion to the Proc. ACM Int. Conf. Object-Oriented Programming, Systems, Languages, and Applications (SPLASH ’10), ACM Press, Oct. 2010, pp. 105–114, doi: 10.1145/1869542.1869560.
E65	Slinker, G., 2007. “Code debt, product market debt, and customer debt”. In: Digerati Illuminatus, Available from: http://digerati-illuminatus.blogspot.com/2007/11/code-debt-product-market-debt-and.html (accessed 26.09.11). (Online).
E66	Snipes, W., Robinson, B., Guo, Y., Seaman, C., 2012. “Defining the decision factors for managing defects: a technical debt perspective”. In: Third International Workshop on Managing Technical Debt, ICSE 2012, Zurich, Switzerland, June 5, 2012.
E67	Sonar. Available on April 9, 2012 from http://www.sonarsource.org
E68	Spínola, R.O., Vetrò, A., Zazworka, N., Seaman, C., Shull, F. “Investigating technical debt folklore: Shedding some light on technical debt opinion”. In: 2013 4th International Workshop on Managing Technical Debt, MTD 2013 – Proceedings 6608671, pp. 1-7Managing_TD_in_practice_an_industrial_report
E69	Szykarski, A., 2012. “Ted Theodorpoulos on Managing Technical Debt Successfully”, Available from: http://www.ontechnicaldebt.com/blog/ted-theodoropoulos-on-managing-technical-debt-successfully/
E70	Tom, et al., 2013. “An exploration of technical debt”. Journal of Systems and Software 86 (6), pp. 1498-1516
E71	Vetro, Antonio. "Using automatic static analysis to identify technical debt."*Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012.
E72	Wiklund, K., Eldh, S., Sundmark, D., Lundqvist, K., 2012. “Technical debt in test automation”. In: Third International Workshop on Managing Technical Debt, ICSE 2012, Zurich, Switzerland, June 5, 2012.
E73	Williams, L. and Cockburn, A. "Agile Software Development: It's About Feedback and Change", Computer, vol. 36, no. 6, 2003, pp. 39-43.
E74	Wirfs-Brock, R., 2008a. “Designing Then and Now”. IEEE Software 25, 29, *A14.

E75	Wirfs-Brock, R.J., 2008b. “Enabling change”. IEEE Software 25, 70–71, *A15.
E76	Zazworka, N., 2011. “Prioritizing design debt: investment opportunities”. In: SecondInternational Workshop on Managing Technical Debt, ICSE 2011, Waikiki, Honolulu, Hawaii, USA, 23 May, 2011.
E77	Zazworka, N., Shaw, M.A., Shull, F., Seaman, C., “Investigating the impact of design debt on software quality”. Proceedings - International Conference on Software Engineering, 2011, pp. 17-23
E78	Zazworka, N., Spínola, R.O., Vetro, A., Shull, F., Seaman, C., “A case study on effectively identifying technical debt”. In: ACM International Conference Proceeding Series, 2013, pp. 42-47