UNIFACS UNIVERSIDADE SALVADOR
MESTRADO ACADÊMICO EM SISTEMAS E COMPUTAÇÃO

NICOLLI SOUZA RIOS ALVES

ORGANIZAÇÃO DO CORPO DE CONHECIMENTO SOBRE DÍVIDA TÉCNICA: TIPOS, INDICADORES, ESTRATÉGIAS DE GERENCIAMENTO E CAUSAS

Salvador
2016

NICOLLI SOUZA RIOS ALVES

ORGANIZAÇÃO DO CORPO DE CONHECIMENTO SOBRE DÍVIDA TÉCNICA: TIPOS, INDICADORES, ESTRATÉGIAS DE GERENCIAMENTO E CAUSAS

Dissertação apresentada ao mestrado Mestrado Acadêmico em Sistemas e Computação da UNIFACS Universidade Salvador – Laureate International Universities, como requisito parcial para obtenção do título de Mestre.

Orientador: Prof. Dr. Rodrigo Oliveira Spínola

Salvador
2016

FICHA CATALOGRÁFICA
Elaborada pelo Sistema de Bibliotecas da UNIFACS Universidade Salvador, Laureate Internacional Universities

Alves, Nicolli Souza Rios

Organização do corpo de conhecimento sobre dívida técnica: tipos, indicadores, estratégias de gerenciamento e causas. / Nicolli Souza Rios Alves. Salvador, 2016.

161 f.: il.

Dissertação apresentada ao Curso de Mestrado em Sistemas e Computação, UNIFACS Universidade Salvador, Laureate Internacional Universities, como requisito parcial para obtenção do grau de Mestre.

Orientador Prof. Dr. Rodrigo Oliveira Spínola.

1. Desenvolvimento de software. I. Sínola, Rodrigo Oliveira, orient. II. Título.

CDD: 004.6

TERMO DE APROVAÇÃO

NICOLLI SOUZA RIOS ALVES

ORGANIZAÇÃO DO CORPO DE CONHECIMENTO SOBRE DÍVIDA TÉCNICA: TIPOS, INDICADORES, ESTRATÉGIAS DE GERENCIAMENTO E CAUSAS

Dissertação aprovada como requisito parcial para obtenção do título de Mestre em Sistema de Computação, UNIFACS Universidade Salvador, Laureate International Universities, pela seguinte banca examinadora:

Rodrigo Oliveira Spínola – Orientador _____
Doutor e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ
UNIFACS Universidade Salvador, Laureate International Universities

Artur Henrique Kronbauer _____
Doutor em Ciência da Computação, na área de Interação Humano-Computador pela Universidade Federal da Bahia (UFBA)
UNIFACS Universidade Salvador, Laureate International Universities

Guilherme Horta Travassos _____
Doutor em Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro (UFRJ)
Universidade Federal do Rio de Janeiro – COPPE/UFRJ

Salvador de de 2016.

À Deus, minha família, namorado, amigos, colegas e orientador pelo apoio, força, incentivo, companheirismo e amizade. Sem eles nada disso seria possível.

AGRADECIMENTOS

Nesses dois anos de mestrado, de muito estudo, esforço e empenho, sou grata a algumas pessoas que me acompanharam, apoiaram e foram fundamentais para a realização dessa etapa.

Agradeço aos meus familiares pelo apoio e compreensão necessários para a conclusão desta pesquisa.

Ao meu namorado, Arthur, pelo companheirismo, compreensão e por ter aguentado meus momentos de estresse e ansiedade nos meses de dedicação ao mestrado.

Em especial ao grande professor e orientador Rodrigo, pelos ensinamentos, amizade, dedicação e principalmente pela confiança depositada em mim ao longo de todos esses anos de trabalho que se iniciaram ainda na graduação.

À professora Carolyn Seaman, pelo apoio e sabedoria nessa pesquisa. Aos amigos e colegas de mestrado, pelo apoio, amizade e conhecimentos compartilhados. A Thiago pela grande ajuda na execução do mapeamento sistemático da literatura. A Rodrigo Araújo, pela ajuda no desenvolvimento da ferramenta TD Wiki. A Josiane Melo, sempre disposta a ajudar.

Aos professores Guilherme Travassos e Artur Kronbauer, pela disponibilidade em avaliar o trabalho e contribuições para a melhoria da versão final entregue.

Àqueles que participaram dos estudos dessa pesquisa e, por último e não menos importante, a Deus por me iluminar e guiar nos momentos mais difíceis.

RESUMO

O conceito de dívida técnica contextualiza o problema das tarefas de desenvolvimento pendentes como um tipo de dívida que traz um benefício a curto prazo para o projeto, mas que poderão ter de ser pagas com juros mais tarde no processo de desenvolvimento. A consequência desses problemas é observada em atrasos inesperados na realização de modificações necessárias e na dificuldade em atingir os critérios de qualidade acordados para o projeto. É comum que um projeto de software incorra em dívidas durante o processo de desenvolvimento. No entanto, sua presença traz riscos para o projeto e dificulta sua gestão. A identificação, medição e gerenciamento da dívida apoia os gestores a tomarem decisões fundamentadas, resultando em maior qualidade do software e maior produtividade na execução das atividades. No entanto, antes de identificar, medir e/ou gerenciar a dívida, é necessário entender quais são os tipos existentes, quais são seus indicadores, quais técnicas de gerenciamento estão sendo propostas e o que leva as equipes a incorrerem a dívida. Este trabalho organizou o corpo de conhecimento em dívida técnica considerando seus tipos, indicadores, estratégias de gerenciamento e causas para sua ocorrência. Em complemento, compartilhou o corpo de conhecimento organizado através da infraestrutura TD Wiki. Para alcançar estes objetivos, a pesquisa seguiu duas linhas de trabalho: (1) realização de um mapeamento sistemático da literatura complementado por um *survey* para a coleta de evidências que permitissem organizar o conhecimento sobre DT; (2) estruturação das informações identificadas a partir dos estudos em uma taxonomia de tipos de dívida e em uma infraestrutura que apoia o compartilhamento e evolução colaborativa do conhecimento – TD Wiki.

Palavras chaves: Dívida técnica. Tipos de dívida. Indicadores de Dívida. Estratégias de Gerenciamento. Causas. Mapeamento Sistemático. Taxonomia. *Survey*. Compartilhamento do Conhecimento

ABSTRACT

The concept of technical debt contextualizes the problem of pending development tasks as a type of debt which brings a short-term benefit for the project, but that may have to be paid later with interest in the development process. The consequences of these problems can be observed in unexpected delays, changes, and in the difficulty for achieving the quality criteria agreed for the project. It is usual for a software project to incur debt during its development process. However, its presence brings risks for the project and hampers its development. Its identification, measurement, and management helps project managers to take decisions, resulting in improved software quality and productivity in performing development activities. But, before identifying, measuring and / or managing the debt, the development team need to understand what are the existing types and their indicators, which management techniques are being proposed, and what leads professionals to incur the debt. This work proposes the organization of a body of knowledge for the technical debt area considering its types, indicators, management strategies, and causes for its occurrence. In addition, we shared the organized body of knowledge through the TD Wiki infrastructure. To achieve these goals, the research followed two lines of work: (1) performing a systematic mapping study complemented by a survey study to collect evidence that would allow the organization of knowledge about technical debt; (2) structuring the information identified from the studies in a taxonomy of types of debt and in an infrastructure that supports its collaborative sharing and evolution.

Keywords: Technical Debt. Types of Debt. Debt Indicators. Management Strategies. Causes. Systematic Mapping. Taxonomy. Survey Study. Knowledge Sharing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DT	Dívida Técnica
MS	Mapeamento Sistemático
ASA	Análise Estática Automática
ESE	Engenharia de Software Experimental
ES	Engenharia de Software

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo são apresentados o contexto do trabalho, o que motivou essa pesquisa e seus objetivos. São também apresentados a metodologia de pesquisa utilizada, o histórico do trabalho realizado e como este texto está estruturado.

1.1 INTRODUÇÃO

Durante o desenvolvimento, a qualidade do software diminui quando se considera aspectos como sua estrutura interna, adesão a normas, documentação e facilidade de entendimento para futuras manutenções (LIENTZ *et al.*, 1978) (LEHMAN; BELADY, 1985) (PARNAS, 1994). Um motivo para isto acontecer é que as atividades de desenvolvimento são frequentemente realizadas sob forte restrição de tempo e recursos, sendo investida a quantidade mínima de esforço e tempo necessário para sua realização. Esse cenário traz impactos na produtividade uma vez que uma modificação em um software de baixa qualidade geralmente é mais difícil de ser realizada do que em um de alta qualidade (LEHMAN; BELADY, 1985).

Lidar com esta questão considerando a metáfora da Dívida Técnica (DT) tem ajudado alguns profissionais a debater questões associadas ao desenvolvimento do software. A metáfora da DT descreve o efeito de artefatos imaturos no desenvolvimento do software, que traz um benefício a curto prazo para o projeto em termos de maior produtividade e menor esforço, mas que poderão ter de ser ajustados com juros (esforço extra necessário para ajustar o item da dívida ou realizar atividades de desenvolvimento em um software com sua qualidade interna degenerada) mais tarde (BROWN *et al.*, 2010) (GUO; SEAMAN, 2011) (KRUCHTEN *et al.*, 2012) (AVGERIOU *et al.*, 2016). A metáfora da DT tem potencial para ser amplamente utilizada na indústria de software por representar de forma eficaz um entendimento da dinâmica de projetos de desenvolvimento de software (FOWLER, 2003) e devido à facilidade de entendimento de seus conceitos (SPÍNOLA *et al.*, 2013).

Desde sua citação inicial em (CUNNINGHAM, 1992), que utilizou o termo "*going into debt*" para explicar que uma pequena dívida pode acelerar o desenvolvimento de software mas que cada minuto extra gasto com o código mal construído conta como juros sobre essa dívida, a metáfora da DT tem se expandindo e abrange, atualmente, diferentes artefatos gerados ao longo do ciclo de vida do software. Neste contexto, diferentes pesquisas têm sido realizadas para tratar a dívida técnica sob diferentes perspectivas (KRUCHTEN *et al.*, 2012) (ZAZWORKA *et al.*, 2013)

(GUO *et al.*, 2014): estratégias de gerenciamento, identificação da dívida, quantificação da dívida e aspectos financeiros da dívida, dentre outros.

1.2 CONTEXTO E MOTIVAÇÃO

O conceito de dívida técnica contextualiza o problema das tarefas de desenvolvimento pendentes (por exemplo, não execução de testes, refatoração de código pendente, atualização de documentação pendente) como um tipo de dívida que traz um benefício a curto prazo para o projeto (normalmente em termos de maior produtividade ou menor tempo de liberação de versões do software), mas que poderão ter de ser pagas com juros mais tarde no processo de desenvolvimento. A consequência desses problemas é observada em atrasos inesperados na realização de modificações necessárias e na dificuldade em atingir os critérios de qualidade acordados para o projeto (SPÍNOLA *et al.*, 2013)(ZAZWORKA *et al.*, 2013).

É comum que um projeto de software incorra em dívidas durante o processo de desenvolvimento (BROWN *et al.*, 2010). No entanto, sua presença traz riscos para o projeto e dificulta sua gestão uma vez que os gerentes terão que decidir se a dívida será paga e, em caso positivo, quanto da dívida deve ser paga e quando (GUO *et al.*, 2014). Para tomar essas decisões baseadas em fatos, é necessário que a equipe tenha conhecimento sobre o valor presente e futuro da DT associada ao projeto. Portanto, a identificação, medição e gerenciamento da DT ajudaria os gestores a tomar decisões fundamentadas, resultando em maior qualidade do software e maior produtividade na execução das atividades de desenvolvimento (GUO *et al.*, 2014).

No entanto, antes de identificar, medir e/ou gerenciar a dívida, é necessário entender quais são os tipos existentes, quais são seus indicadores, quais técnicas de gerenciamento estão sendo propostas e o que leva as equipes a incorrerem a dívida. A organização destas informações permite que equipes trabalhem de uma forma mais controlada em atividades de prevenção, monitoramento e gerenciamento da dívida, além de estabelecer um arcabouço bem fundamentado sobre o qual novas pesquisas possam ser realizadas na área.

Este cenário motivou este trabalho a organizar um corpo de conhecimento em dívida técnica considerando em sua estrutura os seguintes itens: tipos de dívida, indicadores de dívida, estratégias de gerenciamento e causas para a sua ocorrência.

1.3 OBJETIVO E QUESTÕES DE PESQUISA

Muito se tem pesquisado sobre estratégias de identificação, gerenciamento, monitoramento e quantificação da dívida existente nos projetos. Porém, esse conhecimento desenvolvido está espalhado na literatura técnica dificultando seu uso por pesquisadores e profissionais. Assim, é importante organizar um corpo de conhecimento sobre DT de forma que ele possa, efetivamente, apoiar a aplicação prática dos conceitos desenvolvidos e também direcionar melhor novos esforços de pesquisa na área. Este trabalho tem como objetivo **propor a organização do corpo de conhecimento em dívida técnica levando em consideração seus tipos, indicadores, estratégias de gerenciamento e causas para sua ocorrência.**

Em busca deste objetivo, foram definidas as seguintes questões de pesquisa:

- **(Q1) "Quais são as estratégias que têm sido propostas para identificar ou gerenciar DT em projetos de software?"**. As seguintes questões complementares foram derivadas dessa questão inicial:
    - **(QC1) Quais são os tipos de DT?** Esta questão de pesquisa tem como objetivo identificar os diferentes tipos de DT encontrados na literatura.
    - **(QC2) Quais são as estratégias propostas para identificar DT?** Uma vez que são conhecidos os tipos de DT, também é importante saber como eles podem ser identificados em projetos de software. Esta questão tem como objetivo, para cada tipo de dívida, identificar os indicadores que têm sido propostos para permitir sua identificação.
        - **(QC2.1) Quais avaliações experimentais foram realizadas?** O propósito desta questão é identificar quais tipos de estudos experimentais têm sido utilizados para avaliar as estratégias de identificação de DT propostas na literatura.
        - **(QC2.2) Quais são as fontes de dados e artefatos que têm sido propostos para identificar a DT?** O objetivo desta questão de pesquisa é identificar quais artefatos e fontes de dados têm sido considerados em atividades de identificação da DT.
        - **(QC2.3) Quais técnicas de visualização de software têm sido propostas para apoiar a identificação de DT?** Esta questão de pesquisa busca

identificar se e quais técnicas de visualização têm sido propostas para apoiar a identificação de DT.

- **(QC3) Quais estratégias têm sido propostas para apoiar o gerenciamento da DT?** O objetivo desta questão de pesquisa é realizar uma caracterização das estratégias que têm sido propostas para o gerenciamento de DT.
    - **(QC3.1) Quais avaliações experimentais foram realizadas?** O propósito desta questão é identificar quais tipos de estudos experimentais têm sido utilizados para avaliar as abordagens de gerenciamento de DT propostas na literatura.
    - **(QC3.2) Quais técnicas de visualização de software têm sido propostas para apoiar o gerenciamento da DT?** Esta questão de pesquisa busca identificar se, e quais técnicas de visualização têm sido propostas para apoiar o gerenciamento da DT.
- **(Q2) Quais causas levam à ocorrência de dívida técnica?** Essa questão tem como objetivo identificar possíveis causas que levam a equipe de desenvolvimento a incorrer em dívida técnica.
- **(Q3) As causas ocorrem de forma encadeada ou isolada?** O objetivo dessa questão é investigar se as causas da dívida ocorrem em cadeia. Isso pode ser um indicador sobre quanto as medidas de prevenção devem ser focadas ou mais abrangentes.
- **(Q4) A dívida técnica pode ser evitada?** O objetivo dessa questão é saber se é possível evitar a DT. A resposta a esta pergunta pode apoiar equipes de desenvolvimento na decisão de investir em atividades com objetivo de prevenir a dívida (caso seja possível evitá-la) ou em atividades de monitoramento/gerenciamento (caso não seja possível prevenir sua ocorrência).
- **(Q5) Em termos de esforço, é melhor evitar a dívida ou incorrê-la para pagar depois?** Essa questão tem como objetivo investigar se prevenir a dívida teria sido mais caro do que incorrer a dívida. Sua resposta poderá indicar qual caminho a equipe de desenvolvimento deve seguir: trabalhar para prevenir ou, simplesmente, incorrer a dívida e pagá-la posteriormente.

Com a finalidade de atingir o objetivo do trabalho e responder às questões de pesquisa, foram definidos quatro objetivos específicos:

1. **Buscar na literatura técnica quais são as estratégias que têm sido propostas para identificar ou gerenciar DT em projetos de software.** Para alcançar esse objetivo, foi executado um mapeamento sistemático da literatura, apresentado no Capítulo 3.

2. **Organizar os tipos de dívida conhecidos.** Para alcançar esse objetivo, foi elaborada uma taxonomia de tipos de DT, apresentada no Capítulo 4.

3. **Investigar as causas que levam a equipe de desenvolvimento a incorrer DT em projetos de software.** Para alcançar esse objetivo, foi planejado e executado um *survey* baseado em entrevistas, apresentado no Capítulo 5.

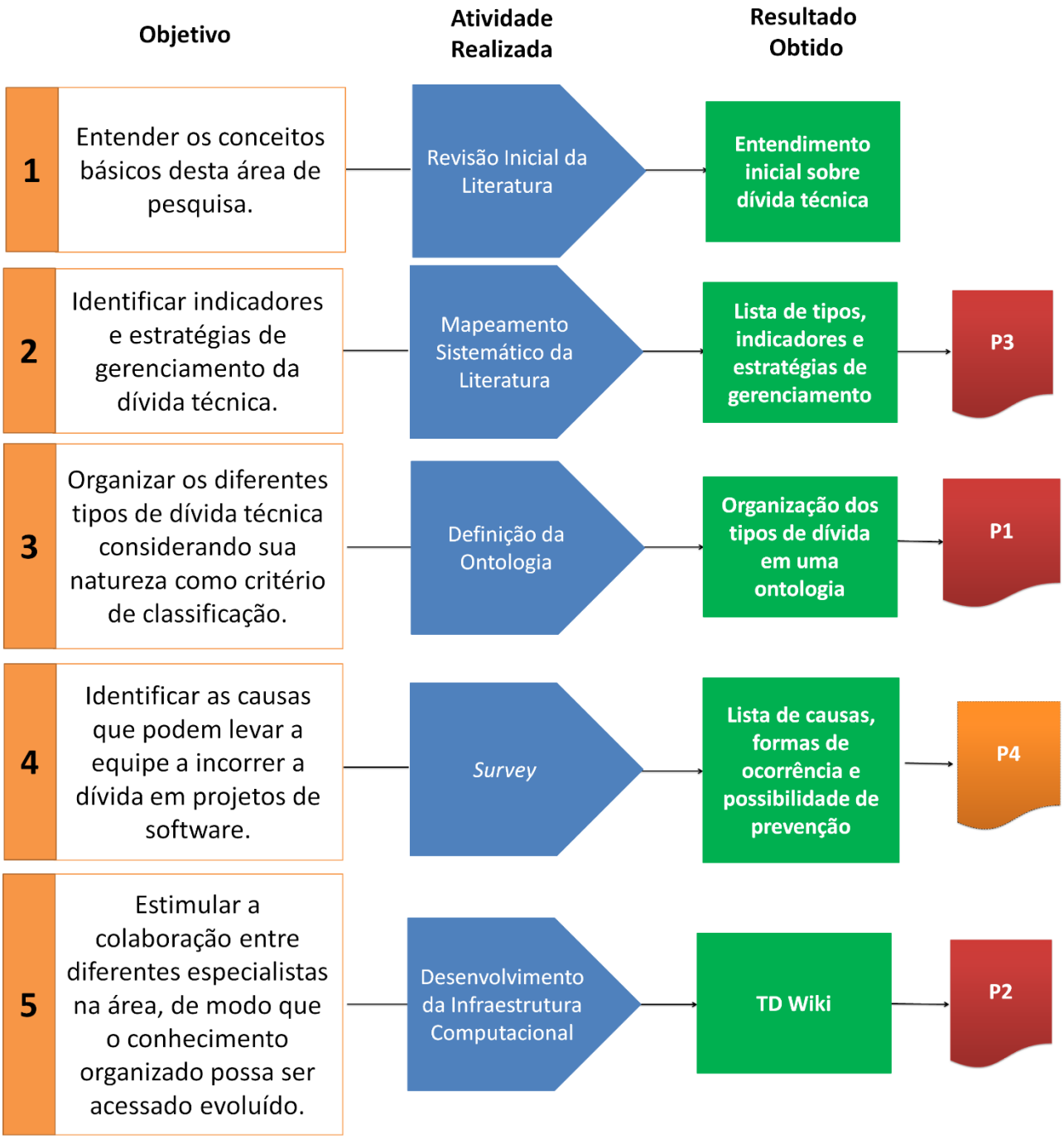
4. **Compartilhar o corpo de conhecimento organizado de forma que ele possa ser acessado e evoluído pela comunidade de pesquisa e profissionais da área.** Para alcançar esse objetivo, foi desenvolvida uma ferramenta que apoia o compartilhamento e evolução colaborativa do conhecimento estruturado sobre DT, apresentada no Capítulo 6.

1.4 METODOLOGIA E HISTÓRICO DA PESQUISA

As atividades realizadas neste trabalho foram fundamentadas no paradigma da Engenharia de Software Experimental (BASILI, 1992). A Engenharia de Software Experimental busca aprimorar a engenharia de software aplicando a abordagem científica na construção e/ou evolução de métodos, teorias e técnicas, sejam eles novos ou já existentes, para apoiar o desenvolvimento de software (BRILLIANT ; KNIGHT, 1998).

Para alcançar o objetivo do trabalho, dois estudos complementares foram planejados e executados. A Figura ilustra os passos realizados, sendo que para cada um deles são descritos os respectivos objetivos e resultados. Os pentágonos azuis representam as atividades realizadas, com seus respectivos objetivos (retângulos brancos) e resultados (retângulos verdes). Os ícones na cor vermelha representam artigos publicados relacionados à atividade realizada. Quando na cor laranja, indica um artigo que está em fase de escrita para submissão.

Figura 1 - Estratégia de pesquisa



A seguir, tem-se uma breve descrição das atividades desempenhadas:

1. **Revisão Inicial da Literatura:** nessa atividade, foi realizada uma revisão informal da literatura para entendimento dos conceitos básicos necessários para o desenvolvimento desta pesquisa;

2. **Mapeamento Sistemático da Literatura:** executada no segundo semestre de 2013 e primeiro de 2014, nesta atividade realizou-se o mapeamento sistemático da literatura seguindo o processo padrão para a realização deste tipo de estudo, definido em

(PETERSON *et al.*, 2008). O foco desta revisão foi a identificação de *indicadores e técnicas de gerenciamento da dívida técnica*. Como resultado, tipos de dívida, indicadores e estratégias de gerenciamento da dívida foram identificados;

3. **Definição da Taxonomia:** nessa atividade, uma taxonomia foi definida para organizar os diferentes tipos de dívida técnica considerando a classificação proposta por Alves *et al.*, (2016);

4. ***Survey*:** executado no segundo semestre de 2015, através deste estudo foi possível caracterizar as causas que levam a equipe de desenvolvimento a incorrer a dívida técnica em projetos de software. O estudo foi efetuado com a participação de um grupo de 10 profissionais e pesquisadores da área de Engenharia de Software. Ao final foi possível extrair uma lista de causas, se elas ocorriam de forma isolada ou encadeada, e investigou-se a possibilidade de prevenir a dívida;

5. **Desenvolvimento da Infraestrutura Computacional:** nessa atividade foi desenvolvido TD Wiki - uma infraestrutura computacional de apoio ao compartilhamento e evolução colaborativa do conhecimento sobre DT;

A partir da realização das atividades apresentadas na Figura 1, os seguintes resultados foram obtidos:

- **Organização do corpo de conhecimento sobre dívida técnica:**
  - **P3:** ALVES, N.S.R., MENDES, T.S., MENDONÇA, M.G., SPÍNOLA, R.O., SHULL, F., SEAMAN, C.. Identification and Management of Technical Debt: A Systematic Mapping Study. Information and Software Technology, 70, 100 – 121, 2016.
- **Estruturação do corpo de conhecimento organizado:**
  - **P1:** ALVES, N. S. R., RIBEIRO, L. F., CAIRES, V., MENDES, T. S., & SPÍNOLA, R. O. 2014. "Towards an Ontology of Terms on Technical Debt". In Proceedings of the 2014 Sixth International Workshop on Managing Technical Debt (MTD '14). IEEE COMPUTER SOCIETY, Washington, DC, USA, 1-7. DOI=10.1109/MTD.2014.9 http://dx.doi.org/10.1109/MTD.2014.9.
- **Compartilhamento do corpo de conhecimento organizado de forma que ele possa ser acessado e evoluído pela comunidade de pesquisa e profissionais da área:**
  - **P2:** ALVES, N.S.R.; ARAÚJO, R.S.; SPÍNOLA, R.O. . A Collaborative Computational Infrastructure for Supporting Technical Debt Knowledge Sharing and

Evolution. In: Americas Conference on Information Systems, 2015, Puerto Rico. 2015 Americas Conference on Information Systems, 2015.

Por fim, os resultados do *survey* estão sendo preparados para submissão.

1.5 ORGANIZAÇÃO DA DISSERTAÇÃO

Este capítulo apresentou a motivação para a realização desta dissertação, seu objetivo e a metodologia de pesquisa utilizada. A organização do texto deste trabalho segue a linha de tempo em que as atividades de pesquisa foram realizadas (Figura) e está estruturado segundo os itens a seguir:

Capítulo 1. Introdução: contextualiza e descreve os motivos para a realização deste trabalho, juntamente com os objetivos que o trabalho pretende alcançar;

Capítulo 2. Dívida Técnica: apresenta uma revisão bibliográfica sobre DT considerando definições importantes para o entendimento desta dissertação;

Capítulo 3. Identificação e Gerenciamento da Dívida Técnica: um Mapeamento Sistemático: apresenta um mapeamento sistemático da literatura conduzido para responder à seguinte questão de pesquisa: "Quais são as abordagens que têm sido propostas para identificar ou gerenciar DT em projetos de software?";

Capítulo 4. Taxonomia de Tipos de Dívida Técnica: apresenta uma taxonomia de tipos de dívida técnica cujo objetivo foi organizar um vocabulário comum para a área. Os conceitos organizados foram extraídos a partir dos resultados do mapeamento sistemático apresentado no Capítulo 3;

Capítulo 5. Causas que Levam à Inserção da Dívida Técnica em Projetos de Software: apresenta um *survey* que teve como objetivo identificar os motivos que levam a equipe de desenvolvimento a incorrer a dívida em projetos de software;

Capítulo 6. TD Wiki - Uma Infraestrutura Computacional para Apoiar o Compartilhamento e a Evolução Colaborativa do Conhecimento sobre Dívida Técnica: descreve os requisitos, arquitetura e a ferramenta desenvolvida para apoiar o compartilhamento e evolução colaborativa do conhecimento sobre DT;

Capítulo 7. Considerações Finais: apresenta as considerações finais sobre o trabalho considerando suas principais contribuições, limitações e perspectivas de trabalhos futuros.

Além disto, este trabalho contém três apêndices:

APÊNDICE A – Artigos Selecionados no Mapeamento Sistemático da Literatura: apresenta a lista de artigos selecionados pela revisão controlada da literatura executada neste trabalho e que foi descrita no Capítulo 3;

APÊNDICE B – OWL da Taxonomia de Tipos de Dívida Técnica: apresenta o código OWL da taxonomia definida no Capítulo 4;

APÊNDICE C – Instrumentos Utilizados no *Survey*: apresenta os instrumentos utilizados no *survey* executado neste trabalho e que foi descrito no Capítulo 5.

2 DÍVIDA TÉCNICA

Neste capítulo são apresentados os conceitos e definições sobre dívida técnica utilizados neste trabalho. Serão discutidos o que é dívida técnica, seus tipos e como ela pode ser gerenciada e identificada.

2.1 INTRODUÇÃO

A metáfora da dívida técnica descreve o efeito de artefatos imaturos no desenvolvimento do software que traz um benefício a curto prazo para o projeto em termos de maior produtividade e menor esforço, mas que poderão ter de ser ajustados com juros mais tarde. O resultado desses artefatos é observado em atrasos inesperados na realização de modificações necessárias e na dificuldade em atingir os critérios de qualidade acordados para o projeto (SPÍNOLA *et al.*, 2013) (ZAZWORKA *et al.*, 2013).

Estratégias de gerenciamento e indicadores têm sido investigados com o objetivo de identificar e monitorar a dívida incorrida em projetos de software (GUO; SEAMAN, 2011) (SEAMAN *et al.*,2012) (SEAMAN *et al.*, 2012) (CURTIS, 2012) (HOLVITIE; LEPPANEN, 2013). Essas estratégias avaliam o momento mais adequado para que a dívida seja paga de forma a minimizar maiores transtornos ao projeto.

Com o objetivo de apresentar os conceitos e definições sobre DT, além desta introdução, este capítulo foi organizado em mais cinco seções. Inicialmente, na seção 2.2, a metáfora da dívida técnica é discutida. Já a seção 2.3 apresenta alguns tipos de dívida. A seção 2.4 discorre sobre o gerenciamento da dívida e a seção 2.5 sobre sua identificação. Finalmente, a seção 2.6 traz as considerações finais deste capítulo.

2.2 A METÁFORA DA DÍVIDA TÉCNICA

Um motivo comum para o declínio da qualidade do software em atividades de desenvolvimento é que estas são frequentemente realizadas sob forte restrição de tempo e recursos, sendo investida a quantidade mínima de esforço e tempo necessário para sua realização. Assim, existe uma lacuna entre a quantidade mínima de esforço exigida e a quantidade necessária para realizar as modificações ao mesmo tempo em que se mantém o nível de qualidade do software. Por razões de negócios, nem sempre é possível preencher completamente essa lacuna uma vez que o esforço necessário pode não estar previsto no orçamento do projeto.

Por outro lado, algumas vezes, pode até fazer sentido não preenchê-la. Por exemplo, consideremos que uma determinada parte do sistema não possua defeitos, então, nenhum dano seria causado ao se tomar a decisão de não testá-lo para economizar recursos. Da mesma forma, caso um módulo em particular nunca venha a ser modificado no futuro, então, não atualizar sua documentação irá poupar algum tempo durante a manutenção do módulo sem quaisquer consequências adversas. A dificuldade para estes dois cenários é que raramente é conhecido se uma parte específica do sistema possui ou não defeitos, ou se um módulo em particular possuirá necessidades de modificações futuras. Dessa forma, percebe-se que este problema é semelhante ao que ocorre ao se realizar a análise de risco e tomar decisões sobre quais tarefas em atraso têm de ser cumpridas e quando.

Atualmente, os gerentes e líderes dos esforços de desenvolvimento de software realizam esta espécie de análise de risco de forma implícita. No entanto, em sistemas de grande porte e/ou complexos, é muito fácil perder o controle das tarefas atrasadas ou não compreender o impacto que elas podem trazer. O resultado disso é observado, muitas vezes, em atrasos inesperados na conclusão de modificações necessárias e no comprometimento da qualidade.

Lidar com esta questão considerando a metáfora da DT tem ajudado alguns profissionais a debatê-la (BROWN *et al.*, 2010). O termo DT foi cunhado por Ward Cunningham em (CUNNINGHAM, 1992), no qual ele apresentou a metáfora de "going into debt" toda vez que uma nova versão de um sistema é fornecida. O ponto discutido era que uma pequena dívida pode acelerar o desenvolvimento de software no curto prazo, mas cada minuto extra gasto com o código mal construído conta como juros sobre essa dívida (CUNNINGHAM, 1992). Essa metáfora tem sido estendida para se referir a qualquer artefato mal elaborado durante o ciclo de vida do software.

Assim, a DT inclui aquelas tarefas internas que você escolhe não fazer agora, mas que correm um risco de causar problemas futuros se não forem efetuadas (GUO; SEAMAN, 2011) (KRUCHTEN *et al.*, 2012). Esta metáfora contextualiza o problema das tarefas de desenvolvimento atrasadas (por exemplo, não execução de testes, refatoração de código pendente, atualização de documentação pendente) como um tipo de dívida que traz um benefício a curto prazo para o projeto (normalmente em termos de maior produtividade ou menor tempo de liberação de versões do software), mas que poderão ter de ser pagas com juros mais tarde no processo de desenvolvimento. O valor base da dívida é a quantidade de esforço necessário para eliminar a dívida (ou seja, concluir a tarefa), enquanto que o juros é a provável penalidade (em termos de aumento de esforço e redução de produtividade) que terá de ser paga no futuro como resultado da não conclusão dessas tarefas no momento em que elas surgiram (BROWN *et al.*, 2010). Por exemplo, ao codificar uma classe do

projeto de forma inadequada, além de ajustar aquela classe (valor da dívida), pode ser necessário ajustar outras classes que se relacionam com ela e que sofreriam efeitos colaterais de uma alteração nela (juros da dívida).

Dessa forma, a DT se refere ao efeito trazido por qualquer artefato incompleto, imaturo ou inadequado no ciclo de desenvolvimento do software. Ao permanecerem ao longo do ciclo de vida do software, estes artefatos irão afetar as atividades de desenvolvimento subsequentes. Por isso, eles podem ser vistos como um tipo de dívida que os desenvolvedores devem ao sistema cujo pagamento pode ser exigido mais cedo ou mais tarde.

O conceito de dívida tem estado alinhado com o interesse de profissionais da área porque ele representa de forma eficaz uma compreensão intuitiva da dinâmica de projetos de desenvolvimento de software (FOWLER, 2003). Muitos engenheiros consideram este conceito intuitivamente atraente e útil ao analisar estas questões (SPÍNOLA *et al.*, 2013).

Embora o termo DT seja recente, o conceito por trás dele não é novo. Ele está relacionado à noção de Lehman e Belady (LEHMAN; BELADY, 1985) de decaimento do software (complexidade crescente devido à mudança) e ao envelhecimento do software de Parnas (PARNAS, 1994) (fracasso de um produto em continuar a atender às necessidades de mudança). Estas questões têm sido estudadas há décadas por pesquisadores da área de manutenção e qualidade de software. A introdução do conceito de DT fornece uma nova maneira de falar, gerenciar e medir estes conceitos relacionados.

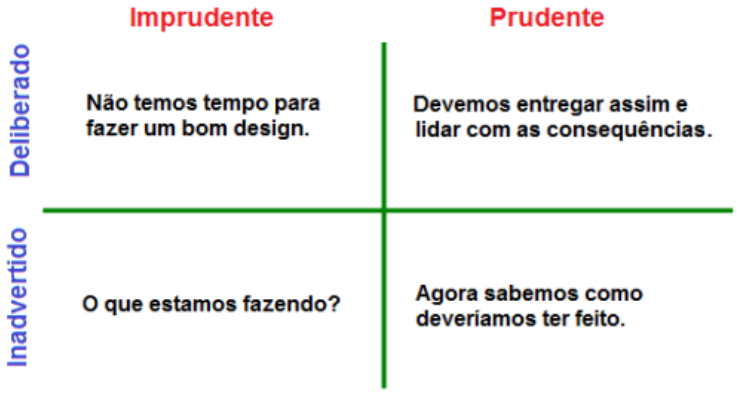
2.3 TIPOS DE DÍVIDA TÉCNICA

Existem algumas iniciativas com objetivo de organizar os diferentes tipos de DT que podem afetar um projeto. Fowler (2009) classificou as dívidas considerando as características: imprudente/prudente e deliberado/inadvertido. Estas características compõem o que ele chamou de *Technical Debt Quadrant* (conforme ilustra a Figura 2) e permitem classificar a dívida quanto ao fato dela ter sido inserida de forma intencional ou não e, para ambos os casos, se ela pode ser considerada o resultado de uma falta de cuidado ou foi inserida com cautela.

Na classificação Prudente/Deliberado, verifica-se que a equipe tem consciência que está assumindo uma dívida que será paga futuramente. Na classificação Prudente/Inadvertido, a equipe não teria conhecimento da inserção da dívida, mas ao saber de sua existência, trabalha para pagá-la e assim, adquirem experiência para uma futura análise caso necessário. Já na classificação Imprudente/Deliberado, a equipe sabe que está incorrendo a dívida no projeto, no entanto não efetua

a devida avaliação do aumento dos juros e os impactos futuros que poderão ocorrer no projeto. Na classificação Imprudente/Inadvertido, a equipe, além de inicialmente não saber sobre a inserção de dívida no projeto, também não utiliza os resultados negativos da dívida no projeto para análises futuras.

Figura 2 - Technical Debt Quadrant



Fonte: adaptado de Fowler (2009).

Próxima à classificação de Martin Fowler, Steve McConnell (2007) classificou a dívida em dois grupos: intencional e não intencional. A dívida não intencional ocorre devido a uma falta de atenção ou desconhecimento sobre a forma mais correta de desempenhar uma tarefa (por exemplo, falta de aderência aos padrões de desenvolvimento ou não uso de um padrão de projeto). Já a dívida intencional é originada de forma proativa por razões táticas ou estratégicas com o objetivo de cumprir o prazo de entrega.

Por fim, a DT também pode ser classificada em termos da fase em que ocorre no ciclo de vida de software, por exemplo, dívida de *design* ou dívida de teste (ROTHMAN, 2006). A dívida de *design* se refere à integridade do projeto em relação ao código fonte. Já a dívida de teste se refere a testes que não foram desenvolvidos ou executados. Sob essa perspectiva, tem-se um conjunto de formas sob as quais a dívida se manifesta. Alguns exemplos são:

- documentação incompleta;
- lista de pendências do projeto crescente;
- pequenas mudanças exigem um esforço desproporcional;
- o código não pode ser atualizado por dificuldade no entendimento;
- dificuldade em manter a consistência da arquitetura entre diferentes partes do sistema.

Estas formas podem ser vistas como sintomas da DT e têm estimulado a realização de pesquisas focadas no desenvolvimento de estratégias para sua identificação e gerenciamento.

2.4 GERENCIAMENTO DE DÍVIDA TÉCNICA

É comum que projetos de software incorram em dívidas durante o processo de desenvolvimento uma vez que pequenas quantidades de dívida podem aumentar a produtividade. No entanto, sua presença traz riscos para o projeto e dificulta sua gestão uma vez que os gerentes, além de suas tarefas diárias, também terão que decidir se a dívida será paga e, em caso positivo, quanto da dívida deve ser paga e quando.

Com o objetivo de assegurar a produtividade no curto prazo e um menor esforço da versão atual do projeto, técnicas de gerenciamento de DT começaram a ser elaboradas (BROWN *et al.*, 2010). Estas técnicas possuem uma preocupação geral: identificar e monitorar os itens de dívida técnica inseridos no projeto de forma que eles estejam explícitos e que sejam pagos no momento correto para evitar transtornos maiores. Dessa forma, essas técnicas ajudam os gestores a tomar decisões baseadas em fatos, resultando em maior qualidade do software em desenvolvimento e maior produtividade na execução das atividades de desenvolvimento.

Uma das possíveis formas de se fazer isso é gerenciar as mudanças na produtividade da construção do software durante seu processo de desenvolvimento. Em muitos casos, as empresas de software deixam sua dívida sair do controle e gastam a maior parte do seu esforço de desenvolvimento pagando os "juros" sob a forma de um software mais difícil de manter e de qualidade inferior. Uma vez que o pagamento de juros prejudica a produtividade de uma equipe, a diminuição da produtividade pode refletir o quanto de dívida uma organização tem.

Além disso, uma vez que a DT está intimamente ligada à qualidade do software, métricas de qualidade de software também podem ser usadas para medi-la. Por exemplo, se o software possui defeitos ou não preenche todos os requisitos do sistema, como consequência, os defeitos eventualmente terão de ser corrigidos e os requisitos implementados. Assim, o número de defeitos do sistema em um dado momento ou o número de requisitos pendentes também podem ser um indicador de dívida. Como outro exemplo, caso o software seja difícil de manter ou extremamente complexo, então mudanças futuras serão mais caras. Deste ponto de vista, acoplamento, coesão, complexidade e profundidade de decomposição são métricas que podem ser aplicadas ao lidar para medir a dívida existente. Por fim, em uma última análise, um bom *design* é julgado pela forma como seu resultado lida com as mudanças (FAIRLEY, 1994), assim, tempo e esforço necessários para lidar com as mudanças são também um indicador da presença da dívida.

Por fim, a DT também pode ser medida pelo esforço para se pagar a dívida. Por exemplo, depois de determinar que tipo de trabalho está pendente (por exemplo, reprojetar a arquitetura,

refatorar o código, documentação), o tempo e o esforço associados podem ser estimados para corrigir os problemas identificados. Esse esforço reflete o montante de DT presente atualmente no sistema.

Estratégias para pagar a DT técnica também têm sido discutidas (BROWN *et al.*, 2010). Uma delas é pagar os itens centrais da dívida em primeiro lugar. Em outras palavras, essa estratégia se concentra nos itens de dívida que possuem um grande potencial de impacto futuro no projeto. Se os itens com juros altos são pagos assim que identificados, o projeto evita sua ocorrência, que em alguns casos pode levar ao fracasso do projeto. Em termos do montante total dos juros que o projeto tem de pagar, esta estratégia não é necessariamente a que leva aos melhores resultados, mas pode reduzir o nível de risco do projeto e manter a dívida sob controle.

Outra estratégia para pagar a DT é determinar um ponto de equilíbrio ao longo do cronograma do projeto (FOWLER, 2003). Esta estratégia se baseia na lógica de que a DT remanescente do sistema prejudica a produtividade, que pode ser medida pela quantidade de funcionalidades que a equipe de desenvolvimento pode implementar por unidade de tempo. Por exemplo, suponhamos que um gerente de projeto de software, no planejamento da implementação de uma melhoria, tenha atribuído relativamente pouco tempo para analisar e documentar o projeto inicial. Como resultado, o projeto entregou mais funcionalidades ao mercado na versão em que a melhoria foi executada. No entanto, como consequência, os módulos adicionados e alterados para implementar essa melhoria são mais difíceis de trabalhar a longo prazo. Esta é, então, a DT acumulada para o projeto, que poderá atrasar a produtividade do seu desenvolvimento mais tarde uma vez que um bom projeto exige tempo e esforço inicial e, em contrapartida, permite o desenvolvimento de um software flexível e fácil de manter por um longo período de tempo.

Para entender melhor este exemplo, suponha que possamos estimar que a produtividade (P), para um projeto em que todas as melhorias são projetadas e documentadas cuidadosamente, é de 10 requisitos atendidos por dia (supondo-se, neste contexto, que requisitos por dia seja uma medida razoável de produtividade) e este número se mantém estável nos três primeiros meses. Assim, considerando 30 dias por mês, o número acumulado de requisitos que P pode implementar é de 300 no primeiro mês, 600 nos primeiros dois meses e 900 nos primeiros três meses.

Em contrapartida, o mesmo projeto (P’) através de uma estratégia considerando a ocorrência de dívidas, tem produtividade mais elevada no primeiro mês, mas esta diminui à medida que o desenvolvimento prossegue. Neste segundo cenário, P’ entrega 500 requisitos no primeiro mês, 700 nos dois primeiros meses e 900 nos primeiros três meses. Com esta tendência, P' não pode mais prevalecer em termos de funcionalidades entregues após o terceiro mês. Portanto, neste exemplo, o

fim do terceiro mês é o ponto de equilíbrio do projeto. Antes deste ponto, ter DT traz benefícios (menor tempo de resposta para o mercado) para P'. Por outro lado, após este ponto, o efeito negativo da dívida sobrepõe tal benefício e, portanto, ela deve ser paga (que, neste caso, implicaria na reengenharia e/ou redocumentação do projeto e, possivelmente, refatoração do código).

Esta estratégia realça a necessidade de se determinar até que ponto é ou não rentável incorrer em dívida, quando esta deveria ser paga, e quanto vai ser pago de cada vez de forma a minimizar esforços ou maximizar o lucro do projeto.

2.5 IDENTIFICAÇÃO DE DÍVIDA TÉCNICA

O gerenciamento da DT é importante para aprimorar a produtividade da equipe e a qualidade do produto. Porém, antes mesmo de gerenciar a dívida, é necessário identificá-la. No entanto, identificar itens da dívida não é uma tarefa simples uma vez que itens da dívida podem estar presentes em diferentes artefatos do projeto (ZAZWORKA *et al.*, 2013).

Estudos têm demonstrado que diferentes estratégias automatizadas podem ser utilizadas para apoiar a identificação da DT obtendo resultados de valor para os desenvolvedores (SCHUMACHER *et al.*, 2010) (GUO *et al.*, 2011). Duas estratégias que se destacam são: identificação de *code smells* e problemas descobertos através de análise estática automática (ASA). *Code smells* são violações dos bons princípios da programação orientada a objetos no código fonte e podem ser identificados pela comparação de valores de métricas de software com limites previamente definidos (LANZA *et al.*, 2005). Já ASA consiste em extrair informações sobre um programa a partir do seu código fonte usando ferramentas automáticas (BINKLEY, 2007). Ferramentas ASA procuram por problemas em termos de violações de práticas recomendadas de programação e potenciais defeitos que podem causar falhas ou podem degradar requisitos da qualidade de software (como a capacidade de manutenção, eficiência). Os problemas devem ser removidos através de refatoração para evitar danos futuros ao projeto.

Apesar do fato destas estratégias apontarem para fragmentos de código do sistema que necessitam de melhoria, não está claro ainda se elas apontam para os itens da dívida mais importantes sob a perspectiva dos *stakeholders*. Por outro lado, as estratégias manuais são susceptíveis a serem mais demoradas, mas possuem vantagens (pelo menos em teoria) sobre as automatizadas. Uma destas vantagens é que elas podem ser mais exatas, isto é, possuem mais chances de identificar itens da dívida, enquanto análises automatizadas podem revelar muitas anomalias que acabam não sendo relevantes para o projeto.

Nesse sentido, algumas pesquisas já abordaram a questão do quão perto as estratégias automatizadas podem chegar dos resultados da identificação manual de DT. Em um estudo recente, Zazworka *et al.* (2013) investigaram a identificação da DT considerando estratégias manuais e automatizadas. Para isso, eles pediram a uma equipe de desenvolvimento que identificasse itens da dívida em artefatos de um projeto de software em que eles estavam trabalhando. Em paralelo, pesquisadores realizaram a identificação automatizada de DT utilizando ferramentas que permitissem detectar *code smells*, problemas ASA e algumas métricas de projeto. Os resultados da detecção automatizada foram organizados e comparados aos resultados da detecção manual realizada pela equipe de desenvolvimento. Observou-se que as ferramentas utilizadas são especialmente úteis para a identificação de dívida de defeito e projeto, mas não podem ajudar na identificação de outros tipos de dívida, assim, torna-se necessário envolver pessoas no processo de identificação.

2.6 CONSIDERAÇÕES FINAIS

Neste capítulo foram discutidas definições relacionadas à DT, sua classificação, estratégias de gerenciamento e identificação da dívida. Esses conceitos serão a base para o entendimento dos demais capítulos desta dissertação.

Em termos de pesquisa, DT ainda é considerada uma área recente. Ainda não existe um consenso sobre os tipos de DT existentes, uma vez que as informações estão dispersas em vários estudos. Além disso, diversos trabalhos têm sido realizados em diferentes áreas, como indicadores de dívida e estratégias de gerenciamento. Sendo assim, é necessário mapear os tipos de dívida existentes, identificar indicadores que podem ser utilizados para a descoberta da dívida, identificar as estratégias de gerenciamento existentes e entender o nível de maturidade das propostas através de uma análise dos estudos experimentais que têm sido realizados e estão descritos na literatura técnica. Este é o objetivo do mapeamento sistemático da literatura discutido no próximo capítulo.

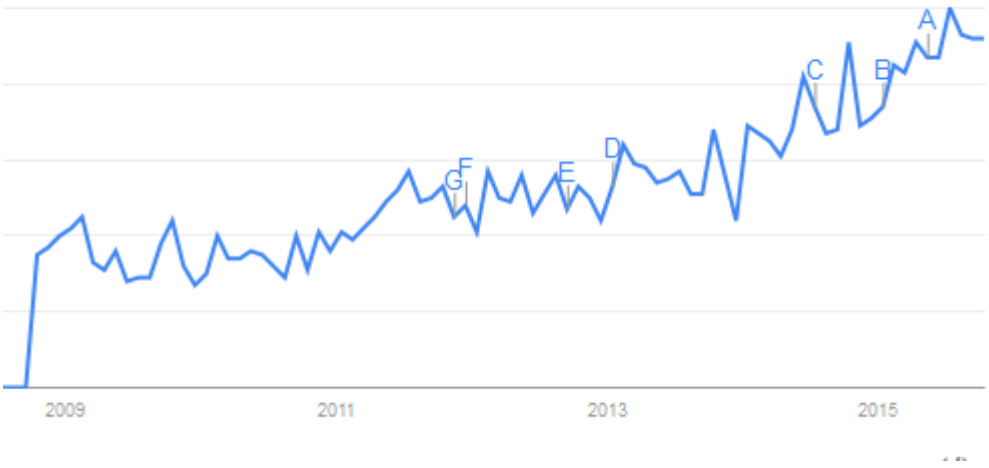
3 IDENTIFICAÇÃO E GERENCIAMENTO DA DÍVIDA TÉCNICA: UM MAPEAMENTO SISTEMÁTICO

Neste capítulo são apresentados os resultados de um mapeamento sistemático da literatura conduzido com o objetivo de identificar quais são as estratégias que têm sido propostas para identificar ou gerenciar a dívida técnica em projetos de software.

3.1 INTRODUÇÃO

Antes de poder trabalhar efetivamente na gestão da dívida técnica, é preciso saber quais tipos de dívida podem ser incorridos, como eles podem ser identificados e quais estratégias podem ser usadas para gerenciá-los. Embora a DT esteja cada vez mais sendo discutida, como reportado em trends.google.com (indicando que ao longo dos últimos sete anos cada vez mais usuários do Google têm procurado pelo termo *Technical Debt*) (ver Figura 3), ainda é difícil ter uma compreensão ampla da área porque as informações sobre ela estão espalhadas na literatura técnica.

Figura 3 - Interesse no termo “Technical Debt” com o passar do tempo no Google Trends



Neste contexto, este capítulo apresenta um mapeamento sistemático da literatura conduzido com o objetivo de responder a seguinte questão de pesquisa: **"Quais são as estratégias que têm sido propostas para identificar ou gerenciar a DT em projetos de software?"**. Um mapeamento sistemático (MS) é um tipo de estudo secundário que se destina a mapear uma determinada área de pesquisa através de um procedimento sistemático que possui como finalidade identificar a extensão e natureza dos estudos primários disponíveis na área mapeada (BUDGEN *et al.*, 2008).

As seguintes questões de pesquisa complementares foram derivadas da questão principal:

- **(Q1)** Quais são os tipos de DT?
- **(Q2)** Quais são as estratégias propostas para identificar a DT?
    - **(Q2.1)** Quais avaliações experimentais foram realizadas?
    - **(Q2.2)** Quais são as fontes de dados e artefatos que têm sido propostos para identificar a DT?
    - **(Q2.3)** Quais técnicas de visualização de software têm sido propostas para apoiar a identificação de DT?
- **(Q3)** Quais estratégias têm sido propostas para apoiar o gerenciamento da DT?
    - **(Q3.1)** Quais avaliações experimentais foram realizadas?
    - **(Q3.2)** Quais técnicas de visualização de software têm sido propostas para apoiar o gerenciamento da DT?

Ao responder estas questões, foram identificados os tipos de DT, os indicadores de sua existência em projetos e as estratégias que têm sido desenvolvidas para apoiar a gestão dessa dívida. Além disso, o grau de maturidade das propostas existentes foi investigado através de uma análise das avaliações experimentais que foram realizadas. Também foi pesquisado como recursos de visualização de software têm sido utilizados para apoiar a identificação e monitoramento da DT, identificando quais metáforas visuais têm sido propostas e quais são as plataformas que estão sendo usadas para apresentar os diferentes tipos de dívida.

Os resultados deste estudo são benéficos para pesquisadores e profissionais. Para a comunidade científica, esse mapeamento fornece informações sobre o estado atual da pesquisa sobre DT, bem como temas que requerem mais investigação. Para os profissionais, foram mapeados os tipos de DT considerados atualmente, bem como estratégias para a sua identificação e gerenciamento. Os profissionais podem utilizar essas informações como base para a adaptação e desenvolvimento de estratégias para controlar a DT em seus projetos.

Além desta introdução, este capítulo possui outras sete seções. A seção 3.2 discute alguns trabalhos relacionados. Na seção 3.3, a metodologia utilizada neste trabalho é apresentada. A seção 3.4 apresenta a execução do estudo, incluindo o processo de definição das questões de pesquisa, o processo de seleção e o esquema de classificação que foi utilizado. Em seguida, na seção 3.5, os resultados do mapeamento sistemático são apresentados. A seção 3.6 discute os resultados alcançados, comparando-os com trabalhos relacionados, e apresenta suas implicações para profissionais e pesquisadores. A seção 3.7 apresenta as ameaças à validade do estudo. Finalmente, a seção 3.8 apresenta as considerações finais deste capítulo.

3.2 TRABALHOS RELACIONADOS

Dívida técnica tem sido cada vez mais investigada nos últimos anos. Um indicador desta tendência é a existência de outros cinco estudos secundários na área já publicados. Nesta seção serão discutidos, em ordem cronológica, os objetivos e os resultados de cada um desses estudos.

Tom *et al.* (2012) apresentaram o primeiro estudo secundário sobre DT. Através de uma revisão sistemática, os autores buscaram proporcionar um entendimento consolidado sobre a DT (questões de pesquisa: Quais são os elementos que compõem a dívida técnica? Por que a dívida técnica surge?) estruturando-o em um quadro teórico, e discutir os resultados positivos e negativos da DT (questão de pesquisa: Quais são as vantagens e desvantagens de permitir o acúmulo da dívida técnica?). De acordo com os autores, o quadro teórico resultante retratou uma visão holística da DT que incorpora um conjunto de precedentes e resultados, bem como o próprio fenômeno (comportamentos, metáforas e elementos).

Em outro estudo secundário da área, Villar e Matalonga (2013) realizaram um mapeamento sistemático inicial respondendo às seguintes questões de pesquisa:

- Quais são as definições atuais de dívida técnica e dívida de desenvolvimento?
- Quais atividades de pesquisa têm sido realizadas na área?
- Como a área tem evoluído ao longo do tempo?
- Quem são os principais pesquisadores na área?

Como resultados, os autores apresentaram:

- 11 definições de DT;
- o número de artigos publicados ao longo dos anos;
- quatro categorias (trabalhos genéricos sobre DT, dívida de código, outros tipos de dívida, *stakeholders* que lidam com DT em seus projetos) que foram criadas para agrupar os artigos analisados, e;
- lista de autores mais ativos (em termos de artigos publicados) na área.

Tom *et al.* (2013) estenderam o trabalho de Tom *et al.* (2012) através da realização de um estudo de caso exploratório que envolveu uma revisão da literatura multivocal que considerou textos disponíveis em blogs na internet, *white papers* e magazines, complementadas por entrevistas com profissionais de software e acadêmicos para estabelecer os limites da metáfora da DT. O objetivo deste estudo foi consolidar a compreensão da natureza da dívida técnica e suas implicações

para o desenvolvimento de software estabelecendo, assim, os limites da metáfora e um quadro teórico mais completo para facilitar futuras pesquisas. Este quadro teórico consiste em um conjunto de dimensões da DT, atributos precedentes e resultados, bem como a própria metáfora e uma taxonomia que descreve e abrange diferentes formas de DT. No entanto, esta pesquisa não fornece uma taxonomia abrangente nem dos tipos de DT, nem de indicadores que possam ser utilizados para apoiar a identificação de diferentes tipos. Além disso, a forma como o estudo foi realizado dificulta sua replicação por outros pesquisadores.

Mais recentemente, Ampatzoglou *et al.* (2015) realizaram uma revisão sistemática focada na análise dos esforços de pesquisa sobre a dívida técnica concentrando-se em seu aspecto financeiro. Especificamente, a análise foi realizada com respeito à forma como termos financeiros são definidos no contexto da DT e como eles se relacionam com os conceitos subjacentes de engenharia de software. Os autores identificaram que os termos financeiros mais comumente utilizados na pesquisa sobre DT são valor e juros da dívida. Além disso, identificaram que as abordagens financeiras que têm sido mais frequentemente aplicadas no gerenciamento da dívida técnica são *options*, gerenciamento de portfólio, análise de custo/benefício e análise baseada em valor. Os autores também destacaram que a aplicação de tais abordagens carece de consistência, ou seja, a mesma abordagem é aplicada de forma diferente em diferentes estudos e, em alguns casos, não há um mapeamento claro entre os conceitos da engenharia software e os da área financeira.

Em outro trabalho relacionado, LI *et al.* (2015) realizaram um mapeamento sistemático com objetivo de analisar estudos sobre DT e gerenciamento de DT, realizando uma classificação e análise temática sobre eles. Seus principais objetivos foram obter uma compreensão abrangente do conceito de DT, uma visão geral do estado atual da pesquisa sobre gerenciamento de DT, identificar os atributos de qualidade que são comprometidos quando DT é incorrida, e oportunidades de pesquisa promissoras. Como resultado, DT foi classificada em 10 tipos, 8 atividades de gerenciamento de DT (por exemplo: identificação de DT, pagamento, prevenção, comunicação e monitoramento) foram identificadas e 29 ferramentas para gerenciamento de DT foram coletadas.

Uma análise sobre a relação de sobreposição e complementariedade entre estes trabalhos relacionados e o mapeamento sistemático realizado nesta pesquisa será apresentada na seção 3.6.1. Comparação dos trabalhos relacionados.

3.3 MAPEAMENTO SISTEMÁTICO DA LITERATURA

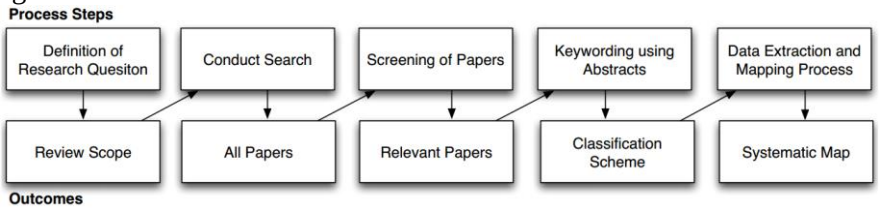
Mapeamento sistemático da literatura é uma ferramenta útil para alcançar a qualidade da informação em uma revisão da literatura. Ele fornece um meio para realizar revisões de literatura abrangentes e imparciais, proporcionando valor científico. O mapeamento sistemático (MS) é um tipo de estudo secundário que tem como objetivo caracterizar uma determinada área de pesquisa através de um procedimento sistemático que visa identificar a extensão e a natureza dos principais estudos disponíveis na área (BUDGEN *et al.*, 2008).

Enquanto uma revisão sistemática (outra abordagem controlada para realizar estudos secundários) é uma forma de identificar, avaliar e interpretar toda a pesquisa relevante disponível sobre uma questão específica (KITCHENHAM *et al.*, 2007) (BIOLCCHINI *et al.*, 2005), o MS pretende mapear a investigação em vez de responder uma questão de pesquisa em detalhe. Budgen *et al.* (2008) afirmam que os estágios iniciais de um estudo de mapeamento são geralmente muito semelhantes aos de uma revisão sistemática da literatura, embora seja possível que a própria questão de pesquisa seja muito mais ampla a fim de abordar adequadamente o escopo mais amplo do estudo. Neste trabalho, optou-se por realizar o mapeamento sistemático ao invés da revisão sistemática devido ao escopo mais amplo do estudo. Pretende-se pesquisar na literatura para determinar quais tipos de estudos foram realizados abordando a questão de pesquisa sobre a identificação e gerenciamento de DT, onde eles são publicados, em quais bibliotecas digitais foram indexados e que tipo de resultados eles têm avaliado.

Um conjunto bem organizado de diretrizes e procedimentos para a realização de MS no contexto da engenharia de software é definido em (PETERSEN *et al.*, 2008) (BUDGEN *et al.*, 2008), que estabelecem as bases para o estudo apresentado neste capítulo. Os principais motivos para executar um MS são identificar sistematicamente as lacunas no corpo atual da pesquisa e apoiar o planejamento de novas pesquisas, evitando a duplicação desnecessária de esforços e erros (BUDGEN *et al.*, 2008). Vale notar que a importância e o uso do MS na área de engenharia de software está crescendo (PETERSEN *et al.*, 2008) (BUDGEN *et al.*, 2008), mostrando a relevância e o potencial do método.

A comunidade de pesquisa de engenharia de software experimental definiu um processo padrão para a realização deste tipo de estudo (PETERSON *et al.*, 2008). A Figura 4 apresenta as fases do MS utilizadas neste estudo. A execução de cada fase será explicada em detalhes nas seções seguintes.

Figura 4 - Fases do MS



Fonte: Petersen, adaptado de *et al.*, (2008).

3.4 IMPLEMENTAÇÃO DO ESTUDO

Nesta seção é explicado como o processo apresentado na Figura 4 foi implementado, incluindo as questões de pesquisa, estratégia de busca, os critérios de inclusão e exclusão, o processo de seleção e o esquema de classificação utilizado.

3.4.1 Definição das questões de pesquisa

Para este trabalho foi definida uma questão de pesquisa principal: **"Quais são as estratégias que têm sido utilizadas para identificar ou gerenciar a dívida técnica em projetos de software?"**.

As seguintes questões complementares foram definidas a partir da questão principal. Ao responder estas questões, será possível ter uma caracterização detalhada dos estudos identificados:

Q1. Quais são os tipos de DT?

Muitas iniciativas têm sido realizadas com o objetivo de investigar a área de DT sob diferentes perspectivas. Uma delas envolve a definição dos diferentes tipos de dívida que podem ser encontrados em projetos de software. É necessário organizar esse conhecimento sobre os tipos existentes de modo que a comunidade de pesquisa de DT possa compartilhar um vocabulário comum. Assim, esta questão de pesquisa tem como objetivo identificar os diferentes tipos de DT encontrados na literatura.

Q2. Quais são as estratégias propostas para identificar DT?

Uma vez que são conhecidos os tipos de DT, também é importante saber como eles podem ser identificados em projetos de software. Existem várias maneiras de identificar os diferentes tipos de dívida. Algumas delas envolvem simplesmente tornar explícitas dívidas como documentação desatualizada ou testes incompletos. Outras abordagens de identificação envolvem o uso de

ferramentas para analisar o código fonte ou outros artefatos para encontrar itens da dívida ocultos no projeto, por exemplo, código mal estruturado, problemas na arquitetura, etc. Cada abordagem de identificação tem associado a ela algum tipo de "indicador", às vezes uma métrica ou algo menos formal, que pode ser utilizado para apontar para áreas com tipos específicos de dívida. Esta questão de pesquisa objetiva, para cada tipo de dívida, identificar os indicadores que têm sido propostos para a sua identificação. Esta associação entre os tipos de dívida e os seus respectivos indicadores é importante porque possibilita, ao se optar por gerenciar um determinado tipo de dívida, escolher indicadores apropriados que podem ser utilizados para sua localização.

Q2.1 Quais avaliações experimentais foram realizadas?

Além de propor tecnologias para a identificação de DT, é importante avaliar as tecnologias que foram desenvolvidas de modo que sua eficácia seja caracterizada. Diferentes estratégias de avaliações podem ser utilizadas tais como questionário, estudo de caso e experimento controlado.

Neste contexto, o propósito desta questão é identificar quais tipos de estudos experimentais têm sido utilizados para avaliar as estratégias de identificação de DT propostas na literatura. Obter essas informações é importante para se ter uma ideia do nível de maturidade das propostas existentes.

Q2.2. Quais artefatos e fontes de dados têm sido propostos para identificar a DT?

Diferentes tipos de dívida podem ser identificados em diferentes artefatos (por exemplo, requisitos, código-fonte) gerados durante o desenvolvimento do software. Além disso, vários tipos de fontes de dados (onde os artefatos são organizados) podem conter informações relevantes para que a descoberta da DT seja possível. Sistemas de controle de versão e gerenciamento de defeito são exemplos de possíveis fontes de dados que podem ser analisados para localizar a dívida em um projeto. O objetivo desta questão de pesquisa é identificar quais artefatos e fontes de dados têm sido considerados em atividades de identificação da DT.

Q2.3. Quais técnicas de visualização de software têm sido propostas para apoiar a identificação de DT?

O software está se tornando cada vez mais complexo em termos de inovação e tamanho. A consequência disso é a crescente quantidade de informação gerada nas atividades de desenvolvimento. Isso torna ainda mais desafiadora a tarefa de analisar os artefatos gerados com objetivo de identificar itens da dívida e, uma vez identificados, monitorar o seu comportamento durante a evolução do software. Uma das ferramentas que podem ser utilizadas para tornar esta tarefa mais fácil é a visualização de software. Técnicas de visualização permitem a representação de informações que muitas vezes são difíceis de analisar em formato textual ou tabular. Considerando

este cenário, esta questão de pesquisa busca identificar se e quais técnicas de visualização têm sido propostas para apoiar a identificação de DT.

Q3. Quais estratégias têm sido propostas para apoiar o gerenciamento da DT?

Tão importante quanto identificar os itens de DT em um projeto é a implementação de uma estratégia de gestão eficiente e eficaz para eles. Técnicas como a análise de custo/benefício e teoria moderna de portfólio (GUO; SEAMAN, 2011) foram propostos com esta finalidade. O objetivo desta questão de pesquisa é realizar uma caracterização das estratégias que têm sido propostas para o gerenciamento de DT. Responder esta questão irá ajudar a entender, por exemplo, quais critérios podem ser utilizados na decisão de pagar a dívida ou mantê-la no projeto.

Q3.1. Quais avaliações experimentais foram realizadas?

O propósito desta questão é identificar quais tipos de estudos experimentais têm sido utilizados para avaliar as abordagens de gerenciamento de DT propostas na literatura.

Q3.2. Quais as técnicas de visualização de software têm sido propostas para gerenciar TD?

Esta questão de pesquisa busca identificar se, e quais técnicas de visualização têm sido propostas para apoiar o gerenciamento da DT.

3.4.2 Condução da Busca

Primeiro, foram escolhidas as seguintes palavras-chave para realizar a pesquisa nas bibliotecas digitais:

- **População**:
  - *Software Project, Software;*
- **Intervenção:**
  - *Technical Debt*
- **Resultados:**
  - *Practice, technique, method, process;*
  - *Identification, identify, gathering, detection, discovery;*
  - *Management, Monitoring.*

A Tabela 1 apresenta a *string* de busca derivada dessas palavras-chave.

Tabela 1 - *String* de Busca

(“Software Project” OR “Software”) **AND**
("technical Debt") **AND**
(“Practice” OR “Technique” OR “Method” OR “Process”) **OR** (“Identification” OR “Identify” OR “Gathering” OR “Detection” OR “Discovery”) **OR** (“Management” OR “Monitoring”)

No entanto, depois de realizados alguns testes em bibliotecas digitais, observou-se que a *string* de busca estava muito restritiva, retornando uma quantidade pequena de artigos. Isto traria o risco de se gerar um mapeamento incompleto. Desta forma, optou-se por considerar uma estratégia de pesquisa mais geral, considerando apenas as seguintes palavras-chave:

- **População:**
  - *Software.*
- **Intervenção:**
  - *Technical Debt;*

Para essas palavras-chave, definiu-se a *string* de busca apresentada na Tabela 2. Essa string de busca foi aplicada para títulos e resumos. Optou-se por não realizar pesquisa de texto completo uma vez que esta resulta em um grande número de estudos de outros domínios.

Tabela 2 - *String* de busca genérica

(“Software”) **AND** (“Technical Debt”)

As decisões tomadas na definição da *string* de busca deste mapeamento trazem algumas

ameaças à validade do estudo. Estas ameaças estão discutidas e justificadas na seção 3.7 Ameaças à Validade.

3.4.3 Fontes de dados

Na escolha das fontes de dados, buscou-se incluir periódicos e conferências relevantes sobre o tema de pesquisa. A pesquisa foi restringida para estudos publicados até dezembro de 2014. Foram incluídas publicações indexadas pelas seguintes bibliotecas digitais e motores de busca: ACM Digital Library, IEEE Xplorer, Science Direct, Engineering Village, Springer Link, Scopus, Citeseer e DBLP. De acordo com (BRERETON *et al.*, 2007), estas são as fontes de dados de artigos recomendadas para pesquisadores de engenharia de software uma vez que elas fornecem acesso aos periódicos e conferências relevantes da área.

3.4.4 Seleção do Estudo

Como a *string* de busca era genérica, a busca retornou uma grande quantidade de artigos que não eram relevantes para a pesquisa. Assim, foi necessário filtrar os estudos retornados considerando critérios de inclusão e exclusão.

Os critérios de inclusão foram:

- Trabalhos publicados que descrevem como identificar ou gerir a dívida técnica;
- Quando vários trabalhos relatam o mesmo estudo, apenas o mais recente será incluído;
- Quando vários estudos foram reportados no mesmo artigo, cada estudo será tratado separadamente.

Os critérios de exclusão foram:

- Trabalhos que não possuem informação sobre como eles lidam com a identificação ou gerenciamento de DT;
- Documentos que estão disponíveis apenas sob a forma de relatórios de workshop/conferência, resumos ou apresentações em Power Point;
- Artigos duplicados.

A identificação e a filtragem dos artigos foi dividida em cinco etapas, conforme mostra a Figura 5. O primeiro passo consistiu na busca por artigos usando a *string* de busca em cada uma das bibliotecas digitais selecionadas para este estudo (ACM Digital Library, IEEE Xplore, Science Direct, Engenharia Village, Springer Link, Scopus, CiteSeer e DBLP).

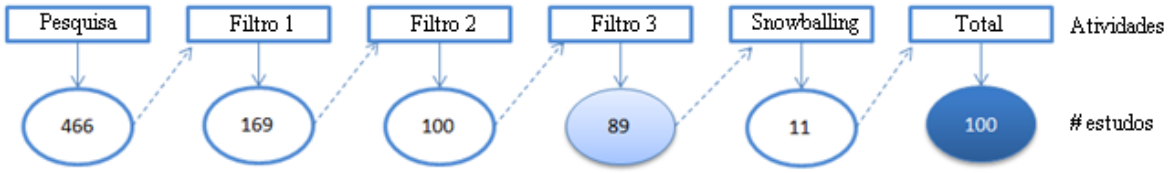
No segundo passo, a primeira filtragem (Figura 5 – Filtro 1) aconteceu e foi realizada por um único pesquisador. Neste processo, foram utilizados os critérios de exclusão, resultando na remoção de todos os relatórios de workshops/conferências, apresentações em Power Point e artigos duplicados.

No terceiro passo, a segunda filtragem foi aplicada (Figura 5 – Filtro 2). Cada artigo foi analisado por dois pesquisadores e, no caso de conflito, um terceiro analisaria e tomaria a decisão de incluir ou não o estudo. A filtragem ocorreu através da leitura dos títulos, abstracts e introdução (caso fosse necessário) usando os critérios de inclusão e exclusão. Quando a decisão sobre incluir ou não o artigo ainda não era possível, este passava para a próxima etapa onde era lido na íntegra. Após esta etapa, 100 estudos foram selecionados para o próximo estágio de filtragem.

Na quarta etapa, o último filtro (Figura 5 – Filtro 3) foi aplicado. Desta vez, os trabalhos selecionados foram lidos na íntegra. Ao final desta fase, mais 12 artigos foram excluídos por não possuírem informação suficiente sobre identificação ou gerenciamento de DT.

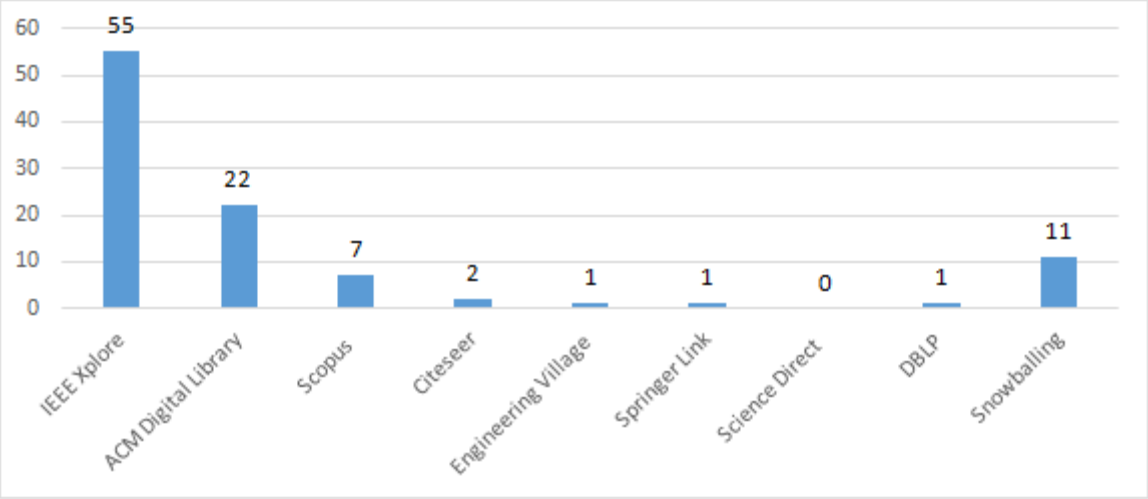
Finalmente, na quinta etapa foi aplicado o *snowballing*, verificando as referências de cada estudo selecionado (89) com o objetivo de não perder quaisquer estudos potencialmente relevantes (BUDGEN *et al.*, 2008). Ao final, os estudos selecionados (11) a partir do processo de *snowballing* foram agregados ao resultado final da seleção de estudos. Os 100 estudos (Apêndice A) selecionados foram utilizados para extrair as informações para responder às questões de pesquisa.

Figura 5 - Processo de filtragem dos artigos



O número final de trabalhos selecionados de cada biblioteca digital é apresentado na Figura 6. A maioria dos trabalhos selecionados vieram das bibliotecas digitais ACM Digital Library e IEEE Xplore. Embora outras bibliotecas digitais também indexem artigos presentes na ACM Digital Library e IEEE Xplore, identificou-se neste trabalho que a indexação de artigos em outras bibliotecas, como a Scopus, ocorre depois dessas duas. Por este motivo, optou-se por priorizar os estudos retornadas por elas quando houvesse qualquer tipo de sobreposição no retorno das buscas.

Figura 6 - Número de artigos incluídos por biblioteca digital



3.4.5 Esquema de Classificação

O esquema de classificação foi estruturado em seis categorias para permitir uma melhor análise das informações: metadados sobre os estudos selecionados, tipos de DT, indicadores de DT, estratégias de gerenciamento, estudos de avaliação e estratégias para visualização software. Cada categoria está relacionada com uma ou mais questões de pesquisa e é apresentada a seguir.

− Metadados dos estudos

A primeira categoria contém dados sobre os estudos selecionados tais como: locais onde os estudos foram publicados, os autores e suas afiliações, tipo de artigo (por exemplo: resumo, trabalho completo ou artigo de periódico) e ano de publicação.

Para coletar as informações para esta categoria, o número de artigos por local de publicação foi computado, bem como o tipo de local (por exemplo, conferências, workshop ou periódico) onde foram publicados. Além disso, os artigos foram classificados em dois tipos: resumos e trabalhos completos. Artigos com até 4 páginas foram considerados resumos; artigos longos são considerados trabalhos completos.

− Tipos de DT(Q1)

Esta categoria inclui informações sobre os tipos de DT. Seus atributos são os tipos identificados e suas definições.

A DT pode ocorrer em diferentes artefatos ao longo do ciclo de vida de um produto. Vários tipos de DT foram estudados e apresentados na literatura. Eles foram extraídos dos artigos das seguintes formas:

- Diretamente: nome do tipo + a palavra dívida (por exemplo: dívida de design, dívida de defeito) ou;
- Indiretamente: identificados em frases como *o desenvolvedor incorre um tipo de dívida relacionado com a arquitetura do projeto* (dívida de arquitetura) ou *este tipo de dívida é derivado de decisões finais sobre a formação de pessoas ou contratação de desenvolvedores* (dívida de pessoas).

Assim, a fim de identificar os diferentes tipos de dívida, três pesquisadores documentaram os tipos e suas definições usando a terminologia extraída diretamente/indiretamente dos artigos. Ao final, esta informação foi consolidada para cada tipo de dívida que foi encontrado.

− Indicadores de DT (Q2, Q2.1, e Q2.2)

Na terceira categoria, informações sobre os indicadores de DT mapeados são representadas. Seus atributos são a lista de indicadores, sua relação com cada tipo de DT, o tipo de avaliação experimental realizada em cada indicador e onde (fonte de dados) cada indicador pode ser encontrado em projetos de software.

A fim de identificar os diferentes indicadores de DT, seguindo a terminologia adotada nos artigos, três pesquisadores coletaram os indicadores mencionados, assim como os tipos de dívida a que eles estavam associados e o artefato de desenvolvimento de software em que eles podem ser identificados. Ao final, as informações coletadas foram consolidadas.

− Estratégias de Gerenciamento da DT (Q3 e Q3.1)

A quarta categoria representa as estratégias de gerenciamento de DT. Todas as estratégias descritas na literatura com o objetivo de gerenciar qualquer tipo de DT no projeto foram listadas. Além das estratégias e suas definições, também foi investigado o tipo de avaliação experimental realizado sobre elas.

A fim de identificar as diferentes estratégias, três pesquisadores coletaram as estratégias mencionadas e suas definições considerando a terminologia adotada nos artigos. Para ser considerada uma estratégia de gerenciamento, foi adotado o seguinte critério: a estratégia precisa apoiar decisões sobre quando e se um item de DT deve ser pago. Ao final, as informações coletadas foram consolidadas para cada estratégia de gestão encontrada.

− Estudos de Avaliação de DT (Q2.1 e Q3.1)

A engenharia de software tem como objetivo apoiar o desenvolvimento de sistemas de software dentro de limites previamente estabelecidos de tempo, custo e qualidade (PFLEEGER, 2007). No entanto, de acordo com Kitchenham *et al.* (2004), o uso de bons processos não é suficiente para melhorar a qualidade durante o desenvolvimento do software. Isso ocorre porque o desenvolvimento é dependente de tecnologias que muitas vezes não apresentam evidências suficientes de suas potenciais vantagens, limitações, esforço de implementação e riscos associados. Para lidar com isso, Kitchenham *et al.* (2004) argumentam que o uso de evidências permitiria a caracterização de uma tecnologia antes da sua adoção em projetos de software por parte da indústria. Isso permitiria determinar, com níveis razoáveis de confiança, a viabilidade de sua utilização aos se considerar cenários específicos de uso.

Existem diferentes tipos de estudos experimentais que podem ser usados para obter evidências sobre a viabilidade e a eficácia de uma abordagem. A aplicação deste tipo de estudo e, consequentemente, a utilização do paradigma experimental para apoiar a avaliação em engenharia de software é importante uma vez que contribui para um maior nível de maturidade. Isto também se aplica à pesquisa sobre DT.

Neste contexto, a quinta categoria apresenta as informações coletadas sobre estudos experimentais encontrados na literatura. Para classificar os tipos de avaliações realizadas, foi utilizada a seguinte taxonomia (WOHLIN *et al.*, 2000):

- **Estudo de caso**: é usado para monitorar projetos, atividades ou tarefas que visam traçar um atributo específico ou estabelecer relações entre diferentes atributos sem muito controle formal sobre as atividades relacionadas para o método experimental;
- **Experimento controlado:** uma investigação experimental que manipula um fator ou variável do ambiente estudado. Com base na aleatorização, diferentes tratamentos são aplicados para ou por assuntos diferentes, mantendo outras variáveis constantes. Ao final, os efeitos sobre as variáveis do resultado são medidos;
- **Estudo etnográfico**: é aplicado para compreender o comportamento do usuário detalhadamente. A ação é observada em uma situação real ao invés de simulada.

Para a análise dos indicadores de DT, os artigos foram classificados em dois tipos: artigos que analisam (aqueles que apresentam indicadores e os descrevem em detalhes) e artigos que apenas citam (aqueles que não entram em detalhes sobre o indicador citado).

− Técnicas de Visualização de Software de DT (Q2.3 e Q3.2)

Técnicas de visualização de software têm sido investigadas em engenharia de software para ajudar no entendimento, manutenção, testes e evolução de sistemas de software (DIEHL, 2007) (NOVAIS *et al.*, 2013).

Técnicas de visualização de software podem apoiar o desenvolvedor na identificação e/ou gerenciamento dos diferentes tipos de DT em projetos de software. A sexta categoria reflete os artigos que fazem uso de técnicas de visualização de software para identificar e gerenciar DT em projetos de software. As técnicas de visualização de software foram classificadas de acordo com (NOVAIS *et al.*, 2013).

3.5 RESULTADOS DO MAPEAMENTO

Para a extração de dados, os estudos incluídos foram lidos na íntegra por três pesquisadores que estiveram diretamente envolvidos no mapeamento. Uma planilha foi usada para coletar e analisar os dados.

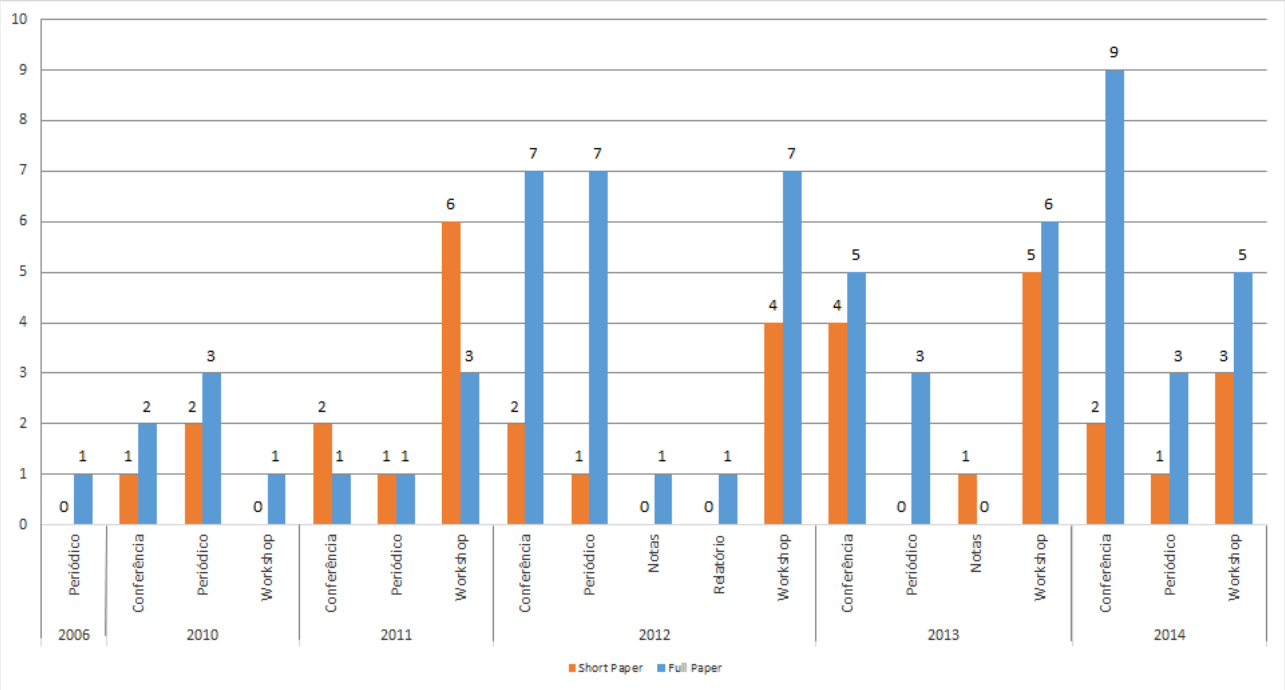
Nesta seção, será apresentada a análise dos dados extraídos dos estudos selecionados. Primeiro é mostrada a análise dos metadados relacionados com publicações e autores. Nas subseções seguintes, os tipos de DT, seus indicadores, estratégias de gerenciamento, estudos de avaliação identificados e tipos de estratégias de visualização de software utilizados serão apresentados.

A maior parte (38 em 100) dos artigos selecionados está concentrada no Workshop Internacional sobre Gerenciamento de DT. Outro pequeno grupo de artigos (5) foi originado de uma edição especial da IEEE Software sobre DT. No entanto, publicações sobre DT já se encontram disponíveis em diversas outras conferências como Agile Conference, IEEE Annual Computer Software and Applications Conference, International Conference on Software Engineering (ICSE), IEEE International Conference on Software Maintenance and Evolution (ICSME), International Symposium on Empirical Software Engineering and Measurement (ESEM) e o Software Quality Journal. Ao total, foram identificados 51 locais diferentes de publicação com um ou dois artigos cada.

Como pode ser visto na Figura 7, a distribuição do estudos sobre os tipos de publicação é de 40% (40 estudos) em workshops, 35% (35 estudos) em conferências, e 24% (24 estudos) em

periódicos. Além disso, tem havido um aumento no número de trabalhos completos e publicações em periódicos, o que indica que a área está amadurecendo.

Figura 7 - Estudos por tipo de artigo, forma de publicação e ano



Os 100 artigos selecionados foram escritos por 115 autores diferentes, mostrando um interesse sobre este assunto na comunidade de pesquisa de engenharia de software. No entanto, verificou-se que apenas 10 pesquisadores (Antonio Vetro', Carolyn Seaman, Clemente Izurieta, Forrest Shull, Ipek Özkaya, Nico Zazworka, Rami Bahsoon, Robert Nord, Yuanfang Cai e Yuepu Guo) foram envolvidos em mais de 5 artigos, cada. Desse grupo, os cinco autores mais ativos são todos colaboradores muito próximos (22 dos 100 trabalhos tiveram pelo menos um desses autores).

3.5.1 Tipos de Dívida Técnica (Q1)

Nesta seção, os tipos de DT e suas definições encontrados neste mapeamento são apresentados ordenados pela frequência que foram citados nos artigos:

- **Dívida de Projeto**: refere-se à dívida que pode ser descoberta através da análise do código fonte e identificação de violações de bons princípios da orientação a objetos (por exemplo, classes muito grandes ou fortemente acopladas) (GUO; SEAMAN, 2011) (IZURIETA *et al.*, 2012);
- **Dívida de Arquitetura**: refere-se a problemas encontrados na arquitetura do software, por exemplo, violação de modularidade, o que pode afetar os requisitos arquiteturais (desempenho, robustez, entre outros) do projeto. Normalmente, este tipo de

dívida não pode ser paga com intervenções simples no código, implicando em atividades de desenvolvimento mais abrangentes (KRUCHTEN *et al.*, 2012);

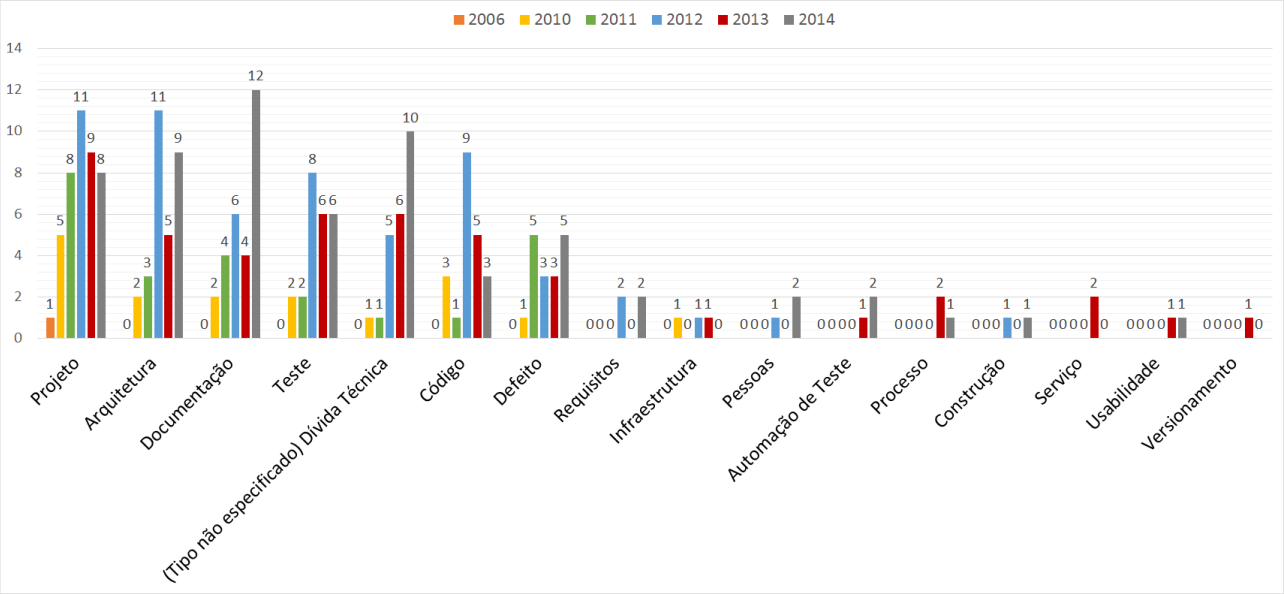
- **Dívida de Documentação**: refere-se a problemas encontrados na documentação de projetos de software e pode ser identificado procurando-se por documentação inexistente, inadequada ou incompleta (GUO ; SEAMAN, 2011);
- **Dívida de Teste**: refere-se a problemas encontrados em atividades de teste que podem afetar a qualidade dessas atividades. Exemplos deste tipo de dívida são testes previstos que não foram executados ou deficiências conhecidas no conjunto de testes (por exemplo, baixa cobertura dos testes) (GUO ; SEAMAN, 2011);
- **Dívida de Código**: refere-se a problemas encontrados no código fonte que podem afetar negativamente a legibilidade do código tornando-o mais difícil de manter. Normalmente essa dívida pode ser identificada ao examinar o código fonte em busca de questões relacionadas a más práticas de codificação (BOHNET ; DЦLLNER, 2011);
- **Dívida de Defeito**: refere-se a defeitos conhecidos, geralmente identificados pelas atividades de teste ou pelo usuário e reportados em sistemas de rastreamento de defeitos, que o comitê de controle de configuração (CCB) concorda devem ser reparados mas que devido a prioridades concorrentes e recursos limitados, têm de ser adiados para mais tarde. As decisões tomadas pelo CCB para adiar o tratamento de defeitos podem acumular uma quantidade significativa de DT em um produto, tornando mais difícil corrigi-los mais tarde (SNIPES *et al.*, 2012);
- **Dívida de Requisitos:** refere-se a decisões tomadas com relação a quais requisitos a equipe de desenvolvimento precisa implementar ou como implementá-los. Alguns exemplos deste tipo de dívida são: requisitos que são implementados apenas parcialmente; requisitos que são implementadas, mas não para todos os casos; requisitos que são implementados, mas de uma maneira que não satisfaz plenamente todos os requisitos não-funcionais (por exemplo, segurança, desempenho, etc.) (KRUCHTEN *et al.*, 2012);
- **Dívida de Infraestrutura**: refere-se a questões de infraestrutura que podem atrasar ou impedir algumas atividades de desenvolvimento. Exemplos deste tipo de dívida são atrasos no ajuste ou atualização da infraestrutura (SEAMAN ; SPÍNOLA, 2013);
- **Dívida de Pessoas**: refere-se a questões relacionadas a pessoas que podem atrasar ou impedir a realização de algumas atividades de desenvolvimento. Um exemplo deste tipo de dívida é o conhecimento concentrado em poucas pessoas como resultado de treinamentos e/ou contratações tardias (SEAMAN ; SPÍNOLA, 2013);
- **Dívida de Automação de Testes**: refere-se ao trabalho envolvido na automatização

de testes de funcionalidades previamente desenvolvidas para apoiar a integração contínua e ciclos mais rápidos de desenvolvimento (CODABUX ; WILLIAMS, 2013). Essa dívida pode ser considerada um subtipo da dívida de teste;

- **Dívida de Processo**: refere-se a processos ineficientes. Por exemplo: o processo definido pode não ser mais o mais apropriado para as atividades atuais da organização (CODABUX ; WILLIAMS, 2013);
- **Dívida de Build (Construção):** refere-se a questões que tornam a tarefa de compilação mais difícil e desnecessariamente demorada. O processo de compilação pode conter código que não possui valor agregado para o cliente. Além disso, se o processo de construção precisa ser executado com dependências mal definidas, o processo se torna desnecessariamente lento. Quando isto ocorre, é possível identificar dívida de construção (MORGENTHALER *et al.*, 2012);
- **Dívida de Serviço:** refere-se à seleção e substituição inadequada de serviços web que levam à incompatibilidade entre as características do serviço e os requisitos da aplicação. Este tipo de dívida leva à sub/super utilização do sistema através da integração de um serviço que não utiliza os recursos do sistema da maneira esperada (por exemplo, falta de memória devido a um serviço que não segue o processo de processamento de dados esperado, ou falta de desempenho devido a um serviço que não usa a memória disponível para a tarefa). Este tipo de dívida é relevante para sistemas baseados em arquiteturas orientadas a serviços (ALZAGHOUL ; BAHSOON, 2013);
- **Dívida de Usabilidade**: refere-se a decisões inadequadas de usabilidade que terão de ser ajustadas mais tarde. Exemplos dessa dívida são a falta de padrão de usabilidade e inconsistência entre os aspectos de navegação do software (ZAZWORKA *et al.*, 2013). (POTDAR ; SHIHAB, 2014);
- **Dívida de Versionamento**: refere-se a problemas de versionamento do código fonte, como uso desnecessário de *forks* (GREENING, 2013).

A Figura 8 apresenta o número de artigos que analisaram ou mencionaram cada tipo de dívida. Observa-se também que alguns artigos não discutiram um tipo específico de dívida e, portanto, estão representados no gráfico pelo termo "Dívida Técnica". Nesses artigos, os autores reportavam algum estudo que analisava a DT sob uma perspectiva mais geral (por exemplo: ponto de vista de desenvolvedores sob o impacto da dívida em projetos, discussão sobre estratégias para gerenciamento da dívida).

Figura 8 - Artigos por tipo de DT ao longo dos anos



A Figura 8 também mostra que em 2010 e 2011 os artigos eram muito concentrados nas dívidas de arquitetura, projeto e documentação. Outros tipos de dívida eram discutidos em menor intensidade: código e teste. Além disso, pode ser observado que os tipos de dívida foram expandidos com o tempo, incluindo o surgimento dos seguintes tipos: serviço, processo, usabilidade e versionamento. Também é possível observar que há uma alta concentração de estudos sobre os tipos de dívida relacionados o código fonte (design, arquitetura, código e defeito). Uma possível explicação para esse comportamento é que já existem ferramentas que executam a análise do código fonte e podem ser utilizadas para apoiar a detecção de DT a partir dele.

3.5.2 Indicadores da Dívida Técnica (Q2, Q2.1, Q2.2, Q2.3)

Indicadores de DT permitem a descoberta de itens da dívida ao analisar os diferentes artefatos criados durante o desenvolvimento de um projeto de software.

A Tabela 3 apresenta os indicadores que foram identificados neste estudo organizado pelo tipo de DT que eles estão associados. Os indicadores identificados foram explicitamente mencionados nos artigos como uma maneira de localizar um tipo específico de dívida, por exemplo: *god classes foram usadas para apoiar a identificação de dívida de projeto ou uma alternativa para identificar problemas de arquitetura é olhar para violações de modularidade.*

Pode-se observar que alguns tipos de dívida, como o de projeto, já possuem um bom número de indicadores. Por outro lado, não foram identificados indicadores para alguns tipos de dívida: processo, infraestrutura, pessoas e usabilidade. Observa-se também que, assim como os tipos de dívida não são ortogonais, também é possível ter alguns indicadores que são mapeados para mais de

um tipo. Por exemplo, a linha que separa dívida de projeto e dívida de código é, algumas vezes, tênue. Dívida de projeto é geralmente mais concentrada no uso de práticas da orientação a objetos. Por outro lado, a dívida de código está mais relacionada às boas práticas de codificação. No entanto, como o projeto se reflete no código, claramente tem-se alguma sobreposição entre estes tipos e, como consequência, também é possível que haja alguma sobreposição entre seus indicadores.

Tabela 3 - Indicadores organizados por tipo de DT

Indicadores	Artigos que analisaram	Artigos que somente citaram	Tipo de DT
Violação de modularidade	***6***	*3*	
Problemas na arquitetura do software	***7***	*2*	
Betweenness Centrality	***1***	*-*	
Augmented Constraint Network (CAN)	***1***	*-*	*Dívida de Arquitetura*
Pairwise-Dependency Relation (PWDR)	***1***	*-*	
Index of Package Changing Impact (IPCI)	***1***	*-*	
Index of Package Goal Focus (IPGF)	***1***	*-*	
Dependências estruturais	***4***	*1*	*Dívida de Arquitetura / Dívida de Projeto / Dívida de Construção*
Problemas de construção (build issues)	***3***	*-*	*Dívida de Construção*
Código sem padrões	***3***	*5*	
Algoritmo lento	*-*	*1*	*Dívida de Código*
Multithread Correctness	***1***	*1*	
Code Metrics (not specified)	***3***	*3*	
Automatic Static Analysis (ASA) Issues	***7***	*2*	*Dívida de Projeto / Dívida de Código*
Code Smells	***38***	*14*	
Grime	***3***	*2*	
Problemas de projeto de software	***2***	*2*	
Low External / Internal Quality	***1***	*2*	*Dívida de Projeto*
Afferent / Efferent Couplings (AC / EC)	***1***	*-*	

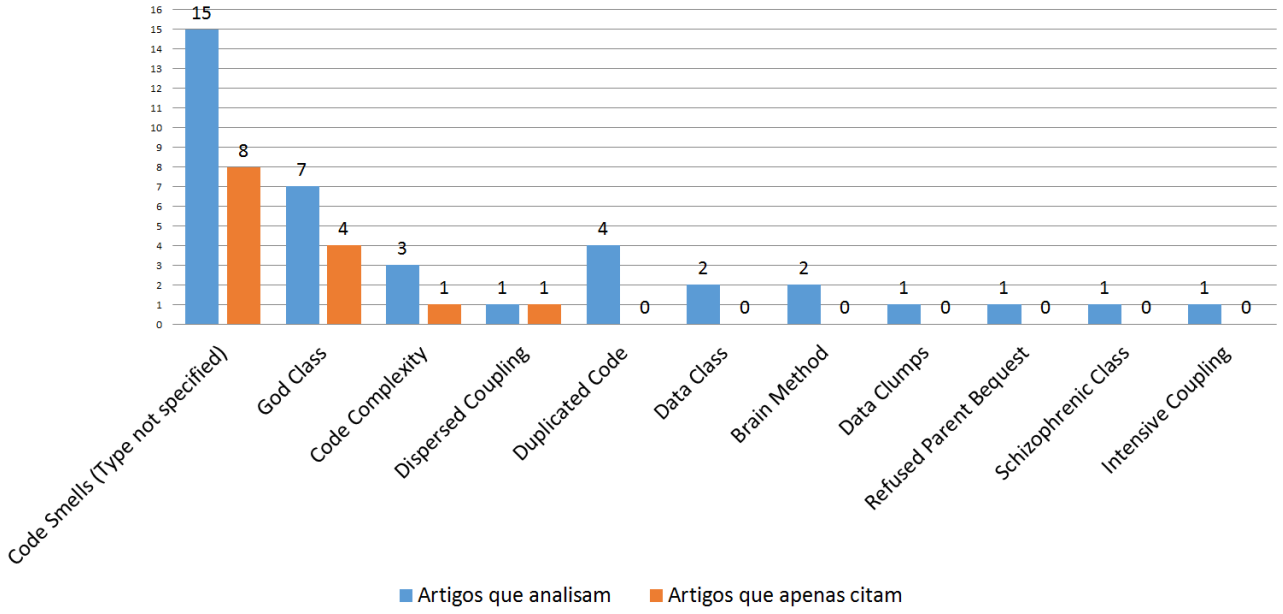
Indicadores	Artigos que analisaram	Artigos que somente citaram	Tipo de DT
Depth of Inheritance Tree (DIT)	***1***	-	
Referential Integrity Constraints (RICs)	***1***	-	
Defeitos conhecidos não corrigidos	***3***	*3*	*Dívida de Defeito / Dívida de Teste*
Comentários insuficientes no código	-	*1*	
Falta de documentação	***1***	-	*Dívida de Documentação*
Comentários (hack, fixme, is problematic, ...)	***1***	-	
Problemas na documentação	***6***	*11*	
-	**-**	-	*Dívida de Infraestrutura*
-	**-**	-	*Dívida de Pessoas*
-	**-**	-	*Dívida de Processo*
Lista de pendências em requisitos	***1***	-	*Dívida de Requisitos*
Serviços web que não atendem as restrições de qualidade do projeto	***1***	-	*Dívida de Serviço*
Falta de teste automatizado	***1***	-	*Dívida de Automação de Teste*
Testes incompletos	***3***	*4*	
Correção de defeitos adiada	***3***	-	
Cobertura de código insuficiente	***1***	-	*Dívida de Teste*
Falta de documentação de casos de teste	***1***	-	
Falta de planejamento de casos de teste	***1***	-	
-	**-**	-	*Dívida de Usabilidade*
Uso desnecessário de forks	**-**	1	*Dívida de Versionamento*

A Tabela 3 também mostra o número de estudos que consideram cada indicador. Para esta análise, os artigos foram classificados em dois tipos: artigos que analisam (aqueles que apresentam indicadores e os descrevem em detalhes) e os artigos que apenas citam (aqueles que não entram em

detalhes). Pode-se observar que, com algumas exceções (por exemplo, code smells, *ASA issues*, documentação, padrões de codificação e violação de modularidade), existem poucos artigos que analisam em detalhes cada indicador. A maioria dos indicadores identificados ou são citados ou são analisados por um único artigo. Isso indica que mais estudos precisam ser realizados a fim de investigar os reais benefícios e limitações de cada indicador ao apoiar a identificação de itens da DT.

Os resultados mostram que o indicador de DT mais citado e analisado é o *code smell*. A Figura 9 apresenta os diferentes tipos de *code smell* que foram considerados como indicadores. Alguns autores não especificam o tipo de *code smell*, nestes casos, os artigos foram agrupados como "Tipo não especificado". Pode-se observar que o tipo que tem sido mais investigado é *God Class*. Uma possível explicação para isso é que *God Classes* são conceitualmente simples de entender, são até 13 vezes mais susceptíveis de serem afetadas por defeitos e até sete vezes mais propensas a passar por mudanças do que classes que não são afetadas por este *smell*, o que os tornam um bom candidato ao iniciar a detecção de DT a partir do código fonte (OLBRICH *et al.*, 2010) (ZAZWORKA *et al.*, 2011).

Figura 9 - Artigos por *Code Smell*



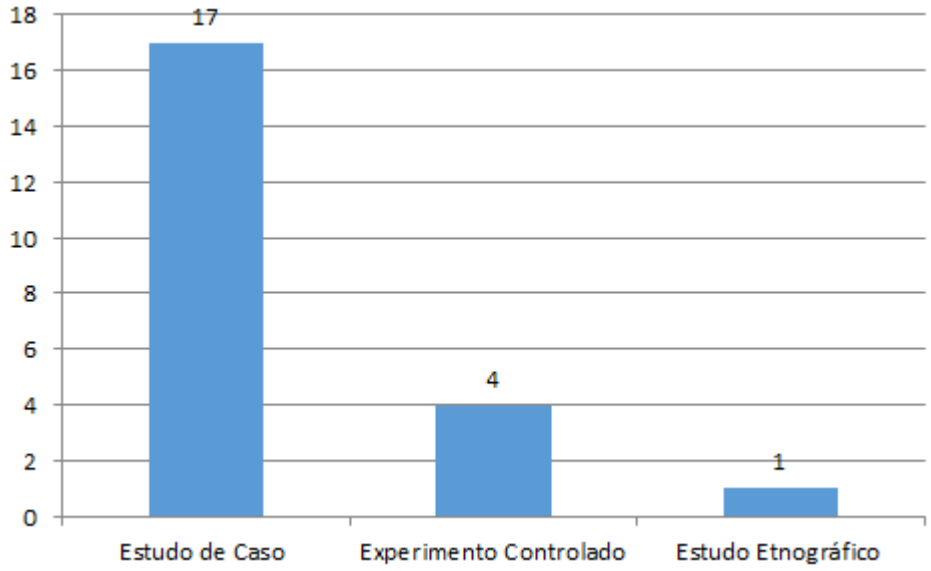
3.5.2.1 Estudos de Avaliação

A comunidade de pesquisa publicou artigos sobre 45 indicadores de DT diferentes ao longo dos anos que foram considerados. Em 2010, quatro indicadores de DT foram identificados, 7 indicadores em 2011, 15 em 2012, 8 em 2013, e 11 em 2014. Também foi observado que à medida que novos tipos de dívida são considerados, novos indicadores também são propostos.

Embora muitos indicadores tenham sido identificados neste estudo, observou-se que apenas alguns deles passaram por algum tipo de avaliação experimental a respeito de usa utilidade ou validade. Para classificar os tipos de avaliação realizados, foi considerada a taxonomia definida em (WOHLIN *et al.*, 2000): estudo de caso, experimento controlado e estudo etnográfico.

Os resultados mostram que estudos de casos e experimentos controlados são os tipos de estudo mais comumente usados entre os indicadores avaliados. A Figura 10 apresenta a distribuição do número de artigos que utilizaram cada tipo de estudo experimental. O método de estudo de caso foi o mais utilizado, sendo identificados apenas quatro experimentos controlados. Isto indica que a maioria dos indicadores propostos ainda requerem mais avaliações de forma que seus benefícios e limitações possam ser conhecidos com mais confiança.

Figura 10 - Artigos por tipo de estudo



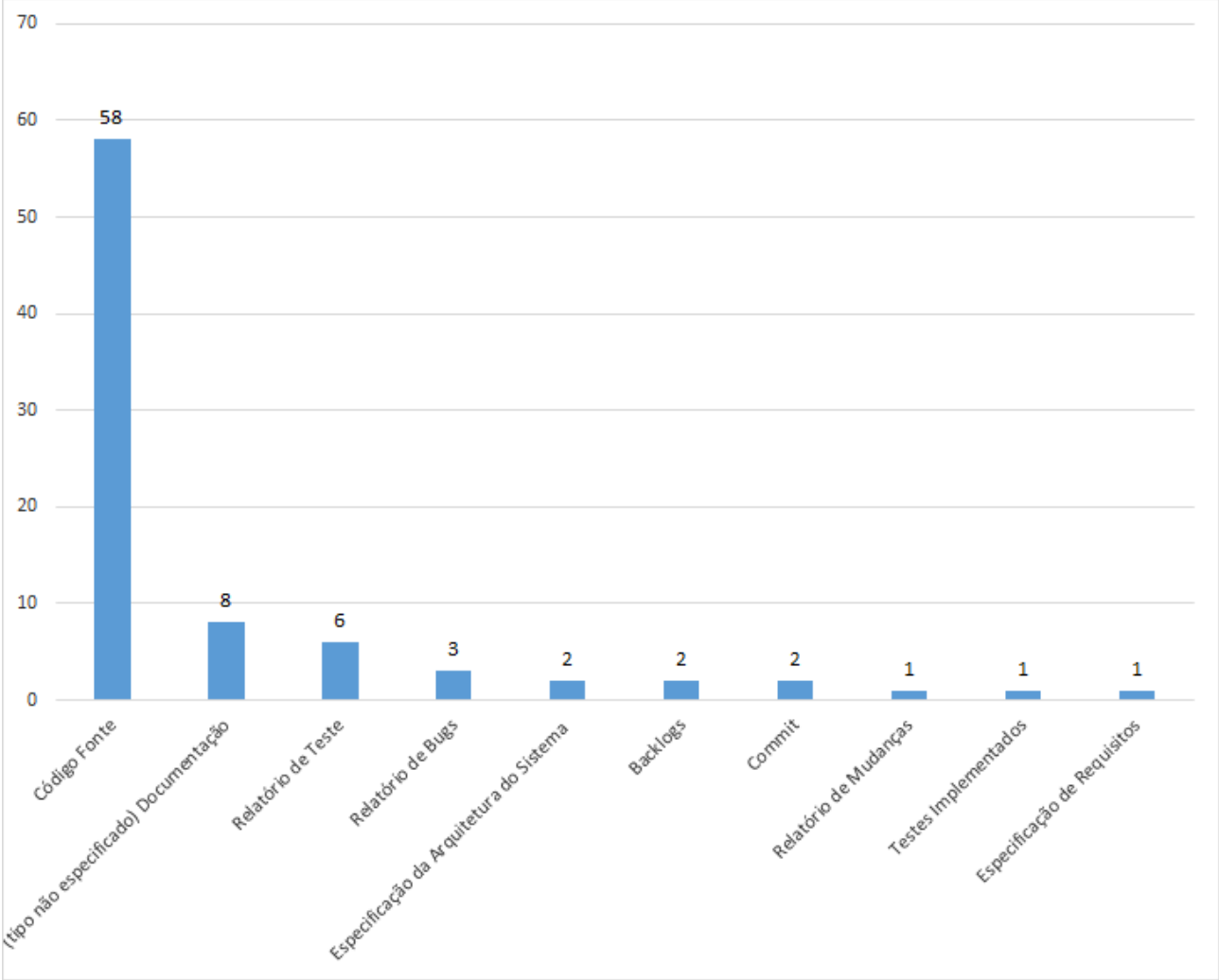
Entre os indicadores avaliados, pode-se destacar *god class* avaliado por seis estudos (três experimentos controlados e três estudos de caso). Isto sugere que a potencial utilidade de *god class* no apoio às atividades de identificação de DT em projetos de software já é bem conhecida. Outro indicador que se destaca é ASA *issues*, que foi avaliado através de cinco estudos de caso e dois experimentos controlados. Além disso, Problemas na Arquitetura de Software e Documentação foram avaliados em cinco estudos de caso. Os indicadores violação de modularidade e dependência estrutural foram avaliados em quatro estudos de caso cada.

3.5.2.2 Artefatos e Fontes de dados

Os indicadores de DT também podem ser classificados de acordo com o artefato no qual eles são identificados. A Figura 11 apresenta os artefatos que foram mais frequentemente utilizados de

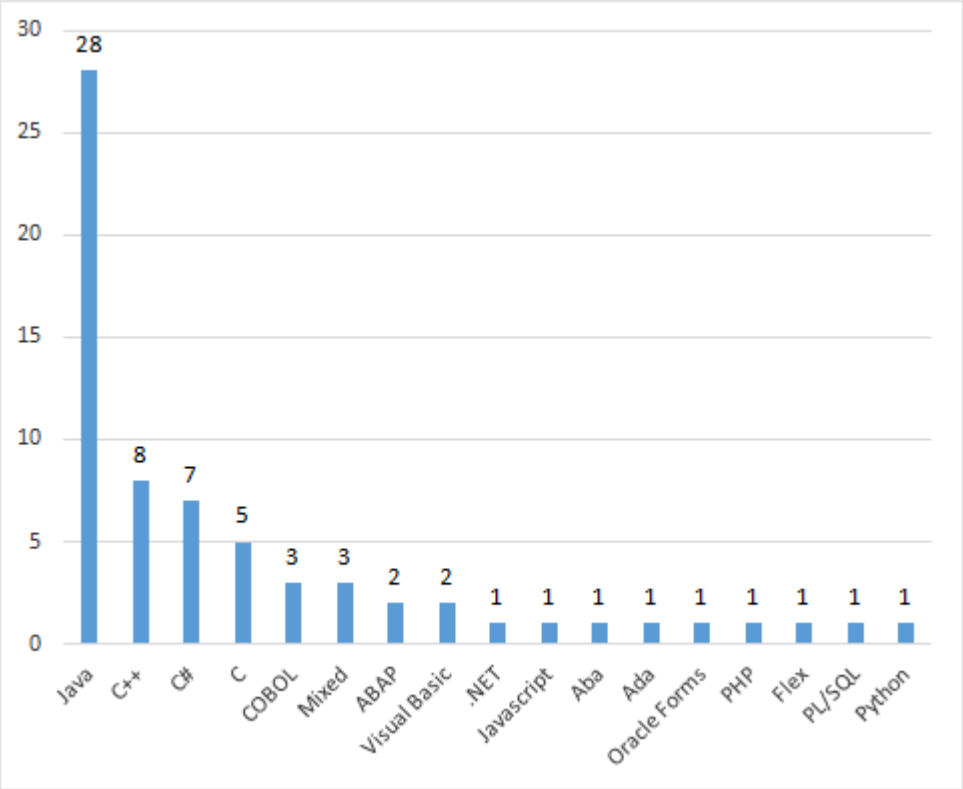
acordo com os estudos analisados. Notou-se um foco maior em estratégias para identificar itens da dívida a partir do código fonte. Outros artefatos foram considerados apenas ocasionalmente. Mais uma vez, uma possível explicação para isto é que já existe uma série de ferramentas que realizam a análise de código fonte e que podem ser utilizadas para apoiar a detecção de DT.

Figura 11 - Quantidade de artigos por artefato considerado



Como complemento aos resultados apresentados na Figura 11, a Figura 12 apresenta as diferentes linguagens de programação utilizadas onde os indicadores de DT foram investigados no código fonte. Embora diferentes linguagens tenham sido consideradas, existe uma maior concentração em Java. Uma possível explicação para isso é a grande quantidade de ferramentas que realizam análise de código Java.

Figura 12 - Artigos por linguagem de programação



Por fim, a Figura 13 apresenta as fontes de dados que têm sido relatadas na literatura como fontes de informação útil para o processo de identificação de DT. Por exemplo, se a DT está localizada no código, a fonte de dado se refere ao local onde o código se encontra armazenado. As fontes de dados identificadas foram:

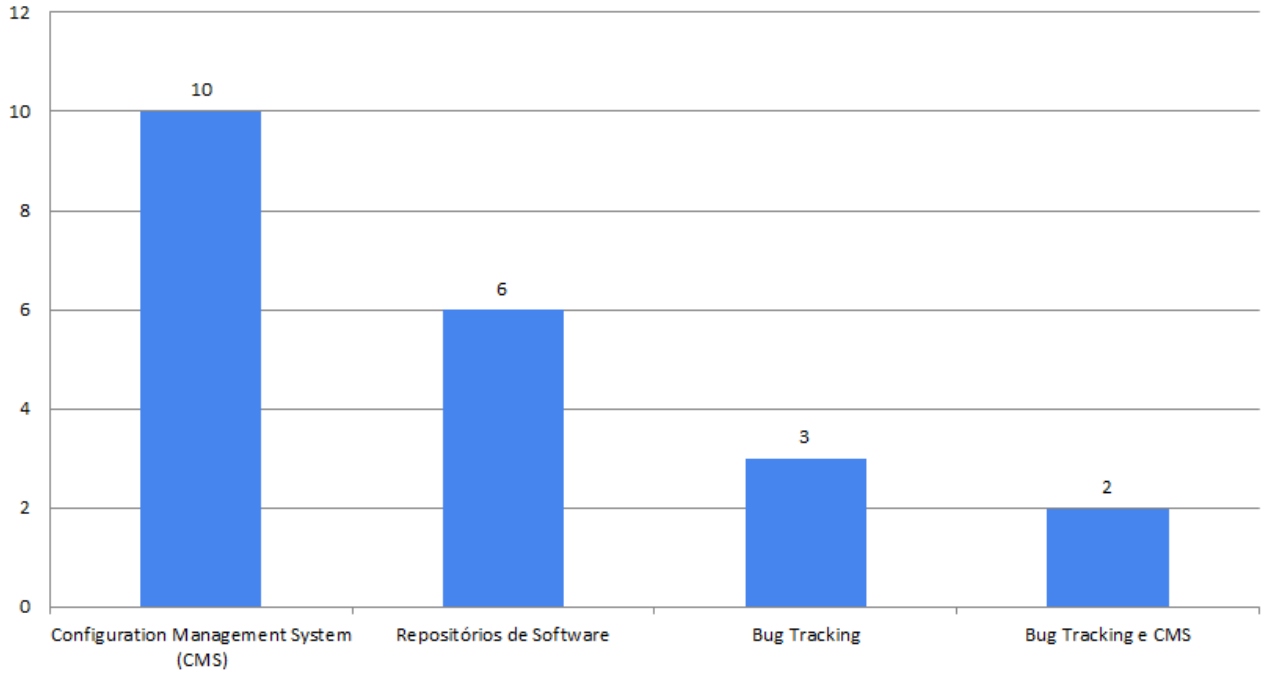
- **Sistemas de Rastreamento de *Bug* (BTS)**: sistemas que registram solicitações de manutenção e mudanças durante o ciclo de vida do software (por exemplo, Bugzilla[1]);
- **Sistemas de Gerenciamento de Configuração (CMS)**: software que permite que a equipe do projeto trabalhe de uma forma controlada e organizada sobre os artefatos criados durante o desenvolvimento do software (por exemplo, sistemas SVN[2] e GIT[3]);
- **Repositório de Aplicações**: refere-se a repositórios que possuem informações sobre softwares que podem ser utilizados, gratuitamente, para a coleta de métricas e realização de estudos (por exemplo, Google Repository[4]);

O tipo de fonte de dados mais comumente utilizado foi Repositórios de Aplicações e CMS, em seis artigos. Repositórios de aplicações são importantes na condução dos estudos, no entanto, o

[1] www.bugzilla.org
[2] https://subversion.apache.org/
[3] http://git-scm.com/
[4] https://code.google.com/

impacto prático de tais estudos é questionável, uma vez que as aplicações são muito diferentes daquelas mantidas por profissionais da indústria de software. Por outro lado, observou-se que BTS também foram utilizados em alguns estudos.

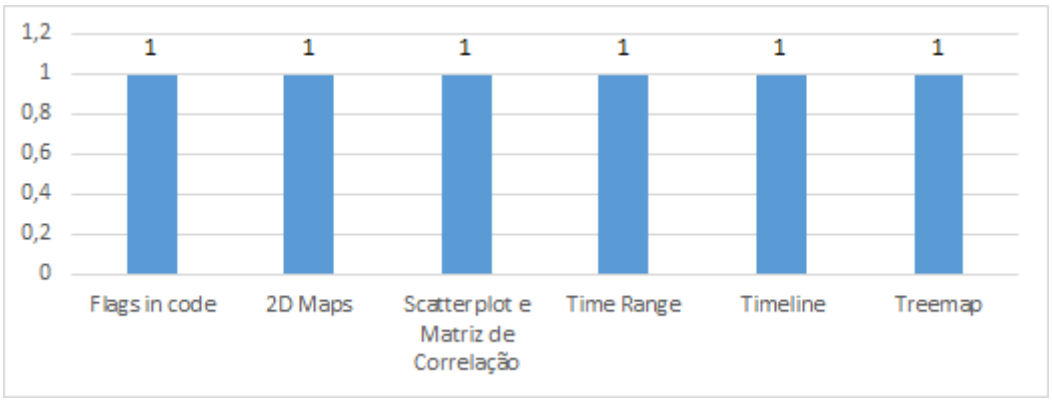
Figura 13 - Quantidade de artigos por fonte de dados identificada



5.2.3 Técnicas de Visualização de Software

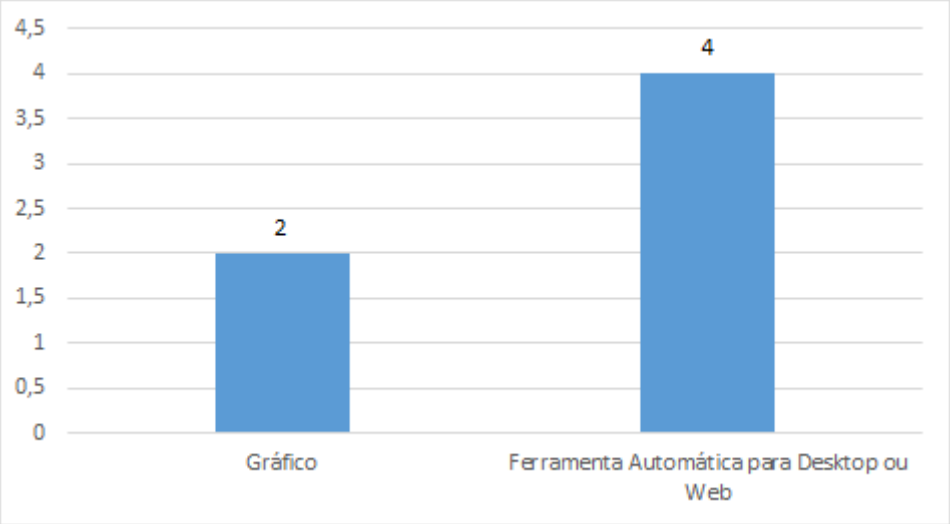
Apenas 6 dos 100 estudos selecionados para análise propuseram técnicas de visualização de software no contexto de identificação de DT, indicando um uso bastante limitado de visualização nesta área (Figura 14).

Figura 14 - Técnicas de visualização de software propostas para apoiar a identificação de DT



Outro resultado associado ao uso de técnicas de visualização pode ser observado na Figura 15. Identificou-se que, apesar de existirem soluções manuais para alimentar os dados da visualização utilizando planilhas eletrônicas, a maior parte dos estudos identificados utilizou algum apoio ferramental para automatizar o uso dos recursos de visualização. Este é um resultado esperado porque as atividades de identificação de DT geralmente envolvem uma grande quantidade de dados do projeto.

Figura 15 - Plataformas utilizadas nas técnicas de visualização de software



3.5.3 Estratégias de Gerenciamento da Dívida Técnica (Q3, Q3.1, Q3.2)

Nesta seção, as estratégias de gerenciamento de DT identificadas na literatura são apresentadas juntamente com suas definições. As estratégias com duas ou mais referências foram:

- **Análise de Custo Benefício**: esta estratégia avalia se o juros esperado é alto o suficiente para justificar o pagamento da dívida. A taxa de juros é composta de duas partes: a probabilidade do juros e o seu valor. A primeira refere-se à probabilidade de que a dívida, se não paga, resulte em custo adicional para o projeto. A segunda é uma quantidade estimada de trabalho adicional que será necessário caso este item não seja pago (SEAMAN *et al.*, 2012);
- **Abordagem de Portfólio**: o conceito central desta estratégia é a lista de itens de DT que contém itens de dívida identificados para o projeto. Esta lista é organizada em uma tabela que contém a localização da dívida, o momento em que foi identificada, a pessoa responsável, a razão pela qual é considerada DT, uma estimativa dos valores da dívida e do juros, e as correlações deste item com outros itens de DT. Durante o planejamento de cada incremento do software, uma análise é realizada sobre o que deve ser pago e o que pode ser

tratado mais adiante (GUO; SEAMAN, 2011);

- **Processo Analítico Hierárquico (AHP):** AHP fornece um método para a estruturação de um problema comparando alternativas de solução em relação aos critérios especificados e determinando uma pontuação total para cada uma das alternativas. Quando aplicado à DT, as alternativas de decisão seriam os vários casos identificados da dívida técnica e o resultado do processo seria um *ranking* priorizando esses itens, indicando aqueles que devem ser pagos inicialmente (SEAMAN *et al.*, 2012);
- **Cálculo do Valor da DT**: esta estratégia foca no valor estimado da dívida. O objetivo é usar um processo definido para estimar o valor da DT e associar os problemas identificados com diferentes atributos de qualidade (ISO 9126). Segundo o autor, associar o valor da dívida a atributos de qualidade auxilia gestores a tomarem melhores decisões (CURTIS, 2012);
- **Marcação de Dependências e Problemas no Código**: esta é uma estratégia utilizada para gerenciar problemas e dependências no código fonte do projeto. O objetivo é inserir *tags* no código do projeto de uma forma que se torne fácil para a equipe de desenvolvimento visualizar onde está inserida a DT e, assim, decidir quando pagá-la com base no esforço envolvido e na disponibilidade de tempo do projeto (HOLVITIE; LEPPANEN, 2013).

Para mais informações sobre as demais estratégias de gerenciamento, observe as referências na Tabela 4. Nela é possível observar que as estratégias de gerenciamento "**Abordagem de Portfólio**" e "**Análise de custo-benefício**" foram as mais citadas. Mas, como elas não foram avaliadas, não é possível afirmar se as duas estratégias são mais eficazes do que as demais.

Tabela 4 - Referências de cada estratégia de gerenciamento

Estratégia de Gerenciamento (Abordagens, Métodos e Modelos)	**Referências**	**Número de Artigos**
Análise de custo benefício	(GUO *et al.*, 2011) (SEAMAN *et al.*, 2012) (STOCHEL, 2012) (HOLVITIE; VILLE, 2013) (MONTEITH e MCGREGOR, 2013) (GRIFFITH *et al.*, 2014) (ALZAGHOUL; BAHSOON, 2014) (OJAMERUAYE ; BAHSOON, 2014) (RAMASUBBU ; KEMERER, 2014) (GUO *et al.*, 2014) (HOLVITIE, 2014)	11
Abordagem de portfólio	(GUO ; SEAMAN, 2011) (SEAMAN *et al.*, 2012) (STOCHEL, 2012) (SNIPES *et al.*, 2012) (POWER, 2013) (ZAZWORKA *et al.*, 2013) (KEN, 2013) (GRIFFITH *et al.*, 2014) (ALZAGHOUL ; BAHSOON, 2014) (OJAMERUAYE ; BAHSOON, 2014)	10
Options	(ZAZWORKA *et al.*, 2011) (SEAMAN *et al.*, 2012) (ALZAGHOUL ; BAHSOON, 2013) (GRIFFITH *et al.*, 2014) (ALZAGHOUL ; BAHSOON, 2014) (OJAMERUAYE ; BAHSOON, 2014)	6

Estratégia de Gerenciamento (Abordagens, Métodos e Modelos)	**Referências**	**Número de Artigos**
Processo analítico hierárquico	(SEAMAN *et al.*, 2012) (ALZAGHOUL; BAHSOON, 2014) (OJAMERUAYE; BAHSOON, 2014)	3
Cálculo do DT-principal	(CURTIS *et al.*, 2012) (CURTIS *et al.*, 2012b)	2
Marcação das dependências e problemas no código	(MORGENTHALER *et al.*, 2012) (TOM *et al.*, 2013)	2
Debt Symptoms Index	(MARINESCU, 2012)	1
Modelo experimental de dívida técnica e juros	(NUGROHO *et al.*, 2011)	1
Abordagem formal de tomada de decisão de dívida técnica	(SCHMID, 2013)	1
Game theoretic competitive source control approach	(MORRISON-SMITH *et al.*, 2012)	1
Measuring symptom severity on a smell thermometer	(LIGU *et al.*, 2013)	1
Métrica para gerenciamento da dívida técnica arquitetural	(NORD *et al.*, 2012)	1
Modelo RE-KOMBINE	(ERNST, 2012)	1
Método SQALE	(LETOUZEY; ILKIEWICZ, 2012)	1
Supply Chain Management	(ALZAGHOUL ; BAHSOON, 2013)	1

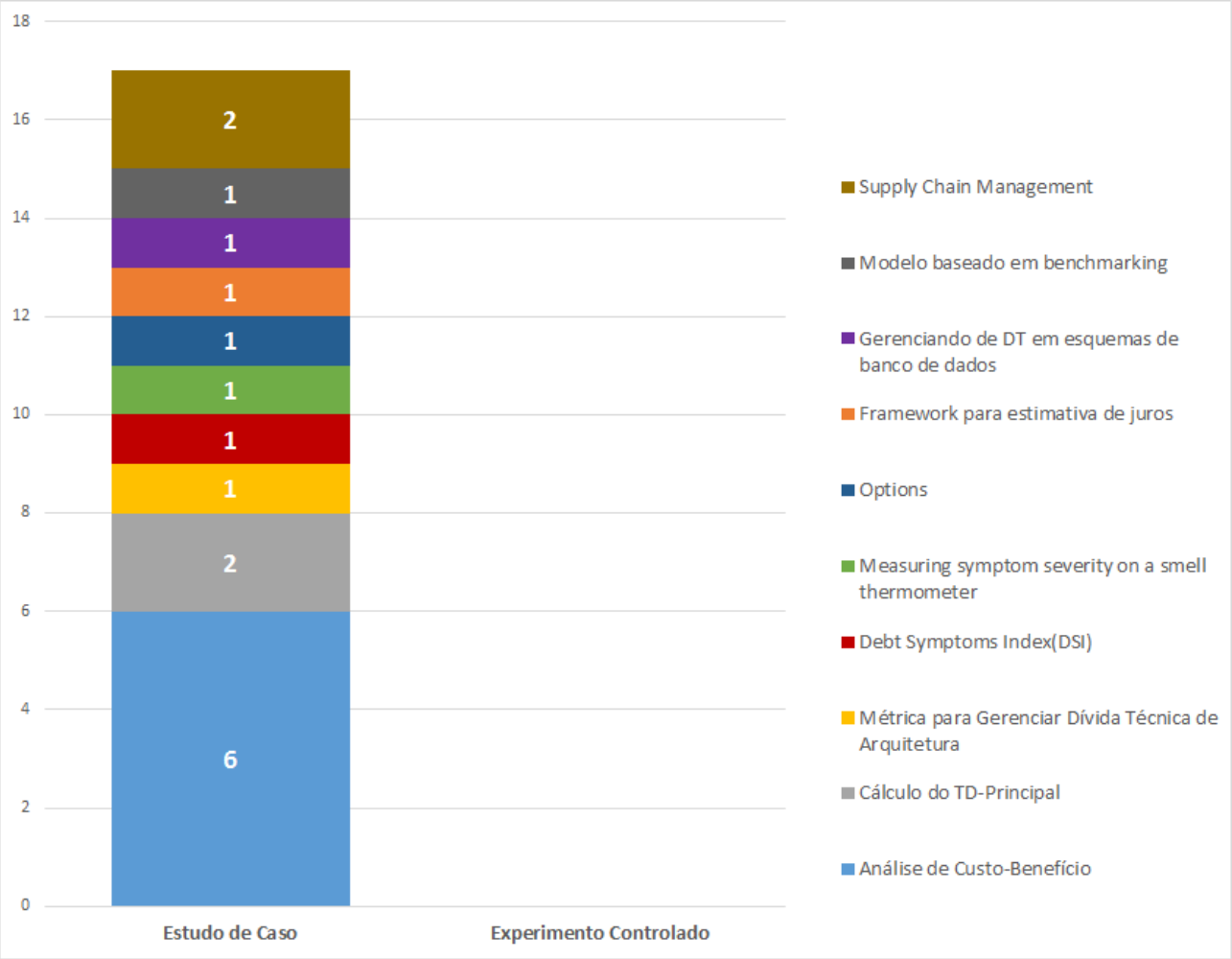
Estratégia de Gerenciamento (Abordagens, Métodos e Modelos)	Referências	Número de Artigos
Framework para estimativa de juros	(SINGH *et al.*, 2014)	1
Gerenciando de DT em esquemas de banco de dados	(WEBER *et al.*, 2014)	1
Modelo baseado em *benchmarking*	(MAYR *et al.*, 2014)	1
Modelo para otimizar dívida técnica	(RAMASUBBU; KEMERER, 2013)	1
Práticas de finanças e contabilidade	(CONROY, 2012)	1

O número de artigos publicados por ano discutindo abordagens de gerenciamento também foi investigado. Em 2010, nenhuma abordagem de gerenciamento foi identificada. Apenas alguns poucos artigos sobre o gerenciamento da DT foram publicados em 2011 introduzindo ideias como a análise de custo benefício e o gerenciamento de portfólio. Em 2012, o número de artigos e o número de estratégias de gerenciamento cresceu bastante, totalizando 16 artigos publicados naquele ano. Em 2013, foi identificado um total de 10 artigos. Finalmente, em 2014, o número de artigos que exploram o gerenciamento da DT também atingiu um número alto de 17 artigos.

3.5.3.1 Estudos de Avaliação

A Figura 16 apresenta o número de artigos sobre estratégias de gerenciamento e o tipo de avaliação realizada. Pode-se observar que os estudos se concentram no tipo estudo de caso. Além disso, nem todas as estratégias foram avaliadas. Isto implica que a maioria das propostas na área de gerenciamento da DT ainda requerem mais avaliações de modo que seus benefícios e limitações possam ser conhecidos.

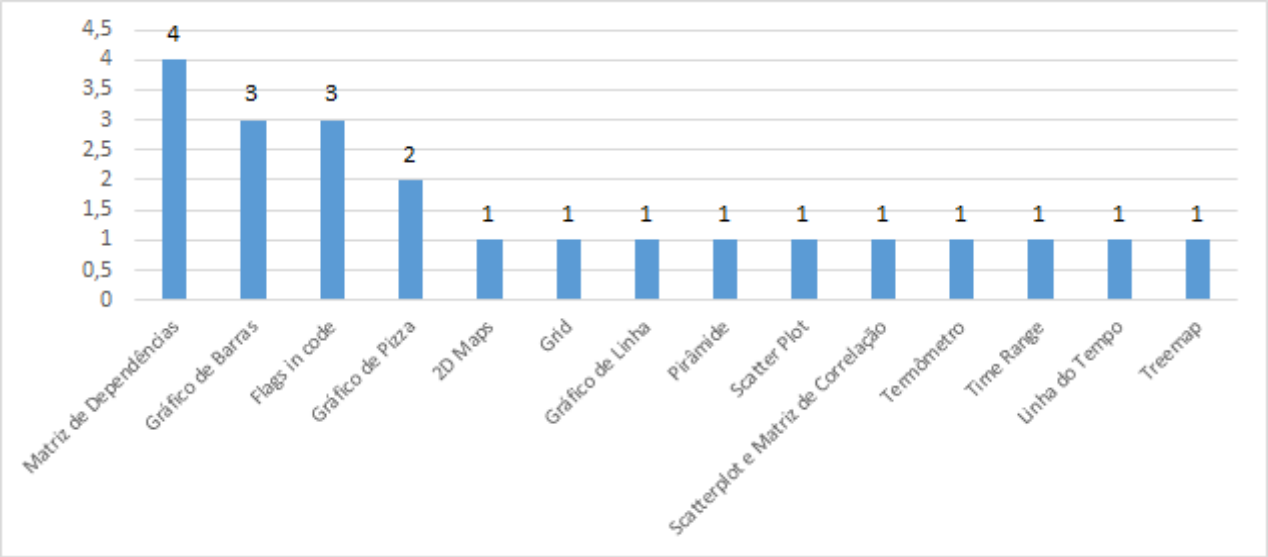
Figura 16 - Artigos por estratégia de gerenciamento e tipo de estudo



3.5.3.2 Técnicas de Visualização de Software

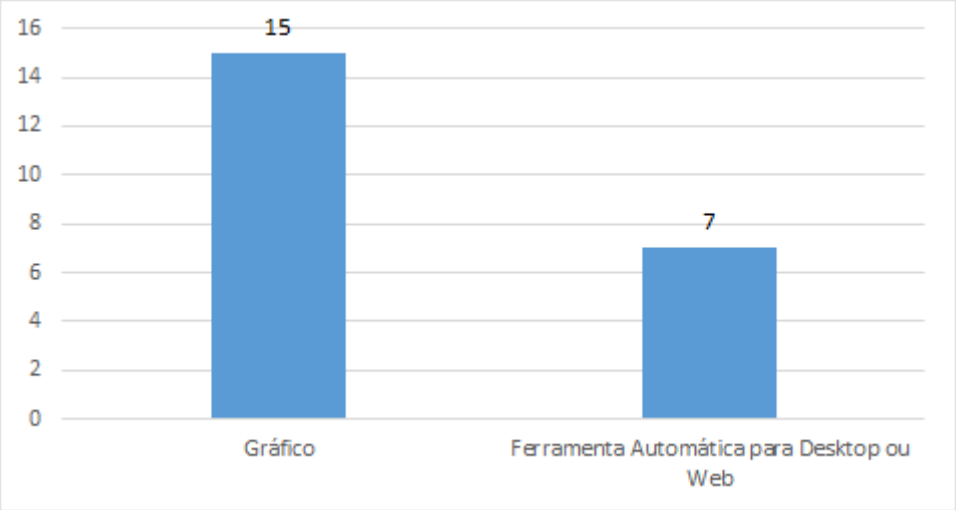
Como pode ser observado na Figura 17, 22 dos 100 estudos propuseram técnicas de visualização de software no contexto da gestão da DT. As técnicas de visualização mais consideradas foram matriz de dependência, gráfico de barras e gráfico de pizza. Uma oportunidade identificada é investigar diferentes tipos de técnicas de visualização de software já propostos em outros contextos da manutenção e evolução de software, e adaptá-los para apoiar o gerenciamento da DT.

Figura 17 - Tipos de técnicas de visualização de software propostas para gerenciar DT



Em complemento, a Figura 18 mostra que o tipo mais comum de plataforma proposta para manipular as visualizações é planilha eletrônica. Este tipo de solução manual é longe do ideal uma vez que requer um grande esforço para manipular os dados extraídos do projeto de software e mantê-los atualizados.

Figura 18 - Tipos de plataformas propostas para exibir as técnicas de visualização de software



3.6 DISCUSSÃO

3.6.1 Comparação dos trabalhos relacionados

Nesta seção será discutido como este mapeamento sistemático difere dos demais estudos secundários identificados como trabalhos relacionados. A Tabela 5 apresenta a correspondência entre as questões de pesquisas encontradas em cada estudo.

No que diz respeito à "*(Q1) Quais são os tipos de DT?*", há uma sobreposição parcial entre os estudos. A sobreposição está associada com o objetivo da questão, mas analisando os resultados de cada estudo, é possível observar uma relação complementar entre eles. Tom *et al.* (2012) classificaram a dívida em sete elementos e, em seguida Tom *et al.* (2013), categorizaram em cinco dimensões. Já o trabalho de (LI *et al.*, 2015) e este estudo de mapeamento identificaram 10 tipos de dívida em comum, mas diferem em 5 identificados apenas neste trabalho. A Figura 19 representa a interseção entre os estudos. Dessa forma, os tipos de dívida identificados neste trabalho complementam outras categorizações relevantes da DT realizadas nos últimos anos (TOM *et al.*, 2013) (LI *et al.*, 2015). No entanto, é importante observar que os tipos identificados de dívida ainda requerem algum tipo de avaliação por parte da comunidade de pesquisa e profissionais de software.

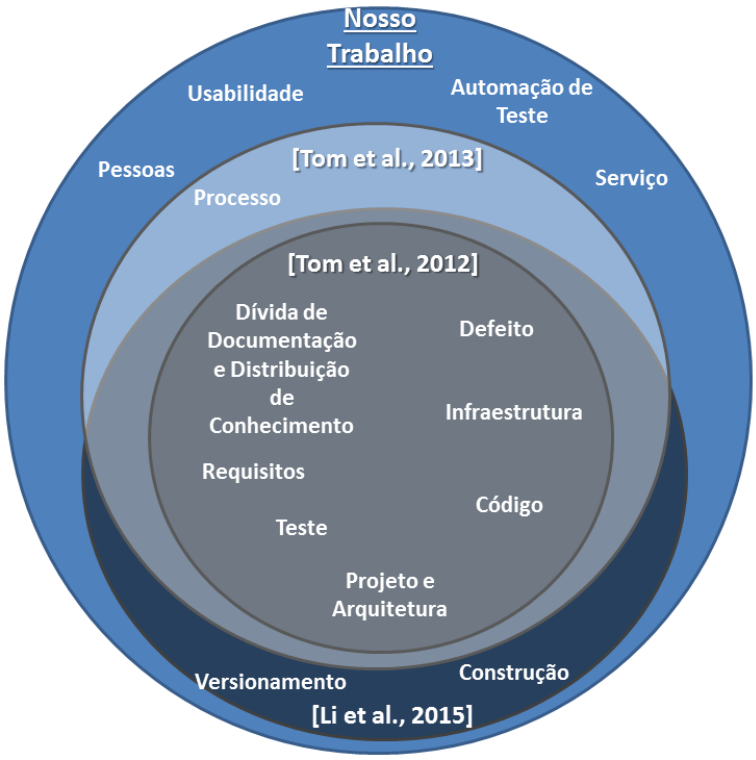
Tabela 5 - Correspondência entre questões de pesquisa

	Mapeamento realizado	**(TOM *et al.*, 2012)**	**(VILLAR MATALONGA, 2013)**	**; TOM *et al.*, 2013)**	**(AMPATZOGLOU *et al.*, 2015)**	**al; LI *et al.*, 2015)**
Questões de Pesquisa	**(Q1)** Quais são os tipos de DT?	**(RQ1)** Quais são os elementos da dívida técnica?	-	**(RQ1)** Quais são as dimensões da dívida técnica?	-	**(RQ1)** Quais são os tipos de DT e quais não são considerados como DT? **(RQ2)** Quais tipos de DT têm sido mais considerados por pesquisadores e profissionais e quais tipos são menos estudados?
	(Q2) Quais são as estratégias propostas para identificar DT?	-	-	-	-	-
	(Q2.1) Quais avaliações experimentais foram realizadas?	-	-	-	-	-
	(Q2.2) Quais artefatos e fontes de dados têm sido propostos para identificar DT?	-	-	-	-	-

Mapeamento realizado	(TOM *et al.*, 2012)	(VILLAR MATALONGA, 2013)	; TOM *et al.*, 2013)	(AMPATZOGLOU *et al.*, 2015)	al; LI *et al.*, 2015)
(Q2.3) Quais técnicas de visualização de software têm sido propostos para apoiar a identificação da DT?	-	-	-	-	-
(Q3) Quais estratégias têm sido propostas para apoiar o gerenciamento da DT?	-	-	-	**(RQ2)** Quais abordagens financeiras têm sido aplicadas na gestão da dívida técnica?	**(RQ6)** Quais são as diferentes atividades do gerenciamento da DT? **(RQ7)** Quais abordagens são usadas em cada atividade do gerenciamento da DT? **(RQ8)** Quais ferramentas são usadas no gerenciamento da DT e quais atividades são apoiadas por estas ferramentas?

	Mapeamento realizado	(TOM *et al.*, 2012)	(VILLAR MATALONGA, 2013)	; TOM *et al.*, 2013)	(AMPATZOGLOU *et al.*, 2015)	al; LI *et al.*, 2015)
	(Q3.1) Quais avaliações experimentais foram realizadas?					
	(Q3.2) Quais técnicas de visualização de software têm sido propostas para gerenciar DT?	-	-	-	-	-
Artigos publicados até	2014	2012 (parcialmente)	2012 (parcialmente)	2012 (parcialmente)	2013 (parcialmente)	2013

Figura 19 - Interseção entre os estudos para a Questão de Pesquisa 1 (tipos de DT)



Sobre "*(Q2) Quais são as estratégias propostas para identificar DT?*" e suas sub questões (Q2.1, Q2.2, e Q2.3), não há sobreposição entre seus objetivos e as pesquisas realizadas nos outros trabalhos relacionados.

Para "*(Q3) Quais estratégias têm sido propostas para apoiar o gerenciamento da DT?*", os objetivos e os resultados de cada estudo são diferentes. Ampatzoglou *et al.* (2015) apresentaram uma lista de estratégias de gerenciamento de DT baseadas em finanças. Este trabalho obteve uma lista mais abrangente de estratégias de gestão (incluindo as estratégias encontradas por Ampatzoglou *et al.* (2015)) porque foi considerada qualquer estratégia de gerenciamento de DT. O trabalho de LI *et al.* (2015) identificou atividades que são executadas (por exemplo: identificação da DT, pagamento, prevenção, comunicação e monitoramento) para gerenciar a DT. Já este trabalho concentra-se em estratégias que têm sido propostas para apoiar o gerenciamento da DT (por exemplo: análise de custo benefício, abordagem de portfólio e Método SQALE). Assim, esta questão de pesquisa possui uma forte relação de complementaridade entre os diferentes estudos e pequena sobreposição entre eles como pode ser observado na Tabela 6. Além disso, não existe sobreposição relacionada à questão de pesquisa "*(Q3.2) Quais técnicas de visualização de software têm sido propostas para apoiar o gerenciamento da DT* ".

No que diz respeito a "*(Q2.1 e Q3.1) Quais avaliações experimentais foram realizadas?*", apesar de não existir uma questão de pesquisa relacionada diretamente com este item nos trabalhos relacionados, cada questão de pesquisa dos demais trabalhos foi analisada a fim de identificar possíveis interseções. Li *et al.* (2015) foi o único trabalho que levou em consideração se a técnica descrita foi avaliada. Os autores usaram essa informação para representar uma visão geral dos níveis de evidência dos estudos selecionados para análise. Em contrapartida, neste trabalho, essas informações foram usadas para caracterizar o nível de confiança que temos sobre os indicadores e estratégias de gerenciamento de DT.

Finalmente, o estudo de mapeamento apresentado neste trabalho considera trabalhos publicados até 2014. Especificamente, isso significou 27 artigos adicionais publicados no ano de 2014 selecionados para análise.

Tabela 6 - Interseção entre os estudos para a Questão de Pesquisa 3 (gerenciamento de DT)

		Mapeamento realizado	**(AMPATZOGLOU *et al.*, 2015)**	**(LI *et al.*, 2015)**
Abordagens de gerenciamento de DT	**Análise de Custo-benefício**	**x**	**x**	
	Abordagem de Portfolio	**x**	**x**	
	Options	**x**	**x**	
	Processo Analítico Hierárquico	**x**	**x**	
	Cálculo do DT-principal	**x**		
	Marcação das dependências e problemas no código	**x**		
	Debt Symptoms Index	**x**		
	Modelo Experimental de Dívida Técnica e Juros	**x**	**x**	
	Abordagem Formal de Tomada de Decisão	**x**		

			Mapeamento realizado	(AMPATZOGLOU *et al.*, 2015)	(LI *et al.*, 2015)
		de Dívida Técnica			
		Game Theoretic Competitive Source Control Approach	**x**		
		Medir a gravidade dos sintomas em um termômetro de *smell*	**x**		
		Métrica para Gerenciamento da Dívida Técnica Arquitetural	**x**		
		Modelo RE-KOMBINE	**x**		
		Método SQALE	**x**		
		Supply Chain Management	**x**		
		Framework para estimar juros	**x**		
		Gerenciamento de DT em *schemas* de banco de dados	**x**		
		Modelo baseado em *Benchmarking*	**x**		
		Modelo para otimizar dívida técnica	**x**	**x**	
		Práticas de finanças e contabilidade	**x**	**x**	
Atividade de gerenciamento da DT		**Prevenção**			**x**
		Comunicação			**x**
		Representação/Documentação			**x**

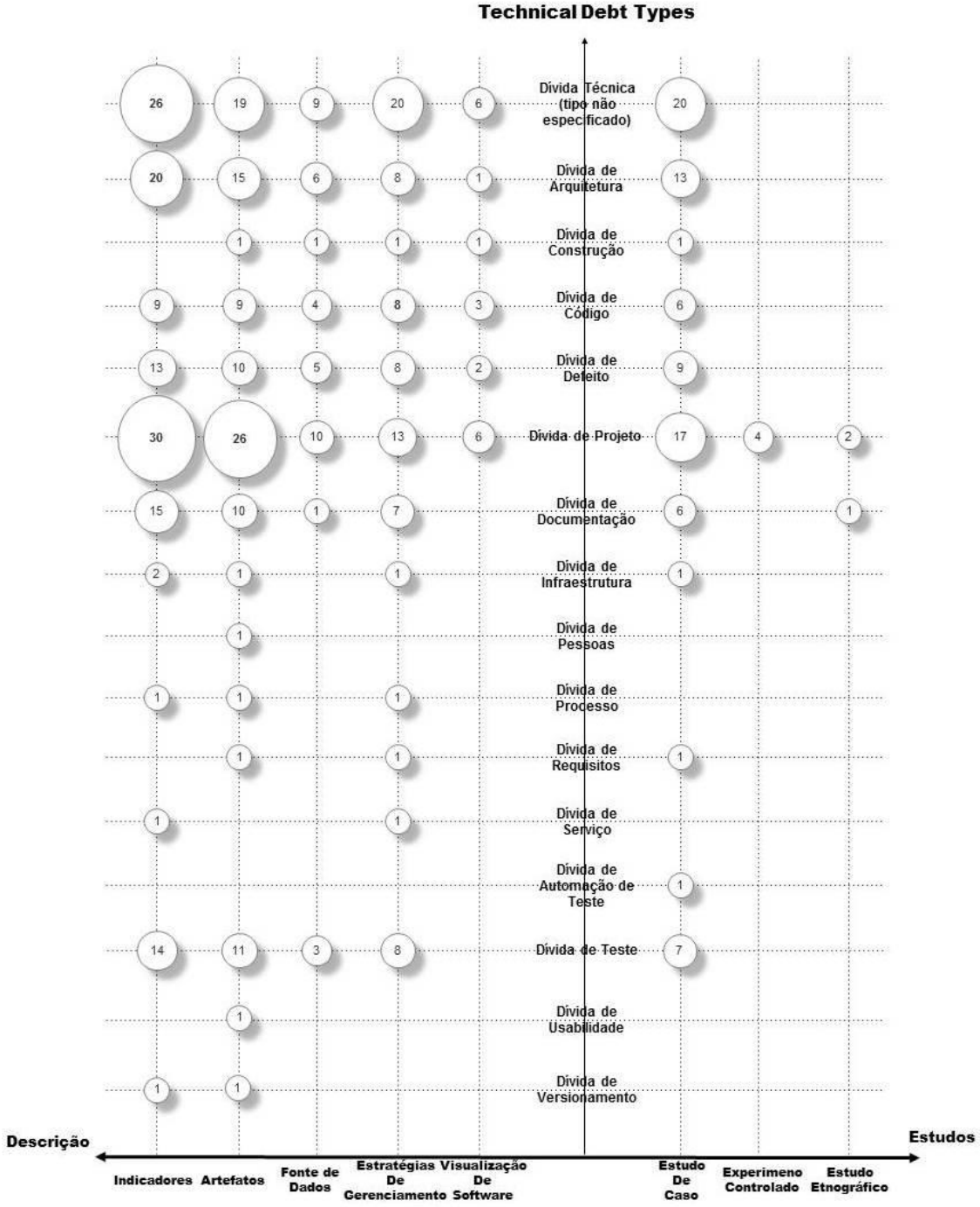
	Mapeamento realizado	(AMPATZOGLOU *et al.*, 2015)	(LI *et al.*, 2015)
Reembolso		**x**	**x**
Monitoramento		**x**	**x**
Priorização		**x**	**x**
Medição		**x**	**x**
Identificação		**x**	**x**

3.6.2 Análise dos resultados

O objetivo deste mapeamento sistemático foi identificar os tipos de DT e as estratégias que têm sido propostas para identificar ou gerenciar a DT em projetos de software. Para consolidar os resultados obtidos, a Figura 20 detalha a relação entre o número de artigos analisados por assunto (indicadores, artefatos, fontes de dados, estratégia de gerenciamento, visualização de software), os tipos de DT e o número de estudos experimentais realizados.

Pode-se observar que a maioria dos estudos lida com DT no nível de código fonte, ou seja, dívida de projeto, defeito, código e arquitetura (área superior esquerda do gráfico de bolhas).

Figura 20 - Visualização dos resultados do mapeamento sistemático utilizando bubble chart



Também é possível observar a existência de tipos de dívida ao longo de todo o ciclo de vida do projeto. Assim, garantir a qualidade do código fonte do projeto não é a única maneira de melhorar a qualidade do projeto. Apesar disso, grande parte dos estudos identificados se concentram em analisar os problemas existentes no código. A ênfase no código pode ser explicada pelo fato de que já há um conjunto considerável de métricas e ferramentas de suporte automatizado para extrair informações que podem ser utilizados como indicadores de

DT (por exemplo, ASA *issues*, *code smells*, entre outros). Outra possível explicação é que o corpo de conhecimento em DT ainda está se consolidando. Por exemplo, tipos de DT como infraestrutura, processo, serviços e automação de teste foram citados apenas em 2013. Estes tipos mais recentes ainda não são totalmente conhecidos pela comunidade. Assim, é muito cedo para dizer se esses tipos mais recentes serão aceitos como tipos "reais" de dívida, ou serão considerados apenas um uso “forçado” do conceito de DT para outras situações.

Este trabalho também identificou vários estudos sobre estratégias de gerenciamento da DT. No entanto, embora várias estratégias tenham sido identificadas, apenas cinco delas (Abordagem de Portfólio, Análise de Custo-Benefício, Processo Analítico Hierárquico, Cálculo do DT-principal, e Marcação das dependências e problemas de código) foram citados em mais de dois trabalhos e algumas delas foram avaliadas. Isso mostra que a maioria dos autores propuseram novas estratégias, mas poucos estão realizando estudos para avaliar a sua real aplicabilidade.

É possível observar na Figura 20 que ainda são poucos os estudos realizados sobre o uso de técnicas de visualização de software para apoiar as atividades relacionadas à DT. No entanto, acredita-se que o uso dessas técnicas pode facilitar o trabalho de identificação e gerenciamento de DT na evolução de projetos de software.

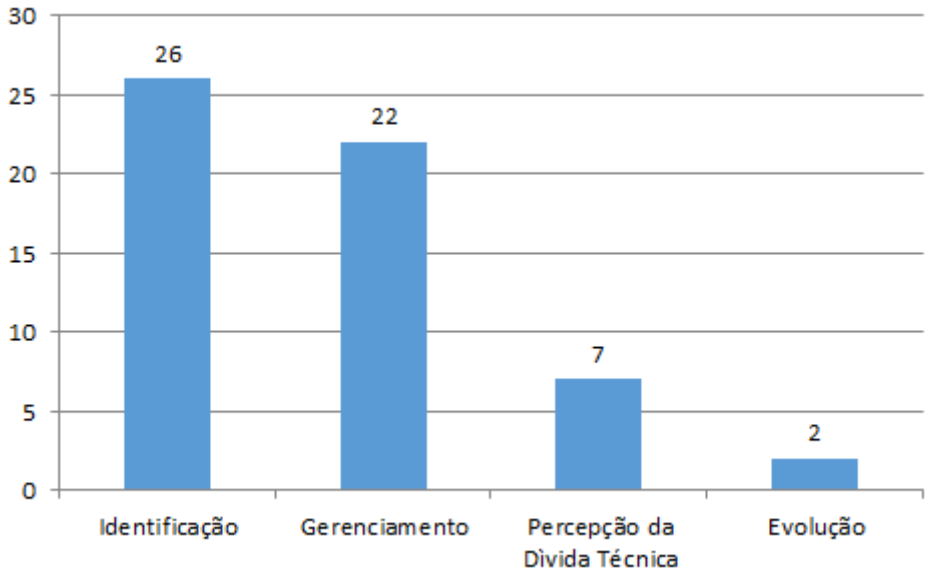
Além disso, quando se observa o lado direito do gráfico na Figura 20, onde os estudos experimentais identificados são apresentados, é possível notar que o conhecimento sobre os reais benefícios e limitações do que foi proposto pela comunidade de pesquisa ainda é limitado. Em complemento, a Figura 21 resume o número de estudos por tópico de pesquisa. As categorias utilizadas para agrupar os tópicos de pesquisa foram extraídos do objetivo de cada estudo experimental analisado:

- **Evolução**: pesquisa que investiga o comportamento dos itens de DT durante a evolução do software;
- **Identificação**: estudos cujo foco é sobre a definição de processos/atividades e indicadores que permitem a descoberta de itens de DT em projetos de software;
- **Gerenciamento**: trabalhos que propõem estratégias para medir a quantidade e o valor das DT incorridas, bem como critérios para definir o melhor momento para pagar a dívida;
- **Percepção da DT**: estudos que investigam DT sob uma perspectiva mais

genérica e lidam com questões como a percepção do desenvolvedor sobre a DT.

Os resultados indicam que há um equilíbrio entre os esforços que visam a avaliação das estratégias de gerenciamento e de identificação da DT. Este resultado é esperado, já que essas atividades são as primeiros que a equipe de desenvolvimento precisa considerar ao se preocupar com a dívida incorrida em seus projetos. No entanto, pouco esforço tem sido investido em estudos relacionados à evolução da dívida durante o desenvolvimento e manutenção do projeto.

Figura 21 - Artigos por tópicos de pesquisa



3.6.3 Implicações para profissionais e pesquisadores

Este mapeamento sistemático da literatura buscou, principalmente, orientar futuras pesquisas sobre identificação e gerenciamento de DT. Entretanto, seus resultados também têm implicações importantes para profissionais, particularmente para aqueles que buscam na literatura orientação sobre como gerenciar DT em projetos reais:

- DT pode ser encontrada em muitos artefatos diferentes produzidos durante o processo de desenvolvimento de software. Como consequência, uma variedade de estratégias deve ser empregada se o objetivo é encontrar todos os tipos de DT que podem trazer um impacto negativo no projeto;
- Existem diferentes indicadores para cada tipo de DT. Assim, os desenvolvedores possuem opções para definir uma estratégia para identificar e rastrear a DT em seus projetos, e

deveriam definir critérios para a escolha dos indicadores mais adequados;

- A visualização de DT através de técnicas de visualização de software ainda é uma tarefa difícil devido à falta de ferramentas de apoio;
- A maioria das pesquisas sobre identificação de DT é relacionada a dívida de código. Isto poderia sugerir que se concentrar em atividades de identificação de DT considerando, inicialmente, a dívida relacionada com código (dívida de código, projeto e arquitetura) faria sentido. No entanto, este tipo de decisão deve ser assumido como sendo um risco porque a dívida pode estar oculta no projeto de formas diferentes (não apenas relacionada a código) e a dívida não relacionada a código também pode trazer impactos negativos significativos para o projeto. Assim, sugere-se evitar limitar o escopo de identificação da dívida àquelas relacionadas apenas a código;
- Foram identificadas diferentes estratégias de gerenciamento de DT. A maioria delas ainda requer uma investigação mais aprofundada e avaliação experimental. No entanto, elas podem ser um bom ponto de partida para a customização ou a definição de uma estratégia de gerenciamento de DT em um projeto de software real.

Para pesquisadores, os resultados deste mapeamento apontam para as seguintes implicações:

- Existem diferentes tipos de DT e alguns indicadores para cada um deles, mas não foi identificada nenhuma evidência sobre como utilizar este conjunto de informações para orientar iniciativas de identificação de DT em situações reais. Apesar dos progressos em diferentes áreas da DT, ainda há uma necessidade de analisar o todo e investigar abordagens holísticas para gerir de forma eficaz a DT na indústria de software;
- Apesar da visualização de software beneficiar o processo de compreensão do software, ainda há pouca pesquisa relacionando DT a essa área;
- Poucos estudos experimentais foram realizados em ambientes reais. Este é um indicador de que, em algumas áreas, ainda não é possível entender completamente todos os custos ou benefícios dos indicadores de DT e estratégias de gerenciamento propostas. Muitas dessas propostas exigem uma investigação mais profunda. Algumas delas foram apenas citadas em alguns trabalhos;
- A pesquisa sobre DT é altamente concentrada em alguns tipos de dívida (projeto, arquitetura, código, e defeito). Isto indica uma lacuna que poderia ser explorada nos próximos anos.

Os resultados obtidos indicam que DT é uma área de pesquisa ativa e produtiva que continua a crescer e que ainda necessita de maior maturidade em termos de conceitos consolidados e novas avaliações experimentais. Para que resultados de pesquisa sobre DT tenham uma contribuição útil para a indústria de software, a comunidade de pesquisa deve encontrar maneiras de orientar os profissionais a utilizar estratégias que tenham maior probabilidade de serem úteis em um contexto particular e a adaptá-las para uma determinada situação.

3.7 AMEAÇAS À VALIDADE

Os resultados deste mapeamento sistemático podem ter sido afetados por algumas ameaças à validade, tais como:

- **Questão de Pesquisa**: o conjunto de questões definidas nesse estudo podem não cobrir toda a área de DT. Para lidar com este risco, as questões formuladas foram analisadas por pelo menos dois pesquisadores, sendo que um deles atuou como revisor externo do protocolo. Além disso, o protocolo foi apresentado na 1ª Escola Latino Americana de Engenharia de Software[5] e foi avaliado por pelo menos mais dois pesquisadores independentes. Todas as observações foram consideradas na definição da lista final de questões de pesquisa;
- **Viés de Publicação**: não é possível garantir que todos os artigos relevantes foram retornados nas buscas. Para minimizar esta ameaça, as principais bibliotecas digitais na área da computação foram consideradas e o processo de *snowballing* foi efetuado;
- **String de Busca**: Existem duas principais preocupações em relação à string de busca. Ambas estão relacionadas com a utilização do termo "*technical debt*" como parte da string:
    - Em primeiro lugar, há alguns estudos relevantes potencialmente publicados antes do termo "*technical debt*" ter sido amplamente utilizado. Por exemplo, *God classes* são consideradas um bom indicador da dívida de projeto, mas elas já existiam antes de terem sido associadas à presença deste tipo de dívida. Assim, existem alguns trabalhos que discutem *God*

[5] www.inf.ufrgs.br/elaes2013

classes que não mencionam DT em seu texto e, por isso, não estão incluídas nesse estudo de mapeamento. No entanto, o escopo do presente estudo foi limitado à forma como esses assuntos foram discutidos sob a perspectiva da DT. Assim, em vez de apresentar uma visão abrangente do que a literatura técnica tem dito sobre *God classes*, havia interesse em saber como a comunidade de pesquisa relaciona *God classes* e DT;

- Segundo: existe um risco de exclusão de alguns artigos que utilizam apenas o termo "dívida" ou um determinado tipo de dívida sem usar o termo "dívida técnica". Para analisar a extensão desse risco, antes de efetuar a procura neste estudo, a *string* de busca foi testada utilizando a string ("*debt*" AND "software"). O resultado era muito genérico e retornou um número muito grande de artigos (por exemplo, 1738 e 5032 documentos nas bibliotecas digitais ACM Digital Library e Science Direct, respectivamente) que não estavam necessariamente relacionados ao objetivo da pesquisa. Por outro lado, para a busca usando os termos ("*technical debt*" AND "software"), os resultados foram mais restritivos (152 na ACM Digital Library e 34 na Science Direct). Estes dados foram analisados através de uma busca manual com o objetivo de verificar se a string mais restritiva estava causando a perda de estudos pertinentes. O resultado foi negativo. Apesar de não existir garantia de que esta busca manual seja 100% exata, acredita-se que ela indicou que é possível usar a string de busca mais restritiva.

- **Extração de Dados**: não se pode garantir que todos os estudos primários relevantes tenham sido selecionados para esse mapeamento e nem que os estudos retornados foram analisados apropriadamente. Para reduzir esse risco, a classificação e extração das informações de cada artigo foi realizada por pelo menos dois pesquisadores.
- **Taxonomia de Estudos**: a escolha da taxonomia de (WOHLIN *et al.*, 2000) para classificar os estudos experimentais identificados durante o mapeamento pode ser considerada uma ameaça à validade, uma vez que existem outros tipos de estudos que têm sido propostos pela comunidade de pesquisa em engenharia de software experimental. Entretanto, a taxonomia proposta em (WOHLIN *et al.*, 2000) é bem aceita e difundida pela comunidade científica.

Por fim, também é importante considerar que o termo *technical debt* é novo. Os artigos começaram a ser publicados recentemente. Entretanto, pesquisas relacionadas podem ter sido realizadas anteriormente. Como esta é uma variável complexa de ser avaliada e não pode ser controlada facilmente, optou-se neste trabalho por considerar válidos apenas artigos desenvolvidos pela comunidade de pesquisa que tratam do tema sob a perspectiva da dívida técnica.

3.8 CONSIDERAÇÕES FINAIS

Este capítulo apresentou o planejamento e os resultados da execução de um mapeamento sistemático da literatura cujo objetivo foi investigar estratégias que têm sido propostas para identificar e gerenciar DT no ciclo de vida do software. Através do mapeamento foi possível fornecer uma visão abrangente sobre o estado atual da pesquisa sobre DT.

Foram identificados 100 estudos primários abrangendo estratégias, técnicas e/ou ferramentas para lidar com DT em projetos de software. Os resultados variam entre terminologia, descrições, artefatos, indicadores, estratégias de gerenciamento, estudos experimentais, fontes de dados e técnicas de visualização de software para identificar e/ou gerenciar a DT. Ao final, as seguintes contribuições foram obtidas:

(i) uma taxonomia inicial de tipos de DT existentes;

(ii) uma lista de indicadores utilizados para apoiar a identificação de DT;

(iii) uma lista de estratégias propostas para o gerenciamento da DT;

(iv) uma análise dos tipos de avaliação experimental realizados nos estudos;

(v) uma lista de fontes de dados utilizadas em atividades de identificação de DT;

(vi) uma lista de técnicas de visualização de software utilizadas para apoiar a identificação e gerenciamento da DT.

Além disso, caracterizou-se o estado atual da arte em DT, identificando possíveis lacunas e temas onde novos esforços de pesquisa podem ser direcionados.

O estudo realizado mostra o crescente interesse de pesquisadores na área de DT. Além disso, o número de novas propostas para indicadores de DT e tipos também está crescendo. Contudo, a avaliação experimental dessas novas propostas tem ficado em segundo plano. Isso

indica que a área de pesquisa de DT está em uma fase de expansão e inovação, mas apenas iniciando uma fase de avaliação estreitando, dessa forma, o campo para práticas mais eficazes. Além disso, novos tipos de dívida aumentam a necessidade de novos indicadores para ajudar a encontrá-los e estratégias de gerenciamento para controlá-los. Apesar de já existir um bom número de estratégias propostas, é necessário realizar mais estudos de avaliação de forma que seja possível apoiar, com maior segurança, os desenvolvedores a controlar a DT em seus projetos.

No Capítulo 4, os tipos de dívida identificados e suas respectivas definições serão organizados em uma taxomonia para a área de DT.

4 TAXONOMIA DE TIPOS DE DÍVIDA TÉCNICA

Neste capítulo é apresentada uma taxonomia de tipos de dívida técnica elaborada para organizar um vocabulário comum para a área. Também é discutido como a taxonomia foi avaliada.

4.1 INTRODUÇÃO

Embora a DT venha sendo cada vez mais discutida, o conhecimento sobre ela está disperso na literatura técnica tornando difícil a definição de um vocabulário comum para a área e a indicação de qual direção seguir a fim de identificar a dívida existente em projetos de software. Existem algumas iniciativas com o objetivo de organizar os diferentes tipos de DT. Por exemplo, as iniciativas de Martin Fowler e Steve McConnell (que consideram dívidas intencionais e não intencionais) foram apresentadas na seção 2.3 do Capítulo 2. Em outros trabalhos, Tom *et al.* (2012) classificaram a dívida em sete elementos e, em seguida Tom *et al.* (2013), categorizaram em cinco dimensões. Já o trabalho de Li *et al.*, (2015) identificou 10 tipos de dívida. Em um trabalho mais recente, Alves *et al.*, (2016) organizaram os tipos de dívida considerando sua natureza como critério de classificação (uma discussão mais completa entre os tipos de dívida identificados neste trabalho e as demais classificações existentes foi apresentada na seção 6.1 do Capítulo 3).

Sistemas de representação e organização do conhecimento são considerados fundamentais em meio à crescente produção de informação (FOGL, 1979). Taxonomias e ontologias são ferramentas que têm sido utilizadas como base para definição destes sistemas. Ontologia, segundo Chandrasekaran *et al.* (1999), é um vocabulário de representação, frequentemente especializado para algum domínio ou assunto. Já taxonomia permite organizar a informação e/ou conhecimento em relações hierárquicas entre os termos (ADAMS, 2000). Neste trabalho optou-se por utilizar taxonomia como mecanismo para representar o conhecimento organizado porque se buscou organizar apenas os conceitos mapeados e suas propriedades. Além disso, não serão definidas restrições e/ou axiomas que levem a uma definição mais formal de cada tipo de dívida organizado. Alguns desses tipos ainda são muito recentes e existe pouca evidência associada a eles de forma que não é possível realizar uniformemente uma análise mais detalhada sobre eles.

Este capítulo apresenta a definição de uma taxonomia para organizar os diferentes tipos de DT considerando a classificação proposta por Alves *et al.*, (2016) e apresentada no Capítulo 3 desta dissertação. Depois de definida, a taxonomia foi avaliada em duas etapas. Na primeira delas, os critérios de qualidade adaptados de Gruber (1995) foram considerados. Em um segundo momento, um especialista da área, que não participou do desenvolvimento da taxonomia, fez uma avaliação inicial do conhecimento organizado.

Além desta introdução, este capítulo possui mais três seções. Na seção 4.2 será apresentado o processo utilizado no desenvolvimento da taxonomia definida neste trabalho. Em seguida, na seção 4.3, a taxonomia de tipos de dívida técnica será apresentada. Por fim, a seção 4.4 apresenta as considerações finais deste capítulo.

4.2 DESENVOLVIMENTO DE TAXONOMIAS

A construção de taxonomias não é uma tarefa simples e deve ser apoiada por práticas da engenharia de software. Neste trabalho foi utilizado um processo estruturado para guiar a sua definição. O processo foi uma adaptação da abordagem SABiO definida por FALBO *et al.* (1998) para apoiar a construção de ontologias. Embora existam abordagens específicas para apoiar a definição de taxonomias, SABiO foi escolhida por já ter sido utilizada no apoio à definição de diferentes vocabulários de conhecimento (FALBO, 2004). Ela considera as seguintes atividades (adaptadas para o contexto de definição de uma taxonomia):

- **Identificação de propósitos e especificação de requisitos:** identifica claramente o propósito e uso previsto da taxonomia;
- **Captura da taxonomia:** captura a conceituação de domínio com base na competência da taxonomia. Além disso, utiliza um modelo que considera uma linguagem gráfica para facilitar a comunicação com os especialistas no domínio;
- **Formalização da taxonomia:** atividade na qual a taxonomia é formalizada. Deve ser utilizado um formalismo para representá-la;
- **Integração de taxonomias existentes:** durante a captura e/ou processo de formalização, pode ser necessário integrar a taxonomia atual com outras já existentes a fim de aproveitar conceituações previamente estabelecidas;

- **Avaliação da taxonomia:** a taxonomia deve ser avaliada para determinar se a especificação satisfaz suas exigências. Além disso, a taxonomia também pode ser avaliadas considerando critérios de qualidade de concepção (GRUBER, 1995);
- **Documentação da taxonomia:** todo o desenvolvimento da taxonomia deve ser documentado incluindo propósitos, requisitos e cenários de motivação, descrições textuais de conceituação, a taxonomia formal e os critérios de projeto adotados.

Este processo é iterativo. Por exemplo, a etapa de captura da taxonomia pode apontar novos requisitos ou, durante a avaliação, pode-se perceber que os termos identificados não são suficientes para finalizar sua definição.

Na próxima seção, a definição da taxonomia para o domínio de dívida técnica será apresentada. Serão discutidas as três fases do processo utilizado: como ocorreu a identificação de propósitos e especificação de requisitos, a captura e formalização da taxonomia e, por fim, sua avaliação.

4.3 TAXONOMIA DOS TIPOS DE DT

4.3.1 Identificação de Propósitos e Especificação de Requisitos

Essa é a primeira atividade realizada na construção de uma taxonomia. Nela é identificada a competência da taxonomia, ou seja, seu uso e propósito, através da delimitação do que é relevante ou não para ela. Neste trabalho, o propósito da taxonomia é **organizar os diferentes tipos de dívida técnica considerando sua natureza como critério de classificação**. Dessa forma, a seguinte questão de competência foi definida:

Quais são os tipos de dívida técnica existentes e que podem ser considerados em projetos de software?

A resposta para esta pergunta é importante porque, até o momento da realização desse trabalho, não há nenhuma outra iniciativa com o objetivo de organizar os tipos de DT em uma taxonomia considerando sua natureza como uma perspectiva de classificação.

4.3.2 Captura e Formalização da Taxonomia

A captura da taxonomia envolve a identificação e especificação de conceitos e seus relacionamentos. Os tipos de DT definidos na taxonomia proposta nesse trabalho, assim como

suas definições, foram identificados a partir do resultado do mapeamento sistemático apresentado no Capítulo 3. Estes tipos não são mutuamente exclusivos e a relação existente entre eles será discutida mais adiante.

Associadas a cada um dos tipos, outras informações relevantes também foram especificadas: indicadores que podem ser utilizados para identificar a DT em projetos e referências onde as informações foram extraídas. A Tabela 3 apresentada no Capítulo 3 detalha os indicadores identificados na literatura técnica para cada um dos tipos de dívida considerado. Ao analisar essa tabela, é possível observar que, embora para alguns tipos de dívida (processo, infraestrutura, automação de teste, pessoas e usabilidade) ainda não tenham sido identificados indicadores, a quantidade deles já é relativamente grande.

Uma vez capturados os conceitos que definiam a taxonomia, iniciou-se sua formalização. Neste trabalho optou-se pelo uso da OWL (Web Ontology Language) (W3C, 2012) (MCGUINNESS; HARMELEN, 2004). Embora seja um padrão para o desenvolvimento de ontologias, optou-se pelo seu uso por ela ser uma ferramenta difundida para representação de conhecimento, além de ter sido projetada para ser usada por aplicações que necessitem processar o conteúdo de informações ao invés de somente apresentar sua visualização. O Apêndice B apresenta a taxonomia definida em OWL. Os tipos de DT foram definidos como subclasses de *Technical Debt* e as demais informações (definição, indicadores e referências) foram especificadas utilizando *Annotations*.

Além disso, ao organizar os tipos de dívida, também é importante considerar que podem existir sobreposições entre classes de dívida, ou seja, indivíduos podem ser classificados como pertencentes a mais de uma classe. Por exemplo, existem indicadores de dívida que permitem identificar dívidas do tipo projeto ou arquitetura. A fim de definir explicitamente esta restrição e separar um grupo de classes dos demais (quando necessário), é necessário torná-los disjuntos um do outro. Isto assegura que um indivíduo que tenha sido classificado como membro de uma das classes no grupo não possa ser membro de qualquer outra classe presente. Por exemplo, um item de DT não pode ser categorizado, ao mesmo tempo, como uma dívida de processo e dívida de teste porque isto não faz sentido. Assim, dívida de teste e dívida de processo são consideradas classes disjuntas. Por outro lado, é possível, em alguns casos, que itens da dívida sejam categorizados em mais de um tipo. Por exemplo, um problema em nível de projeto pode ser considerado, ao mesmo tempo, como sendo uma dívida de projeto ou arquitetura. Para situações como esta, a restrição de disjunção

não foi definida entre os conceitos. A Tabela 7 apresenta as classes disjuntas para cada tipo de DT identificado. As células marcadas indicam que há uma restrição de disjunção entre os tipos.[6]

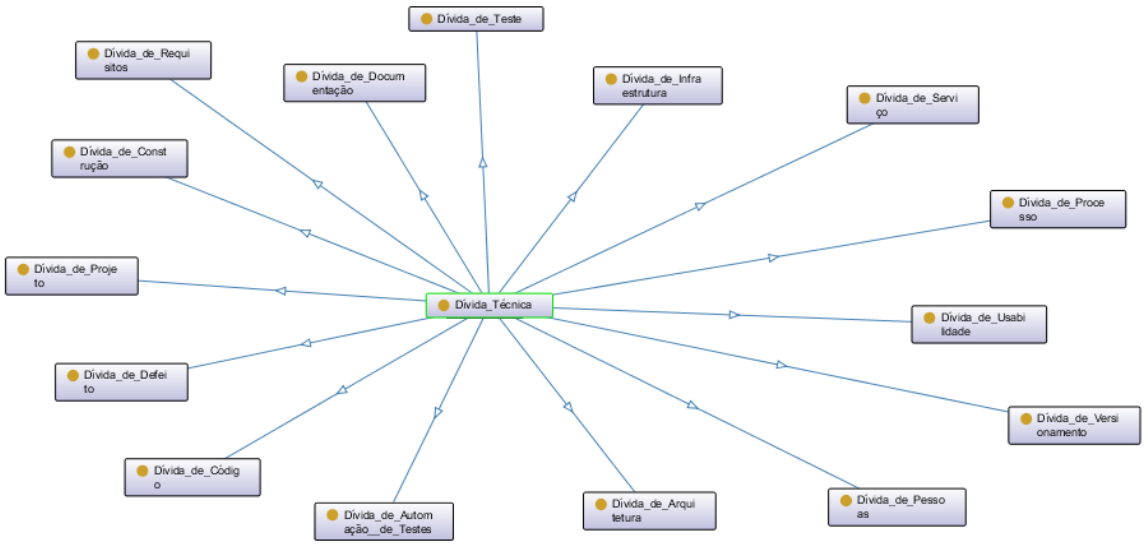
A ferramenta Protégé[1] foi utilizada para apoiar a formalização da taxonomia em OWL. A Figura 22 apresenta a representação visual da taxonomia.

A quarta atividade definida no processo SABiO, Integração de taxonomias existentes, não foi realizada porque não existem outras taxonomias na área de dívida técnica.

Tabela 7 - Definição de disjunção entre os tipos de DT

	Dívida de Arquitetura	Dívida de Construção	Dívida de Código	Dívida de Defeito	Dívida de Projeto	Dívida de Documentação	Dívida de Processo	Dívida de Infraestrutura	Dívida de Automação de Teste	Dívida de Pessoas	Dívida de Requisitos	Dívida de Serviços	Dívida de Teste	Dívida de Usabilidade	Dívida de Versionamento
Dívida de Arquitetura		X	X	X		X	X	X	X	X	X	X	X	X	X
Dívida de Construção	X			X	X	X	X	X	X	X	X	X	X	X	X
Dívida de Código	X			X			X	X	X	X	X	X	X	X	X
Dívida de Defeito	X	X	X		X	X	X	X	X	X	X	X	X	X	X
Dívida de Projeto		X		X		X	X	X	X	X	X	X	X	X	X
Dívida de Documentação	X	X		X	X		X	X	X	X	X	X	X	X	X
Dívida de Processo	X	X	X	X	X	X		X	X	X	X	X	X	X	X
Dívida de Infraestrutura	X	X	X	X	X	X	X		X	X	X	X	X	X	
Dívida de Automação de Teste	X	X	X	X	X	X	X	X		X	X	X		X	X
Dívida de Pessoas	X	X	X	X	X	X	X	X	X		X	X	X	X	X
Dívida de Requisitos	X	X	X	X	X	X	X	X	X	X		X	X	X	X
Dívida de Serviços	X	X	X	X	X	X	X	X	X	X	X		X	X	X
Dívida de Teste	X	X	X	X	X	X	X	X		X	X	X		X	X
Dívida de Usabilidade	X	X	X	X	X	X	X	X	X	X	X	X	X		X
Dívida de Versionamento	X	X	X	X	X	X	X		X	X	X	X	X	X	

Figura 22 - Representação gráfica da taxonomia definida no Protegé



1. Protegé, Software Protegé. Disponível em: http://protege.stanford.edu/, 2011.

4.3.3 Avaliação da Taxonomia

A taxonomia definida foi avaliada considerando seus requisitos e alguns critérios de qualidade. Este processo foi realizado em duas etapas: (1) avaliação a partir dos critérios de qualidade definidos em (GRUBER, 1995); (2) avaliação por um especialista na área.

4.3.3.1 Avaliação a partir dos critérios de qualidade

Para a primeira etapa, a taxonomia foi enviada para dois pesquisadores do grupo de pesquisa sobre dívida técnica TDResearchTeam. Os pesquisadores foram convidados a avaliá-la considerando os seguintes resultados possíveis para os critérios de avaliação definidos (ver Tabela 8):

- **Atendido Totalmente (AT)**: a taxonomia está em conformidade com todas as variáveis consideradas no critério;
- **Atendido Parcialmente (AP)**: a taxonomia está parcialmente em conformidade com as variáveis consideradas no critério;
- **Não Atendido (NA)**: a taxonomia não está em conformidade com todas variáveis consideradas no critério.

Se o resultado da avaliação fosse parcialmente atendido ou não atendido, os avaliadores deveriam relatar os problemas identificados para que a taxonomia pudesse ser melhorada.

O resultado da primeira etapa da avaliação pode ser observado na Tabela 8. Esta apresenta os critérios de qualidade adaptados de Gruber (1995) e o resultado de sua aplicação. No geral, alguns pequenos ajustes foram solicitados de forma a tornar mais clara a diferença entre alguns tipos de DT e também a melhoria na definição de alguns exemplos. A partir destes resultados, melhorias foram efetuadas na taxonomia de forma que ela pudesse atender às não conformidades identificadas.

Tabela 8 - Avaliação da taxonomia segundo os critérios definidos

Critérios	Avaliação		Comentários
	Pesquisador 1	Pesquisador 2	
Clareza: a taxonomia deve comunicar efetivamente o significado pretendido dos termos definidos. As definições devem ser objetivas. Todas as definições devem ser documentadas com a linguagem natural.	AT	AT	Os tipos de DT mapeados foram definidos e documentados para esclarecer a diferença entre alguns tipos de DT.
Extensibilidade: Uma taxonomia deve ser projetada para antecipar o uso do vocabulário compartilhado. Assim, outras pessoas devem ser capazes de definir novos termos com base no vocabulário já existente sem a necessidade de revisar as definições existentes.	AT	AP	A taxonomia organizada neste capítulo é o primeiro passo em busca de uma organização mais abrangente sobre o conhecimento em dívida técnica considerando seus tipos. Dessa forma, espera-se que ela seja evoluída a partir das definições formalizadas. Nesse contexto, um dos avaliadores solicitou que algumas definições fossem melhoradas de forma a facilitar sua extensão.
Viés de codificação mínimo: A conceituação deve ser especificada no nível de conhecimento sem que haja uma limitação por conta da tecnologia de representação do conhecimento utilizada.	AT	AT	Os conceitos definidos e documentados na taxonomia definida não sofreram influência de limitações existentes na linguagem (OWL) escolhida para sua representação.
Compromissos mínimos: Uma taxonomia deve exigir o mínimo compromisso suficiente para apoiar o conhecimento pretendido de atividades compartilhadas.	AT	AT	A taxonomia definida organiza um vocabulário comum para a área de DT, é extensível e não faz uso de um formalismo muito extenso de maneira que seu uso seja facilitado para a comunidade de pesquisa.

Fonte: Gruber (1995).

4.3.3.2 Avaliação por um especialista na área

A segunda etapa da avaliação realizada considerou as recomendações de (GRUBER, 1995) (GUARINO, 1998) que indicam que representações de conhecimento devem ser embasadas no consenso de um grupo de especialistas da área. Neste trabalho, esta atividade foi desempenhada por um especialista na área de DT externo ao grupo de pesquisa TDResearchTeam. Para isso, os tipos identificados de DT e suas respectivas definições foram organizados em um formulário e enviados para avaliação. O especialista que participou desta etapa é um pesquisador sênior sendo, atualmente, um dos mais atuantes na área de DT e estudos qualitativos em engenharia de software experimental.

Como resultado da avaliação pelo especialista, um conjunto de ajustes foram sugeridos no sentido de tornar mais claras algumas definições que, em alguns casos, estavam descritas sob diferentes pontos de vista ("Algumas de suas definições descrevem um tipo de dívida a partir do ponto de vista de como a dívida é identificada, enquanto outras definições são mais sobre como a dívida foi incorrida"). Além disso, foi indicado também que os tipos identificados faziam sentido, nenhum novo foi sugerido ("Acho que as descrições são muito claras [...] Existem alguns tipos de dívida que você tem aqui que eu não havia pensado antes, mas eles fazem sentido. Não consigo imaginar quaisquer outros tipos que você não tenha incluído."). Finalmente, o especialista mencionou que ainda não há um consenso na comunidade de pesquisa de DT se a dívida de requisitos pode ser realmente considerada como um tipo de dívida técnica ("Uma outra questão é a dívida requisitos. Não existe um consenso na comunidade de pesquisa sobre se ela deve ser considerada dívida técnica ou não"). Apesar disso, o especialista na área considerou importante manter os conceitos na taxonomia proposta uma vez que esta pode ser um importante meio para discussão pela comunidade de pesquisa.

4.4 CONSIDERAÇÕES FINAIS

Os resultados apresentados neste capítulo contribuem para a evolução do *Technical Debt Landscape* (JIE *et al.*, 2006) (KRUCHTEN *et al.*, 2012) através da organização dos diferentes tipos de dívida técnica que têm sido considerados pela literatura técnica em uma taxonomia (Alves *et al.*, 2014). Esta permite o compartilhamento de um vocabulário comum para comunidade de pesquisa em DT e profissionais da área.

A construção de taxonomia é um grande desafio, especialmente quando se espera que a ela tenha relevância e valor para um público amplo (MORGENTHALER *et al.*, 2012). Embora a organização dos tipos de DT em uma taxonomia seja uma tentativa inicial de organizar o conhecimento sobre DT, a lista dos tipos considerados pode não ser completa, levando à necessidade de exclusão ou inclusão de alguns deles. A fim de obter uma taxonomia melhor definida, a colaboração entre diferentes especialistas na área precisa ser estimulada, permitindo que cada um deles participe ativamente de seu desenvolvimento (JIE *et al.*, 2006) (ROSPOCHER *et al.*, 2014). Para apoiar esta tarefa, foi desenvolvida uma infraestrutura web para permitir o compartilhamento e evolução colaborativa desse conhecimento sobre DT. Essa infraestrutura será abordada no Capítulo 6.

Contudo, antes de apresentar a infraestrutura desenvolvida, este trabalho também buscou identificar possíveis motivos que levam a equipe de desenvolvimento a incorrer a dívida em seus projetos. Para isso, um *survey* foi executado para coletar as causas da ocorrência da DT. O planejamento e resultados do estudo serão apresentados no próximo capítulo.

5 CAUSAS QUE LEVAM À INSERÇÃO DA DÍVIDA TÉCNICA EM PROJETOS DE SOFTWARE

Neste capítulo são apresentados os resultados de um survey baseado em entrevistas conduzido com o objetivo de identificar causas que levam à inserção da dívida técnica em projetos de software. Serão apresentados o objetivo, metodologia e resultados obtidos do estudo.

5.1 INTRODUÇÃO

Os Capítulos 3 e 4 desta dissertação apresentaram a organização de um corpo de conhecimento sobre DT considerando seus tipos, indicadores e técnicas de gerenciamento. Embora esse conjunto de informações seja importante, visto que pode ser utilizado para apoiar atividades de identificação e gerenciamento de itens da dívida existente nos projetos, ele é ainda limitado no que diz a respeito à compreensão dos motivos que levam à ocorrência da DT nos projetos.

A identificação das causas que podem levar a equipe a incorrer a dívida é importante uma vez que, com este conhecimento, pode-se trabalhar na definição de atividades que apoiarão a prevenção da dívida. Nessa etapa da pesquisa, buscou-se identificar as causas que levam equipes de desenvolvimento a incorrerem em dívida técnica. Para isso, optou-se pela realização de um *survey* cujo planejamento e resultados obtidos serão apresentados neste capítulo.

O objetivo do estudo é **analisar** os diferentes tipos de dívida técnica, **com o propósito de** caracterizar, **com respeito às** causas que levam a sua inserção, **no contexto de** projetos de desenvolvimento de software **sob o ponto de vista de** profissionais da área. Alinhado a este objetivo, foram definidas as seguintes questões de pesquisa:

- **Q1. Quais causas levam à ocorrência de dívida técnica?**
  Essa questão tem como objetivo identificar possíveis causas que levam a equipe de desenvolvimento a incorrer em dívida técnica.
- **Q2. As causas ocorrem de forma encadeada ou isolada?**
  O objetivo dessa questão é investigar se os motivos que levam a equipe de desenvolvimento a incorrer em dívida ocorrem em cadeia. Sua resposta poderá ser um

indicador sobre o quanto as medidas de prevenção devem ser focadas ou mais abrangentes.

- **Q3. A dívida técnica pode ser evitada?**
  O objetivo dessa questão é saber se, na opinião dos entrevistados, é possível evitar a DT. A resposta dessa questão poderá fundamentar esforços de equipes de desenvolvimento tanto no desenvolvimento de atividades com objetivo de prevenir a dívida (caso seja possível evitá-la) quanto em atividades de monitoramento/gerenciamento (caso não seja possível prevenir sua ocorrência).
- **Q4. Em termos de esforço, é melhor evitar a dívida ou incorrê-la para pagar depois?**
  Essa questão tem como objetivo investigar se, na opinião dos entrevistados, prevenir teria sido mais caro do que incorrer a dívida. Sua resposta poderá fornecer indícios sobre qual caminho a equipe de desenvolvimento deve seguir: trabalhar no desenvolvimento de atividades de prevenção ou incorrer a dívida e pagá-la posteriormente.

As entrevistas foram realizadas com 10 profissionais de diferentes níveis de experiência. Elas permitiram identificar 61 causas que levam a equipe de desenvolvimento a incorrer a dívida. Além disso, para a maioria dos tipos de dívida, essas causas ocorrem de forma encadeada. Foi indicado também que a dívida pode ser evitada, sendo melhor trabalhar em atividades que apoiam a prevenção do que incorrê-la para pagamento posterior.

Além desta introdução, este capítulo possui mais quatro seções. Na seção 5.2 será apresentado o procedimento do estudo, bem como seu planejamento e execução. Em seguida, na seção 5.3, os resultados serão reportados. A seção 5.4 apresenta uma discussão sobre os resultados obtidos e suas possíveis implicações, e limitações do estudo. Por fim, a seção 5.5 apresenta as considerações finais deste capítulo.

5.2 METODOLOGIA

O *survey* foi planejado e executado considerando profissionais da área. A escolha dos participantes foi realizada por conveniência e considerou parceiros industriais do Time de Pesquisa em Dívida Técnica[7]: Centro de Projetos Fraunhofer para Sistemas e Engenharia de

[7] www.tdresearchteam.com

Software e SENAI Cimatec. Ao total, 10 engenheiros de software foram convidados a participar do estudo via *e-mail* e foram entrevistados presencialmente.

Os tipos de dívida identificados no Capítulo 3 foram divididos em dois grupos: A e B. O grupo A continha os tipos considerados mais comuns (os seis tipos mais citados nos artigos identificados no mapeamento). Já o grupo B foi formado pelos tipos de dívida mais específicos. A divisão foi realizada da seguinte forma:

- **Grupo A:** (1) Dívida de Arquitetura, (2) Dívida de Código, (3) Dívida de Defeito, (4) Dívida de Projeto, (5) Dívida de Documentação, (6) Dívida de Teste;

- **Grupo B:** (7) Dívida de Construção, (8) Dívida de Serviço, (9) Dívida de Infraestrutura, (10) Dívida de Pessoas, (11) Dívida de Processos, (12) Dívida de Automação de Teste, (13) Dívida de Requisitos, Dívida de Usabilidade (14) e Dívida de Versionamento (15).

Antes de realizar a entrevista, um formulário de caracterização e consentimento foi preenchido pelos participantes do estudo. O resultado desta caracterização foi utilizado para apoiar a escolha dos tipos de dívida sobre os quais cada participante deveria responder. Essa escolha foi realizada através de um sorteio controlado que seguiu o processo:

1. Cada tipo de dívida recebeu um número identificador de 1 a 15;
2. Para cada participante do estudo:
    a. Utilizou-se um gerador de número randômicos para sortear 2 valores para o grupo A (por serem os tipos de dívida mais comumente pesquisados e conhecidos) e 1 valor para o grupo B (por serem tipos de dívida que ainda que necessitam de mais evidências para que se possa entender melhor seu escopo);
    b. Caso um dos números sorteados se referisse a um tipo de dívida cuja a caracterização do participante indicasse que ele não possuia conhecimento na área, um novo sorteio era realizado para subsitituir aquele tipo. Por exemplo, se o participante nunca trabalhou com programação, dívida de código seria descartada e, dessa forma, um outro tipo deveria ser sorteado.
3. Ao final do processo, cada participante possuía três tipos de dívida associados a ele.

Ao optar-se por utilizar o sorteio controlado, buscou-se evitar que: (1) entrevistados respondessem a questionamentos sobre tipos de dívida cuja sua experiência prática dificultasse seu entendimento sobre os questionamentos e, como consequência, prejudicasse

suas respostas; (2) tipos de dívida ainda pouco explorados pela comunidade de pesquisa (Grupo B) tivessem uma participação maior no estudo, e; (3) a escolha dos tipos fosse totalmente direcionada pelos pesquisadores responsáveis pela execução do estudo.

A Tabela 9 apresenta o resultado da caracterização dos participantes. Na coluna "áreas de conhecimento", listou-se as áreas cujo participante indicou ter um alto nível de conhecimento. Observa-se que o nível de experiência dos candidatos pode ser considerado de médio para alto. Além disso, eles já possuem conhecimento prático em diferentes áreas da engenharia de software.

Tabela 9 - Caracterização dos participantes

Participante	**Nível**	**Função na Empresa**	**Tempo de Experiência**	**Áreas de Conhecimento**
A	Graduado	Analista de Requisitos	15 anos	- arquitetura de software - compilação ou integração contínua de software - *design* (projeto) do software - acompanhamento e correção de defeitos do software - documentação de software - processos de software - infraestrutura do software - gestão da equipe de desenvolvimento do software - requisitos de software - teste de software - automação de teste de software
B	Graduado	Programador	7 anos	- arquitetura de software - compilação ou integração contínua de software - codificação de software - acompanhamento e correção de defeitos do software - documentação de software - software orientado a serviço - teste de software - automação de teste de software
C	Mestre	Engenheiro de Software	5 anos	- arquitetura de software - codificação de software - *design* (projeto) do software - acompanhamento e correção de defeitos do software - documentação de software - processos de software - teste de software
D	Mestre	Gerente de Projetos	7 anos	- arquitetura de software - codificação de software - *design* (projeto) do software - acompanhamento e correção de defeitos do software - documentação de software
E	Mestre	Analista de Requisitos	36 anos	- codificação de software - *design* (projeto) do software - acompanhamento e correção de defeitos

Participante	Nível	Função na Empresa	Tempo de Experiência	Áreas de Conhecimento
F	Mestrando	Programador	4 anos	do software - documentação de software - gestão da equipe de desenvolvimento do software - requisitos de software - teste de software - codificação de software - *design* (projeto) do software - acompanhamento e correção de defeitos do software - documentação de software - processos de software - requisitos de software - teste de software
G	Mestre	Analista de Requisitos	39 anos	- arquitetura de software - compilação ou integração contínua de software - codificação de software - acompanhamento e correção de defeitos do software - documentação de software - infraestrutura do software - processos de software - gestão da equipe de desenvolvimento do software - requisitos de software - teste de software
H	Mestre	Programador	5 anos	- compilação ou integração contínua de software - codificação de software - acompanhamento e correção de defeitos do software - documentação de software - processos de software - requisitos de software - teste de software - automação de teste de software
I	Mestre	Pesquisador	7 anos	- arquitetura de software - codificação de software - design (projeto) do software - acompanhamento e correção de defeitos do software - documentação de software - processos de software - gestão da equipe de desenvolvimento do software - requisitos de software
j	Mestre	Programador	12 anos	- arquitetura de software - codificação de software - *design* (projeto) do software - acompanhamento e correção de defeitos do software - documentação de software - processos de software - infraestrutura do software - gestão da equipe de desenvolvimento do software

Participante	Nível	Função na Empresa	Tempo de Experiência	Áreas de Conhecimento
				- requisitos de software - software orientado a serviço - teste de software

As entrevistas foram realizadas individualmente, sendo que cada entrevistado respondeu às perguntas sobre três tipos de dívida (2 tipos escolhidos do Grupo A e 1 tipo escolhido do Grupo B). Como cada entrevistado respondeu a cada pergunta três vezes (cada uma para um tipo de dívida) e o estudo contou com a participação de 10 engenheiros de software, ao final, 30 respostas foram obtidas para cada questão realizada.

No início de cada entrevista foi realizada uma breve apresentação do tema dívida técnica, que durou cerca de 10 minutos, apresentando conceitos básicos e, caso o participante tivesse alguma dúvida, esta seria sanada. Em seguida, para cada tipo de dívida definido para o entrevistado, os passos apresentados a seguir foram considerados:

1. A definição do tipo de dívida selecionado foi apresentada;
2. Foi pedido ao entrevistado que ele fornecesse um exemplo, com base em sua experiência, do tipo de dívida que estava sendo discutido, para confirmar o entendimento do entrevistado sobre o tipo considerado;
   2.1. Se o entrevistado desse um exemplo que não fosse do tipo de dívida em discussão, seria pedido um segundo exemplo explicando novamente a definição daquele tipo;
      2.1.1. Se o entrevistado não conseguisse pensar em um exemplo sobre o tipo de dívida escolhido, este tipo deveria ser substituído por um outro do mesmo grupo (A ou B).
   2.2. Se o entrevistado apresentasse um exemplo que representasse o tipo de dívida discutido, a entrevista prosseguia para o próximo passo;
3. Foi perguntado ao entrevistado o que levou a equipe de desenvolvimento a inserir o item de dívida descrito em seu exemplo;
   3.1. Em seguida, foi perguntado se algum outro motivo teria levado ou contribuído para o motivo citado. Esta pergunta foi realizada repetidas vezes de forma que detalhes mais específicos sobre o que poderia ter levado a equipe a incorrer na dívida fosse identificado (o objetivo aqui foi fugir do óbvio e tentar entender se as causas para a inserção da dívida ocorrem em cadeia);
4. Foi perguntado ao entrevistado o que, na opinião dele, teria evitado a DT;

4.1. Em seguida, foi perguntado se valeria a pena ter investido na prevenção da dívida considerando o que ele indicou na pergunta anterior ou se fazer isso seria mais caro do que incorrer na própria dívida.

Quanto ao sorteio controlado dos tipos para cada entrevistado, dívida de documentação foi selecionada cinco vezes. Já dívida de arquitetura, projeto, teste, pessoas e processos foram consideradas quatro vezes, cada. Além disso, dívida de código e serviço foram selecionadas três e duas vezes, respectivamente. Os demais tipos de dívida não foram contemplados.

As entrevistas foram gravadas com autorização dos participantes. Depois de realizadas, elas foram transcritas para que a análise dos dados pudesse ser realizada. Os textos transcritos das entrevistas passaram por um processo de codificação e, então, as evidências coletadas foram avaliadas frente às questões de pesquisa. Para isso, foi utilizado um formulário de planilha eletrônica.

A codificação foi realizada por um pesquisador e, depois, avaliada por um segundo de acordo com o esquema de codificação apresentado na Tabela 10. Este esquema partiu de um conjunto inicial de códigos e evoluiu à medida que a análise dos dados foi realizada.

Os resultados do estudo foram formulados com base na síntese dos fragmentos codificados de cada questão. Assim, os resultados foram fundamentados nos dados do estudo, em vez de formulados anteriormente e validados através dos dados.

Tabela 10 - Esquema de codificação

CÓDIGO	DESCRIÇÃO
EXDT	**Exemplo DT** Moderador pergunta por uma situação que o participante tenha passado sobre um tipo específico de DT em algum projeto real na indústria de software. O participante responde à pergunta e apresenta um exemplo daquele tipo de dívida.
CDT	**Causas DT** Moderador pergunta sobre as possíveis causas que tenham levado à equipe de desenvolvimento a incorrer a DT no projeto de software reportado na EXDT. O participante responde à pergunta e explica as causas relatadas.
NCDT	**Novas Causas DT** Moderador pergunta sobre novas possíveis causas que tenham levado a equipe de desenvolvimento a incorrer a DT no projeto de software reportado na EXDT. O participante responde à pergunta e explica as causas relatadas.
FCDTI	**Forma de Ocorrência de Causas DT – Isolada** Moderador pergunta se as possíveis causas da DT, reportadas na CDT e NCDT, ocorrem de forma encadeada ou de forma isolada. O participante responde à pergunta e indica que as causas ocorrem de forma isolada.
FCDTE	**Forma de Ocorrência de Causas DT – Encadeada** Moderador pergunta se as possíveis causas da DT, reportadas na CDT e NCDT, ocorrem de forma encadeada ou de forma isolada. O participante responde à pergunta e indica que as causas ocorrem de forma encadeada.
MCMD	**Mais Causas Mais Dívida** Entrevistado indica que quanto mais causas ocorrerem, mais dívida o projeto terá.
EDT	**Evitar DT** Moderador pergunta se é possível evitar um tipo específico de DT. O participante responde à pergunta e explica como é possível evitar um tipo específico de DT.
NEDT	**DT que não pode ser evitada** Moderador pergunta se é possível evitar um tipo específico de DT. O participante responde à pergunta e explica por que não é possível evitar um tipo específico de DT.
EIDT	**Evitar Incorrer a Dívida Técnica** Entrevistado indica que é melhor evitar a ocorrência da dívida técnica.
EMEDT	**Esforço Maior para Evitar Dívida Técnica** Entrevistado indica que o esforço de evitar a dívida é maior do que o de incorrer e pagar depois.
PDT	**Pagar Dívida Técnica** Entrevistado indica que é melhor incorrer a dívida e pagá-la depois do que trabalhar no sentido de evitá-la.
CPD	**Causas com Pesos Diferentes** Entrevistado indica que causas diferentes podem ter pesos diferentes.

5.3 RESULTADOS DO ESTUDO

Nesta seção serão apresentados os resultados do estudo. Eles estão organizados segundo as quatro questões de pesquisa definidas.

5.3.1 Quais causas levam à ocorrência de dívida técnica?

As Tabelas 11 a 18 apresentam as causas identificadas neste estudo organizadas por tipo de dívida. As tabelas também indicam a quantidade de citações que cada causa obteve para cada tipo de dívida e considerando todos os tipos em conjunto. Para chegar a estes valores, dois pesquisadores analisaram cada fragmento da entrevista associado aos códigos NCDT e CDT e, para cada um deles, foi identificada a causa através da cópia direta do termo utilizado na entrevista (eventualmente, por questão de padronização na nomenclatura, pequenos ajustes foram feitos sem alterar sua semântica). Em seguida, as causas foram filtradas com a finalidade de eliminar repetições.

Ao observar a Tabela 11, percebe-se que, para dívida de arquitetura, as causas mais citadas pelos participantes foram prazo, documentação inexistente, alta rotatividade na equipe, falta de conhecimento da tecnologia utilizada e não compartilhamento do conhecimento. Já sobre a dívida de código (Tabela 12), a causa mais citada foi prazo.

Tabela 11 - Causas identificadas para Dívida de Arquitetura

Causas	Quantidade de ocorrências para o tipo de dívida	Quantidade de ocorrências Geral
Documentação inadequada	1	1
Documentação desatualizada	1	1
Pressão do gerente sobre a equipe	1	3
Falta de maturidade da equipe	1	3
Postergar resoluções de problemas	1	1
Descuido do programador	1	1
Falta de conhecimento do domínio	1	2
Falta de conhecimento da arquitetura	1	1
Falta de investimento nos funcionários	1	2
Falta de comunicação na equipe	1	1
Pressão externa	1	1
Prazo	2	13
Documentação inexistente	2	4
Alta rotatividade na equipe	2	4
Falta de conhecimento da tecnologia	2	3
Não compartilhamento do conhecimento	2	3

Tabela 12 - Causas identificadas para Dívida de Código

Causas	Quantidade de ocorrências para o tipo de dívida	Quantidade de ocorrências Geral
Falta de metodologia	1	1
Falta de padronização da infraestrutura	1	1
Falta de padronização do código	2	2
Falta de conhecimento da tecnologia	1	3
Falta de equipe qualificada	1	2
Falta de treinamento	1	1
Falta de revisão do código	1	1
Falta de conhecimento	1	3
Falta de experiência	1	3
Ajuste de modificações apenas no código	1	1
Falta de interesse do desenvolvedor	1	1
Falta de rastreabilidade	1	1
Sobrecarga da equipe	1	1
A equipe não dá importância para a documentação	1	1
A empresa negligencia o pessoal responsável pela documentação	1	1
Falta de sincronismo da equipe de documentação com a equipe de desenvolvimento e de análise	1	1
Prazo	2	13

Para a dívida de documentação, Tabela 13, as causas mais citadas foram prazo e esforço. Já para dívida de pessoas, prazo e falta de recurso para treinamento se destacaram (Tabela 14). As causas mais citadas sobre a dívida de processo (Tabela 15) foram falta de adaptação do processo ao cenário real, falta de maturidade para acompanhar o processo e falta de um processo bem definido.

Tabela 13 - Causas identificadas para Dívida de Documentação

Causas	Quantidade de ocorrências para o tipo de dívida	Quantidade de ocorrências Geral
Falta de um padrão de documentação	1	1
Alterações no desenvolvimento do projeto	1	2
Desconhecimento do cliente das suas próprias necessidades	2	3
Não priorizar atividades de documentação	1	1
Necessidade de esperar a atualização do sistema para então tornar a documentação consistente com ele	1	1
Programadores não gostam de documentar	1	1
Preguiça	1	3
Cliente não tem o compromisso de homologar os requisitos	1	1
Esforço	3	6
Prazo	3	13

Tabela 14 - Causas identificadas para Dívida de Pessoas

Causas	Quantidade de ocorrências para o tipo de dívida	Quantidade de ocorrências Geral
Falta de conhecimento	1	3
Preguiça	1	3
Acúmulo de atividades	1	1
Documentação inexistente	1	4
Planejamento inadequado	1	1
Má alocação de recursos	1	1
Falta de recurso para contratação	1	1
Alta rotatividade na equipe	1	4
Não compartilhamento do conhecimento	1	3
Falta de nivelamento técnico na equipe	1	1
Falta de investimento nos funcionários	1	2
Prazo	2	13
Falta de recurso para treinamento	2	2

Tabela 15 - Causas identificadas para Dívida de Processo

Causas	Quantidade de ocorrências para o tipo de dívida	Quantidade de ocorrências Geral
Falta de adaptação do processo ao cenário real	7	7
Prazo	1	13
A gerência não ouve a opinião das pessoas envolvidas	1	1
Pressão do gerente sobre a equipe	1	3
Falta de percepção da importância de se ter um processo	1	1
Falta de experiência	1	3
Falta de maturidade para acompanhar o processo	5	6
Falta de papéis bem definidos para acompanhar o processo e a garantia da qualidade	1	1
Falta de um processo bem definido	3	3

Para dívida de projeto, na Tabela 16, tem-se prazo, falta de conhecimento no domínio e esforço, como as causas mais citadas. Sobre a dívida de teste, Tabela 17, as causas mais citadas pelos participantes foram prazo e falta de aplicação de técnicas para planejamento de testes. Por fim, sobre a dívida de serviço (Tabela 18), destaca-se falta de planejamento do projeto.

Tabela 16 - Causas identificadas para Dívida de Projeto

Causas	Quantidade de ocorrências para o tipo de dívida	Quantidade de ocorrências Geral
Design mal projetado	1	1
Alta rotatividade na equipe	1	4
Desconhecimento do cliente das suas próprias necessidades	1	3
Mudança no projeto	1	1
Estimativa de tempo imprecisa	1	1
Mudança de requisitos	1	1
Preguiça	1	3
Pressão do gerente sobre a equipe	1	3
Falta de experiência	1	3
Requisito impreciso	1	1
Falta de conhecimento no domínio	3	3
Esforço	2	6
Prazo	3	13

Tabela 17 - Causas identificadas para Dívida de Teste

Causas	Quantidade de ocorrências para o tipo de dívida	Quantidade de ocorrências Geral
Falta de conhecimento do domínio	1	2
Falta de equipe qualificada	1	2
Execução inadequada do teste	1	1
Teste não automatizado	1	1
Falta de ferramentas adequadas	1	1
Documentação inexistente	1	4
Falta de aplicação de técnicas para planejamento de testes	3	3
Prazo	3	13

Tabela 18 - Causas identificadas para Dívida de Serviço

Causas	Quantidade de ocorrências para o tipo de dívida	Quantidade de ocorrências Geral
Alterações no desenvolvimento do projeto	1	2
Alterações no código do serviço	1	1
Falta de planejamento do projeto	2	2
Serviço inadequado em termos de desempenho	1	1
Esforço	1	6
Falta de confiabilidade no código do serviço	1	1

Ao total foram identificadas 61 causas distintas neste estudo. Destas, as mais citadas foram prazo (13 citações), falta de adaptação do processo ao cenário real (7 citações), falta de maturidade para acompanhar o processo (6 citações), esforço (6 citações), documentação inexistente (4 citações) e alta rotatividade na equipe (4 citações). É possível observar também que a maioria das causas citadas estavam associadas especificamente a um tipo de dívida. A presença de causas compartilhadas e específicas indica que ações para prevenir a dívida podem ter um efeito específico ou contribuírem para o tratamento de diferentes tipos de dívida.

Ao analisar as causas identificadas para cada tipo, também foi possível perceber que muitas delas estavam relacionadas e, desta forma, poderiam ser organizadas em categorias que refletissem o problema sobre o qual elas tratavam. Este agrupamento permite se ter uma visão mais abrangente sobre fontes de causas que levam à ocorrência da dívida. A Tabela 19 apresenta o resultado do agrupamento realizado. As categorias identificadas foram:

- **Documentação:** agrupa causas relacionadas à documentação do software em qualquer fase do seu desenvolvimento. Exemplos de causas associadas a este grupo são: falta de documentação e documentação desatualizada;
- **Falta de conhecimento:** refere-se às causas relacionadas à falta de conhecimento da equipe para desenvolvimento do projeto. Exemplos incluem a falta de conhecimento da tecnologia e do domínio;
- **Fatores externos:** reúne causas que são relacionadas a problemas externos ao desenvolvimento e à organização. Alguns exemplos são: pressões externas e problemas com cliente;
- **Metodologia:** agrupa causas relacionadas a processos e metodologias utilizadas no desenvolvimento do projeto. Exemplos de causas associadas a este grupo são: falta de processos e alta rigidez do processo;
- **Organizacional:** reúne causas relacionadas à organização, como problemas ligados à alocação de recursos, investimentos e processos da empresa;
- **Pessoas:** refere-se às causas relacionadas diretamente às pessoas envolvidas no projeto. Alguns exemplos são: falta de maturidade da equipe e falta de interesse do desenvolvedor;
- **Planejamento e gerência:** agrupa causas relacionadas ao planejamento e gerenciamento do projeto. Exemplos de causas associadas a este grupo são: prazo e esforço;
- **Projeto:** agrupa as causas que ocorrem durante o desenvolvimento do projeto, como mudanças no projeto e *design* mal elaborado.

Tabela 19 - Agrupamento das causas

Planejamento e Gerência	Pressão do gerente sobre a equipe
	Postergar resoluções de problemas
	Sobrecarga da equipe
	Acúmulo de atividades
	A gerência não ouve a opinião das pessoas envolvidas
	Prazo
	Esforço
	Estimativa de tempo imprecisa
	Planejamento inadequado
	Falta de planejamento do projeto
Falta de conhecimento	Falta de conhecimento do domínio
	Falta de conhecimento da arquitetura
	Falta de conhecimento da tecnologia
	Falta de conhecimento
	Falta de experiência
	Não compartilhamento do conhecimento
	Falta de maturidade para acompanhar o processo
	Falta de maturidade da equipe
	Falta de nivelamento técnico na equipe Descuido do programador
Pessoas	Falta de interesse do desenvolvedor
	A equipe não dá importância para a documentação
	Programadores não gostam de documentar
	Preguiça
	Falta de equipe qualificada
Metodologia	Execução inadequada do teste
	Falta de aplicação de técnicas para planejamento de testes
	Falta de comunicação na equipe Falta de revisão do código
	Falta de sincronismo da equipe de documentação com a equipe de desenvolvimento e de análise
	Necessidade de esperar a atualização do sistema para então tornar a documentação consistente com ele
	Teste não automatizado
	Falta de padronização do código
	Ajuste de modificações apenas no código
	Falta de metodologia
	Falta de rastreabilidade
	Falta de adaptação do processo ao cenário real
	Falta de um processo bem definido
	Falta de papéis bem definidos para acompanhar o processo e a garantia da qualidade
Documentação	Documentação inadequada
	Documentação inexistente
	Documentação desatualizada
	Falta de um padrão de documentação
	Não priorizar atividades de documentação

Organizacional	Alta rotatividade na equipe
	Falta de investimento nos funcionários
	Falta de padronização da infraestrutura
	Falta de ferramentas adequadas
	A empresa negligencia o pessoal responsável pela documentação
	Má alocação de recursos
	Falta de recurso para contratação
	Falta de treinamento
	Falta de recurso para treinamento
	Falta de percepção da importância de se ter um processo
Projeto	*Design* mal projetado
	Mudança no projeto (*design*)
	Mudança de requisitos
	Requisito impreciso
	Alterações no desenvolvimento do projeto
Fatores Externos	Pressão externa
	Desconhecimento do cliente das suas próprias necessidades
	Cliente não tem o compromisso de homologar os requisitos
	Alterações no código do serviço (*web service*)
	Serviço (*web service*) inadequado em termos de desempenho
	Falta de confiabilidade no código do serviço (*web service*)

5.3.2 As causas ocorrem de forma encadeada ou isolada?

Os resultados indicam que, de uma forma geral, as causas que levam a equipe de desenvolvimento a incorrer em DT ocorrem em cadeia, ou seja, uma série de fatores leva à presença da dívida no projeto. Os seguintes trechos ilustram a opinião dos entrevistados sobre esta questão: "*...eu acho que era um* ***conjunto*** *de não ter uma documentação, pessoas que documentassem bem o que deveria ser feito e avaliar o impacto de cada mudança nos módulos.";* "*...****mas também em conjunto****, tanto a falta de conhecimento e falta de documentação irão contribuir para aumentar a dívida técnica de arquitetura*"; "***Encadeada*** *também... se você não tem maturidade para o processo, você não vai ter processo ou pelo menos alguém para avaliá-lo. Se você tem maturidade para o processo, mas não tem maturidade para avaliar a qualidade do processo ou se a equipe está seguindo ele, isso pode somar com o fato de não se ter papeis específicos para esse trabalho*"; "*Eu acho que* ***é tudo integrado****... **são em conjunto, não é um só, não é uma coisa só***"; "*Acho que acontece em* ***conjunto***"; "*Eu acho que* ***é um conjunto de coisas***. ***Não posso dizer que é isolado***."; "*Acho que ocorre de forma* ***encadeada****..*"

Contudo, apesar de grande parte dos entrevistados informarem que as causas ocorrem de forma encadeada, os resultados também indicaram que elas podem afetar o projeto de forma isolada conforme reportado nos seguintes fragmentos: "Elas podem acontecer **isoladamente**..."; "*Mas **nada impede** que uma situação específica **aconteça de forma isolada**.*"; "***Pode ser isolada** ou pode ser em conjunto*."; "*Depende muito. Você pode encontrar todas **juntas ou isoladas**, não tem receita de bolo aí não*."; "*Eu acho que **isoladamente elas já contribuiriam para o surgimento e grande impacto da dívida** no projeto, mas o fato delas acontecerem em conjunto ou não depende muito do cenário que você tá aplicando*".

Além disso, foi indicado que embora as causas possam afetar o projeto de forma isolada, quando elas ocorrem em cadeia a consequência delas é potencializada: "*Eu acho que nesse caso é isoladamente, mas **juntas** elas são **potencializadas***"; "*...**quanto mais incidem esses tipos de causas**, **maior será a dívida**. Acho que pode ser **proporcional** a questão aí*".

Este conjunto de resultados sugere que, como diferentes causas podem levar à ocorrência da dívida de forma encadeada ou isolada, diferentes ações devem ser consideradas no sentido de preveni-la uma vez que causas diferentes podem levar à necessidade de tratamentos diferentes. Este tratamento deve ser efetuado mais cuidadosamente quando causas em cadeia afetarem o projeto uma vez que isso potencializa a ocorrência da dívida e o seu impacto.

Por fim, foi indicado também que as causas podem possuir pesos diferentes na contribuição para a ocorrência da dívida: "*Eu acho que **cada causa tem um peso diferente**... Eu acho que é uma **equação**, em que cada uma tem um determinado peso...*". Como as causas podem possuir diferentes pesos, tratar as causas com objetivo de evitar a dívida sem considerar seus pesos pode levar a uma estratégia ineficiente de prevenção.

5.3.3 A dívida técnica pode ser evitada?

Quando perguntados se a dívida técnica poderia ser evitada, os entrevistados indicaram em sua maioria que sim. De um total de 28 respostas, 20 indicaram que poderia ser evitada e 6 que não poderiam ser evitadas totalmente, mas amenizadas. Apenas dois participantes indicaram que a dívida não poderia ser evitada.

Para aquelas que indicaram que a dívida não poderia ser evitada totalmente, o motivo estava associado à existência de uma causa que não poderia ser tratada. Os seguintes fragmentos ilustram esta situação: "***Eu não sei se evitar 100%...*** *A gente tem um projeto de aproximadamente um ano e foi definido que na arquitetura íamos usar uma série de definições. Iríamos parar um mês para treinar a equipe (no uso da tecnologia), mas isso não foi feito por questão de tempo. A partir disso, a equipe começou a usar uma tecnologia que não conhecia. Como não tivemos esse tempo para realizar o treinamento, não teríamos como evitar a dívida*"; "***Acho que pode ser minimizada***. *Acho que pode ser mitigada... preparando a equipe utilizando boas ferramentas... Prazo não sei se tem como negociar*". Em ambos os fragmentos, foi colocado que problemas provenientes de prazo são difíceis de serem tratados embora outras causas que também impactam no projeto possam ser controladas.

Para aqueles que consideram possível evitar a dívida, observou-se que as explicações feitas também giravam em torno de uma causa que poderia ser tratada: "*Teve projeto que a gente precisava de especialistas muitos específicos em uma tecnologia e encontra-los localmente é muito difícil e trazer de fora é muito caro. Então, fazer o que? Treinar é mais fácil? Mais barato? Sim. Temos tempo pra isso? Quando temos tempo é uma solução, quando não temos, contratar é uma solução.*"; "*Eu acho que sempre é possível, melhor com certeza. É necessário mostrar que se você não fizer uma boa documentação, você terá depois um esforço de manutenção. Pelo menos de descolamento, de atendimento, que também não vai ser um esforço baixo...*"; "*É possível se você começar bonitinho, seus processos definidos, descrevendo tudo o que cada um tem que fazer, como fazer, tendo instrumentos e ferramentas que ajudem não só melhorar o desempenho das pessoas, mas que as pessoas façam as coisas de forma mais correta.*". Para o primeiro fragmento apresentado, a causa tratada seria a questão de se ter mão de obra qualificada para o projeto. Já no segundo, é considerada a questão da documentação. No último fragmento de exemplo, foi indicado que seria possível tratar a dívida de processo através de um processo bem definido na organização.

Ainda neste contexto, uma observação interessante reportada foi que embora seja possível evitar a dívida, não temos como garantir que ela não exista no projeto: "*... acho* ***a gente pode evitar****, mas garantir que ela exista, não podemos! Você pode usar práticas, melhorar os processos, ter as melhores pessoas experientes com requisitos... para evitar, mas garantir que ela não exista é complicado.".* Isto indica que mesmo em ambientes controlados

onde existem iniciativas para evitar a ocorrência da dívida, ainda assim é importante possuir mecanismos para detecção da dívida existente no projeto.

Por fim, duas respostas indicaram situações em que a dívida não poderia ser evitada. Na primeira delas, "*Não vejo como é possível a questão do evitar, pois é um sistema terceiro. Só tem ele, não tem como...*", o entrevistado reportou esta questão ao falar da necessidade de integração com um módulo específico de um sistema. Como só havia uma opção de integração, não havia muito o que fazer. Já na segunda situação, foi colocado que não é possível evitar a dívida quando a equipe é pega de surpresa: "*Eu acho difícil ser evitado, sabe? Eu não sei como, porque é uma coisa que você recebe sem você ter nenhum conhecimento prévio disso. Não existia isso pra gente."*.

Os resultados das entrevistas sugerem que o fator que leva uma dívida a poder ser ou não evitada é a causa que levou à sua existência. Existem causas de DT que, por serem imprevisíveis ou não poderem ser facilmente controladas, levam a situações em que a DT não pode ser evitada (ou totalmente evitada).

5.3.4 Em termos de esforço, é melhor evitar a dívida ou incorrê-la para pagar depois?

Os resultados indicaram que é melhor e mais barato a equipe de desenvolvimento investir na prevenção da dívida ao invés de incorrê-la e pagá-la depois: "*Acho* ***melhor prevenir***."; "*Acho* ***melhor prevenir*** *esse tipo de dívida, porque geralmente o esforço de corrigir uma coisa que a gente fez (de forma inadequada) é muito alto (depois). Isso pode levar a situações em que o que poderia ser feito em um dia, precisará de uma semana para ser resolvido porque será necessário refazer outras coisas que dependem daquilo. É necessário repensar muita coisa."*; "*Eu* ***evitaria*** *essa dívida. Fica* ***mais barato*** *evitar do que pagar.*"; "*Teria sido possível prevenir e seria menos custoso prevenir do que corrigir, porque na correção você necessariamente vai ter que alterar uma boa parte do que já foi feito e você acaba deixando de fazer outras funcionalidades.*"; "*Evitar, claro que evitar. Porque retrabalho é uma coisa muito chata, ela tira a motivação das pessoas. Isso deixava a gente totalmente desmotivado.*". Considerando estes fragmentos, é possível observar alguns fatores que pesam positivamente na decisão de prevenir a dívida: esforço de correção tardia dos problemas inseridos no projeto, o nível de retrabalhado necessário para eliminar a dívida e a

motivação da equipe. Os dois últimos fatores impactam diretamente na produtividade da equipe de desenvolvimento.

Um outro fator que é considerado determinante no sentido de se trabalhar para prevenir a dívida é o quão crítico é o artefato para as atividades de desenvolvimento: "*Eu acho que é melhor evitar. Porque eu estou falando muito de documentação de requisitos, que é o fio condutor de todo processo. Então, com certeza, se você propaga o esforço da dívida de requisitos ao longo do processo, esse esforço vai se multiplicando várias vezes. Então eu diria que é uma dívida que, pela experiência e conhecimento que tenho, com certeza vale a pena evitar do que pagá-la depois.*"

Por outro lado, também existiram aqueles que indicaram que podem existir situações em que é melhor incorrer a dívida e pagá-la depois: "*Pode ser evitado, porém o* ***custo (de evitar) é maior****. Pode ser evitado, mas às vezes não vale a pena... em fábricas de software, como tudo é imediato, ou o sistema está muito bem parametrizado ou não vai ser possível evitar.*"; "*É* ***melhor incorrer*** *e pagar depois. Independente do tipo de dívida, para mim aquela que vale a pena incorrer e pagar depois são as que demandam tempo, seja para fazer ou pensar...".* Considerando estes fragmentos, é possível observar que esforço e tempo podem ser fatores que impossibilitam a prevenção da dívida. Quando o esforço de evitar a dívida é maior do que o de inserir e pagar depois, é natural que a equipe siga pelo caminho de ter a dívida no projeto. Já a restrição de tempo também contribui para que nem sempre a melhor solução (que está livre de dívida) seja a utilizada no projeto. Contudo, a decisão de incorrer a dívida deve ser consciente de forma que ela possa ser monitorada. Do contrário, a dívida pode sair do controle: "*...mas essa decisão (de incorrer a dívida para pagá-la depois) só irá valer a pena se você continuar monitorando essa dívida para que ela não chegue a um ponto em que* ***se torne irreversível****... É necessário manter o monitoramento e, junto com o gerente de projeto, analisar quando ela será tratada.*".

5.4 DISCUSSÃO

5.4.1 Implicações para profissionais e pesquisadores

Este *survey* buscou identificar: (1) possíveis causas que levam a equipe de desenvolvimento a incorrerem em dívida técnica; (2) se os motivos que levam a equipe de desenvolvimento a incorrerem em dívida ocorrem em cadeia; (3) se, na opinião dos

entrevistados, prevenir a dívida teria sido mais caro do que incorrer a dívida. Seus resultados sugerem algumas implicações importantes para profissionais, particularmente para aqueles que buscam na literatura informações sobre motivos que podem levar à ocorrência da dívida e como lidar com eles:

- É válido investir em estratégias de prevenção. Dívidas que trazem grande impacto ao projeto tendem a trazer problemas que custam mais caro para serem ajustados do que o esforço necessário para preveni-las;
- A quantidade diversificada de causas leva à necessidade de se definir critérios para escolha daquelas mais relevantes para a equipe. Caso contrário, as estratégias de prevenção podem se tornar ineficientes e não contribuírem para evitar os tipos de dívida mais relevantes para a equipe;
- Algumas causas podem levar à ocorrência de diferentes tipos de dívida. Trabalhar no tratamento delas pode ser considerado um caminho natural quando o objetivo for aumentar a cobertura dos tipos de dívida prevenidos, enquanto se busca minimizar os esforços de prevenção;
- Também é importante que causas em cadeia sejam tratadas adequadamente uma vez que sua ocorrência potencializa a ocorrência e o impacto da dívida no projeto;
- É necessário definir quais causas tendem a impactar mais diretamente no projeto. Esta definição é essencial ao se definir uma estratégia de prevenção, uma vez que monitorar causas sem considerar seus pesos pode levar a uma estratégia ineficiente de prevenção;
- Não há como garantir que dívidas não existam no projeto mesmo investindo-se em atividades de prevenção. Desta forma, estas atividades devem ser combinadas com estratégias de identificação e monitoramento da dívida;

Para pesquisadores, os resultados deste estudo sugerem diferentes possibilidades de investigação:

- Cadeias de causas merecem ser investigadas em maiores detalhes, uma vez que podem ser um indicador sobre quanto as medidas de prevenção devem ser focadas ou mais abrangentes;
- Dada a diversidade das causas, é interessante desenvolver estratégias que direcionem melhor a seleção de causas que, se tratadas, trarão impactos mais positivos para o projeto;
- Diferentes perfis de profissionais (gerentes, analistas, desenvolvedores, dentre outros) podem ter uma percepção diferente sobre dívidas que podem ou não ser prevenidas. Isto

ocorre porque cada um deles possui uma visão diferente sobre as restrições que impactam a realização de uma atividade na qual um item da dívida pode ser inserido. É importante, em pesquisas futuras, levar em consideração estes diferentes perfis;

- É interessante que sejam realizados estudos que permitam avaliar o custo benefício de estratégias de prevenção. Embora os entrevistados tenham indicado que no geral elas são válidas, estudos precisam ser realizados de forma que esta afirmação seja fundamentada em evidências (seja através de análises quantitativas ou estudos envolvendo um número maior de entrevistados e análises qualitativas mais detalhadas).

5.4.2 Limitações do estudo

O pequeno número de entrevistados (dez) é a principal limitação desse estudo. Essa limitação impossibilitou que análises mais específicas fossem realizadas, por exemplo: ainda não é possível avaliar se diferentes perfis de profissionais (ex: gerente de projetos e desenvolvedores) que possuem visões distintas sobre as restrições do projeto, também possuem uma percepção diferente sobre dívidas que podem ser ou não evitadas e suas causas. Por exemplo, é possível que desenvolvedores julguem que certos itens de dívida não possam ser evitados por acreditar que restrições de prazo inviabilizariam ações de mitigação. Contudo, o gerente de projeto, por possuir informações mais precisas sobre prazos, poderia indicar que o cronograma poderia ser ajustado de forma a prevenir estes mesmos itens.

Alguns tipos de dívida não foram sorteados para serem considerados nas entrevistas e outros foram selecionados apenas uma vez. Este cenário limita a análise específica por tipo de dívida e demanda que o estudo seja replicado de forma que mais evidências sejam coletadas.

Por outro lado, mesmo com estas limitações, ainda assim foi obtido um bom número de respostas visto que cada participante foi entrevistado três vezes, totalizando trinta transcrições para análise. Embora os resultados não possam ser generalizados e não representem todos os tipos de dívida conhecidos, eles fornecem bons indícios a respeito das conclusões alcançadas.

5.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou um estudo sobre causas que levam a equipe de desenvolvimento a incorrerem em DT. Para isso, foi realizado o planejamento e execução de

um *survey* cujos resultados permitiram ter uma melhor investigação sobre as causas da dívida, se elas ocorrem em cadeia ou podem afetar isoladamente o projeto, além de uma análise sobre a possibilidade de prevení-la.

O próximo capítulo irá apresentar TD Wiki, ferramenta desenvolvida para permitir o compartilhamento e evolução colaborativa do conhecimento organizado nesta dissertação sobre DT. Através de TD Wiki busca-se facilitar o acesso ao corpo de conhecimento apresentado nos Capítulos 3, 4 e 5 deste trabalho, tanto para pesquisadores quanto para profissionais da área.

6 TD WIKI: UMA INFRAESTRUTURA COMPUTACIONAL PARA APOIAR O COMPARTILHAMENTO E A EVOLUÇÃO COLABORATIVA DO CONHECIMENTO SOBRE DÍVIDA TÉCNICA

Neste capítulo é apresentado TD Wiki, uma infraestrutura computacional que apoia o compartilhamento e a evolução colaborativa do conhecimento sobre DT. Serão descritos seus requisitos, arquitetura e funcionalidades implementadas.

6.1 INTRODUÇÃO

Mais do que desenvolver novos sistemas de informação, mantê-lo útil e tratar sua evolução de forma que acompanhe as modificações organizacionais é uma tarefa complexa. As dificuldades envolvidas nesta tarefa são frequentemente associadas ao fato de que os sistemas de informação têm sido desenvolvidos sem uma grande preocupação com sua manutenibilidade. A manutenibilidade é uma característica de qualidade do software que indica a facilidade com que o software pode ser entendido, corrigido, adaptado e/ou melhorado (PFLEEGER e ATLEE, 2009). Vários fatores podem influenciar positivamente ou negativamente na presença desta característica no projeto. Alguns exemplos são: a utilização de bons princípios da orientação a objetos, padrões de projeto, comentários no código e documentação do projeto atualizada. O monitoramento da DT, mantida dentro de níveis que possam ser administrados, é uma maneira de minimizar os efeitos da baixa capacidade de manutenção em projetos de software.

Mas mesmo antes que a equipe de desenvolvimento possa trabalhar sobre o monitoramento da dívida, ela precisa compreender que tipos de dívida podem ser incorridos, como ela pode ser identificada e o que pode levar a equipe de desenvolvimento a incorrê-la no projeto. O Capítulo 4 desta dissertação apresentou uma organização dos diferentes tipos de dívida considerando sua natureza como critérios de classificação. Esta organização resultou em uma taxonomia de termos sobre DT. Em sua primeira versão, a taxonomia mapeou os tipos de dívida e os seus indicadores. Esta taxonomia foi evoluída considerando os resultados discutidos no Capítulo 5, sendo adicionadas algumas das causas que levam a equipe de desenvolvimento a incorrer em cada tipo de dívida.

Apesar da taxonomia ser um mecanismo poderoso para representar o conhecimento, é difícil para pessoas que não são especialistas em mecanismos de representação do conhecimento compreendê-la e manipulá-la. Esta preocupação é ainda mais crítica quando o conhecimento precisa ser compartilhado com um público grande como o representado por equipes de desenvolvimento de software. Assim, o compartilhamento e a evolução desse conhecimento precisam ser estimulados de forma que tanto pesquisadores quanto profissionais tenham acesso a ele e possam usá-lo em suas atividades diárias.

Neste contexto, esse Capítulo apresenta TD Wiki, uma infraestrutura computacional de apoio ao compartilhamento e evolução colaborativa do conhecimento sobre DT. Existem duas expectativas principais para esta infraestrutura: (1) criar um canal para que o assunto DT seja efetivamente tratado na indústria de software através do compartilhamento das informações em um formato adequado para uso por profissionais; (2) permitir a evolução do conhecimento de forma colaborativa. Ao fornecer as informações necessárias para que as equipes de desenvolvimento comecem a monitorar a dívida em seus projetos, TD Wiki também pode contribuir positivamente para melhoria da qualidade dos sistemas de informação.

Além desta introdução, este Capítulo possui mais quatro seções. Na próxima seção serão apresentadas algumas definições utilizadas no desenvolvimento de TD Wiki. Em seguida, será discutido o conhecimento organizado sobre DT. Com base nesse conhecimento, TD Wiki será apresentado considerando seus requisitos, arquitetura e a infraestrutura computacional implementada. Por fim, tem-se as considerações finais deste capítulo.

6.2 REPRESENTAÇÃO DO CONHECIMENTO

Sistemas de representação do conhecimento são essenciais em meio à crescente produção de informação (FOGL, 1979). Isso inclui informações que precisam ser compartilhadas durante o desenvolvimento de grandes sistemas. Existem diferentes ferramentas que podem ser utilizadas para apoiar o desenvolvimento deste tipo de sistemas, incluindo taxonomias e ontologias. Se por um lado, a ontologia é uma representação de vocabulário que é muitas vezes especializada em algum domínio ou assunto (CHANDRASEKARAN *et al.*, 1999), por outro, as taxonomias permitem organizar a informação e/ou conhecimento em relações hierárquicas entre os termos.

Apesar da sua importância, para ser útil, a taxonomia precisa ser representada em um formato mais intuitivo. Uma maneira de fazer isso é usar técnicas de visualização de conhecimento. De acordo com TERGAN *et al.* (2006), visualização de conhecimento é um campo de estudo que investiga o poder de formatos visuais para representar o conhecimento com o objetivo de apoiar o processo cognitivo de gerar, representar, estruturar, recuperar e compartilhar conhecimento. É composto por um conjunto de técnicas e teorias que ajudam a melhorar a maneira como o conhecimento é compartilhado entre duas ou mais pessoas através de características visuais. Visualização do conhecimento usa técnicas de visualização de informação como uma ferramenta para implementar seus conceitos (BURKHARD, 2004).

A definição da taxonomia é uma tarefa difícil, principalmente quando se espera que tenha relevância e valor a um público amplo (CHEN *et al.*, 2008). A fim de obter uma taxonomia mais bem definida, a colaboração entre diferentes especialistas precisa ser estimulada, permitindo que cada um deles participe ativamente no seu desenvolvimento (JIE *et al.*, 2006) (ROSPOCHER *et al.*, 2014). O uso de sistemas colaborativos pode apoiar o alcance deste objetivo. A ideia central de sistemas colaborativos é apoiar um grupo de pessoas que buscam objetivos similares a alcançar esses objetivos, de modo que essas pessoas ajudem uns aos outros, direta ou indiretamente, através do sistema.

Segundo Coleman (1997), sistemas colaborativos podem ser classificados nos seguintes tipos: de gerenciamento de conteúdo, gestão de conhecimento, ferramentas de colaboração em tempo real, ferramentas da equipe virtual, CRM colaborativo, portais e comunidades online. De acordo com esta taxonomia, a infraestrutura proposta neste trabalho é classificada como um Sistema de Gestão do Conhecimento Colaborativo, porque o seu principal objetivo é recolher (colaboração), armazenar, analisar e fornecer conhecimento sobre DT de uma forma interativa e visual.

6.3 CONHECIMENTO SOBRE DT REPRESENTADO

A taxonomia definida em (ALVES *et al.*, 2014) foi evoluída através do acréscimo de causas para cada tipo de dívida e foi representada através de uma árvore. A Tabela 20 representa a versão atualizada da taxonomia considerando os tipos de DT, indicadores e causas.

Tabela 20 - Informações definidas na taxonomia de termos sobre DT considerando seus tipos, indicadores e causas

Causas da DT	Tipos de DT	Indicadores da DT
- Documentação inadequada; - Documentação desatualizada; - Pressão do gerente sobre a equipe; - Falta de maturidade da equipe; - Postergar resoluções de problemas; - Descuido do programador; - Falta de conhecimento do domínio; - Falta de conhecimento da arquitetura; - Falta de investimento nos funcionários; - Falta de comunicação na equipe; - Pressão externa; - Prazo; - Documentação inexistente; - Alta rotatividade na equipe; - Falta de conhecimento da tecnologia; - Não compartilhamento do conhecimento.	Dívida de Arquitetura	- Violação de modularidade - Problemas na arquitetura do software - *Betweenness Centrality* - *Augmented Constraint Network (CAN)* - *Pairwise-Dependency Relation (PWDR)* - *Index of Package Changing Impact (IPCI)* - *Index of Package Goal Focus (IPGF)* - Dependências estruturais
- Falta de Experiência de desenvolvedores; - Baixa qualidade de código ou aplicação para liberar rapidamente o projeto.	Dívida de Construção	- Dependências estruturais - Problemas de construção
- Falta de metodologia; - Falta de padronização da infraestrutura; - Falta de padronização do código; - Falta de conhecimento da tecnologia; - Equipe desqualificada; - Falta de treinamento; - Falta de revisão do código; - Falta de conhecimento; - Falta de experiência; - Ajuste de modificações apenas no código; - Falta de interesse do desenvolvedor; - Falta de rastreabilidade; - Sobrecarga da equipe; - A equipe não dá importância para a documentação; - A empresa negligencia o pessoal responsável pela documentação; - Falta de sincronismo da equipe de documentação com a equipe de desenvolvimento e de análise; - Prazo.	Dívida de Código	- Código sem padrões - Algoritmo lento - *Multithread Correctness* - *Code Metrics (not specified)* - *Automatic Static Analysis (ASA) Issues* - *Code Smells*
- Acumulação de defeitos ao longo do tempo; - Defeitos e solicitações de mudança diferidos em versões posteriores; - Baixa qualidade de código ou aplicação para liberar rapidamente o projeto.	Dívida de Defeito	- Defeitos conhecidos não corrigidos
- Design mal projetado;	Dívida de Projeto	- *Code Metrics (not specified)*

Causas da DT	Tipos de DT	Indicadores da DT
- Alta rotatividade na equipe; - Desconhecimento do cliente das suas próprias necessidades; - Mudança no projeto; - Estimativa de tempo imprecisa; - Mudança de requisitos; - Preguiça; - Pressão do gerente sobre a equipe; - Falta de experiência; - Requisito impreciso; - Falta de conhecimento no domínio; - Esforço; - Prazo.		- *Automatic Static Analysis (ASA) Issues* - *Code Smells* - *Grime* - Problemas de projeto de software - *Low External / Internal Quality* - *Afferent / Efferent Couplings (AC / EC)* - *Depth of Inheritance Tree (DIT)* - *Referential Integrity Constraints (RICs)*
- Falta de um padrão de documentação; - Alterações no desenvolvimento do projeto; - Desconhecimento do cliente das suas próprias necessidades; - Não priorizar atividades de documentação; - Necessidade de esperar a atualização do sistema para então tornar a documentação consistente com ele; - Programadores não gostam de documentar; - Preguiça; - Cliente não tem o compromisso de homologar os requisitos; - Esforço; - Prazo.	Dívida de Documentação	- Comentários insuficientes no código - Falta de documentação - Comentários (*hack*, *fixme*, *is problematic*, ...) - Problemas na documentação
- Falta de experiência de desenvolvedores.	Dívida de Infraestrutura	-
- Falta de conhecimento; - Preguiça; - Acúmulo de atividades; - Documentação inexistente; - Planejamento inadequado; - Má alocação de recursos; - Falta de recurso para contratação; - Alta rotatividade na equipe; - Problemas financeiros; - Não compartilhamento do conhecimento; - Falta de nivelamento técnico na equipe; - Falta de investimento nos funcionários; - Prazo; - Falta de recurso para treinamento.	Dívida de Pessoas	-
- Falta de adaptação do processo ao cenário real; - Prazo; - A gerência não ouve a opinião das pessoas envolvidas;	Dívida de Processo	-

Causas da DT	Tipos de DT	Indicadores da DT
- Pressão do gerente sobre a equipe; - Falta de percepção da importância de se ter um processo; - Falta de experiência; - Falta de maturidade para acompanhar o processo; - Falta de papéis bem definidos para acompanhar o processo e a garantia da qualidade; - Falta de um processo bem definido.		
- Dependência de requisitos - Falta de experiência de desenvolvedores.	Dívida de Requisitos	- Lista de pendências em requisitos
- Alterações no desenvolvimento do projeto; - Alterações no código do serviço; - Falta de planejamento; - Serviço inadequado em termos de desempenho; - Esforço; - Falta de confiabilidade no código do serviço.	Dívida de Serviço	- Serviços web que não atendem as restrições de qualidade do projeto
- Prazo apertado; - Cobertura baixa/inadequada; - Falta de experiência de desenvolvedores.	Dívida de Automação de Teste	- Falta de teste automatizado
- Falta de conhecimento do domínio; - Falta de equipe qualificada; - Execução inadequada do teste; - Teste não automatizado; - Falta de ferramentas adequadas; - Documentação inexistente; - Falta de aplicação de técnicas para planejamento de testes; - Prazo.	Dívida de Teste	- Defeitos conhecidos não corrigidos - Testes incompletos - Correção de defeitos adiada - Cobertura de código insuficiente - Falta de documentação de casos de teste - Falta de planejamento de casos de teste
-	Dívida de Usabilidade	
-	Dívida de Versionamento	- Uso desnecessário de *forks*

6.4 TD WIKI: COMPARTILHAMENTO E EVOLUÇÃO COLABORATIVA DO CONHECIMENTO SOBRE DÍVIDA TÉCNICA

TD Wiki é uma infraestrutura computacional que faz uso de técnicas de visualização de conhecimento para apoiar o compartilhamento e a evolução colaborativa do conhecimento sobre DT. Tipos de dívida técnica, indicadores, causas e estudos de avaliação foram

organizados para compor a base de conhecimento. Esse conhecimento poderá ser evoluído da seguinte forma:

- **Tipo de DT:** inclusão de novos tipos e identificação dos mais utilizados por pesquisadores e profissionais;
- **Indicador da DT:** inclusão de novos indicadores e identificação daqueles considerados mais apropriados para identificar cada tipo de dívida;
- **Causa da DT:** inclusão de novas causas e identificação daquelas que mais contribuem para que um determinado tipo de dívida seja incorrida no projeto;
- **Estudos de Avaliação:** inclusão de novos estudos que foram realizados para avaliar um determinado tipo de dívida.

O compartilhamento destas informações permite que pesquisadores e profissionais: (1) conheçam os tipos de dívida, bem como os seus indicadores e causas; (2) tenham uma ideia sobre o nível de conhecimento atual sobre um determinado tipo de dívida com base nos estudos experimentais que o avaliaram.

6.4.1 Requisitos

Os atores que irão manipular a infraestrutura são:

- **Usuário não registrado:** refere-se a qualquer usuário que visite a página de TD Wiki na Internet;
- **Usuário registrado: refere-se a pesquisadores/profissionais que estão registrados na infraestrutura;**
- **Moderador:** pesquisadores/profissionais responsáveis por avaliar a informação enviada a TD Wiki antes de serem incorporadas à base de conhecimento, e;
- **Administrador:** responsável pela definição dos moderadores.

Este conjunto de atores forma uma hierarquia (administrador -> moderador -> usuário registrado -> usuário não registrado) onde os níveis hierárquicos mais elevados têm acesso às funcionalidades dos níveis mais baixos. A Tabela 21 apresenta os requisitos definidos para TD Wiki e os atores que têm acesso a cada um deles.

Tabela 21 - Lista de requisitos do TD Wiki

Ator	Requisito
Usuário não Registrado	**(REQ01)** Visualizar a taxonomia de dívida técnica de forma interativa;
	(REQ02) Visualizar um resumo contendo as características básicas (breve descrição e alguns de seus indicadores) sobre cada tipo de dívida;
	(REQ03) Visualizar em detalhes as informações sobre cada tipo de dívida técnica incluindo definição, indicadores, causas, estudos de avaliação e referências;
Usuário Registrado	**(REQ04)** Criar um perfil pessoal contendo informações sobre seus trabalhos e suas pesquisas;
	(REQ05) Adicionar informações relevantes à infraestrutura como novos tipos de dívida, novos indicadores, novas causas, novas referências;
	(REQ06) Indicar, com base em experiências pessoais, quais indicadores são mais comuns para a localização de determinado tipo de dívida em projetos de software;
	(REQ07) Indicar, com base em experiências pessoais, quais são as causas que mais contribuem para que um tipo de dívida seja incorrido no projeto;
Moderador	**(REQ08)** Avaliar conhecimento/informação inserido (inserção de novos tipos, indicadores, causas e referências);
Administrador	**(REQ09)** Definir usuário moderador;

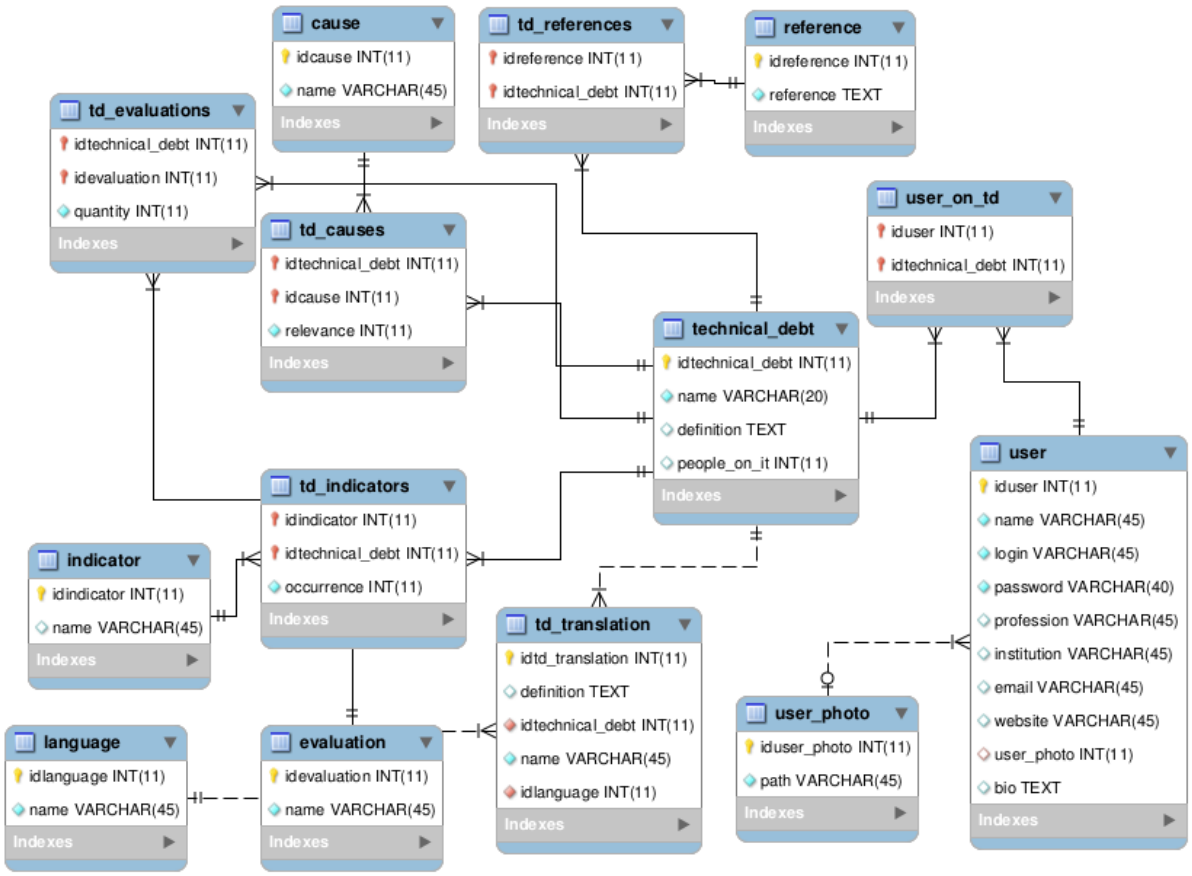
6.4.2 Arquitetura

O modelo de dados apresentado na Figura 23 foi criado de forma a atender aos requisitos funcionais e representar o conhecimento organizado em um esquema relacional. Para isso, foram criadas entidades para representar os principais conceitos (tipos de DT, indicadores, causas, métodos de avaliação, e referências) e as relações entre eles.

Alguns dos aspectos colaborativos da infraestrutura foram capturados por meio de tabelas como *user_on_td*, que armazena o número de usuários que estão pesquisando e/ou trabalhando em um determinado tipo de dívida. Já os campos *relevance* (da tabela *td_causes*), *occurrence* (da tabela *td_indicators*) e *quantity* (da tabela *td_evaluations*) permitem armazenar a indicação de causas e indicadores de dívida mais relevantes considerando a opinião de cada usuário registrado em TD Wiki. Por fim, a tabela *td_translation* foi criada de forma que seja possível traduzir o conhecimento mapeado para outras línguas.

Este modelo de dados foi inicialmente preenchido com o conhecimento mapeado nos Capítulos 3, 4 e 5.

Figura 23 - Modelo de dados do TD Wiki

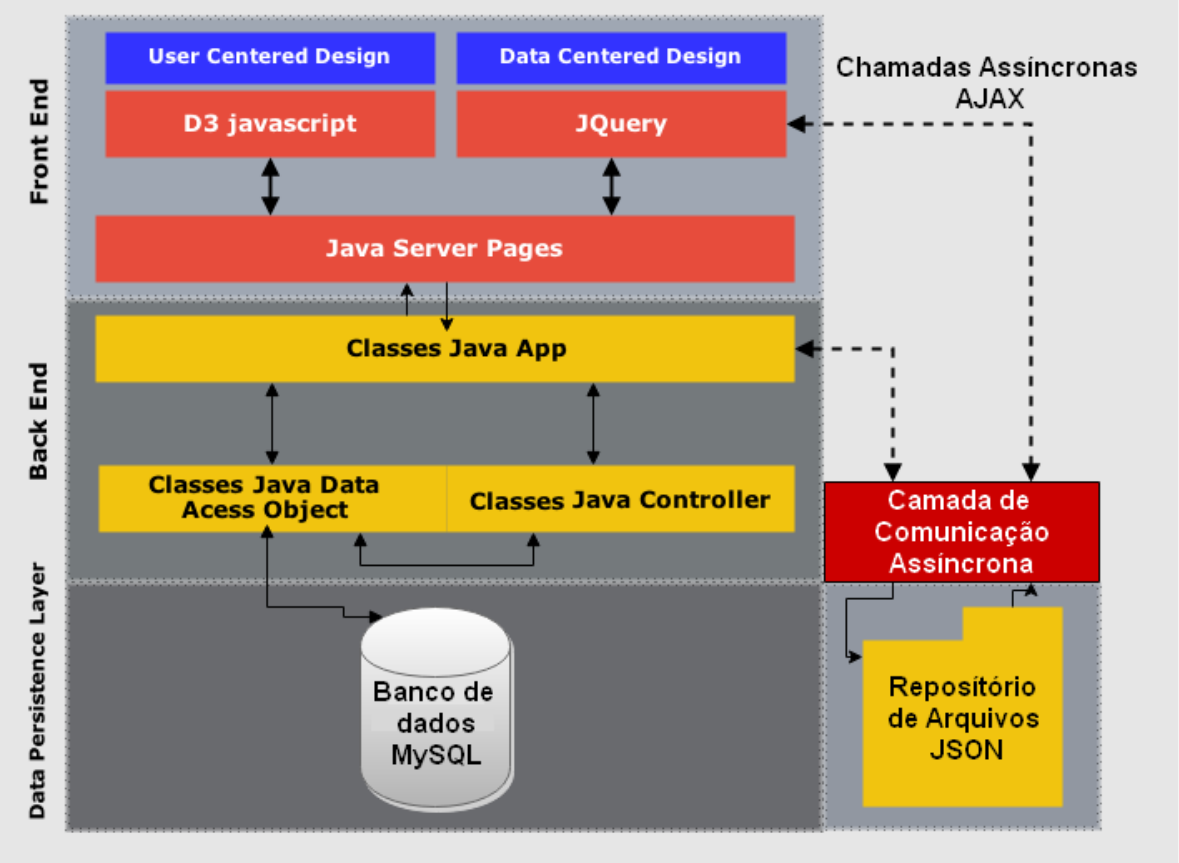


Em seguida, uma arquitetura de quatro camadas (*FrontEnd*, *BackEnd*, Camada de Persistência de Dados e a Camada de Comunicação Assíncrona) foi definida (Figura 24).

Na camada *FrontEnd*, foram utilizadas as seguintes tecnologias:

- JavaServer Pages (JSP): para a renderização de páginas web dinâmicas;
- D3 JavaScript: uma biblioteca focada em disponibilizar experiências ricas para visualização da informação. Esta tecnologia foi utiliza para representar o conhecimento sobre DT;
- JQuery: biblioteca utilizada para permitir uma maior abstração das funcionalidades do JavaScript e utilizar chamadas AJAX possibilitando que a troca de informações seja realizada com um baixo tempo de resposta.

Figura 24 - Arquitetura de TD Wiki



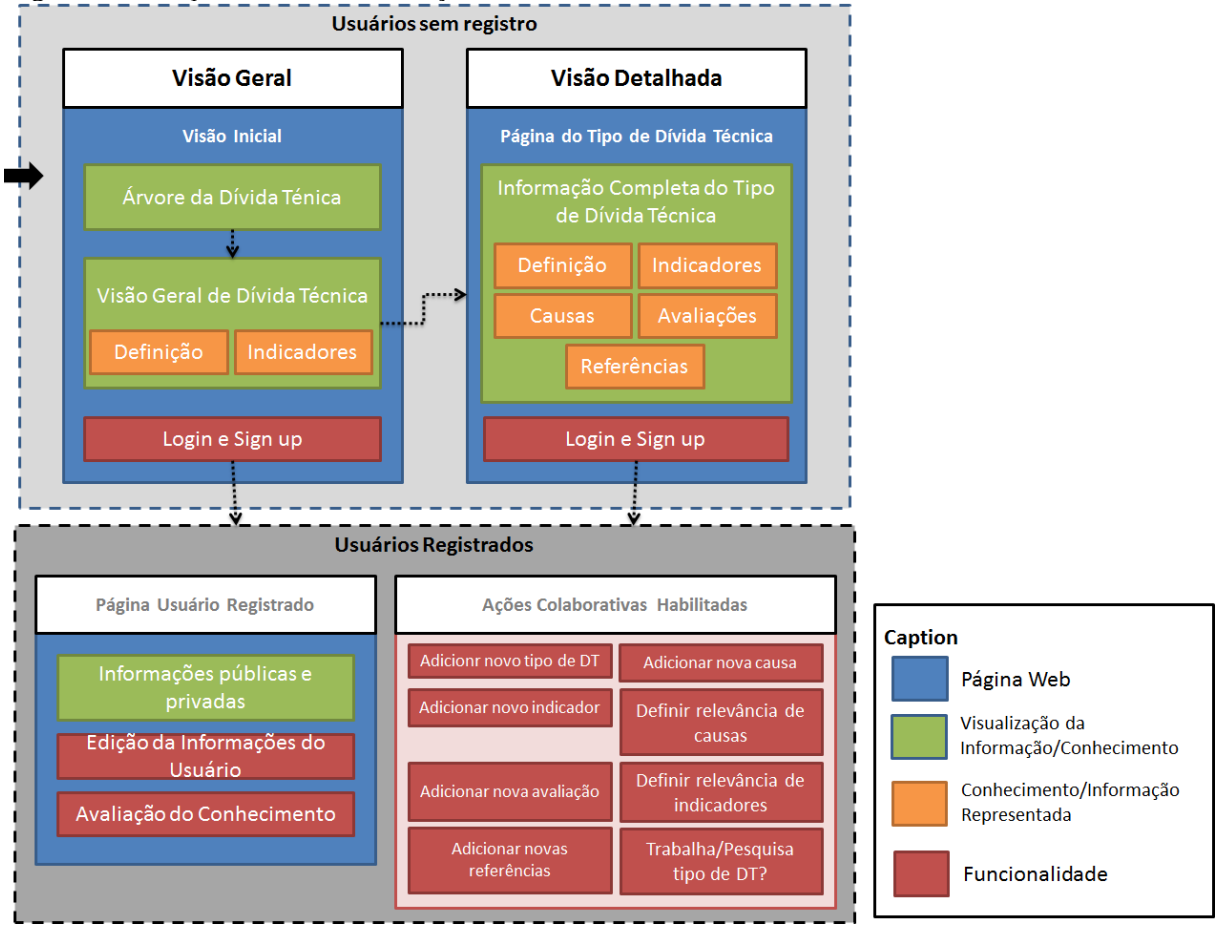
Já na camada *BackEnd* foi utilizado o Java JEE. A organização das classes seguiu os padrões *Data Access Object* (DAO), *JavaBean* e *Controller*, de forma a facilitar futuras extensões e manutenções no código. Além disso, foi utilizada uma camada intermediária de comunicação assíncrona entre as camadas *FrontEnd* e *BackEnd*. Essa camada foi necessária para enriquecer a experiência de uso da aplicação. O AJAX permite uma melhora no tempo de resposta da aplicação, porque não é necessário que uma página seja completamente carregada a cada requisição. Como as chamadas AJAX são eficazes quando trabalham com objetos JSON e a manipulação dos recursos visuais de TD Wiki demandam um baixo tempo de resposta, parte das informações persistidas no banco MySQL também foram persistidas como o JSON no JSON *File Repository*. As informações replicadas nessa base auxiliar são aquelas utilizadas para alimentar os recursos de visualização do sistema.

Por fim, tem-se a camada de persistência de dados utilizando o MySQL. Todas as informações a respeito dos usuários registrados forma persistidas de forma criptografada, tornando os dados ofuscados em caso de acesso indevido.

6.4.3 Acesso às informações

O conhecimento sobre DT foi organizado em duas camadas: visão geral e visão detalhada (Figura 25). A camada de visão geral está representada na página inicial de TD Wiki. Nesta página, são apresentadas uma breve descrição de cada tipo de dívida e alguns de seus indicadores. Além disso, esta página conterá uma árvore representando os tipos conhecidos de DT. É a partir desta árvore que o usuário pode obter mais detalhes sobre cada tipo. Na camada de conhecimento detalhado, o usuário tem acesso aos indicadores conhecidos de cada tipo de dívida, as causas que podem levar à sua ocorrência, estudos de avaliação realizados e algumas referências.

Figura 25 - Arquitetura da informação



Quando autenticados na infraestrutura, os usuários terão acesso a outros recursos como a sua página pessoal. Eles podem contribuir, com base na sua experiência, para a evolução do conhecimento sobre DT por meio da adição de novos tipos, indicadores, causas e também através da definição da importância de cada tipo de indicador ou causa.

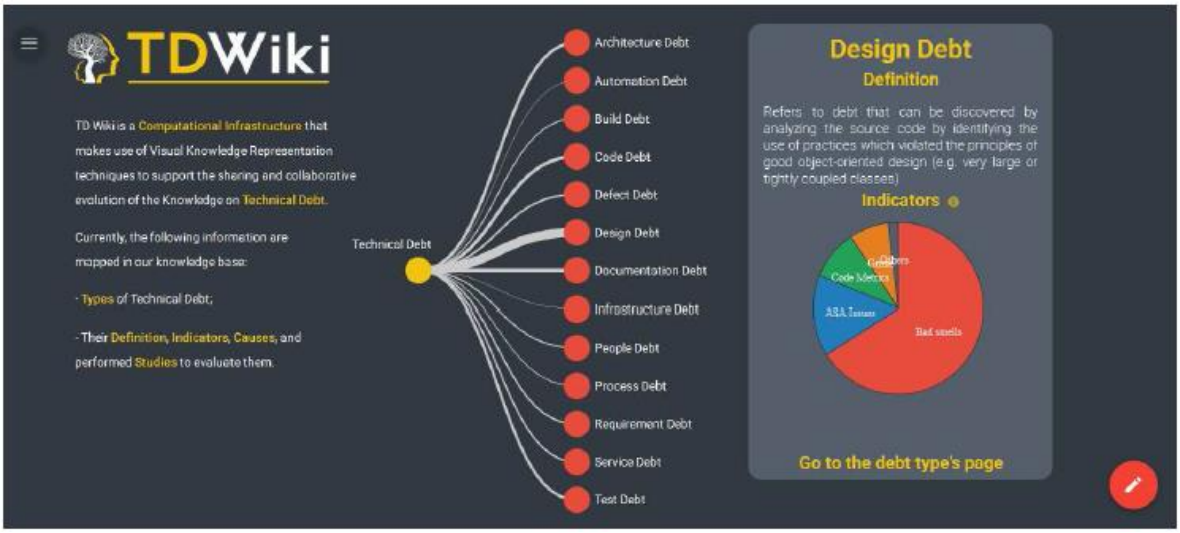
6.4.4 TD Wiki

Nesta seção serão apresentadas as funcionalidades de TD Wiki. Elas foram agrupadas em três categorias: Compartilhamento do Conhecimento, Evolução do Conhecimento e Funcionalidades de Apoio.

6.4.4.1 Compartilhamento do Conhecimento

TD Wiki permite o compartilhamento do conhecimento sobre DT utilizando técnicas de representação visual de conhecimento. Ao acessar TD Wiki, a primeira tela apresentada é a *initial view* (Figura 26). Nela é possível visualizar a taxonomia em formato de árvore que apresenta todos os tipos de DT cadastrados na infraestrutura (REQ01). Além disso, a espessura da linha que associa os tipos de dívida à raiz da árvore indica o quanto aquele tipo tem sido considerado no monitoramento da dívida na indústria de software ou investigado em atividades de pesquisa. Quanto mais espessa a linha, mais aquele tipo tem sido considerado.

Figura 26 - Página Inicial – Visão Geral da Camada de Conhecimento



Ao passar o *mouse* sobre um determinado tipo, um breve resumo será exibido contendo sua definição e alguns de seus indicadores (REQ02). Ao final desse resumo, tem-se a opção '*Go to the debt's page*' que direciona o usuário para a *Technical Debt Type's Page* (Figura 27) correspondente (REQ03). Nesta página é possível ter acesso às seguintes informações sobre a DT selecionada:

- **Definição do tipo de DT:** contém a definição do tipo de dívida;

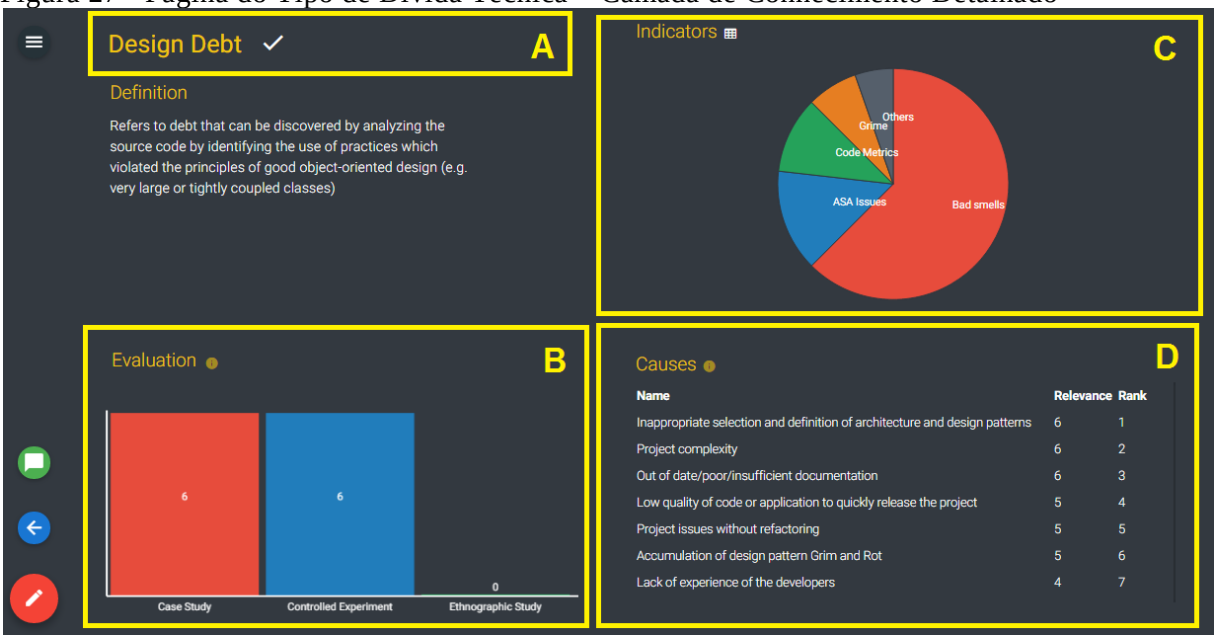
- **Indicadores da DT:** contém indicadores que têm sido utilizados para apoiar a localização de itens da dívida em projetos de software. Também é possível saber quais indicadores têm sido mais utilizados ou recomendados para apoiar a tarefa de identificar aquele tipo de dívida. Quanto mais alto na lista de indicadores, mais recomendado ele é. Nesta área também temos o gráfico de pizza que mostra a proporção das recomendações feitas;

- **Causas da DT:** contém possíveis causas que levam a equipe de desenvolvimento a incorrer aquele tipo de dívida. Uma lista foi utilizada para representar as causas identificadas. Quanto maior o valor do campo "*Relevance*", mais relevante ela é considerada como fator que leva aquele tipo de dívida a ser inserido no projeto;

- **Avaliação:** representa os estudos que avaliaram aquele tipo de DT. A quantidade de estudos pode ser um indicador sobre o quanto se conhece sobre um tipo de DT;

- **Referências:** contém uma lista de referências onde mais informações podem ser obtidas a respeito dos dados representados por TD Wiki.

Figura 27 - Página do Tipo de Dívida Técnica – Camada de Conhecimento Detalhado



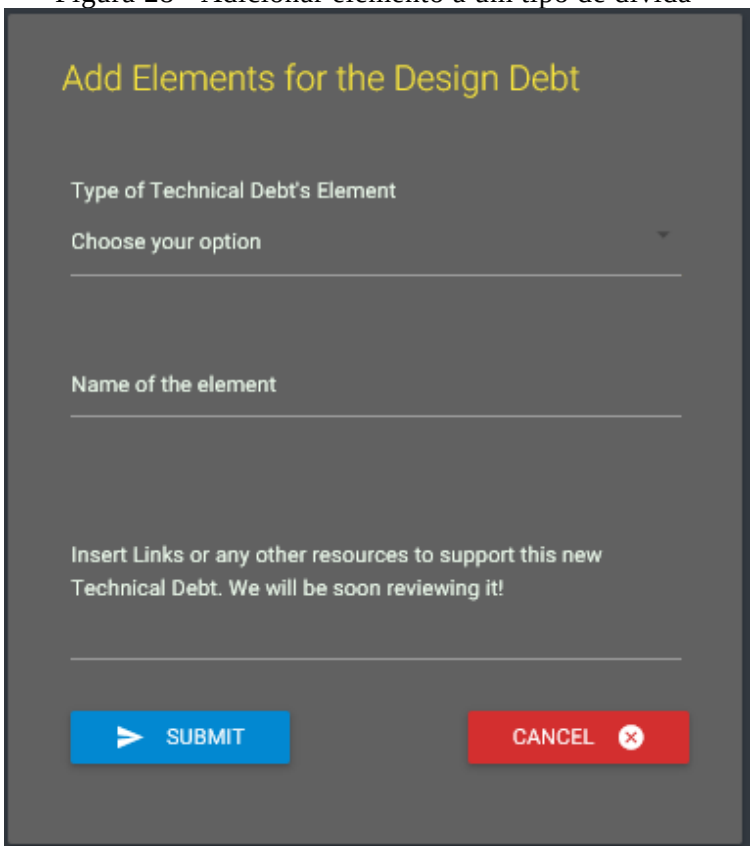
6.4.4.2 Evolução do conhecimento

TD Wiki permite a evolução colaborativa do conhecimento sobre DT possibilitando que usuários registrados adicionem novos tipos de dívida, novos indicadores, novas causas e novas referências (REQ05). Para cada uma dessas funcionalidades de inserção, um grupo de três moderadores avalia as informações fornecidas. Se aprovada, a nova informação é incorporada à base de conhecimento. Este controle é necessário porque DT tem sido usada às vezes como uma *buzzword*, o que pode levar a interpretações errôneas de seu significado.

Além disso, TD Wiki também permite aos usuários indicar que tipo de DT eles estão trabalhando e, com base em sua experiência, quais indicadores são mais adequados para a identificação de um determinado tipo (REQ06). Além disso, também é possível indicar quais são as principais causas que contribuem para que um determinado tipo de dívida seja incorrido (REQ07). Na Figura 27, quadro A, o usuário pode sinalizar que está trabalhando/pesquisando sobre um determinado tipo de dívida clicando sobre o "*check*". O resultado desta ação será apresentado na árvore ontológica através da espessura da linha que relaciona cada tipo à raiz da árvore.

No quadro B, o usuário pode visualizar os estudos que avaliaram esse tipo especificando o tipo de estudo e as suas referências. Novos estudos podem ser adicionados clicando-se em Adicionar (botão amarelo no canto inferior esquerdo da tela), informando a opção, nesse caso "Avaliação" e, em seguida, seu nome (campo Nome do Elemento) e referências para apoiar a avaliação (feita pelos moderadores) dos novos itens, conforme mostra a Figura 28.

Figura 28 - Adicionar elemento a um tipo de dívida



No quadro C, o usuário pode informar a relevância de um indicador particular de DT. Para isso, o usuário atribui um valor para cada indicador, clicando nas setas para baixo (diminui o valor) e para cima (aumenta o valor). Como resultado, o *rank* será atualizado, considerando a classificação definida por todos os usuários da infraestrutura. O usuário pode definir, para cada indicador, se ele é relevante (+1 ponto), neutro (sem pontuação) ou sem relevância (-1 ponto). Assim, ao final o *rank* é calculado através do somatório de todos os valores informados por todos os usuários para cada indicador.

Além disso, os indicadores também poderão ser visualizados através de um gráfico de pizza clicando-se no ícone correspondente ao lado do nome “Indicators”. Novos indicadores podem ser adicionados clicando-se em Adicionar (botão amarelo no canto inferior esquerdo da tela), informando a opção, nesse caso “Indicador” e, em seguida, seu nome (campo Nome do Elemento) e referências, conforme mostra a Figura 28.

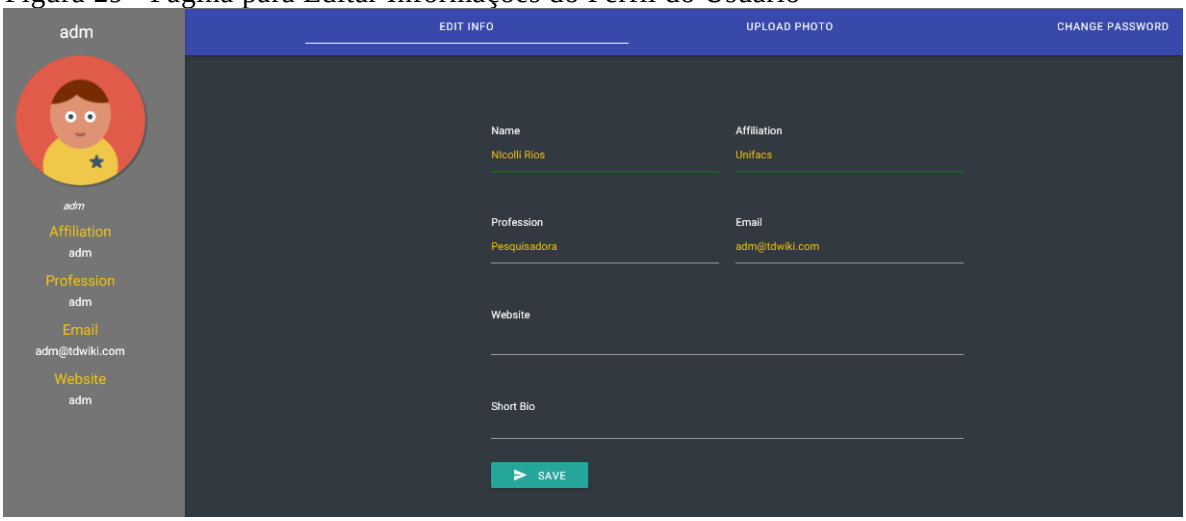
Finalmente, no quadro D, o usuário pode indicar a relevância das causas no que diz a respeito àquelas que mais levam à ocorrência de um determinado tipo de dívida. Ao fazê-lo, TD Wiki irá recalcular o *rank* (de forma semelhante ao realizado para os indicadores de

dívida). Novas causas também podem ser adicionadas da mesma forma que indicadores, informando a opção “Causas” e, em seguida, seu nome (campo Nome do Elemento) e referências, conforme mostra a Figura 28.

6.4.4.3 Funcionalidades de Apoio

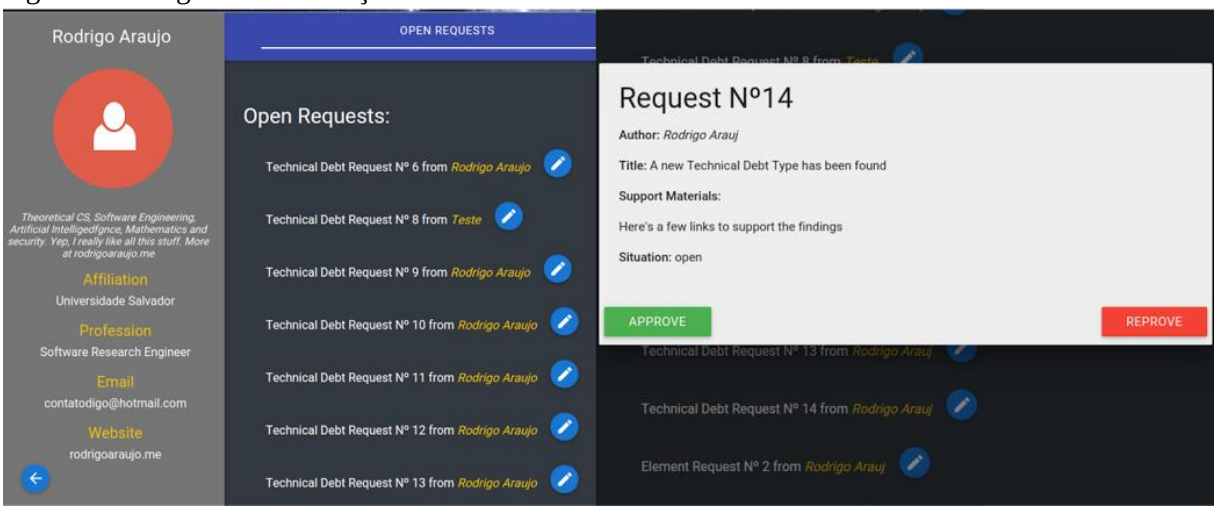
O controle de acesso permitirá o registro de novos usuários e o acesso a funcionalidades de evolução do conhecimento a usuários já cadastrados. Novos usuários poderão criar perfis pessoais, contendo informações sobre seus trabalhos e suas pesquisas, nome completo, instituição, profissão, email e website (REQ4). No perfil do usuário, também será possível editar informações, alterar a senha e enviar e-mail para a equipe do TD Wiki. O usuário ainda poderá visualizar suas atividades recentes (colaborações realizadas) (Figura 29). A ideia de se ter um perfil social dentro da infraestrutura fortalece o conceito de sistemas colaborativos, pois permitirá mapear os profissionais e pesquisadores que têm trabalhado com DT.

Figura 29 - Página para Editar Informações do Perfil do Usuário



Além disso, um grupo de três moderadores, definido pelo administrador do TD Wiki (REQ09), será capaz de avaliar novas informações fornecidas pelos usuários (REQ08) (Figura 30). Para ser incluída na base de conhecimento do TD Wiki, a informação precisa ser aprovada por pelo menos dois moderadores. No caso de não aprovação, o moderador precisa justificar a sua decisão. Ao final, o usuário que solicitou a inclusão será informado sobre o resultado de seu pedido.

Figura 30 - Página de Avaliação do Conhecimento



6.5 CONSIDERAÇÕES FINAIS

O desenvolvimento de sistemas de informação possui uma série de barreiras. O acúmulo de dívida técnica cria um desafio a mais impactando em atividades de evolução do sistema. Assim, identificar e monitorar a dívida são essenciais no sentido de reduzir os riscos envolvidos na construção desses sistemas. Contudo, antes mesmo que a equipe de desenvolvimento possa trabalhar na identificação e monitoramento da DT, é necessário entender quais tipos de dívida podem ser incorridos, como ela pode ser identificada e o que pode levar a equipe de desenvolvimento a incorrê-la no projeto.

Neste contexto, este capítulo apresentou TD Wiki, uma infraestrutura computacional para o compartilhamento e evolução colaborativa do conhecimento sobre DT. Como contribuição principal, TD Wiki fornece as informações necessárias para que equipes de desenvolvimento comecem a monitorar a DT em seus projetos. Isso pode ser considerado um passo importante no sentido de melhorar a qualidade dos sistemas de informação em desenvolvimento. Além disso, TD Wiki cria um canal que estimula a discussão sobre DT por profissionais e pesquisadores, e permite a evolução colaborativa desse conhecimento.

O conhecimento representado na infraestrutura baseia-se no trabalho de Alves *et al.* (2016). Não há garantia de que novos tipos, indicadores e/ou causas tenham sido identificados desde então. Contudo, a infraestrutura já se encontra disponível para uso em www.tdwiki.com e, dessa forma, espera-se que o conhecimento mapeado seja evoluído pela própria comunidade.

7 CONSIDERAÇÕES FINAIS

Neste capítulo são apresentadas as considerações finais sobre o trabalho realizado incluindo suas principais contribuições, os resultados obtidos, limitações os próximos passos a serem realizados com a continuidade desta pesquisa.

7.1 CONSIDERAÇÕES FINAIS

Dívida técnica descreve o efeito negativo que pode existir ao se utilizar o caminho mais fácil e rápido para uma solução de software, objetivando um ganho de tempo imediato, em detrimento de segurança, estabilidade e escalabilidade futuros. Trata-se de uma área que tem crescido consideravelmente. Muito disso se deve ao fato de seus conceitos serem de fácil entendimento tanto por pesquisadores quanto por profissionais da indústria de software.

Diversas pesquisas têm sido realizadas na área em busca do desenvolvimento de estratégias de identificação, gerenciamento, monitoramento e quantificação da dívida existente nos projetos. No entanto, muito do conhecimento que tem sido desenvolvido está espalhado na literatura técnica, tornando-se necessário a organização de um corpo de conhecimento considerando os tipos de dívida conhecidos, seus indicadores e estratégias de gerenciamento, para que seja possível direcionar melhor esforços de pesquisa na área.

Neste contexto, este trabalho propôs a organização de um corpo de conhecimento em dívida técnica considerando seus tipos, indicadores, estratégias de gerenciamento e causas para sua ocorrência. Em complemento, compartilhou o corpo de conhecimento organizado através da infraestrutura TD Wiki. Para alcançar estes resultados, a pesquisa seguiu duas linhas de trabalho: (1) realização de um mapeamento sistemático da literatura complementada por um *survey* para a coleta de evidências que permitissem organizar o conhecimento sobre DT; (2) estruturação das informações identificadas a partir o estudo em uma taxonomia de tipos de dívida e em uma infraestrutura que apoia o compartilhamento e evolução colaborativa do conhecimento – TD Wiki.

7.2 RESULTADOS E CONTRIBUIÇÕES OBTIDAS

Ao analisar os quatro objetivos definidos para este trabalho e descritos na Seção 1.4, os seguintes resultados e contribuições foram obtidos:

1. **Organização de um corpo de conhecimento sobre dívida técnica:** o corpo de conhecimento foi organizado através da execução de um mapeamento sistemático da literatura que permitiu investigar quais são as estratégias que têm sido propostas para a identificação ou o gerenciamento da DT em projetos de software (ALVES *et al.*, 2016). A realização do mapeamento permitiu identificar os tipos de dívida, seus indicadores e estratégias de gerenciamento para a DT em projetos de software. Contribuições específicas desta organização estão descritas na seção 3.8 do Capítulo 3;

2. **Estruturação o corpo de conhecimento organizado:** foi elaborada uma taxonomia que considera os tipos de dívida identificados (ALVES *et al.*, 2014). Esta taxonomia foi avaliada e serviu de base para compor o conhecimento disponibilizado em TD Wiki;

3. **Identificação das causas da dívida técnica:** através da execução de um *survey*, foi possível caracterizar as causas que levam a equipe de desenvolvimento a incorrer a DT em projetos de software. Também foi possível investigar questões associadas à possibilidade de prevenção da dívida e se as causas afetam os projetos de forma isolada ou em cadeia. Os resultados deste estudo complementaram os resultados obtidos com o mapeamento sistemático da literatura. Contribuições específicas deste estudo estão descritas na seção 5.4 do Capítulo 5;

4. **Compartilhamento do corpo de conhecimento organizado de forma que ele possa ser acessado e evoluído pela comunidade de pesquisa e profissionais da área:** foi desenvolvida uma infraestrutura que apoia o compartilhamento e evolução colaborativa do conhecimento organizado sobre DT – TD Wiki (ALVES *et al.*, 2015). A infraestrutura pode ser acessada em www.tdwiki.com.

Em conjunto, estes resultados contribuem para a evolução do *Technical Debt Landscape* (SEAMAN; GUO, 2011) (KRUCHTEN *et al.*, 2012). Além disso, também

contribuem para a aplicação dos conceitos sobre DT na indústria de software, uma vez que engenheiros de software podem utilizar o corpo de conhecimento organizado para:

- Entender os diferentes tipos de dívida que podem afetar projetos de software;
- Conhecer as estratégias de gerenciamento existentes e, como consequência, utilizá-las, adaptá-las ou tomarem elas como base para definir sua própria estratégia para o controle da dívida;
- Conhecer os diferentes indicadores que podem ser utilizados para apoiar a identificação dos tipos de dívida. Este conhecimento é fundamental ao se definir uma estratégia para identificar e monitorar a dívida existente nos projetos;
- Conhecer as diferentes causas que podem levar a equipe a incorrer a dívida no projeto. Este conhecimento pode ser utilizado para apoiar a definição de estratégias que atuem no sentido de prevenir a ocorrência de itens da dívida.

7.3 LIMITAÇÕES

As principais limitações deste trabalho estão relacionadas aos estudos executados e apresentados nos Capítulos 3 e 5, e à não avaliação experimental de TD Wiki:

(1) Abrangência do Mapeamento Sistemático executado

O processo utilizado na execução do MS aumentou a probabilidade do retorno de artigos relevantes para este trabalho, no entanto, como o processo é longo e possui etapas subjetivas, é possível que estudos importantes não tenham sido considerados nesta pesquisa. Além disso, as questões de pesquisa definidas para o mapeamento não capturam outras informações relevantes da área e que poderiam fazer parte do corpo de conhecimento organizado como estratégias para quantificação da dívida e possíveis consequências que a dívida pode trazer para o projeto.

Nesse sentido, seria interessante que os resultados de outros mapeamentos ou revisões sistemáticas realizadas na área fossem agrupados aos resultados obtidos com o mapeamento realizado neste trabalho através da execução de um estudo terciário.

(2) Número e perfil dos participantes do *survey*

O número de participantes do *survey* executado neste trabalho foi baixo, dificultando uma possível generalização dos resultados obtidos. Também devido a esta limitação, não foi possível realizar análises específicas sobre a percepção que perfis diferentes de profissionais têm sobre as causas que levam a equipe a incorrer itens da dívida e se eles poderiam ou não ser evitados.

Para lidar com essas limitações, é interessante a reaplicação do experimento com um número maior de participantes considerando diferentes perfis de profissionais.

(3) Não avaliação experimental de TD Wiki

A infraestrutura computacional passou por atividades internas de testes exploratórios, porém ainda não foi avaliada experimentalmente em relação à sua aplicabilidade para compartilhar e evoluir o conhecimento sobre DT. É interessante que este tipo de avaliação seja realizado com o objetivo de identificar possíveis limitações de TD Wiki e aprimorá-la a partir dos pontos de melhoria levantados. Essa avaliação experimental é um possível trabalho futuro derivado desta dissertação.

7.4 TRABALHOS FUTUROS

Ainda há muito a ser pesquisado e explorado sobre dívida técnica. Considerando este trabalho como ponto de partida, algumas perspectivas futuras de trabalho são:

- Identificar causas e consequências que a dívida pode trazer para o projeto de forma que a equipe possa trabalhar em atividades que apoiem o monitoramento e a prevenção da DT;
- Ainda não se sabe como as causas identificadas estão relacionadas a indicadores de presença da dívida. Algumas relações entre causas e indicadores parecem ser justificadas logicamente, por exemplo: falta de experiência dos desenvolvedores e violação de modularidade; falta de experiência dos desenvolvedores e dependências estruturais. No entanto, uma investigação aprofundada do tipo de relação (causa - consequência) pode ser considerada uma área em aberto e merece estudos experimentais para permitir sua análise;

- Realizar novos estudos para identificar ações que possam ser tomadas para evitar que itens da dívida sejam inseridos no projeto;
- Replicar o *survey* executado neste trabalho. Essa replicação permitirá que os resultados sejam mais generalizáveis e permitirá que análises específicas (por exemplo: percepção da dívida por perfil de profissional) sejam mais bem trabalhadas;
- Realizar avaliação experimental da infraestrutura computacional TD Wiki de modo que sua aplicabilidade seja avaliada;
- Executar um estudo terciário com o intuito de agrupar os resultados de outros mapeamentos ou revisões sistemáticas realizadas na área com os resultados obtidos com o mapeamento realizado neste trabalho. Este estudo permitirá sintetizar uma parcela significativa das evidências geradas até o momento sobre DT.

REFERÊNCIAS

ADAMS, K. C. Immersed in structure: the meaning a function of taxonomies. **Internetworking. USA**, n. 3.2, 2000.

ALVES, N. S. R. et al. Towards an Ontology of Terms on Technical Debt. In: INTERNATIONAL WORKSHOP ON MANAGING TECHNICAL DEBT (MTD '14), 6., IEEE COMPUTER SOCIETY, 6., 2014, Washington, DC, USA. **Proceedings…** 2014. DOI=10.1109/MTD.2014.9 http://dx.doi.org/10.1109/MTD.2014.9.

ALVES, N.S.R.; ARAÚJO, R.S.; SPÍNOLA, R.O. A Collaborative Computational Infrastructure for Supporting Technical Debt Knowledge Sharing and Evolution. In: AMERICAS CONFERENCE ON INFORMATION SYSTEMS, AMERICAS CONFERENCE ON INFORMATION SYSTEMS, 2015, Puerto Rico. **Proceedings…** 2015.

ALVES, N.S.R. et al. Identification and Management of Technical Debt: A Systematic Mapping Study. **Information and Software Technology**, 70, p.100 – 121, 2016.

ALZAGHOUL, E. ; BAHSOON, R. CloudMTD: Using real options to manage technical debt in cloud-based service selection. In: MANAGING TECHNICAL DEBT (MTD), INTERNATIONAL WORKSHOP, 4., 2013. **Proceedings…** 2013. p. 55-62.

AMPATZOGLOU, A. et al. Reducing Friction in Software Development. **Software, IEEE**, v.33, n.1, p.66-73, jan.-feb. 2016. doi: 10.1109/MS.2016.13

AVGERIOU, P. The financial aspect of managing technical debt: A systematic literature review. **Information and Software Technology**, v. 64, apr. 2015, p.52-73. ISSN 0950-5849. http://dx.doi.org/10.1016/j.infsof.2015.04.001.

BASILI, V. R. The Experimental Paradigm in Software Engineering. In: INTERNATIONAL WORKSHOP ON EXPERIMENTAL SOFTWARE ENGINEERING ISSUES: CRITICAL ASSESSMENT AND FUTURE DIRECTIONS, 199, London, UK. **Proceedings…** 1992.

BASILI, V. R., SHULL, F., LANUBILE, F. Building Knowledge Through Families of Experiments. **IEEE Transactions on Software Engineering**, v. 25, n. 4, jul./aug. 1999.

BINKLEY, D. Source code analysis: a road map. **Future of Software Engineering**, 2007, p. 104 –119.

BIOLCHINI, J. et al. **Systematic Review in Software Engineering**. Technical Report ES 679/05. COPPE/UFRJ, 2005.

B. JIE, H. ZHILIANG; C. DOINA, R. JAMES; H. VASANT, G. A Tool for Collaborative Construction of Large Biological Ontologies. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 17., 2006, **Proceedings...** 2006. p. 191–5.

B. P. LIENTZ, et al. Characteristics of Application Software Maintenance. **Communications of the ACM**, v. 21, p. 6, 1978.

BRERETON, P. et al. Lessons from applying the systematic literature review process within the software engineering domain. **Journal of Systems and Software,** v.80, p.571–583, 2007.

BRILLIANT, S. S.; KNIGHT, J. C. Empirical research in software engineering: a workshop. organized by the University Of Virginia the University Of Maryland. **Software Engineering Notes,** v. 24, n.3, p.45-52, jun. 1998.

BROWN, Nanette et al. Managing technical debt in software-reliant systems. In: FSE/SDP WORKSHOP ON FUTURE OF SOFTWARE ENGINEERING RESEARCH. ACM, New York, NY, USA, 2010. **Proceedings... 2010.** p. 47-52.

BUDEGN, D. et al. Using Mapping Studies in Software Engineering. In: PPIG PSYCHOLOGY OF PROGRAMMING INTEREST GROUP, 2008. **Proceedings…** Lancaster University, UK, 2008. p. 195–204.

CAI, Y. Introducing Tool-Supported Architecture Review into Software Design Education. In: CONFERENCE ON SOFTWARE ENGINEERING EDUCATION AND TRAINING (CSEE&T), 26., 2013. **Proceedings…** 2013.

CALERO, Coral; RUIZ, Francisco; PIATTINI, Mario. **Ontologies for software engineering and software technology.** [S.l]: Springer Science & Business Media, 2006.

CUNNINGHAM, Ward. The WyCash portfolio management system. **ACM SIGPLAN OOPS Messenger**, v. 4, n. 2, p. 29-30, 1992.

DE ALMEIDA FALBO, Ricardo. Experiences in using a method for building domain ontologies. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, SEKE'2004. INTERNATIONAL WORKSHOP ON ONTOLOGY IN ACTION, OIA'2004. 2004. **Proceedings...** 2004. p. 474-477.

DE ALMEIDA FALBO, Ricardo; DE MENEZES, Credine Silva; DA ROCHA, Ana Regina C. A systematic approach for building ontologies. In: PROGRESS in Artificial Intelligence—IBERAMIA 98. Berlin: Springer Berlin Heidelberg, 1998. p. 349-360.

DEKLEVA, Sasa M. Software maintenance: 1990 status. **Journal of Software Maintenance:** Research and Practice, v. 4, n. 4, p. 233-247, 1992.

DIEHL, Stephan. **Software visualization:** visualizing the structure, behaviour, and evolution of software.[S.l.]: Springer Science & Business Media, 2007.

FAIRLEY, Richard. Risk management for software projects. **IEEE software**, v. 11, n. 3, p. 57, 1994.

FOGL, Jiri. RELATION OF THE CONCEPTS INFORMATION AND KNOWLEDGE. In: INTERNATIONAL FORUM ON INFORMATION AND DOCUMENTATION. PO BOX 90402, 2509 LK THE HAGUE, NETHERLANDS: INT FEDERAT INFORMATION & DOCUMENTATION, 1979. **Proceedings...** 1979. p. 21-24.

FOWLER, Martin. Technical debt quadrant. **Bliki [Blog].** Available from: <http://www. martinfowler. com/bliki/TechnicalDebtQuadrant. html>, 2009. Acesss in: 1º jul. 2014.

GRUBER, Thomas R. Toward principles for the design of ontologies used for knowledge sharing?. **International journal of human-computer studies**, v. 43, n. 5, p. 907-928, 1995.

GUARINO, Nicola. Formal ontology in information systems. In: INTERNATIONAL CONFERENCE (FOIS'98), 5., 1998, Trento, Italy. **Proceedings...** 1998.

GUO, Yuepu et al. Domain-specific tailoring of code smells: an empirical study. In: ACM/IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 32., 2011. **Proceedings...** 2011. p. 167-170.

GUO, Yuepu; SPÍNOLA, Rodrigo Oliveira; SEAMAN, Carolyn. Exploring the costs of technical debt management–a case study. **Empirical Software Engineering**, v. 21, n. 1, p. 159-182, 2016.

IZURIETA, Clemente et al. On the uncertainty of technical debt measurements. In: INFORMATION SCIENCE AND APPLICATIONS (ICISA), 2013 INTERNATIONAL CONFERENCE ON. IEEE, 2013. **Proceedings…** 2013.p. 1-4

KITCHENHAM, B.; DYBÅ, T.; JORGENSEN, M. Evidence-based software engineering. In: ICSE, 26., 2004, Scotland, UK. **Proceedings…** 2004.

KITCHENHAM, B., CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering. In: **Technical report, Ver. 2.3 EBSE Technical Report. EBSE**. 2007.

KITCHENHAM, Barbara A.; MENDES, Emilia; TRAVASSOS, Guilherme H. Cross versus within-company cost estimation studies: a systematic review.**Software Engineering, IEEE Transactions on**, v. 33, n. 5, p. 316-329, 2007.

KRUCHTEN, P.; NORD, R. L.; OZKAYA, Ipek. Technical debt: from metaphor to theory and practice. **Ieee software**, n. 6, p. 18-21, 2012.

LANZA, M.; MARINESCU, R.; DUCASSE, S. **Object-oriented metrics in practice**. New York: Inc. Secaucus, NJ; USA: Springer-Verlag, 2005.

LEHMAN, Meir M.; BELADY, Laszlo A. **Program evolution:** processes of software change. [S.l.]: Academic Press Professional, Inc., 1985.

LI, Z., AVGERIOU, P., LIANG, P. (2015). A systematic mapping study on technical debt and its management. In Journal of Systems and Software, Volume 101, March 2015, Pages 193–220. doi:10.1016/j.jss.2014.12.027

MCCONNELL, S. **Technical Debt. 10x Software Development** [Blog]. 2007. Available at: <http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx>. Acess in: 10 dec. 2014.

MCGUINNESS, Deborah L. et al. OWL web ontology language overview.**W3C recommendation**, v. 10, n. 10, p. 2004, 2004.

MORGENTHALER, J. David et al. Searching for build debt: Experiences managing technical debt at google. In: INTERNATIONAL WORKSHOP ON MANAGING TECHNICAL DEBT. 3., 2012. **Proceedings...** 2012. p. 1-6.

NOVAIS, Renato Lima et al. Software evolution visualization: a systematic mapping study. **Information and Software Technology**, v. 55, n. 11, p. 1860-1883, 2013.

NUNAMAKER JR, Jay F.; CHEN, Minder; PURDIN, Titus DM. Systems development in information systems research. **Journal of management information systems**, v. 7, n. 3, p. 89-106, 1990.

PARNAS, David Lorge. Software aging. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 16., 1994. **Proceedings...** 1994. p. 279-287.

PETERSEN, Kai et al. Systematic mapping studies in software engineering. In: INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, 12., 2008. Bari, Italy. **Proceedings**... 2008.p. 8. p. 1-10.

PETTICREW, M. ; ROBERTS, H. **Systematic reviews in the social sciences**: a practical guide. New York: Blackwell Publishing, 2006.

PFLEEGER, S. **Software engineering**: theory and practice. 3. ed. New York: Prentice Hall, 2005.

PRESSMAN, R. S. **Software engineering**: a practitioner's approach. New York: McGraw-Hill, 1997. p. 253-259.

ROSPOCHER, M. ; GHIDINI, C. ; DI FRANCESCOMARINO, C. Evaluating wiki collaborative features in ontology authoring. **Knowledge and Data Engineering, IEEE Transactions on**, v. 26, n. 12, p. 2997-3011, 2014.

ROTHMAN, J. An incremental technique to pay off testing technical debt.**stickyminds, Weekly Column**, 2006.

SCHUMACHER, Jan et al. Building empirical support for automated code smell detection. In: ACM-IEEE INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT. ACM, 2010. **Proceedings**... 2010.p. 8.

SEAMAN, C. ; SPÍNOLA, R.O. Managing Technical Debt, (Short Course). In: BRAZILIAN SYMPOSIUM ON SOFTWARE QUALITY, 17., Salvador, Brazil, 2013. **Anais…** 2013.

SHAH, S. et al. Exploratory Testing as a Source of Technical Debt. **IT Professional,** v.16, n.3, p.44-51, 2014.

SPÍNOLA, Rodrigo O. et al. Investigating technical debt folklore: Shedding some light on technical debt opinion. In: INTERNATIONAL WORKSHOP ON MANAGING TECHNICAL DEBT, 4., 2013. **Proceedings...** 2010. p. 1-7.

SOUSA, Maria João Castro; MOREIRA, Helena Mendes. A survey on the software maintenance process. In: SOFTWARE MAINTENANCE, 1998. INTERNATIONAL CONFERENCE ON. IEEE, 1998. **Proceedings...** 1998. p. 265-274.

TOMAS, P.; ESCALONA, M.J.; MEJIAS, M. Open source tools for measuring the Internal Quality of Java software products. **A survey, Computer Standards & Interfaces**, v. 36, n. 1, p. 244-255, 2013.

VILLAR, A.; MATALONGA, S. Definiciones y tendencia de deuda técnica: un mapeo sistemático de la literatura. In: CIBSE13 - CONGRESSO IBERO-AMERICANO EM ENGENHARIA DE SOFTWARE, Montevideo, Uruguai, Abril 8, 9 e 10, 2013. **Anais...** 2013. p 33-46.

W3C. OWL 2 Web Ontology Language – Document Overview. W3C Recommendation. December 2012. Available at:< http://www.w3.org/TR/owl2-overview/>. Acess in: 10 dec. 2014.

WOHLIN, C. et al. **Experimentation in Software Engineering**: an introduction. USA: Kluwe Academic Publishers, 2000.

USCHOLD, Michael; KING, Martin. **Towards a methodology for building ontologies**. Edinburgh: Artificial Intelligence Applications Institute, University of Edinburgh, 1995.

ZAZWORKA, N. et al. A case study on effectively identifying technical debt, EASE 13. In: INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, 17., 2013. **Proceedings…** 2013.

APÊNDICE A – ARTIGOS IDENTIFICADOS NO MAPEAMENTO SISTEMÁTICO DA LITERATURA

Al Mamun, M.; Berger, C. & Hansson, J. (2014), Explicating, Understanding, and Managing Technical Debt from Self-Driving Miniature Car Projects, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 11-18.

Allman, E. (2012), Managing Technical Debt, Queue, 10(3).

Alzaghoul, E. & Bahsoon, R. (2013), CloudMTD: Using real options to manage technical debt in cloud-based service selection, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 55-62.

Alzaghoul, E. & Bahsoon, R. (2013), Economics-Driven Approach for Managing Technical Debt in Cloud-Based Architectures, in Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on, pp. 239-242.

Alzaghoul, E. & Bahsoon, R. (2014), Evaluating Technical Debt in Cloud-Based Architectures Using Real Options, in Software Engineering Conference (ASWEC), 2014 23rd Australian, pp. 1-10.

Barton, B. & Sterling, C. (2010), Manage Project Portfolios More Effectively by Including Software Debt in the Decision Process, Cutter IT Journal, Vol. 23, No. 10, 19-24.

Bohnet, J. & Dцllner, J. (2011), Monitoring code quality and development activity by software maps, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Brondum, J. & Zhu, L. (2012), Visualising architectural dependencies, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 7-14.

Brown, N.; Cai, Y.; Guo, Y.; Kazman, R.; Kim, M.; Kruchten, P.; Lim, E.; MacCormack, A.; Nord, R.; Ozkaya, I.; Sangwan, R.; Seaman, C.; Sullivan, K. & Zazworka, N. (2010), Managing technical debt in software-reliant systems, FoSER 10: Proceedings of the FSE/SDP workshop on Future of software engineering research.

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

Conroy, P. (2012), ‘‘Technical Debt: Where Are the Shareholders’ Interests?’’, Software, IEEE Computer Society, 29 (6), p. 88, November/December 2012.

Curtis, B.; Sappidi, J. & Szynkarski, A. (2012), Estimating the Principal of an Applications Technical Debt, Software, IEEE 29(6), 34-42.

Curtis, B.; Sappidi, J. & Szynkarski, A. (2012), Estimating the size, cost, and types of Technical Debt, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 49-53.

Davis, N., (2013), Driving Quality Improvement and Reducing Technical Debt with the Definition of Done, Syst., Pittsburgh, PA, USA, Agile Conference (AGILE), pp. 164-168.

de Groot, J.; Nugroho, A.; Back, T. & Visser, J. (2012), What is the value of your software?, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 37-44.

Ernst, N. (2012), On the role of requirements in understanding and managing technical debt, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 61-64.

Falessi, D.; Shaw, M.; Shull, F.; Mullen, K. & Keymind, M. (2013), Practical considerations, challenges, and requirements of tool-support for managing technical debt, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 16-19.

Fontana, F.; Ferme, V. & Spinelli, S. (2012), Investigating the impact of code smells debt on quality code evaluation, in, pp. 15-22.

Gat, I. & Heintz, J. D. (2011), From assessment to reduction: how cutter consortium helps rein in millions of dollars in technical debt, MTD'11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Giraldo, F.; España, S.; Pineda, M.; Giraldo, W. & Pastor, O. (2014), Integrating technical debt into MDE, CEUR Workshop Proceedings 1164, 145-152.

Gomes, R.; Siebra, C.; Tonin, G.; Cavalcanti, A.; Silva, F. Q. D.; Santos, A. L. & Marques, R. (2011), An extraction method to collect data on defects and effort evolution in a constantly modified system, MTD'11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Greening, Daniel R. (2013), Release Duration and Enterprise Agility, 46th Hawaii International Conference on System Sciences (HICSS), pp. 4835-4841.

Griffith, I.; Izurieta, C.; Taffahi, H. & Claudio, D. (2014), A Simulation Study of Practical Methods for Technical Debt Management in Agile Software Development, in Proceedings of the 2014 Winter Simulation Conference, IEEE Press, Piscataway, NJ, USA, pp. 1014--1025.

Griffith, I.; Reimanis, D.; Izurieta, C.; Codabux, Z.; Deo, A. & Williams, B. (2014), The Correspondence Between Software Quality Models and Technical Debt Estimation Approaches, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 19-26.

Guo, Y. & Seaman, C. (2011), A portfolio approach to technical debt management, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Guo, Y.; Seaman, C.; Gomes, R.; Cavalcanti, A.; Tonin, G.; da Silva, F.; Santos, A. L. M. & Siebra, C. (2011), Tracking technical debt - An exploratory case study, in Software Maintenance (ICSM), 2011 27th IEEE International Conference on, pp. 528-531.

Guo, Y.; Seaman, C.; Zazworka, N. & Shull, F. (2010), Domain-specific tailoring of code smells: an empirical study, ICSE 10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering.

Guo, Y.; Spínola, R. & Seaman, C. (2014), Exploring the costs of technical debt management – a case study, Empirical Software Engineering, 1-24.

Ho, J. & Ruhe, G. (2014), When-to-Release Decisions in Consideration of Technical Debt, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 31-34.

Holvitie, J. & Leppanen, V. (2013), DebtFlag: Technical debt management with a development environment integrated tool, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 20-27.

Holvitie, J. (2014), Software implementation knowledge management with technical debt and network analysis, in Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on, pp. 1-6.

Holvitie, J.; Laakso, M.-J.; Rajala, T.; Kaila, E. & Leppänen, V. (2013), The Role of Dependency Propagation in the Accumulation of Technical Debt for Software

Implementations, in Ákoss Kiss, ed., 13th Symposium on Programming Languages and Software Tools, University of Szeged, pp. 61–75.

Holvitie, J.; Leppanen, V. & Hyrynsalmi, S. (2014), Technical Debt and the Effect of Agile Software Development Practices on It - An Industry Practitioner Survey, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 35-42.

Izurieta, C.; Griffith, I.; Reimanis, D. & Luhr, R. (2013), On the Uncertainty of Technical Debt Measurements, in Information Science and Applications (ICISA), 2013 International Conference on, pp. 1-4.

Izurieta, C.; Vetro, A.; Zazworka, N.; Cai, Y.; Seaman, C. & Shull, F. (2012), Organizing the technical debt landscape, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 23-26.

Kaiser, M. & Royse, G. (2011), Selling the Investment to Pay Down Technical Debt: The Code Christmas Tree, in Agile Conference (AGILE), 2011, pp. 175-180.

Klinger, T.; Tarr, P.; Wagstrom, P. & Williams, C. (2011), An enterprise perspective on technical debt, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Krishna, V. & Basu, A. (2012), Minimizing Technical Debt: Developers viewpoint, in Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012), International Conference on, pp. 1-5.

Krishna, Vinay; Basu, A. (2013), Software Engineering Practices for Minimizing Technical Debt, Proceedings of the International Conference on Software Engineering Research and Practice (SERP), 1-5.

Kruchten, P.; Nord, R. & Ozkaya, I. (2012), Technical Debt: From Metaphor to Theory and Practice, Software, IEEE 29(6), 18-21.

Ktata, O. & Lévesque, G. (2010), Designing and implementing a measurement program for Scrum teams: what do agile developers really need and want?, C3S2E 10: Proceedings of the Third C* Conference on Computer Science and Software Engineering.

Letouzey, J. & Ilkiewicz, M. (2012), Managing Technical Debt with the SQALE Method, Software, IEEE 29(6), 44-51.

Letouzey, J.-L. (2012), The SQALE method for evaluating Technical Debt, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 31-36.

Li, Z.; Liang, P.; Avgeriou, P.; Guelfi, N. & Ampatzoglou, A. (2014), An Empirical Investigation of Modularity Metrics for Indicating Architectural Technical Debt, in Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures, ACM, New York, NY, USA, pp. 119-128.

Ligu, E.; Chatzigeorgiou, A; Chaikalis, T.; Ygeionomakis, N. (2013), Identification of Refused Bequest Code Smells, Dept. of Appl. Inf., Univ. of Macedonia, Thessaloniki, Greece, 29th IEEE International Conference on Software Maintenance (ICSM), pp. 392-395.

Lim, E.; Taksande, N. & Seaman, C. (2012), A Balancing Act: What Software Practitioners Have to Say about Technical Debt, Software, IEEE 29(6), 22-27.

Lindgren, M. (2012), Bridging the software quality gap, Department of Computing Science, Umea University.

Marinescu, R. (2012), Assessing technical debt by identifying design flaws in software systems, IBM Journal of Research and Development 56(5), 9:1-9:13.

Martini, A.; Bosch, J. & Chaudron, M. (2014), Architecture Technical Debt: Understanding Causes and a Qualitative Model, in Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on, pp. 85-92.

Mayr, A.; Plosch, R. & Korner, C. (2014), A Benchmarking-Based Model for Technical Debt Calculation, in Quality Software (QSIC), 2014 14th International Conference on, pp. 305-314.

McGregor, J. D.; Monteith, J. & Zhang, J. (2012), Technical debt aggregation in ecosystems, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 27-30.

Mo, R.; Garcia, J.; Cai, Y. & Medvidovic, N. (2013), Mapping architectural decay instances to dependency models, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 39-46.

Monteith, J. & McGregor, J. (2013), Exploring software supply chains from a technical debt perspective, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 32-38.

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

Morrison-Smith, S.; Dighans, S.; Daniels, T.; Marmon, C. & Izurieta, C. (2012), Technical debt reduction using a game theoretic competitive source control approach, in , pp. 157 - 162.

Nacimento, C.; Matalonga, S. & Rossa Hauck, J. (2014), Identifying technical debt cost factors in reflection activities of an agile projects, in Computing Conference (CLEI), 2014 XL Latin American, pp. 1-11.

Neill, C. & Laplante, P. (2006), Paying down design debt with strategic refactoring, Computer 39(12), 131-134.

Nord, R.; Ozkaya, I.; Kruchten, P. & Gonzalez-Rojas, M. (2012), In Search of a Metric for Managing Architectural Technical Debt, in Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on, pp. 91-100.

Nugroho, A.; Visser, J. & Kuipers, T. (2011), An empirical model of technical debt and interest, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

OConnor, D. (2010), Technical debt in semiconductor equipment: Its time to pay it down, Solid State Technology 53(7), 34-35.

Ojameruaye, B. & Bahsoon, R. (2014), Systematic Elaboration of Compliance Requirements Using Compliance Debt and Portfolio Theory, in Camille Salinesi & Inge van de Weerd, ed., Requirements Engineering: Foundation for Software Quality, Springer International Publishing, pp. 152-167.

Olbrich, S.K., Cruzes, D.S., and Sjoberg, D. I. K. (2010), "Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems," in Proceedings of the 2010 IEEE International Conference on Software Maintenance, ser. ICSM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. (Online). Available: http://dx.doi.org/10.1109/ICSM.2010.5609564

Potdar, A. & Shihab, E. (2014), An Exploratory Study on Self-Admitted Technical Debt, in Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on, pp. 91-100.

Power, K. (2013), Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 28-31.

Prause, C. R. (2011), Reputation-based self-management of software process artifact quality in consortium research projects, ESEC/FSE'11: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering.

Pugh, K. (2010), The Risks of Acceptance Test Debt, Cutter IT Journal, Vol. 23, No. 10, 23-29.

Ramasubbu, N. & Kemerer, C. (2013), Towards a model for optimizing technical debt in software products, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 51-54.

Ramasubbu, N. & Kemerer, C. (2014), Managing Technical Debt in Enterprise Software Packages, Software Engineering, IEEE Transactions on 40(8), 758-772.

Schmid, K. (2013), A formal approach to technical debt decision making, QoSA 13: Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures.

Schmid, K. (2013), On the limits of the technical debt metaphor some guidance on going beyond, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 63-66.

Schumacher, J.; Zazworka, N.; Shull, F.; Seaman, C. & Shaw, M. (2010), Building empirical support for automated code smell detection, ESEM'10: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement.

Seaman, C. & Guo, Y. (2011), Measuring and Monitoring Technical Debt, Advances in Computers 82, 25-46.

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

Shafer, A. C. (2010), Infrastructure Debt: Revisiting the Foundation, Cutter IT Journal, Vol. 23, No. 10, 36-40.

Shah, S.; Torchiano, M.; Vetro, A. & Morisio, M. (2013), Exploratory testing as a source of testing technical debt, IT Professional, pp. (99), 1-1.

Shah, S.; Torchiano, M.; Vetro, A. & Morisio, M. (2014), Exploratory Testing as a Source of Technical Debt, IT Professional 16(3), 44-51.

Sharma, T. (2012), Quantifying Quality of Software Design to Measure the Impact of Refactoring, in Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual, pp. 266-271.

Shull, F. (2011), Perfectionists in a World of Finite Resources, Software, IEEE 28(2), 4-6.

Siebra, C. A.; Tonin, G. S.; Silva, F. Q.; Oliveira, R. G.; Junior, A. L.; Miranda, R. C. & Santos, A. L. (2012), Managing technical debt in practice: an industrial report, ESEM 12: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement.

Siebra, C.; Cavalcanti, A.; Silva, F.; Santos, A. & Gouveia, T. (2014), Applying Metrics to Identify and Monitor Technical Debt Items during Software Evolution, in Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on, pp. 92-95.

Singh, V.; Snipes, W. & Kraft, N. (2014), A Framework for Estimating Interest on Technical Debt by Monitoring Developer Activity Related to Code Comprehension, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 27-30.

Skourletopoulos, G.; Bahsoon, R.; Mavromoustakis, C.; Mastorakis, G. & Pallis, E. (2014), Predicting and quantifying the technical debt in cloud software engineering, in Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on, pp. 36-40.

Snipes, W.; Robinson, B.; Guo, Y. & Seaman, C. (2012), Defining the decision factors for managing defects: A technical debt perspective, in, pp. 54-60.

Spínola, R.; Zazworka, N.; Vetro`, A.; Seaman, C. & Shull, F. (2013), Investigating technical debt folklore: Shedding some light on technical debt opinion, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 1-7.

Stephen Chin, Erik Huddleston, W. B. & Gat, I. (2010), The Economics of Technical Debt, Cutter IT Journal, Vol. 23, No. 10, 11-15.

Stochel, M.; Wawrowski, M. & Waskiel, J. (2012), Adaptive Agile Performance Modeling and Testing, in Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual, pp. 446-451.

Theodoropoulos, T.; Hofberg, M. & Kern, D. (2011), Technical debt from the stakeholder perspective, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Tom, E.; Aurum, A. & Vidgen, R. b. (2013), An exploration of technical debt, Journal of Systems and Software 86(6), 1498-1516.

Tom, E.; Aurum, A. & Vidgen, R. T. (2012), A Consolidated Understanding of Technical debt, in 20th European Conference on Information Systems, ECIS 2012, Barcelona, Spain, June 10-13, 2012, pp. 16.

Vetro, A. (2012), Using automatic static analysis to identify technical debt, ICSE 2012: Proceedings of the 2012 International Conference on Software Engineering.

Wang, P.; Yang, J.; Tan, L.; Kroeger, R. & David Morgenthaler, J. (2013), Generating precise dependencies for large software, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 47-50.

Weber, J.; Cleve, A.; Meurice, L. & Bermudez Ruiz, F. (2014), Managing Technical Debt in Database Schemas of Critical Software, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 43-46.

Wiklund, K.; Eldh, S.; Sundmark, D. & Lundqvist, K. (2012), Technical Debt in Test Automation, in Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, pp. 887-892.

Xuan, J.; Hu, Y. & He, J. (2012), Debt-prone bugs: Technical debt in software maintenance, International Journal of Advancements in Computing Technology 4(19), 453-461.

Yli-Huumo, J.; Maglyas, A. & Smolander, K. (2014), The Sources and Approaches to Management of Technical Debt: A Case Study of Two Product Lines in a Middle-Size Finnish Software Company, in 15th International Conference, PROFES 2014, Springer International Publishing, pp. 93-107.

Zazworka, N. b.; Vetro, A. c.; Izurieta, C.; Wong, S.; Cai, Y.; Seaman, C. g. & Shull, F. (2013), Comparing four approaches for technical debt identification, Software Quality Journal, 1-24.

Zazworka, N.; Seaman, C. & Shull, F. (2011), Prioritizing design debt investment opportunities, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Zazworka, N.; Shaw, M. A.; Shull, F. & Seaman, C. (2011), Investigating the impact of design debt on software quality, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt. New York, NY, USA: ACM, 2011, pp. 17–23. (Online). Available: http://doi.acm.org/- 10.1145/1985362.1985366

Zazworka, N.; Spinola, R. O.; Vetro, A.; Shull, F. & Seaman, C. (2013), A case study on effectively identifying technical debt, EASE 13: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering.

Zazworka, N.; Vetró, A.; Izurieta, C.; Wong, S.; Cai, Y.; Seaman, C. & Shull, F. (2014), Comparing four approaches for technical debt identification, Software Quality Journal 22(3), 403-426.

APÊNDICE B – OWL DA TAXONOMIA DE TIPOS DE DÍVIDA TÉCNICA

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.semanticweb.org/.nicollirios/ontologies/2015/10/untitled-ontology-17"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     ontologyIRI="http://www.semanticweb.org/.nicollirios/ontologies/2015/10/untitled-ontology-
17">
    <Prefix name="" IRI="http://www.semanticweb.org/.nicollirios/ontologies/2015/10/untitled-
ontology-17#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Prefix name="untitled-ontology-17"
IRI="http://www.semanticweb.org/.nicollirios/ontologies/2015/10/untitled-ontology-17#"/>
    <Declaration>
        <Class IRI="#Dívida_Técnica"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Dívida_de_Arquitetura"/>
```

```
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Automação__de_Testes"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Construção"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Código"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Defeito"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Documentação"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Infraestrutura"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Pessoas"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Processo"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Projeto"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Requisitos"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Serviço"/>
</Declaration>
```

```
<Declaration>
  <Class IRI="#Dívida_de_Teste"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Usabilidade"/>
</Declaration>
<Declaration>
  <Class IRI="#Dívida_de_Versionamento"/>
</Declaration>
<Declaration>
  <AnnotationProperty IRI="#indicador"/>
</Declaration>
<SubClassOf>
  <Class IRI="#Dívida_de_Arquitetura"/>
  <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
  <Class IRI="#Dívida_de_Automação__de_Testes"/>
  <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
  <Class IRI="#Dívida_de_Construção"/>
  <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
  <Class IRI="#Dívida_de_Código"/>
  <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
  <Class IRI="#Dívida_de_Defeito"/>
  <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
  <Class IRI="#Dívida_de_Documentação"/>
```

```
    <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Dívida_de_Infraestrutura"/>
    <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Dívida_de_Pessoas"/>
    <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Dívida_de_Processo"/>
    <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Dívida_de_Projeto"/>
    <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Dívida_de_Requisitos"/>
    <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Dívida_de_Serviço"/>
    <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Dívida_de_Teste"/>
    <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Dívida_de_Usabilidade"/>
    <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
```

```
<SubClassOf>
   <Class IRI="#Dívida_de_Versionamento"/>
   <Class IRI="#Dívida_Técnica"/>
</SubClassOf>
<DisjointClasses>
   <Class IRI="#Dívida_de_Arquitetura"/>
   <Class IRI="#Dívida_de_Automação__de_Testes"/>
   <Class IRI="#Dívida_de_Construção"/>
   <Class IRI="#Dívida_de_Código"/>
   <Class IRI="#Dívida_de_Defeito"/>
   <Class IRI="#Dívida_de_Documentação"/>
   <Class IRI="#Dívida_de_Infraestrutura"/>
   <Class IRI="#Dívida_de_Pessoas"/>
   <Class IRI="#Dívida_de_Processo"/>
   <Class IRI="#Dívida_de_Projeto"/>
   <Class IRI="#Dívida_de_Requisitos"/>
   <Class IRI="#Dívida_de_Serviço"/>
   <Class IRI="#Dívida_de_Teste"/>
   <Class IRI="#Dívida_de_Usabilidade"/>
</DisjointClasses>
<DisjointClasses>
   <Class IRI="#Dívida_de_Arquitetura"/>
   <Class IRI="#Dívida_de_Automação__de_Testes"/>
   <Class IRI="#Dívida_de_Construção"/>
   <Class IRI="#Dívida_de_Código"/>
   <Class IRI="#Dívida_de_Defeito"/>
   <Class IRI="#Dívida_de_Documentação"/>
   <Class IRI="#Dívida_de_Infraestrutura"/>
   <Class IRI="#Dívida_de_Pessoas"/>
   <Class IRI="#Dívida_de_Processo"/>
   <Class IRI="#Dívida_de_Projeto"/>
   <Class IRI="#Dívida_de_Requisitos"/>
   <Class IRI="#Dívida_de_Serviço"/>
   <Class IRI="#Dívida_de_Teste"/>
```

```
    <Class IRI="#Dívida_de_Usabilidade"/>
    <Class IRI="#Dívida_de_Versionamento"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#Dívida_de_Arquitetura"/>
    <Class IRI="#Dívida_de_Automação__de_Testes"/>
    <Class IRI="#Dívida_de_Construção"/>
    <Class IRI="#Dívida_de_Código"/>
    <Class IRI="#Dívida_de_Defeito"/>
    <Class IRI="#Dívida_de_Documentação"/>
    <Class IRI="#Dívida_de_Infraestrutura"/>
    <Class IRI="#Dívida_de_Pessoas"/>
    <Class IRI="#Dívida_de_Processo"/>
    <Class IRI="#Dívida_de_Projeto"/>
    <Class IRI="#Dívida_de_Requisitos"/>
    <Class IRI="#Dívida_de_Serviço"/>
    <Class IRI="#Dívida_de_Usabilidade"/>
    <Class IRI="#Dívida_de_Versionamento"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#Dívida_de_Arquitetura"/>
    <Class IRI="#Dívida_de_Automação__de_Testes"/>
    <Class IRI="#Dívida_de_Construção"/>
    <Class IRI="#Dívida_de_Código"/>
    <Class IRI="#Dívida_de_Defeito"/>
    <Class IRI="#Dívida_de_Documentação"/>
    <Class IRI="#Dívida_de_Infraestrutura"/>
    <Class IRI="#Dívida_de_Pessoas"/>
    <Class IRI="#Dívida_de_Processo"/>
    <Class IRI="#Dívida_de_Requisitos"/>
    <Class IRI="#Dívida_de_Serviço"/>
    <Class IRI="#Dívida_de_Teste"/>
    <Class IRI="#Dívida_de_Usabilidade"/>
    <Class IRI="#Dívida_de_Versionamento"/>
```

```
</DisjointClasses>
<DisjointClasses>
   <Class IRI="#Dívida_de_Arquitetura"/>
   <Class IRI="#Dívida_de_Automação__de_Testes"/>
   <Class IRI="#Dívida_de_Construção"/>
   <Class IRI="#Dívida_de_Código"/>
   <Class IRI="#Dívida_de_Defeito"/>
   <Class IRI="#Dívida_de_Documentação"/>
   <Class IRI="#Dívida_de_Pessoas"/>
   <Class IRI="#Dívida_de_Processo"/>
   <Class IRI="#Dívida_de_Projeto"/>
   <Class IRI="#Dívida_de_Requisitos"/>
   <Class IRI="#Dívida_de_Serviço"/>
   <Class IRI="#Dívida_de_Teste"/>
   <Class IRI="#Dívida_de_Usabilidade"/>
   <Class IRI="#Dívida_de_Versionamento"/>
</DisjointClasses>
<DisjointClasses>
   <Class IRI="#Dívida_de_Arquitetura"/>
   <Class IRI="#Dívida_de_Automação__de_Testes"/>
   <Class IRI="#Dívida_de_Construção"/>
   <Class IRI="#Dívida_de_Defeito"/>
   <Class IRI="#Dívida_de_Documentação"/>
   <Class IRI="#Dívida_de_Infraestrutura"/>
   <Class IRI="#Dívida_de_Pessoas"/>
   <Class IRI="#Dívida_de_Processo"/>
   <Class IRI="#Dívida_de_Projeto"/>
   <Class IRI="#Dívida_de_Requisitos"/>
   <Class IRI="#Dívida_de_Serviço"/>
   <Class IRI="#Dívida_de_Teste"/>
   <Class IRI="#Dívida_de_Usabilidade"/>
   <Class IRI="#Dívida_de_Versionamento"/>
</DisjointClasses>
<DisjointClasses>
```

```
    <Class IRI="#Dívida_de_Arquitetura"/>
    <Class IRI="#Dívida_de_Automação__de_Testes"/>
    <Class IRI="#Dívida_de_Código"/>
    <Class IRI="#Dívida_de_Defeito"/>
    <Class IRI="#Dívida_de_Infraestrutura"/>
    <Class IRI="#Dívida_de_Pessoas"/>
    <Class IRI="#Dívida_de_Processo"/>
    <Class IRI="#Dívida_de_Projeto"/>
    <Class IRI="#Dívida_de_Requisitos"/>
    <Class IRI="#Dívida_de_Serviço"/>
    <Class IRI="#Dívida_de_Teste"/>
    <Class IRI="#Dívida_de_Usabilidade"/>
    <Class IRI="#Dívida_de_Versionamento"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#Dívida_de_Arquitetura"/>
    <Class IRI="#Dívida_de_Construção"/>
    <Class IRI="#Dívida_de_Código"/>
    <Class IRI="#Dívida_de_Defeito"/>
    <Class IRI="#Dívida_de_Documentação"/>
    <Class IRI="#Dívida_de_Infraestrutura"/>
    <Class IRI="#Dívida_de_Pessoas"/>
    <Class IRI="#Dívida_de_Processo"/>
    <Class IRI="#Dívida_de_Projeto"/>
    <Class IRI="#Dívida_de_Requisitos"/>
    <Class IRI="#Dívida_de_Serviço"/>
    <Class IRI="#Dívida_de_Teste"/>
    <Class IRI="#Dívida_de_Usabilidade"/>
    <Class IRI="#Dívida_de_Versionamento"/>
</DisjointClasses>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Arquitetura</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">ACN/PWDR</Literal>
```

```
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Arquitetura</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Betweeness_Centrality</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Arquitetura</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Dependências_estruturais</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Arquitetura</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Questões_de_Arquitetura_de_Software</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Arquitetura</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Structural_Analysis</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Arquitetura</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Violação_de_modularidade</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Construção</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Dependências_estruturais</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Construção</IRI>
```

```
    <Literal datatypeIRI="&rdf;PlainLiteral">Questões_de_construção</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Código</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Algoritmo_lento</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Código</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Código_duplicado</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Código</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Código_sem_padrões</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Código</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Multithread_correctness</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Código</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Métricas_de_Código</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Código</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Questões_ASA</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
```

```
    <IRI>#Dívida_de_Defeito</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Defeitos_conhecidos_e_não_corrigidos</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Documentação</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Comentários_insuficientes_no_código</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Documentação</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Questões_de_documentação</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Acoplamento__intenso</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Análise_estrutural</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Classe_de_dados</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Classe_esquizofrênica</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
```

```
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Code_smell</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Código_duplicado</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Dados_aglomerados</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Dispersed_Coupling</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">God_class_(ou_classe_grande)</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Grime</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty IRI="#indicador"/>
    <IRI>#Dívida_de_Projeto</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">Método_brain</Literal>
</AnnotationAssertion>
```

```
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Projeto</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Métricas_de_código</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Projeto</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Problemas_no_design_de_software</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Projeto</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Questões_ASA</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Projeto</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Refused_Parent_Bequest</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Requisitos</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Requirement_Backlog_List</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Serviço</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Seleção/Substituição_de_serviço_web</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
   <AnnotationProperty IRI="#indicador"/>
   <IRI>#Dívida_de_Teste</IRI>
   <Literal datatypeIRI="&rdf;PlainLiteral">Defeitos conhecidos_e_não_corrigidos</Literal>
```

```
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty IRI="#indicador"/>
        <IRI>#Dívida_de_Teste</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">Testes_incompletos</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty IRI="#indicador"/>
        <IRI>#Dívida_de_Versionamento</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">Forks_desnecessários_no_código</Literal>
    </AnnotationAssertion>
</Ontology>


<!-- Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net -->
```

APÊNDICE C – INSTRUMENTOS UTILIZADOS NO *SURVEY*

C.1 Termo de Consentimento

Termos de Consentimento

As informações contidas neste formulário visam firmar acordo por escrito, mediante o qual o profissional autoriza sua participação no experimento, com pleno conhecimento da natureza dos procedimentos a que se submeterá para ser participante do estudo e com capacidade de livre arbítrio e sem qualquer coação. Esta participação é voluntária e o sujeito deste experimento tem a liberdade de retirar seu consentimento a qualquer momento e deixar de participar do estudo, sem qualquer prejuízo ao atendimento a que está sendo ou será submetido.

I – TEMA

Dívida Técnica. Manutenção de Software.

II – OBJETIVOS DO ESTUDO

Identificar (1) possíveis causas que levam a equipe de desenvolvimento a incorrerem em dívida técnica; (2) os motivos que levam a equipe de desenvolvimento a incorrerem em dívida ocorrem em cadeia (“causes” of TD come in chains); 3(3) se, na opinião dos entrevistados, prevenir a dívida teria sido mais caro do que incorrer a dívida (“prevention is more expensive than the debt”), do ponto de vista de estudantes e profissionais de computação no contexto de projetos de desenvolvimento de software.

III – INSTITUIÇÃO RESPONSÁVEL

Universidade Salvador.

IV – PESQUISADORES RESPONSÁVEIS

- Nicolli Rios, PPCOMP/UNIFACS
- Rodrigo Spínola, FPC - PPCOMP/UNIFACS

V - CONFIDENCIALIDADE

Toda informação coletada neste estudo é confidencial, e meu nome não será identificado em momento algum. Da mesma forma, me comprometo a manter sigilo das tarefas solicitadas e dos documentos apresentados e que fazem parte dos experimentos.

VI – CONSENTIMENTO

Eu,_____, certifico que, tendo lido as informações acima e suficientemente esclarecido de todos os itens, estou plenamente de acordo com a realização do experimento. Assim, eu autorizo a execução do trabalho de pesquisa exposto acima.

_____ , _____ de _____ de _____.

Assinatura do Participante: _____

C.2 Formulário de Caracterização

Caracterização do Participante

Nome _____ Nível (Ms.c/D.Sc.): _____

Experiência com desenvolvimento de software

Por favor, informe o número de anos de experiência relevante em desenvolvimento (E.g. “Eu trabalhei por 10 anos como programador na indústria”)

_____
_____
_____

Por favor, sobre desenvolvimento de software, indique o grau de sua experiência nesta seção seguindo a escala de 5 pontos abaixo:

1 = nenhum

2 = estudei em aula ou em livro

3 = pratiquei em 1 projeto em sala de aula

4 = usei em 1 projeto na indústria

5 = usei em vários projetos na indústria

- Experiência com arquitetura de software? 1 2 3 4 5
- Experiência com compilação ou integração contínua de software? 1 2 3 4 5
- Experiência com codificação de software? 1 2 3 4 5
- Experiência com design (projeto) do software? 1 2 3 4 5
- Experiência com acompanhamento e correção de defeitos do software? 1 2 3 4 5
- Experiência com documentação de software? 1 2 3 4 5
- Experiência com processos de software? 1 2 3 4 5
- Experiência com infraestrutura do software? 1 2 3 4 5
- Experiência com gestão da equipe de desenvolvimento do software? 1 2 3 4 5
- Experiência com requisitos de software? 1 2 3 4 5
- Experiência com software orientado a serviço? 1 2 3 4 5
- Experiência com teste de software? 1 2 3 4 5
- Experiência com automação de teste de software? 1 2 3 4 5