UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Nathan Manera Magalhães

Uma abordagem para a seleção de desenvolvedores baseada na reputação e esquecimento em projetos de software

Juiz de Fora
2022

Nathan Manera Magalhães

Uma abordagem para a seleção de desenvolvedores baseada na reputação e esquecimento em projetos de software

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. José Maria Nazar David

Coorientador: Prof. Dr. Marco Antônio Pereira Araújo

Juiz de Fora

2022

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Magalhães, Nathan Manera.
Uma abordagem para a seleção de desenvolvedores baseada na reputação e esquecimento em projetos de software / Nathan Manera Magalhães. -- 2022.
130 p.

Orientador: José Maria Nazar David
Coorientador: Marco Antônio Pereira Araújo
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2022.

1. Seleção de Desenvolvedores. 2. Reputação. 3. Esquecimento. 4. Manutenção de Software. I. David, José Maria Nazar, orient. II. Araújo, Marco Antônio Pereira, coorient. III. Título.

Nathan Manera Magalhães

Uma abordagem para a seleção de desenvolvedores baseada na reputação e esquecimento em projetos de software

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Aprovada em 03 de Março de 2022.

BANCA EXAMINADORA

Prof. Dr. José Maria Nazar David – Orientador

Universidade Federal de Juiz de Fora

Prof. Dr. Marco Antônio Pereira Araújo – Coorientador

Universidade Federal de Juiz de Fora

Prof. Dr. Mario Antonio Ribeiro Dantas

Universidade Federal de Juiz de Fora

Prof. Dr. Tássio Ferenzini Martins Sirqueira

Centro Universitário Academia – UniAcademia

Juiz de Fora, 09/02/2022.

AGRADECIMENTOS

Agradeço primeiramente a Deus, Senhor Criador dos Céus e da Terra. Pois quem o conhece, sabe que é dele que procedem todas as condições necessárias de sustento, de saúde e de capacitação para tornar possível uma conquista como essa. A Ele dou toda a honra, glória e louvor para sempre enquanto eu viver!

Aos meus orientadores, professores José Maria e Marco Antônio, pela confiança depositada em mim, pela perseverança e paciência e, também, por todas as orientações e aprendizados recebidos os quais guardarei por toda a vida.

À minha mãe Luciane e ao meu pai Carlos Alberto, por todo o apoio recebido nos momentos difíceis, por sempre me incentivarem na realização deste trabalho. E, também, por se alegrarem comigo nos momentos de conquista.

Aos meus familiares e amigos próximos, pelo incentivo e apoio recebido durante todo este tempo.

Aos demais professores e colegas do PGCC, que contribuíram de alguma forma para a concretização deste trabalho.

À Universidade Federal de Juiz de Fora e à CAPES pelo apoio financeiro do projeto.

“O coração do homem planeja o seu caminho, mas o SENHOR lhe dirige os passos.” Provérbios 16:9

RESUMO

Tarefas de manutenção são essenciais para manter o pleno funcionamento de um software durante seu ciclo de vida. Porém, conforme um software evolui, seu código-fonte tende a tornar-se cada vez mais complexo e de difícil manutenção, tornando-se mais propício a apresentar defeitos ao usuário final. Com isso, escolher desenvolvedores apropriados para as tarefas de manutenção de software torna-se uma atividade também complexa, com alto consumo de tempo e suscetível a falhas de decisão quando realizada manualmente. Pois, para que uma escolha de desenvolvedores aptos às tarefas seja adequada, devem-se conhecer bem as *expertises* de cada candidato. Também é importante conhecer o quanto esses desenvolvedores têm trabalhado nos conhecimentos (*expertises*) em tecnologia exigidos pelas tarefas, sendo isso averiguado através de suas contribuições em plataformas de apoio ao desenvolvimento de software, tais como o *GitHub*, *StackOverflow* e *TopCoder*. Através dessas contribuições históricas, é possível estabelecer a reputação que os desenvolvedores possuem nas *expertises* trabalhadas como também verificar se esses conhecimentos não estão em desuso por longo tempo (esquecimento). A solução proposta deste trabalho foi desenvolvida com o intuito de apoiar a seleção de desenvolvedores para tarefas de manutenção de software considerando reputação e esquecimento destes nos conhecimentos de tecnologias em projetos de software. Para cada tarefa de software são listadas sugestões de desenvolvedores considerando reputação e esquecimento nos conhecimentos exigidos pela tarefa. Uma avaliação foi realizada para verificar a equivalência dessas sugestões com as atribuições históricas de desenvolvedores ocorridas em tarefas de projetos de software. Foram encontradas evidências a favor do uso da reputação e do esquecimento para o apoio na seleção de desenvolvedores em tarefas de manutenção de software.

Palavras-chave: Seleção de Desenvolvedores. Reputação. Esquecimento. Manutenção de Software.

ABSTRACT

Maintenance tasks are essential to keep the software running during its life cycle. However, as software evolves, its source code tends to become increasingly complex and difficult to maintain, making it more likely to present defects to the end-user. As a result, choosing appropriate developers for software maintenance tasks becomes a complex, time-consuming activity and susceptible to decision failures when performed manually. Because, for suitable developers for the functions, each candidate's expertise must be well known. It is also important to know how much these developers have worked on the knowledge (expertise) in technology required by the tasks, which is verified through their contributions to platforms that support software development, such as GitHub, StackOverflow, and TopCoder. These historical contributions make it possible to establish developers' reputation in their expertise and verify that this knowledge is not in disuse for a long time (forgetfulness). The proposed solution of this work was developed in order to support the selection of developers for software maintenance tasks considering their reputation and forgetfulness in the knowledge of technologies in software projects. For each software task, developers' suggestions are listed, considering reputation and forgetfulness in the knowledge required by the task. An evaluation was conducted to verify these suggestions' equivalence with the developers' historical attributions in software project tasks. Favorable evidence of the use of reputation and forgetfulness was found to support the selection of developers in software maintenance tasks.

Keywords: Developer Selection. Reputation. Forgetfulness. Software Maintenance.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta o contexto e a motivação relacionados ao problema abordado nesta dissertação, o enfoque da solução elaborada, os objetivos almejados no presente trabalho e a sua organização.

1.1 CONTEXTO E MOTIVAÇÃO

Tarefas de manutenção são essenciais para manter o pleno funcionamento de um software durante seu ciclo de vida. Como exemplos tem-se a adição de novas funcionalidades no software com tarefas do tipo, adaptativa e evolutiva tais quais requisições de mudança (*change request*), ou então por meio de tarefas do tipo corretiva para correção de defeitos expostos pelos relatórios de erros (*bug report*).

A fase de manutenção compõe a maior parte do ciclo de vida de um software. Segundo ERLIKH (2000), esta fase despende cerca de 90% de todo o esforço do desenvolvimento de um software. De acordo com LEHMAN (1980), conforme um software evolui, seu código fonte tende a tornar-se cada vez mais complexo e, conforme diz PFLEEGER (2004), um software que apresenta dificuldades em sua manutenção se torna mais propício a apresentar defeitos ao usuário final.

Segundo GOYAL e SARDANA (2017), no passado, os projetos de software eram pequenos em tamanho e a contagem de *bugs* era mínima. Com isso, praticava-se a escolha manual dos desenvolvedores mais aptos a corrigirem os poucos problemas que eram relatados. Porém, com o passar do tempo, os projetos de software aumentaram em escala, tamanho e complexidade diante da variedade de tecnologias que surgiram nos últimos anos. A manutenção de software tornou-se desafiadora devido ao crescente número de problemas (*bugs*) relatados em programas de software complexos e de grande escala.

1.2 PROBLEMA

Escolher desenvolvedores apropriados para cada tarefa de manutenção de software tornou-se então uma atividade também complexa, com alto consumo de tempo e suscetível a falhas de decisão quando realizada manualmente (GOYAL e SARDANA, 2017). Pois para que uma escolha de desenvolvedores aptos às tarefas seja adequada, devem-se conhecer bem quais são as *expertises* de cada candidato. Também é importante saber conciliar o tempo disponível deste com o das exigências da tarefa de manutenção, para que não haja sobrecarga de atividades e culmine em prejuízos. Esses prejuízos podem estar relacionados à qualidade da atividade de manutenção e ao tempo para a entrega dessa atividade. Portanto, ao decidir

pela escolha de desenvolvedores, coordenadores das atividades de manutenção devem estar cientes dos especialistas aptos à tarefa. Como resultado, atribuições de desenvolvedores desprovidos dos conhecimentos exigidos pelas tarefas podem ser minimizadas. Esse despreparo tem por justificativas o desenvolvedor: não possuir os conhecimentos necessários para a tarefa (reputação baixa em relação às *expertises*); estar ocupado com outras atividades; não estar mais apto com os conhecimentos exigidos de uma tarefa devido ao desuso destes conhecimentos (esquecimento).

1.3 ENFOQUE DA SOLUÇÃO

MIGUEL *et al*. (2016) e LÉLIS (2017) ao abordarem o apoio à manutenção de software propuseram, respectivamente, soluções que preveem o esforço de tempo a ser gasto em uma nova tarefa e que enfocam na manutenção colaborativa associada à reputação dos desenvolvedores geograficamente distribuídos. Esses trabalhos fornecem suporte para elaborar uma abordagem de apoio à tomada de decisão na escolha de um desenvolvedor apto a uma tarefa. Isto para sugerir aqueles com maior preparo na referida tarefa, com base na reputação dos candidatos ante suas contribuições anteriores. Porém, os autores não tratam da aplicação de suas propostas em bases abertas (*open source*) e não consideram penalizar relevância das contribuições antigas (fator esquecimento).

Havendo dificuldade de encontrar desenvolvedores com boa reputação em *expertises* num contexto restrito (desenvolvedores de uma empresa, por exemplo), recorre-se à busca destes em um contexto de maior abrangência através do desenvolvimento global de software (DGS). A abordagem de DGS disponibiliza acesso a uma ampla quantidade de programadores de diversas capacitações geograficamente distribuídos em diversas localidades mundo afora. Tal gama de pessoas distribuídas que contribuem para uma atividade é conhecida por *crowdsourcing* (LATOZA e VAN DER HOEK, 2016). Em relação à utilização de *crowdsourcing* para o DGS, DE NEIRA *et al*. (2018) destacam a importância do recrutamento daqueles que são “hiperespecialistas” e que se mostram dispostos a cumprirem desafios de tarefas para ganharem prêmios em dinheiro, notoriedade e até mesmo oportunidades profissionais.

Um estudo secundário, realizado por GOYAL e SARDANA (2017), apresenta diversas soluções que foram propostas na literatura com intuito de tornar a triagem de desenvolvedores para tarefas uma atividade automatizada ou semiautomática, e então minimizar a necessidade de triagem manual. Dentre estas, algumas soluções como as de SHOKRIPOUR *et al*. (2015) e MOHAN *et al*. (2016) consideram aspectos do esquecimento ao proporem abordagens que

penalizam contribuições antigas e assim buscam evitar desenvolvedores inativos nas *expertises*. E ainda, aplicam suas soluções em desenvolvimento *open source*. Entretanto, os aspectos da reputação não foram considerados, nem tampouco a utilização de uma diversidade de repositórios abertos.

Diante do contexto apresentado, busca-se o enfoque da solução com base na existência de abordagens que visam verificar o histórico de contribuições dos desenvolvedores, seja em *commits* de código fonte em repositórios no *GitHub*[1] (YANG *et al*., 2016); (SUN *et al*., 2018); tarefas de manutenção e desenvolvimento concluídas no *TopCoder*[2] (MAO *et al*., 2015); (ZHU *et al*., 2015); e/ou perguntas respondidas no *StackOverflow*[3] (BADASHIAN, 2016); (BADASHIAN *et al*., 2016); (WEI *et al*., 2018). Isso com o intuito de verificar quais os desenvolvedores que já possuem experiência nos domínios de conhecimento das tecnologias exigidas para novas contribuições no contexto de desenvolvimento global de software. A justificativa para a escolha dessas plataformas se resume em: o *GitHub* ser a plataforma que vem sendo cada vez mais utilizada pelos desenvolvedores para compartilhamento de código fonte (SCHETTINO *et al*., 2019), o *StackOverflow* ser uma plataforma popular de perguntas e respostas no contexto do desenvolvimento de software (WEI *et al*., 2018), e o *TopCoder* ser a principal plataforma de apoio ao desenvolvimento de software por *crowdsourcing* (DE NEIRA *et al*., 2018).

1.4 OBJETIVO

Este trabalho tem por objetivo tratar a indicação de desenvolvedores devidamente preparados para as tarefas de manutenção. A solução proposta constitui-se de um serviço que se integra com plataformas de apoio ao desenvolvimento global de software. O trabalho propõe tratar a reputação e o esquecimento dentro do contexto de bases abertas (*open source*) e com isso minimizar o esforço, e tempo, despendidos com a busca por desenvolvedores capacitados para tarefas de correção e aperfeiçoamento de software. Sendo assim, com as atribuições corretas entre tarefas e desenvolvedores, busca-se a redução dos erros de manutenções.

A identificação dos perfis dos desenvolvedores ocorre com base nos conhecimentos aplicados por estes nas tarefas que lhes foram anteriormente atribuídas e finalizadas com sucesso. É também intento da abordagem considerar contribuições externas em plataformas de apoio ao desenvolvimento de software para enriquecer o perfil com informações de

[1] https://github.com/
[2] https://www.topcoder.com/
[3] https://stackoverflow.com/

expertise. Ao identificar os desenvolvedores devidamente capacitados, estes podem ser apresentados como indicados para uma tarefa que exija seus conhecimentos. É também proposta do trabalho sugerir tarefas relevantes para desenvolvedores aptos. A decisão final de escolha cabe ao responsável pelas atribuições e ao próprio desenvolvedor em atender às sugestões de tarefas relevantes que lhes são apresentadas.

Seguindo o modelo GQM (*Goal* / *Question* / *Metric*) proposto por BASILI *et al*. (1994) o escopo do trabalho foi formulado como:

"**Apoiar** (*purpose*) a **seleção** (*object* (*process*)) de desenvolvedores para tarefas de manutenção de software, considerando aspectos de reputação e esquecimento (*issue*) do **ponto de vista** (*viewpoint*) do gerente de projeto."

Por meio deste objetivo definido foi elaborada a seguinte pergunta de pesquisa:

"Como o suporte à reputação e ao esquecimento apoiam a seleção de desenvolvedores para tarefas de manutenção de software?".

1.5 ORGANIZAÇÃO

Este trabalho está organizado em cinco capítulos, incluindo a Introdução apresentada como Capítulo 1. O Capítulo 2 faz menção aos fundamentos teóricos dos conceitos abordados no presente trabalho, bem como os trabalhos relacionados. O Capítulo 3 disserta sobre a proposta desenvolvida para a indicação de desenvolvedores em tarefas de manutenção de software, detalhando os aspectos conceituais como também a sua implementação. O Capítulo 4 tem por objetivo avaliar o funcionamento da abordagem desenvolvida e também apresentar as ameaças à validade desta solução. E, por fim, o Capítulo 5 apresenta uma síntese com as contribuições do trabalho como também perspectivas futuras.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos relacionados com a proposta deste trabalho. São apresentadas considerações sobre, reputação, esquecimento e crowdsourcing para fundamentar a abordagem proposta. Também estão incluídos os trabalhos relacionados à solução proposta neste trabalho.

2.1 REPUTAÇÃO

A reputação é uma forma importante de estabelecer confiança de um desenvolvedor de software para com outros colaboradores em equipes distribuídas globalmente. É uma medida que determina o nível de confiabilidade com base nas classificações de membros da comunidade do qual o desenvolvedor busca fazer parte (LÉLIS, 2017). Um desenvolvedor com boa reputação para com uma equipe é o que transmite confiança mediante as suas habilidades (*expertises*) que possui. ARTZ e GIL (2007) definem seis dimensões que descrevem a confiança: (*Target*) entidade cuja confiança é alvejada; (*Representation*) assinaturas digitais e *tokens* são exemplos de representação da confiança; (*Method*) a verificação do histórico de interações passadas é um exemplo de método para determinar a confiança por meio da reputação; (*Management*) um único serviço pode gerenciar a confiança atuando como um terceiro confiável para mediar o estabelecimento da confiança entre duas entidades desconhecidas; (*Computation*) abordagens podem contabilizar a confiança por meio de valores fixos (confia, não confia, neutro) ou por meio de um intervalo numérico contínuo; e (*Purpose*) a confiança pode ter como exemplo de propósito a proteção dos dados.

O estabelecimento de confiança em equipes de desenvolvimento global de software tem sido explorado na literatura através de estudos empíricos, modelos de confiança e abordagens com implementação de ferramentas.

AL-ANI *et al*. (2011) investigaram por meio de um estudo empírico quais os fatores considerados na busca por alguém que detém determinado conhecimento ou *expertise*. Os resultados obtidos sugerem que esses fatores são determinados pela maneira como uma pessoa articula e compartilha conhecimento. A área da *expertise* e a vontade da pessoa de compartilhar o conhecimento são os fatores considerados. O trabalho também buscou conhecer quais são os critérios considerados para a aceitação do conhecimento compartilhado de pessoa para pessoa. Um dos resultados obtidos indica que as pessoas não consideram como critério a localização geográfica da pessoa. Em vez disso as pessoas buscam considerar qual especialista seria mais confiável no conhecimento que este alega possuir (ou seja, quem seria capaz de articular explicitamente o conhecimento). AL-ANI e REDMILES (2009), porém,

observaram que a confiança é mais difícil de ser estabelecida entre membros de uma equipe globalmente distribuída quando estes são de diferentes culturas, línguas e fusos horários. Também verificaram que quanto maior for uma equipe, mais difícil é o estabelecimento de confiança entre os membros desta equipe.

O uso de *frameworks* para ajuda no estabelecimento da confiança é explorado pelos trabalhos de KINTAB *et al*. (2014) e YE *et al*. (2007). Ambos propõem *frameworks* para ajudar o desenvolvedor a encontrar outros desenvolvedores que possam auxiliá-lo no trabalho da programação em trechos de códigos do qual estejam tendo dificuldades. O uso de heurísticas técnicas e sociais medem os aspectos sociais dos desenvolvedores, incluindo noções de confiança, reputação e utilidade. No contexto técnico, uma abordagem desse tipo busca por desenvolvedores que tenham trabalhado em trechos de código fonte que são semelhantes com os do que o desenvolvedor está buscando ajuda. Dessa forma, a abordagem de YE *et al*. (2007) elabora uma rede de informações entrelaçadas entre código-fonte, documentação e os programadores que os constituíram. Considerando o princípio de que um programador dono de certo código-fonte ou documento é o que detém as *expertises* relacionadas; o respectivo *framework* busca interligar programadores que têm dúvidas sobre determinada *expertise* com aqueles que estão aptos a sanarem essas dúvidas. A abordagem de KINTAB *et al*. (2014) faz o mesmo ao abordar o contexto social para análise das interações passadas entre os desenvolvedores expertos com o desenvolvedor que tem dúvida.

CALEFATO *et al*. (2017) elaboraram um estudo que analisa como a propensão à confiança afeta o sucesso das colaborações em um projeto distribuído. De acordo com o trabalho, propensão à confiança refere-se à tendência geral de um indivíduo em perceber os outros indivíduos como sendo confiáveis. Os resultados principais do estudo atestam que a propensão dos desenvolvedores em confiar ocorre de forma mais geral com base em sua personalidade. Isso inclusive repercute na atribuição de lideranças, conforme diz AL-ANI e REDMILES (2009) ao observar que a confiança de líder é concedida mais prontamente a um membro de equipe autoritário caracterizado por qualidades de liderança dentro de uma equipe distribuída. No contexto do desenvolvimento de software, CALEFATO *et al*. (2017) encontraram evidências de que a propensão de se perceber outros desenvolvedores como confiáveis é um traço de personalidade estável que varia em cada indivíduo. Portanto, desenvolvedores que tendem a confiarem facilmente estão mais propensos em aceitar contribuições externas de outros desenvolvedores para seus projetos.

SOTO *et al*. (2017) propuseram um modelo de confiança para minimizar o efeito da desconfiança entre pessoas que estão isoladas geograficamente. Este modelo utiliza uma

arquitetura multi-agente que gerencia automaticamente o modelo de confiança estabelecido para uma comunidade dessas pessoas de locais diferentes. Segundo os autores o modelo de confiança e a arquitetura multi-agente são úteis na maioria das comunidades on-line, mesmo quando se trata de uma comunidade heterogênea, pois o modelo de confiança considera o problema individual de cada pessoa através de um respectivo agente. Os autores também apresentam uma ferramenta que recomenda fontes de conhecimento e documentação dos quais sejam confiáveis. A contribuição da modelagem de confiança (*trust model*) apoia também a identificação de pessoas especialistas em alguma *expertise*, considerando que estas possuem altos valores de confiança dos quais lhe foram atribuídos ao longo das interações com outros desenvolvedores.

ZHOU *et al*. (2018) observou através de um estudo empírico que no contexto do desenvolvimento de software, as competências técnicas de um desenvolvedor são vitais para a escrita de códigos fonte, como também para a qualidade destes. Nas competências sociais, é vital com que um desenvolvedor tenha a habilidade de colaborar em equipe, de gerenciar um projeto de software e de estar empenhado (motivado) em querer contribuir com o desenvolvimento. Os dados do estudo foram obtidos de projetos públicos do *GitHub* e da plataforma *StackOverflow*. Os resultados apresentados pelos autores indicam que fatores associados à proficiência em colaboração são os mais significantes relacionados com a habilidade técnica da escrita de códigos fonte. E esses fatores associados a esta competência também estão fortemente relacionados com a qualidade do trabalho de um desenvolvedor. Porém foi observado também que existe uma lacuna de associação forte entre competências técnicas e sociais destes.

TAGLIAFERRI *et al*. (2018) apresentam um conjunto taxonômico de modelos computacionais para com a confiança na área da computação. Os autores buscam prover um guia sobre o assunto de modo a facilitar a compreensão das muitas terminologias técnicas presentes na literatura. Após analisarem cinco modelos computacionais de confiança, os autores identificaram quatro características principais descritivas da confiança. Tem-se a **confiança relacional**, do qual diz respeito em sempre haver alguém para confiar (*trustor*) e alguém para ser confiado (*trustee*). A **confiança subjetiva** depende do valor atribuído por uma entidade avaliativa, e diz que confiança é sempre algo pessoal enquanto que reputação é a expressão de uma opinião geral compartilhada por alguém. A **confiança mensurável** marca a diferença entre os conceitos tradicionais de confiança com os conceitos de confiança na computação. Os modelos destes variam conforme a natureza da mensurabilidade e da forma em que a confiança é computada seja, qualitativamente (utilizando palavras) ou

quantitativamente (utilizando números). E por fim, tem-se a **confiança dependente de contexto** do qual, por exemplo, corresponde a alguém ser confiável no contexto da programação orientada a objetos por este ter conhecimentos práticos desta *expertise*, porém não ser confiável no contexto de segurança da informação. É importante ressaltar que o trabalho faz uma observação ao mencionar que confiança e reputação são conceitos distintos entre si.

HACKER *et al*. (2019) realizaram uma revisão sistemática teórica sobre confiança em equipes virtuais. Com os trabalhos encontrados, os autores desenvolveram um modelo integral de confiança em equipes virtuais, e estes também fornecem pressupostos teóricos sobre o comportamento da confiança de equipe quando esta se encontra virtualizada. O modelo integral de confiança apresentado incorpora três categorias amplas de fatores que contribuem para o estabelecimento de confiança (*trust*): **antecedentes**, **consequências** e **moderadores**. Os fatores que antecedem o estabelecimento de confiança são divididos em: atributos de *expertises* dos membros da equipe, estruturação da equipe, gerenciamento da equipe e relações interpessoais dos membros da equipe. A consequência da confiança foi observada e classificada em categorias de fatores que condizem com o desempenho do grupo relacionado com o contexto de trabalho, com o desenvolvimento e manutenção do grupo como um sistema, e com as formas em que um indivíduo é incluído no grupo. Dentre os fatores que antecedem a confiança e os fatores de consequências da confiança, têm-se os fatores de moderação da confiança em um grupo. Estes são importantes para como que a confiança seja estabelecida e traduzida em resultados, dos quais são: cultura, tipo de tarefa, diversidade funcional, e interdependência de tarefas. Alguns dos artigos buscados pelos autores consideram a virtualização como um dos fatores de moderação. Segundo os autores, o apoio de tecnologias de alta sincronia é importante para que haja uma mudança de confiança rápida para uma confiança mais aprofundada entre os membros de um grupo. A confiança rápida (*swift trust*), segundo MEYERSON *et al*. (1996), é uma confiança temporária resultante de quando os membros do grupo conseguem gerir entre si os problemas internos de vulnerabilidade, incertezas, riscos e expectativas. Ao evoluir para uma confiança mais aprofundada torna-se uma condição essencial para que um grupo de pessoas se torne uma equipe bem estabelecida. Pois na medida em que os membros troquem conhecimentos uns com os outros, estes obterão confiança mais sólida entre si de modo que eventuais falhas de comunicação possam ser resolvidas rapidamente. Equipes com confiança fortemente coesa e com líderes inspiradores e participativos tendem a produzir melhor os produtos e proporcionar maior rendimento de seus membros. À medida que o ambiente de trabalho muda, os gerentes

devem considerar o impacto das tendências no desenvolvimento da confiança em equipes virtuais.

2.2 ESQUECIMENTO

Em uma abordagem que busca identificar *expertises* de um desenvolvedor com base em suas contribuições passadas, um fator importante a ser considerado é a não utilização do conhecimento por algum período de tempo. A esse fator denomina-se esquecimento ou obsolescência do conhecimento (CRUZ NAVEA, 2017).

Um desenvolvedor é considerado apto às exigências de uma tarefa quando este possui o conhecimento (*expertise*) necessário para resolvê-la. Porém, segundo SHOKRIPOUR *et al*. (2015), um conhecimento em desuso por longo tempo pode ocasionar em dificuldades na sua utilização por este estar em esquecimento.

De acordo com MOHAN *et al*. (2016), quanto maior for o período de desuso da *expertise* maior o esquecimento aplicado, culminando assim em uma menor relevância do desenvolvedor para uma tarefa que exige esta *expertise*. KHATUN e SAKIB (2016) observam que quanto mais recente um conhecimento tenha sido utilizado, mais apto o desenvolvedor estará na tarefa da respectiva *expertise*, indicando assim maior familiaridade com o conhecimento.

Em abordagens que recomendam desenvolvedores com base em suas contribuições históricas, considerar esse fato da *expertise* ociosa é fundamental para efetuar indicações mais condizentes com o atual estado de conhecimento do desenvolvedor. Segundo WEI *et al*. (2018), isso é para culminar numa escolha mais apurada, de modo a minimizar a escolha de desenvolvedores que, por longo tempo, não tenham utilizado determinada *expertise*.

O esquecimento é um fenômeno natural na mente humana do qual, aprendizados antigos são difíceis de serem relembrados conforme não utilizados. Esse processo é estudado por psicólogos desde o trabalho pioneiro de EBBINGHAUS (1885) *apud* (KRÜGER *et al*., 2018), do qual, posteriormente, derivou-se uma equação matemática (2.1) expressando o esquecimento.

$$R = e^{-\frac{t}{s}} \tag{2.1}$$

A equação (2.1) possui natureza exponencial, do qual representa o esquecimento simulado através do decréscimo da Retenção de Memória (R) em função do Tempo (t) decorrido, em dias, sobre uma Força de Memória (s) aplicada. A fórmula faz uso do Número de Euler (e) também conhecido como Constante de Euler. A Força de Memória (s) expressa o quanto de memória é retida ao longo do tempo. Quanto maior esse valor, menor será o

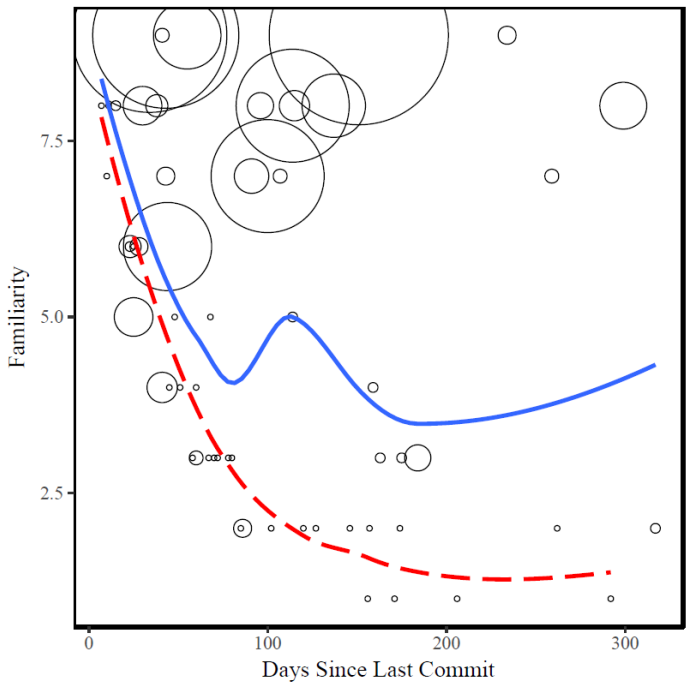
decaimento no decorrer do tempo e assim uma memória estará retida por mais tempo. O valor obtido pela fórmula varia entre 0 e 1 para $t \geq 0$, em que 1 indica retenção máxima da memória, enquanto 0 indica esquecimento total.

No contexto da engenharia de software, existem trabalhos que utilizam a respectiva fórmula para aprimoramento na identificação de programadores aptos em tarefas de manutenção de software (HATTORI *et al*., 2012); (KRÜGER *et al*., 2018); (YANG *et al*., 2018). A equação é utilizada na penalização das contribuições antigas, de modo a se dar maior relevância para os desenvolvedores que trabalharam mais recentemente com as *expertises* exigidas.

Dentre estes trabalhos destaca-se o de KRÜGER *et al*. (2018). Nele é elaborado um estudo empírico para atestar a importância de se considerar que um desenvolvedor perde a familiaridade com o código fonte mediante inatividade. Como resultado, os autores mostram o quanto de esquecimento ocorre em função do tempo. Ainda, segundo os autores, os resultados obtidos podem apoiar abordagens de engenharia de software na questão de se alocar desenvolvedores para tarefas em que estes sejam mais eficientes, de modo a identificar aqueles com mais preparo para certas *expertises*. Entretanto os autores não tratam de verificar os efeitos do esquecimento nos valores de reputação dos desenvolvedores. A fórmula de *Ebbinghaus* (2.1) é utilizada no respectivo trabalho e embora possa atestar a familiaridade de um desenvolvedor ante seu código fonte, esta é limitada em aplicabilidade para um único registro de interação (contribuição) realizada.

A Figura 2.1 ilustra o estudo empírico realizado com 60 desenvolvedores participantes que declaram por si próprios os valores de familiaridade que julgavam possuir sobre seus próprios códigos fonte. Nesse gráfico são exibidos os valores da familiaridade declarada dos participantes em função do tempo decorrido desde o último *commit* (contribuição) realizado sobre o arquivo de código fonte. Cada círculo representa um único desenvolvedor com as variações de tamanho relativos à quantidade de *commits* efetuados sobre o arquivo de código fonte previamente selecionado. A linha contínua de cor azul representa a média dos valores de familiaridade declarada pelos respectivos desenvolvedores, enquanto que a linha tracejada vermelha representa a média desses valores para os que haviam efetuado um único *commit* apenas. Considerando que todos os participantes possuam uma mesma força de memória (s) e que não haja outros fatores influenciando, é de se esperar que a variação da familiaridade seja consistente com a curva da fórmula de *Ebbinghaus*.

Figura 2.1 – Familiaridade dos participantes com um código fonte.



Fonte: KRÜGER *et al.* (2018)

Porém, observa-se na Figura 2.1 que em arquivos com mais de um *commit* efetuado pelo respectivo participante (linha contínua azul), há um aumento dos valores de familiaridade após cerca de 100 dias. Isso também é evidenciado pela concentração de círculos maiores, vinculados a uma familiaridade acima de 50%, e estes na mesma faixa de tempo do ápice registrado. Nota-se também um aumento gradual da média de familiaridade a partir de 200 dias após o último *commit* e coincidente com um círculo, no canto superior direito, representando um arquivo com mais de uma contribuição. Os autores do estudo identificaram, com isso, que múltiplas interações dos desenvolvedores com seus códigos fonte (através dos *commits*), encontram-se fortemente correlacionados com uma alta familiaridade. Além do mais, segundo os autores, foi computada a média dos valores da Força de Memória (s) aplicados na curva de esquecimento sobre as respostas dos participantes. O resultado observado na linha tracejada vermelha condiz com a curva de decaimento gerada pela fórmula de *Ebbinghaus*. E com isso implicaram na conclusão de que essa curva só pode ser aplicada para registros de apenas um único *commit*, isto é, uma única interação do desenvolvedor com o código fonte. Mediante o estudo realizado, os autores definiram um valor de ($s = 65$) para a equação apresentar uma curva de esquecimento do qual seja mais condizente com a realidade contextual do desenvolvimento de software. Os autores, no entanto, atestam a necessidade de novos estudos para validação deste valor estipulado.

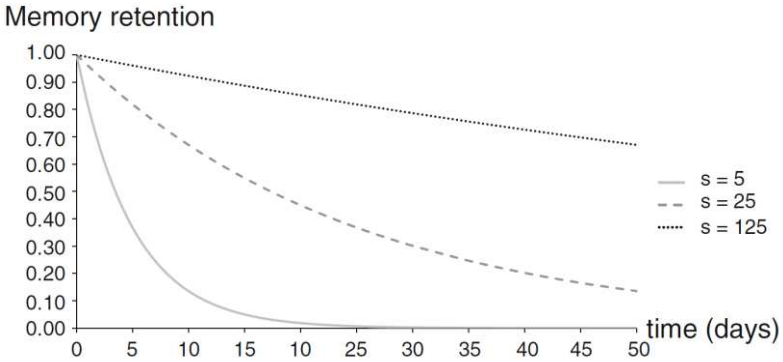
Conforme observado no contexto de desenvolvimento de software por KRÜGER *et al*. (2018), uma adaptação se faz necessária para o uso da respectiva equação, no caso de se considerar registro de mais contribuições do desenvolvedor no código ao longo do tempo. Pois havendo novas interações, o desenvolvedor recupera a familiaridade para esse código, e então os valores obtidos divergem dos valores previstos na curva de esquecimento. Uma opção é usar essa fórmula em cada uma das interações e obter o valor ponderado da soma de seus valores obtidos, tal como feito a seguir por HATTORI *et al*. (2012) e YANG *et al*. (2018).

HATTORI *et al*. (2012) consideram em sua abordagem o fator esquecimento utilizando a fórmula de *Ebbinghaus* (2.1) dentro do contexto do desenvolvimento de software. Este trabalho busca atestar a familiaridade de desenvolvedores com cada artefato de um sistema de software para determinar quem será o proprietário do código (*code ownership*). Este conceito informa a quantidade de conhecimento acumulado por cada desenvolvedor nos artefatos de um sistema de software, de modo a identificar os mais familiarizados com cada um deles. Um refinamento é definido nessa medição para contabilizar a frequência de edições que os desenvolvedores realizaram em cada arquivo, sendo também considerada a noção de perda de memória nessa atribuição de proprietário do código. Isso com o objetivo de mitigar a suposição ingênua dos quais desenvolvedores sempre se lembrariam do código fonte, mesmo após algum tempo não trabalhando neste. Pois, um desenvolvedor que realizou a maioria das edições em um arquivo de código, mas por um longo período posterior não tenha mais feito modificações, este terá gradualmente perdido a familiaridade para com o código. Segundo os autores, o desenvolvedor que realiza mudança mais recentemente se torna o mais experiente para com o código, mesmo que este não tenha realizado tantas edições como o programador anterior. Para consideração do esquecimento, os autores utilizam a fórmula de *Ebbinghaus* (2.1) de modo a penalizar as interações passadas em função do tempo decorrido. No entanto, diferentemente da abordagem de KRÜGER *et al*. (2018), o trabalho não propôs um tamanho ideal para a força de memória aplicada (s), mas sim buscou explicar como este pode ser calibrado de acordo com o cenário aplicado.

Na intenção de estipular valores gerais para (s) e considerando que esse valor deve ser variável, os autores selecionaram os números 5, 25 e 125 conforme a Figura 2.2 mostra. Esses são valores potências de 5 escolhidos devido à função de *Ebbinghaus* ser de natureza exponencial, e com esses três ser possível cobrir uma variabilidade razoável da retenção de memória. Com base nos resultados obtidos do estudo realizado, os autores concluíram que a Força de Memória (s) é um valor subjetivo e que depende de muitas variáveis de cada projeto.

Considerando isto, o estudo evidenciou a possibilidade de se estabelecer um conjunto de valores ideais para (*s*) em cada projeto de software aplicado, porém também mostrou não ser viável estabelecer um valor genérico de (*s*) aplicável para qualquer projeto. Isso devido ao escopo dos fatores que influenciam na intensidade do esquecimento.

Figura 2.2 – Curvas de esquecimento para outros valores de Força de Memória.



Fonte: HATTORI *et al*. (2012)

YANG *et al*. (2018) propuseram uma abordagem de recomendação de revisores para *pull-requests* (códigos que desenvolvedores fora de um projeto solicitam aos donos incluírem-nos em um repositório do *GitHub*). Nessa abordagem há uma consideração do fator esquecimento para obtenção de recomendações mais condizentes com o estado atual de *expertise* dos revisores recomendados. Os autores fazem uso da fórmula de *Ebbinghaus* (2.1), e consideram, porém para a força de memória aplicada (*s*) apenas o valor padrão 1. Isso reflete na prática em um decaimento muito acentuado (de 100% para 37% após um dia) conforme observado por KRÜGER *et al*. (2018). Os autores também reconhecem a necessidade de se considerar que cada desenvolvedor possui valor diferente para a força de memória (*s*), pois isso depende do quanto cada um já tenha usado as *expertises* trabalhadas.

Ao descobrir quem está mais apto em *expertises* exigidas por tarefas de manutenção, HATTORI *et al*. (2012) dizem que, de forma análoga à noção do esquecimento, é também relevante considerar o quanto ocorre de recuperação (ou aprendizado) da memória perdida pelo desenvolvedor. KANG e HAHN (2009) *apud* (KRÜGER *et al*., 2018) sugerem que os efeitos do aprendizado são evidentes para *expertises* de tecnologias enquanto o esquecimento decorre mais nos aspectos metodológicos desses conhecimentos. Com isso, percebe-se que alguém experiente tem maior possibilidade de compreender um código fonte (que nunca viu antes) do que alguém novato na *expertise* necessária para sua compreensão. Ou seja, alguém com mais interações no passado na dita *expertise*, tem mais possibilidade de recuperar os conhecimentos necessários para a compreensão efetiva do software. Isso culmina em menor tempo despendido com o processo de familiarização do código e assim o desenvolvedor

poderá exercer a manutenção de forma a minimizar possíveis erros decorrentes da falta de *expertise* (*bug tossing*).

Semelhantemente, KRÜGER *et al*. (2019) postulam questionamentos sobre quais circunstâncias há um melhor custo-benefício em atribuir o desenvolvedor original (que construiu a maior parte do código no passado), ou aquele com a maior quantidade atual de interações no respectivo código, ou então o programador que contribuiu mais recentemente no desenvolvimento do software. Considerando que a etapa de compreensão do código é a principal e mais custosa atividade nesse contexto, ressaltam ser muito importante o desenvolvedor adquirir familiaridade com o programa de computador. Responder a essas perguntas postuladas e outras relacionadas exige uma investigação para descobrir quais desses desenvolvedores adquirem familiaridade do código com maior facilidade em cada situação.

KRÜGER *et al*. (2018) evidenciaram que a fórmula de *Ebbinghaus* (2.1) não serve para aplicação direta em múltiplas interações, mas mediante uma adaptação desta pode ser utilizada para obtenção da média ponderada dos valores obtidos somados, conforme aplicado por HATTORI *et al*. (2012) e YANG *et al*. (2018). Uma alternativa também é a utilização de outras fórmulas de esquecimento já concebidas para consideração de múltiplas interações (contribuições). Um exemplo disso é o trabalho de GUERCIO *et al*. (2018). Este aplicou uma fórmula em seu trabalho para atestar o quão recente são as contribuições (*commits*) de desenvolvedores em repositórios no *GitHub*, de modo a penalizar as mais antigas, atribuindo maior importância para as contribuições mais recentes, sem deixar de considerar todo o histórico. Fórmula essa que busca identificar os desenvolvedores de um projeto que contribuíram mais vezes e mais recentemente em um projeto de software, somando as diferenças de tempo entre cada data de contribuição com a data em que foram observadas.

Semelhantemente, SCHETTINO *et al*. (2019) penalizaram contribuições antigas através de uma fórmula que busca detectar especialistas que estão com a capacidade de colaboração reduzida em um projeto, isso devido ao tempo decorrido de inatividade destes. A fórmula desenvolvida faz uma penalização exponencial com base na quantidade de dias decorridos desde a interação do especialista em um projeto de software. Para assim atribuir um peso menor em suas contribuições e estas não terem o mesmo peso das contribuições mais recentes.

2.3 CROWDSOURCING

Quando não há desenvolvedores locais disponíveis para serem recrutados, opta-se pela adoção do desenvolvimento global de software, em que participam programadores de diversas

especialidades geograficamente distribuídos. Esse modelo de negócio utiliza o *crowdsourcing* para fazer com que programadores engajados na busca por trabalho se conectem com companhias que estão oferecendo tarefas de desenvolvimento e opcionalmente junto com uma recompensa. De acordo com DE NEIRA *et al*. (2018) o desenvolvimento de projetos de software tem se mostrado eficaz utilizando o *crowdsourcing* de desenvolvedores distribuídos globalmente, cuja plataforma principal é o *TopCoder*. LATOZA e VAN DER HOEK (2016) descrevem três formas diferentes de implementação do *crowdsourcing* em plataformas como o *TopCoder*. A primeira delas, **Peer Production**, envolve contribuições conjuntas de desenvolvedores para a construção de artefatos de código aberto. Em outra, **Competition**, as empresas fazem uma competição pública para buscar as melhores contribuições dos desenvolvedores, sendo essas recompensadas com dinheiro. E a última, **Micro-Task**, trata-se de uma competição cujas tarefas designadas são subdivididas em tarefas menores. As melhores contribuições também são recompensadas monetariamente.

Um desenvolvedor consegue se destacar como notório nas suas *expertises* de atuação quando este apresenta boa reputação na comunidade de *crowdsourcing*. Reputação essa que é construída por meio das boas contribuições apresentadas ao longo da carreira. Segundo BEGEL *et al*. (2013) a plataforma *TopCoder* lida com a reputação de seus usuários através dos parâmetros de classificação e de *feedback*, como também por meio de diferentes tipos de métricas. O sistema de classificação do *TopCoder* (conhecido como *Rating*[4]) é considerado como sendo, provavelmente, o mais influente para a reputação dos usuários do *TopCoder*. Pois, este atribui uma pontuação para cada *submit* individual dos desenvolvedores que participam das competições realizadas pela plataforma. Essas competições são conhecidas como SRM (*Single Round Matches*) e na lista classificatória são atribuídas cores indicativas para toda a comunidade (a cor vermelha por exemplo indica que o usuário está classificado como sendo parte dos 10% melhores classificados). Essa lista de classificação se encontra disponível neste link[5].

A recompensa é uma característica importante observada no contexto das competições em desenvolvimento *crowdsourcing*, e tem como propósito incentivar os programadores a contribuírem com suas soluções. Dinheiro costuma ser o principal fator de recompensa abordado, e seu valor varia dependendo do tipo de contribuição requisitada. Na plataforma *TopCoder* por exemplo, de acordo com BEGEL *et al*. (2013), existem recompensas em dinheiro que variam conforme o tipo da competição. Variação essa que abrange desde um

[4] https://www.topcoder.com/community/competitive-programming/how-to-compete/ratings
[5] https://community.topcoder.com/tc?module=AlgoRank

simples concerto de um *bug* até uma maratona de programação para desenvolver algo complexo em pouco tempo. ILLAHI *et al*. (2019) no entanto, realizaram um estudo empírico para descobrir quais os reais motivos de os desenvolvedores participarem em competições de *crowdsourcing*. O estudo evidenciou que ao invés de recompensa em dinheiro, os participantes têm mais interesse em adquirir aprendizado, conhecer outros desenvolvedores e terem suas habilidades reconhecidas pela comunidade por meio da reputação. Os autores com isso sugerem que plataformas de *crowdsourcing*, tais como o *TopCoder*, tenham que contribuir incentivando o crescimento profissional dos participantes ao invés de somente oferecerem recompensa em dinheiro.

Um problema recorrente observado é a questão de como incluir desenvolvedores que nunca contribuíram antes em projetos amplos de software, e não estão familiarizados com o ambiente de desenvolvimento. Segundo ZANATA *et al*. (2017), os novatos que querem participar em projetos *crowdsourcing* e em projetos de código aberto (*Open Source Software* – OSS) enfrentam problemas tais como: problemas técnicos, documentação escassa ou inexistente, problemas de receptividade da comunidade, descrição confusa das tarefas, e diferenças culturais. A motivação dos desenvolvedores para projetos *crowdsourcing* envolve principalmente ganhos financeiros, havendo assim transferência de intelecto e os participantes são tratados como concorrentes ao invés de colaboradores. Já em projetos de código aberto (OSS) os desenvolvedores contribuem por vontade própria e são motivados por ideologia, autodesenvolvimento, necessidade própria e altruísmo. Também mantêm a autoria de seus próprios trabalhos. O estudo realizado pelos autores identificou seis problemas que impedem a adesão de novatos nos projetos:

(1) Falta de documentação ou diagramas para compreensão prática da tarefa a ser resolvida.

(2) Dificuldade dos usuários em encontrar tarefas adequadas às suas *expertises* devido à escassez no gerenciamento destas e em conseguir estabelecer um tempo para resolução desta.

(3) Dificuldade dos desenvolvedores em compreenderem a estrutura do código fonte dos projetos ou de suas arquiteturas.

(4) Excesso de informação relacionado às tarefas pendentes, dificultando assim o programador em conseguir tomar decisões para planejamento de uma resolução.

(5) A plataforma de *crowdsourcing* tem pouca usabilidade (nesse estudo somente a do *TopCoder* foi verificada).

(6) Se o novato não sabe Inglês, este vai ter dificuldade em usar a respectiva plataforma.

Dificuldade em compreender o código fonte é algo comum em qualquer modelo de desenvolvimento. Porém no *crowdsourcing* essa dificuldade é amplificada.

STEINMACHER *et al*. (2018) elaboraram um estudo que identificou 57 problemas enfrentados pelos novatos (*newcomers*) para contribuírem em projetos de softwares de código aberto (*Open Source Software* – OSS). Esses problemas foram classificados em 7 categorias diferentes. Considerando esses problemas identificados, os autores desenvolveram uma plataforma web para ajudar os novatos serem incluídos em projetos OSS, oferecendo informações relevantes que os facilitam na realização de suas contribuições. Os autores avaliaram a plataforma desenvolvida e com isso elaboraram um conjunto de orientações para ajudar projetos a terem novos usuários contribuintes e guiar os novatos no processo de como contribuir para os projetos OSS.

2.4 TRABALHOS RELACIONADOS

Encontrar especialistas para apoiar o desenvolvimento de software em um contexto de desenvolvimento global é uma atividade que tem sido tratada na literatura. Antes da definição do objetivo definido na presente dissertação, um mapeamento sistemático da literatura (Apêndice A) foi realizado para conhecer o estado-da-arte das soluções existentes que buscam desenvolvedores capacitados nas tarefas de manutenção de software. Os resultados do mapeamento levaram à identificação de uma oportunidade de pesquisa que culminou no desenvolvimento de uma abordagem que viesse a sugerir desenvolvedores em tarefas de manutenção de software. Abordagem essa que, considera a reputação e o esquecimento como forma de os recrutadores ou gerentes de projetos poderem confiar nos desenvolvedores em suas capacidades técnicas. Alguns trabalhos relacionados à abordagem desenvolvida são apresentados a seguir. Entretanto, esses trabalhos apresentam limitações, principalmente na questão de se considerar o esquecimento na busca pela colaboração de desenvolvedores, e mediante análise de suas reputações em diferentes repositórios abertos. Os trabalhos desenvolvidos têm por objetivo principal o apoio à manutenção de software dentro do contexto de desenvolvimento global de software e estão descritos a seguir.

TAVARES *et al*. (2015) desenvolveram uma infraestrutura de nome **GiveMe Infra** para apoio à realização de atividades de manutenção e evolução colaborativa de software, realizadas por equipes co-localizadas ou geograficamente dispersas. Como entrada para sua arquitetura há a indicação de um módulo ou componente de um projeto que se deseja dar manutenção. Em seguida, os dados históricos referentes ao projeto são importados e filtrados para uma análise estatística, bem como para o cálculo das métricas disponíveis. Esta

abordagem gera informações a respeito do tamanho, periodicidade e complexidade dos componentes. No entanto, a abordagem não considera o contexto de bases abertas (*open source*) e nem apoia a formação de equipes.

MIGUEL *et al*. (2016) elaboraram o *framework* **GiveMe Effort**, que utiliza os dados históricos dos projetos de software para apoio à atividade de estimativa de esforço de manutenção, visando calcular o tempo e planejamento das manutenções corretivas, adaptativas ou evolutivas. Para cálculo da estimativa de tempo são utilizadas técnicas que atestam a similaridade entre a requisição de mudança solicitada e o histórico de dados coletados. É verificada também a reputação de um desenvolvedor com base na quantidade de requisições que este resolveu ao longo do tempo. Porém, os dados históricos utilizados são referentes à perspectiva do sistema em manutenção, e não à perspectiva do desenvolvedor responsável por casos anteriores de mudanças pelos quais o software passou. Apesar de o *GiveMe Effort* oferecer suporte para planejar as atividades de manutenção, a tarefa de alocar desenvolvedores às atividades de manutenção não é abordada, e o trabalho não considera o uso de bases abertas (*open source*) de desenvolvimento.

LÉLIS (2017) desenvolveu a infraestrutura **IRID** para apresentar informação dinâmica de reputação, mediante a percepção de que informações de reputação são utilizadas como mecanismo de cálculo da estimativa do tempo das atividades de manutenção de software. É considerada a reputação do desenvolvedor juntamente com seus dados históricos na manutenção e evolução de software em ambientes distribuídos. São considerados seis critérios diferentes para reputação, sendo estes: Opinião, Relacionamento, Comportamento, Confiabilidade, Similaridade e Expertise. O trabalho apresenta uma listagem de desenvolvedores ordenada pelos valores de reputação atribuídos a cada um. Isso considerando informações de grupo e domínio inferidas pela ontologia ArchiRIOnt, desenvolvida por LÉLIS *et al*. (2016). No entanto, a listagem obtida não representa uma ordem de prioridade para análise individual dos desenvolvedores às solicitações de mudança, como se estes fossem de fato os mais relevantes para as respectivas tarefas. Também não considera o ambiente de bases abertas (*open source*) para desenvolvimento e nem o fator esquecimento associado à reputação.

DE NEIRA *et al*. (2018) elaboraram um estudo para identificação de "hiperespecialistas" em uma plataforma de desenvolvimento de software por *crowdsourcing*. Nessa plataforma, de nome *TopCoder*, são disponibilizadas tarefas de manutenção e/ou desenvolvimento de software para desenvolvedores globalmente distribuídos resolverem em modalidade de competição. Prêmios são oferecidos para os usuários que contribuem com as

melhores soluções desenvolvidas. Os autores observaram que os desenvolvedores dessa plataforma apresentam um comportamento de resolverem tarefas de tecnologias específicas, evidenciando o aspecto de se tornarem especialistas nas *expertises* relacionadas. Também observaram que usuários não especialistas contribuem mais em tarefas que exigem maior variedade de *expertises*, e estes desenvolvedores ganham mais prêmios do que os "hiperespecialistas" de tarefas mais específicas. No entanto, o estudo não considerou o esquecimento ao fazer a avaliação das contribuições passadas, e nem consideram a reputação dos desenvolvedores na pesquisa.

TRAINER *et al*. (2018) apresentam um trabalho que trata sobre a influência de fatores da confiabilidade por percepção de suas entidades. Para tanto, propuseram uma ferramenta de apoio à identificação de desenvolvedores confiáveis aos seus recrutadores. A respectiva abordagem coleta informações sobre as atividades de desenvolvimento de software em repositórios de códigos-fonte e em sistemas de rastreio de *bugs*. Esta abordagem agrega as informações obtidas para exibi-las por interface gráfica, de modo a dar apoio à confiabilidade por meio de dois fatores: apresentar informações de disponibilidade (horários de atendimento) e rapidez da resposta dos usuários colaboradores de projetos. Os autores correlacionam esses dois fatores (disponibilidade e rapidez de resposta) como elementos importantes que antecedem a confiança estabelecida. A avaliação feita do protótipo mostrou que este ajuda em aprimorar as atribuições e percepções de confiança feitas sobre os desenvolvedores pelos recrutadores. Os fatores de confiabilidade utilizados no protótipo podem ser utilizados também como indicadores de reputação dos desenvolvedores. Embora o esquecimento não seja tratado diretamente na abordagem, existe a possibilidade desses fatores também serem utilizados para a verificação da relevância da *expertise* no desenvolvedor ante o esquecimento desta. Pois alta disponibilidade e resposta rápida do desenvolvedor atuando em uma *expertise* é indicador de que este é ativo no respectivo conhecimento, e assim, com pouco esquecimento desta. Entretanto, os autores não consideram que os indicadores do conhecimento sobre um tema específico e a reputação do desenvolvedor possam ser obtidos de diferentes repositórios abertos.

OLIVEIRA JR *et al*. (2019) elaboraram uma arquitetura de apoio à recomendação de colaboradores advindos de plataformas externas para atuarem em projetos de desenvolvimento global de software. A abordagem faz uso de informação semântica proveniente de uma combinação de ontologias para buscar contribuições passadas dos desenvolvedores. Informações estas de repositórios presentes em sistemas de rastreio de *bugs* e de versionamento de código fonte, como também em redes sociais para profissionais de

desenvolvimento de software. Os conhecimentos aplicados pelo desenvolvedor são inferidos a partir do registro de uso de dependências dos códigos fontes em projetos atuados por este. A arquitetura faz as recomendações dos colaboradores diferenciando-os e ordenando-os por valores de similaridade dos projetos atuados por estes no passado e comparando os dados com os do projeto em que se quer recrutá-los. Porém, no respectivo trabalho, não é considerado o fator esquecimento na atribuição das *expertises* (conhecimentos) nos desenvolvedores, algo que pode ser um problema ao se recomendar alguém inativo.

SCHETTINO *et al*. (2019) propuseram uma rede colaborativa para encontrar comunidades de desenvolvedores com *expertises* de tecnologias, dentro de projetos aplicados ao desenvolvimento global de software. A abordagem busca encontrar nos desenvolvedores, não apenas conhecimentos técnicos, mas também informações que remetem à capacidade de colaboração destes como também de interações com outros dentro de uma comunidade, mostrando aspectos de sua reputação. Nesta rede há também a penalização de interações (comentários de revisão em *pull-requests*) passadas efetuadas por desenvolvedores que outrora detinham determinada *expertise* correlacionada e, por não estarem mais ativos nesta, suas respectivas contribuições não podem ter a mesma relevância hoje do qual outrora havia (esquecimento). Porém, no contexto de bases abertas, o trabalho se limita em analisar contribuições de desenvolvedores apenas no *GitHub*.

XIE *et al*. (2020) elaboraram uma abordagem, denominada **SoftRec**, que faz recomendação de desenvolvedores de software com base em vários tipos de relações implícitas entre estes e tarefas dos quais tenham anteriormente resolvido. Relações estas que, segundo os autores, estão implícitas de três formas: (1) colaboração entre desenvolvedores; (2) interação entre desenvolvedores e tarefas; (3) associação entre tarefas. A abordagem utiliza uma arquitetura de rede neural profunda para gerar as recomendações de desenvolvedores obtidos a partir da associação desses vários relacionamentos por filtragem colaborativa. Os autores realizaram um experimento para averiguar o uso do *SoftRec* em recomendar desenvolvedores no contexto de repositórios do *GitHub* e do *GitLab*, considerando as interações dos desenvolvedores em contribuir com os projetos por meio de *pull-requests*. A abordagem, porém, não considera a penalização de interações antigas (esquecimento) e nem verifica a reputação dos desenvolvedores para a recomendação.

2.5 CONSIDERAÇÕES FINAIS DO CAPÍTULO

O objetivo deste capítulo foi apresentar os principais conceitos que fundamentam este trabalho. Foram abordados os principais aspectos relativos à reputação, ao esquecimento e ao

uso do *crowdsourcing* no desenvolvimento de software. Foram apresentados também estudos encontrados na literatura, cujas abordagens mencionadas são condizentes com o presente trabalho na parte de apoio à manutenção de software, que estabelecem uma forma de confiança entre desenvolvedores e utilizam fatores como reputação, perfil de *expertise*, busca por contribuições em bases abertas, inferência de *expertises* externas. Porém, grande parte dos trabalhos apresentados não considera diretamente em suas propostas a penalização das contribuições antigas dos desenvolvedores para atribuir o esquecimento. Portanto, o presente trabalho busca apoiar o estabelecimento de confiança de um desenvolvedor ao resolver uma tarefa que demandam determinadas *expertises*, no contexto da manutenção de software. Isso considerando a verificação das *expertises* atribuídas às tarefas anteriormente resolvidas pelo desenvolvedor e também considerando a reputação deste junto com o fator de esquecimento. Além disso, para apoiar as atividades de manutenção, diferentes repositórios abertos podem ser utilizados.

3 SOLUÇÃO PROPOSTA

Este capítulo apresenta a solução desenvolvida para alcançar os objetivos propostos da dissertação. Os aspectos de projeto (requisitos e arquitetura) são apresentados com intuito de apoio à seleção de especialistas para atividades de desenvolvimento de software. É apresentada uma ontologia utilizada para a formação de grupos juntamente com reputação. São descritos os conceitos dos critérios de reputação e esquecimento utilizados na solução desenvolvida. São apresentados o projeto e a implementação da solução considerando-se o suporte de plataformas distribuídas que apoiam o desenvolvimento de software.

3.1 REQUISITOS

Considerando os pressupostos teóricos vistos no capítulo anterior (Reputação, Esquecimento, *Crowdsourcing* e acesso às bases abertas) e os objetivos da pesquisa, foram definidos os requisitos funcionais e não funcionais para a solução desenvolvida neste presente trabalho.

3.1.1 Requisitos funcionais

(RF01). O sistema deve permitir apoio à tomada de decisão de um desenvolvedor responsável por atribuir desenvolvedores para tarefas de manutenção e desenvolvimento de software.

(RF02). O sistema deve ser capaz de extrair os conhecimentos (*expertises*) aplicados em contribuições externas do desenvolvedor advindas das seguintes plataformas de apoio ao desenvolvimento de software: *GitHub*, *StackOverflow,* e *TopCoder*.

(RF03). O sistema deve ser capaz de utilizar os critérios de reputação sobre os dados das contribuições externas e internas do desenvolvedor em suas *expertises*.

(RF04). O sistema deve apresentar o decaimento do conhecimento (esquecimento) conforme o tempo de desuso das *expertises* associadas ao desenvolvedor.

(RF05). O sistema deve apresentar, para cada desenvolvedor, uma lista de tarefas que tenham as *expertises* com que o respectivo desenvolvedor já tenha trabalhado.

(RF06). O sistema deve apresentar, para cada tarefa não finalizada, uma lista de desenvolvedores que já tenham trabalhado nas *expertises* destas tarefas.

(RF07). O sistema deve prover apoio à formação de grupos de desenvolvedores relevantes a uma tarefa como alternativa à atribuição individual de desenvolvedores.

3.1.2 Requisitos não funcionais

(RNF01). **Escalabilidade**: permitir a utilização de diferentes bases de dados (abertas) em função das diferentes complexidades do software.

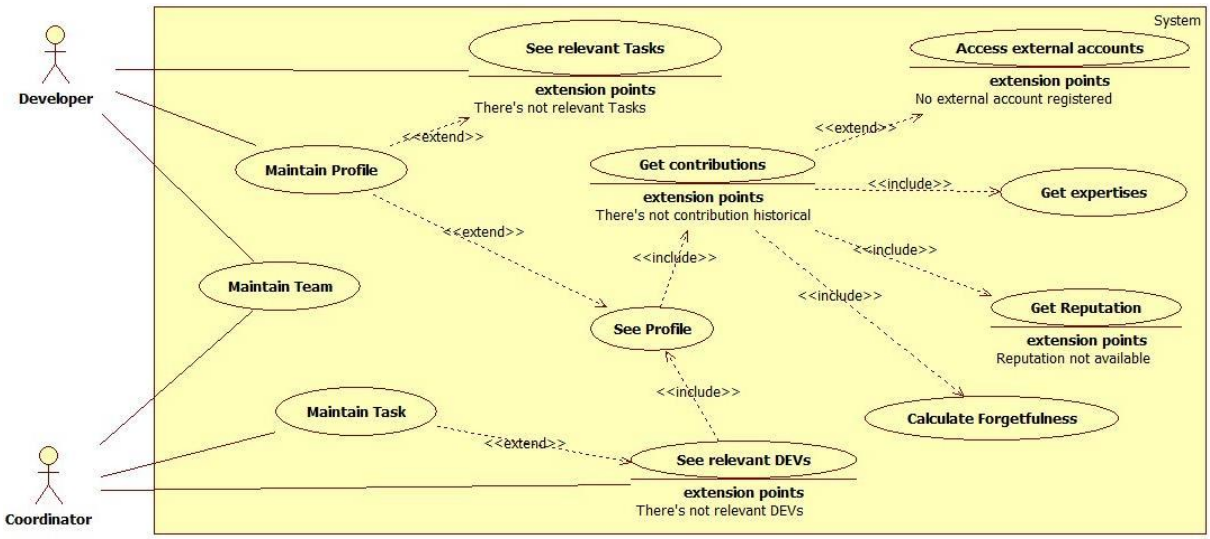
(RNF02). **Extensibilidade**: permitir com que o sistema tenha suporte para incorporação de novas funcionalidades em função da sua evolução.

(RNF03). **Reusabilidade**: permitir a reutilização de módulos do sistema em outras aplicações.

3.1.3 Casos de uso

Um diagrama de casos de uso foi elaborado em conformidade com os requisitos funcionais definidos anteriormente. A Figura 3.1 exibe esse diagrama e, no âmbito de formação de uma equipe para resolução de uma tarefa, foi considerada a definição de dois tipos de atores: o **Coordenador** (*Coordinator*), sendo aquele responsável por criar as tarefas e definir as *expertises* a serem trabalhadas nestas, e o **Desenvolvedor** (*Developer*), que tem as *expertises* para trabalhar na resolução das tarefas.

Figura 3.1 – Diagrama de casos de uso da solução proposta.



Fonte: Elaborado pelo autor (2020).

3.2 ARQUITETURA

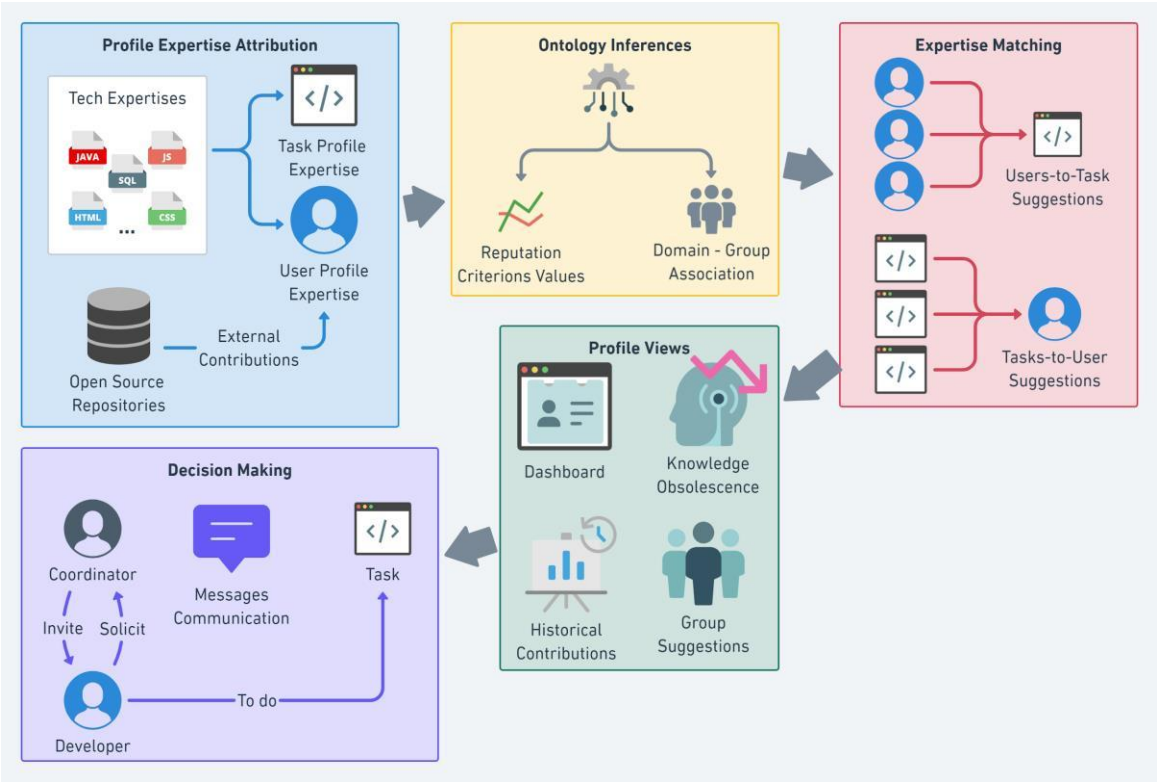
A abordagem proposta tem por objetivo apoiar a escolha de desenvolvedores aptos para tarefas de desenvolvimento de software como também de manutenção do tipo adaptativa ou evolutiva, tais como requisições de mudança (*change request*), e do tipo corretiva ou preventiva para problemas expostos pelos relatórios de erros (*bug report*). Este apoio provém da visualização individual de desenvolvedores com suas respectivas sugestões de tarefas e da

visualização específica de tarefas com suas respectivas sugestões de desenvolvedores. As sugestões apresentadas são provenientes da verificação de *expertises* atribuídas em comum entre desenvolvedor e tarefa. As *expertises* dos desenvolvedores em tecnologias provêm das contribuições advindas de plataformas externas ou localmente, sendo que a reputação e o esquecimento dessas *expertises* são calculados em um módulo à parte.

3.2.1 Arquitetura geral

Para esboçar o planejamento da solução proposta, foi elaborada uma visão geral da arquitetura em conformidade aos requisitos funcionais apresentados. A Figura 3.2 apresenta essa arquitetura.

Figura 3.2 – Visão geral da Arquitetura proposta.



Fonte: Elaborado pelo autor (2020).

O módulo (*Profile Expertise Attribution*) é responsável pela atribuição de *expertises* de tecnologias para as tarefas registradas na plataforma e o preenchimento do perfil de *expertises* dos usuários desenvolvedores. Os desenvolvedores que possuem contribuições em outras plataformas têm a opção de preencherem manualmente o perfil com o registro das *expertises* provenientes de suas contribuições externas. Essas contribuições podem ser advindas de repositórios públicos de código fonte criados, por exemplo, no *GitHub*; tarefas de desafios (*challenges*) em desenvolvimento e manutenção de software, por exemplo, no *TopCoder*; e perguntas e respostas efetuadas, por exemplo, no *StackOverflow*. Para a constituição do perfil

do desenvolvedor, os registros das contribuições externas obtidas devem possuir valores de data e *expertise(s)* associadas, para posterior tratamento com relação ao esquecimento.

O módulo (*Ontology Inferences*) tem como propósito dar apoio à formação de grupos de usuários desenvolvedores que possuem *expertises* com os domínios de conhecimento associados e valores de reputação atribuídos às suas contribuições passadas.

O módulo (*Expertise Matching*) tem por objetivo fornecer para cada nova tarefa uma lista sugestiva de desenvolvedores aptos, e também para cada desenvolvedor, sugestões de tarefas relevantes ao perfil do desenvolvedor. A identificação dos desenvolvedores capacitados para tarefas tem princípio na comparação do perfil de *expertise* de cada tarefa com o perfil de cada desenvolvedor. A abordagem faz então a identificação das *expertises* em comum entre cada tarefa para com um desenvolvedor, e também entre cada desenvolvedor para com uma tarefa.

No módulo (*Profile Views*), cada desenvolvedor pode visualizar essas sugestões de tarefas de acordo com suas *expertises*, assim como cada coordenador de tarefa pode visualizar as sugestões de desenvolvedores com conhecimentos nas *expertises* das tarefas. Outras opções de visualizações incluem o esquecimento (*Knowledge Obsolescence*), a sugestão de formação de grupos (*Group Suggestions*) e o histórico de contribuições (*Historical Contributions*) que são exibidas para cada *expertise* associada.

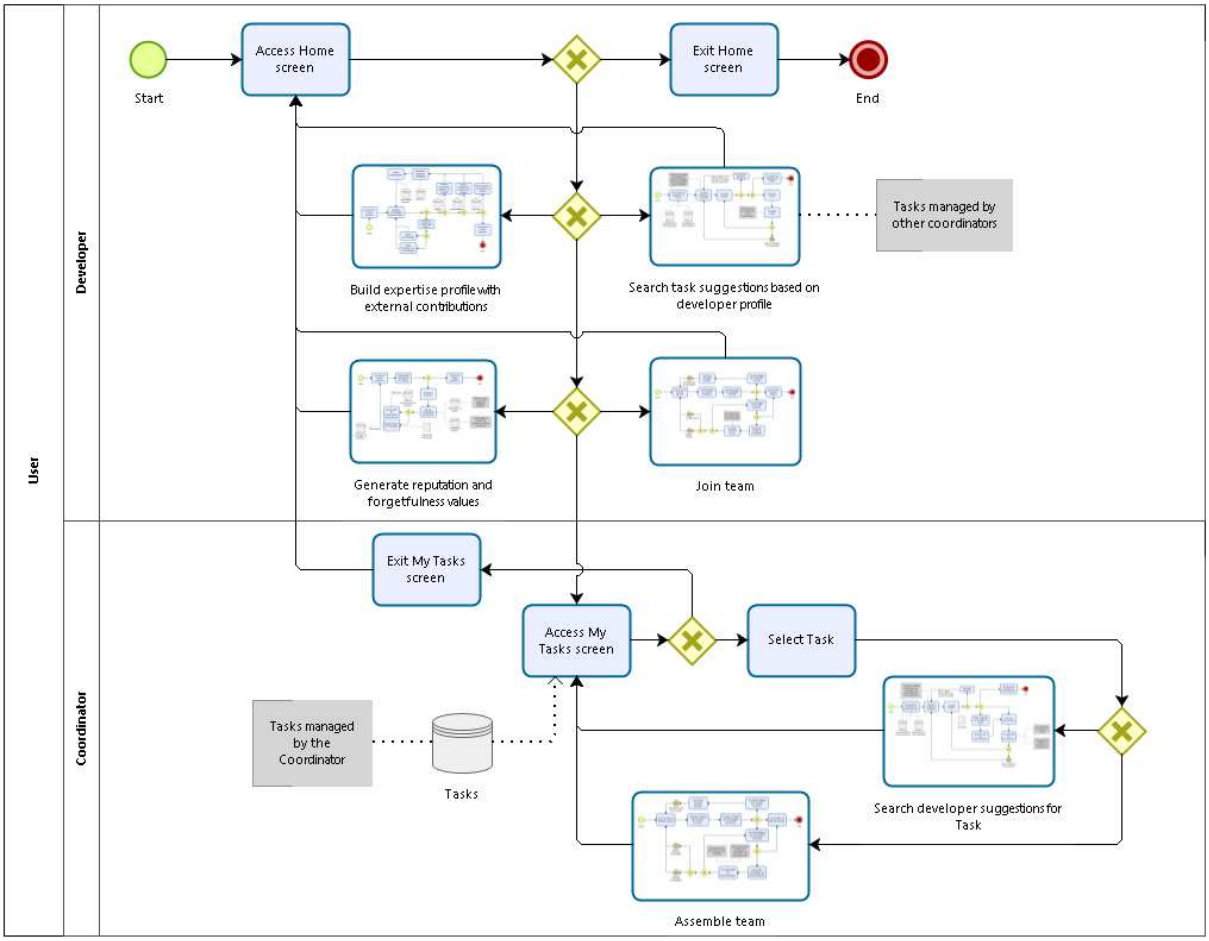
O módulo (*Decision Making*) representa o momento em que usuários desenvolvedores e coordenadores de tarefas interagem para tomada de decisão. Seja um desenvolvedor que vê uma sugestão de tarefa e decide solicitar ao coordenador desta para contribuir, ou então um coordenador que visualiza o perfil de um desenvolvedor sugerido e decide convidá-lo para contribuir na tarefa.

3.2.2 Sistema desenvolvido

O sistema desenvolvido com base nessa arquitetura permite com que um mesmo usuário possa atuar sendo um desenvolvedor à procura de tarefas condizentes com seus conhecimentos, ou então como um coordenador de tarefas à procura de desenvolvedores que trabalham nas *expertises* para suas tarefas criadas. A aplicação tem por objetivo fornecer as seguintes opções para o usuário: elaborar seu perfil de *expertises*, criar tarefas e nestas associar as *expertises* necessárias, e gerenciar as equipes criadas para resolução de suas tarefas elaboradas e ver outras equipes que o próprio esteja participando como desenvolvedor. Para cada uma de suas tarefas criadas o coordenador tem a opção de fazer o sistema buscar sugestões de desenvolvedores, devidamente registrados, que possuam as mesmas *expertises*

da tarefa em seus perfis. O desenvolvedor, após o registro de suas próprias *expertises* em seu perfil, tem a opção de solicitar ao sistema a busca por tarefas com as mesmas *expertises*. O funcionamento geral do processo da solução proposta está ilustrado na Figura 3.3.

Figura 3.3 – Diagrama BPMN do funcionamento geral da implementação da solução proposta.



Fonte: Elaborado pelo autor (2021).

Seis subprocessos estão agregados no processo principal da proposta. Quatro deles estão associados ao papel de desenvolvedor do usuário e os outros dois no papel de coordenador. O usuário quando acessa o sistema na tela inicial, tem as seguintes opções de subprocessos ao seu dispor como desenvolvedor:

- Formar perfil de *expertises* com suas contribuições externas.
- Gerar valores de reputação e esquecimento para suas *expertises*.
- Buscar sugestões de tarefas com base em seu perfil de *expertises*.
- Entrar em equipe de tarefa.

O usuário como coordenador tem as seguintes opções de subprocessos ao gerenciar uma de suas tarefas criadas:

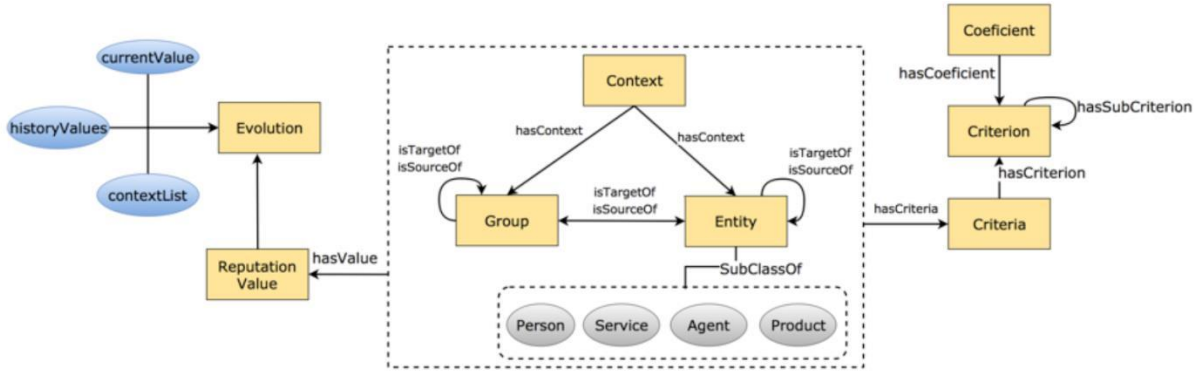
- Buscar sugestões de desenvolvedores para a tarefa.
- Montar equipe da tarefa.

Cada um desses subprocessos são ilustrados e explicados juntamente com a implementação da solução proposta na subseção (3.5).

3.3 ONTOLOGIA ARCHIRIONT

Uma ontologia denominada *ArchiRiOnt*[6], desenvolvida por LÉLIS *et al*. (2016), é utilizada na solução proposta deste presente trabalho para prover apoio à formação de grupos de usuários desenvolvedores que tenham contribuições com os mesmos domínios de conhecimentos. A ontologia se baseia em um modelo conceitual, proposto pelos autores, para representação das informações de reputação e engloba os conceitos de entidade, grupo e domínio. Segundo os autores, esses conceitos representam os pontos de vista dos quais a reputação pode ser percebida pelos avaliadores. A Figura 3.4 apresenta um modelo simplificado da respectiva ontologia com o relacionamento entre as classes definidas.

Figura 3.4 – Modelo simplificado da ontologia *ArchiRIOnt*.



Fonte: LÉLIS *et al*. (2016).

A solução proposta utiliza a ontologia para obter sugestões de formação de grupo inserindo como parâmetro de entrada um usuário desenvolvedor com suas contribuições em tarefas, considerando três contextos diferentes: (1) o da própria tarefa, para retornar sugestões de usuários desenvolvedores que já trabalharam em outras tarefas de um mesmo desenvolvedor sendo analisado; (2) o da categoria de *expertise* trabalhada, retornando sugestões de desenvolvedores que possuam histórico de contribuições em uma mesma categoria; e (3) o da tecnologia de *expertise* trabalhada, ao retornar como sugestões usuários que contribuíram na mesma tecnologia.

A justificativa para a *expertise* ser tratada em dois tipos (categoria e tecnologia) remete ao trabalho de DE NEIRA *et al*. (2018), cujos autores buscam identificar desenvolvedores

[6] https://github.com/pgcc/archiri-ont

“hiperespecialistas” em tecnologias de desenvolvimento de software. Nesse trabalho de pesquisa os autores perceberam haver tecnologias similares ou idênticas, mas com nomes diferentes, associadas aos desenvolvedores, assim como também versões específicas de uma tecnologia ou de um *framework* relacionado a tecnologias específicas. Portanto os autores categorizaram esses registros de tecnologias para facilitar a análise do fenômeno da “hiperespecialidade” no respectivo trabalho.

Na solução proposta é utilizada essa listagem de categorias com suas respectivas tecnologias para junto com a ontologia *ArchiRiOnt* identificar desenvolvedores dos quais que ainda não tenham *expertise* em determinada tecnologia, possam ter *expertise* na categoria pertencente a essa tecnologia (Por exemplo, um desenvolvedor que conhece TypeScript deve também conhecer o JavaScript, pois é conhecimento básico para essa tecnologia. Mas quem conhece JavaScript pode não necessariamente conhecer TypeScript. Nesse caso, o TypeScript é uma tecnologia pertencente à categoria da tecnologia JavaScript).

3.3.1 Critérios de Reputação

Durante o desenvolvimento da solução proposta para o presente trabalho, incorporou-se o cálculo da reputação para o estabelecimento de confiança entre desenvolvedores em projetos de software. O trabalho de LÉLIS (2017) apresenta diferentes aspectos que evidenciam o caráter multicritério do cálculo da reputação. Estes aspectos foram apresentados como seis critérios distintos de reputação, cujas fórmulas e algoritmos utilizados foram implementados na solução proposta. Cada um desses critérios é apresentado a seguir e suas respectivas fórmulas são descritas. Os valores de todos esses critérios variam entre 0 e 1.

3.3.1.1 Critério Opinião (Feedback)

Esse critério trata da opinião de entidades sobre uma em específica, e é base para calcular a reputação dessa entidade alvo. CAVERLEE *et al*. (2010) *apud* (LÉLIS, 2017) estabelece o cálculo do valor de opinião através da equação (3.1), do qual é calculada com base na soma da quantidade de avaliações positivas dividida pela soma da quantidade de avaliações negativas com a soma das positivas.

$$F(j) = \frac{\sum_i \mathcal{I}(v_i(j)^+)}{\sum_i \mathcal{I}(v_i(j)^+) + \mathcal{I}(v_i(j)^-)} \tag{3.1}$$

Na solução proposta, o cálculo de Opinião foi adequado ao contexto de uma *expertise* alvo em que os desenvolvedores trabalharam, para estabelecer um valor de Opinião de cada desenvolvedor nessa *expertise* alvo do qual trabalhou. A quantidade de atribuições efetivadas

de um desenvolvedor alvo em tarefas, em um contexto alvo de *expertise* é normalizada com as quantidades de atribuições dos demais desenvolvedores nessa mesma *expertise*. E o valor obtido é considerado como sendo o valor do critério Opinião do desenvolvedor em uma *expertise* específica.

3.3.1.2 Critério Comportamento (Behavior)

O critério Comportamento considera os valores de reputação anteriores de uma entidade alvo. Este está diretamente relacionado com o critério Opinião e constitui um dos fatores que influenciam a reputação na medida em que afeta a opinião das demais entidades. CAVERLEE *et al*. (2010) *apud* (LÉLIS, 2017) calcula o valor de Comportamento através da equação (3.2), no qual considera a média dos valores anteriores de Opinião ponderado por um peso reduzido dos valores de Opinião mais antigos.

$$Tr_h(i) = \sum_{m=1}^{M} ST(i,m) \cdot \frac{I_m}{\sum_{l=1}^{M} I_l} \tag{3.2}$$

No contexto da solução proposta, o valor do Comportamento é calculado com base nos valores anteriores das quantidades de atribuições de cada desenvolvedor nas tarefas de uma *expertise* alvo. Esses valores anteriores são ponderados por um peso menor conforme a idade da atribuição. O valor obtido é normalizado com base nos valores de atribuições anteriores dos outros desenvolvedores, e assim obtém-se o critério Comportamento.

3.3.1.3 Critério Similaridade (Similarity)

O critério Similaridade busca verificar o quão semelhante é uma entidade com outras em termos de reputação em um mesmo contexto. QU *et al*. (2006) *apud* (LÉLIS, 2017) calcula a Similaridade através da equação (3.3), cujo valor é obtido através da comparação entre as reputações da entidade alvo com as das demais entidades que foram avaliadas em um mesmo contexto. O valor "n" refere-se à quantidade de entidades com o qual a entidade alvo está sendo comparada.

$$CCD_{<j,i>} = 1 - \frac{1}{n} \sum_{t=1}^{n} \left| PR_{<j,t>} - PR_{<i,t>} \right| \tag{3.3}$$

O cálculo da Similaridade foi incorporado na solução proposta e o valor desse critério é obtido a partir da comparação dos valores do critério Comportamento dos desenvolvedores ligados ao desenvolvedor alvo e que trabalharam em um mesmo contexto de uma *expertise* alvo.

3.3.1.4 Critério Relacionamento (Relationship)

O critério Relacionamento pretende medir a qualidade e a força do relacionamento entre os membros de um grupo de entidades avaliadas em reputação sob um mesmo contexto. CAVERLEE *et al*. (2010) *apud* (LÉLIS, 2017) faz o cálculo do valor de Relacionamento através da equação (3.4), cujo valor obtido considera o somatório do critério Opinião de cada entidade relacionada a uma entidade alvo. O valor desse somatório é dividido pela quantidade de entidades e o resultado disso é multiplicado pelo valor de Opinião da entidade alvo.

$$R_{[1]}(i) = F(i) \sum_{j \in rel(i)} F(j)/|rel(i)| \tag{3.4}$$

Por fim, caso essa entidade não esteja relacionada com nenhuma outra entidade, o valor do critério Relacionamento assume o valor do critério Opinião dessa entidade. Na solução proposta, o cálculo do Relacionamento é realizado no contexto de uma *expertise* alvo e compara os valores do critério Opinião do desenvolvedor alvo com o dos outros desenvolvedores que trabalharam nessa mesma *expertise*.

3.3.1.5 Critério Expertise

O critério Expertise visa mostrar o quanto uma entidade domina em conhecimento de um contexto de *expertise*; isso com base nas quantidades de relações com outras entidades nessa mesma *expertise*. ZHANG *et al*. (2007) *apud* (LÉLIS, 2017) faz o cálculo desse critério através da equação (3.5), do qual é um algoritmo iterativo nomeado Expertise Rank.

$$ER(A) = (1\text{-}d) + d\ (ER(U_1)/C(U_1) + \ldots + ER(U_n)/C(U_n)) \tag{3.5}$$

Esse algoritmo calcula o valor de Expertise (ER) da entidade alvo com base no valor de Expertise (ER) das demais entidades relacionadas. O parâmetro (*d*) é uma constante atribuída na fórmula para ser uma solução à partida fria, isto é, quando a entidade avaliada não possuir relações com outras entidades em um mesmo contexto de *expertise*. Os autores atribuíram o valor 0,85 para esse parâmetro. Na solução proposta esse mesmo valor é utilizado no parâmetro (*d*) do algoritmo Expertise Rank; e o valor do critério Expertise é calculado no contexto de desenvolvedores relacionados em cada contexto de *expertise*.

3.3.1.6 Critério Confiabilidade (Trustworthiness)

O critério Confiabilidade representa o quão confiável será uma informação advinda de um desenvolvedor para determinado contexto de *expertise*, isto é, o quanto este

desenvolvedor é digno de confiança para determinado conhecimento. CAVERLEE *et al*. (2010) *apud* (LÉLIS, 2017) faz o cálculo de Confiabilidade através da equação (3.6), no qual envolve o valor dos critérios Relacionamento e Confiabilidade das entidades ligadas à entidade sendo avaliada.

$$Tr_q(i) = \lambda \sum_{j \in rel(i)} R(j) \cdot Tr_q(j)/|rel(j)| + (1 - \lambda)F(i) \tag{3.6}$$

O valor de Opinião da respectiva entidade também é utilizado no cálculo de Confiabilidade. Essa equação possui um parâmetro (*lambda*) como uma constante atribuída na fórmula para ser uma solução à partida fria quando a entidade avaliada não possui relações com outras entidades em um mesmo contexto de *expertise*. Os autores atribuíram o valor 0,5 para esse parâmetro, e na solução proposta esse mesmo valor é utilizado para o respectivo parâmetro no algoritmo. O valor do critério Confiabilidade é calculado no contexto de desenvolvedores relacionados em cada contexto de *expertise*.

3.3.2 Critério de Esquecimento (*Forgetfulness*)

Na solução proposta, o cálculo do Esquecimento mediante a fórmula de *Ebbinghaus* (3.7), mencionada no trabalho de KRÜGER *et al*. (2018), foi agregado como um critério à parte para ser utilizado junto com os critérios de reputação na ontologia *ArchiRiOnt*.

$$R = e^{-\frac{t}{s}} \tag{3.7}$$

Isso com o objetivo de caracterizar o estado atual de conhecimento do desenvolvedor junto com os seus valores de reputação em cada *expertise* trabalhada, considerando suas contribuições de desenvolvimento de software nessas *expertises* de tecnologias e categorias de tecnologias.

A referida equação é de natureza exponencial, representando o esquecimento simulado através do decréscimo da Retenção de Memória (R) em função do Tempo (t) decorrido, em dias, sobre uma Força de Memória (s) aplicada. A Força de Memória (s) expressa o quanto de memória é retida ao longo do tempo, sendo que quanto maior esse valor, menor será o decaimento no decorrer do tempo; significando então memória retida por mais tempo. É utilizado o valor ($s = 65$), aplicado pelos autores na constante Força de Memória (s) da respectiva fórmula. O valor obtido pela fórmula varia entre 0 e 1 para $t \geq 0$, em que 1 indica retenção máxima da memória aplicada, enquanto 0 indica esquecimento total.

Um desenvolvedor pode interagir mais de uma vez no conhecimento de uma *expertise*; porém a fórmula de *Ebbinghaus* (3.7) em si é limitada em aplicabilidade para uma única

interação. No intuito de contornar essa limitação, o cálculo do valor do critério de esquecimento é realizado por um somatório dos valores obtidos dessa fórmula para cada tempo (*t*) decorrido em interações diferentes em uma mesma *expertise*, atribuindo assim uma média de Retenção de Memória como o valor final de esquecimento da tal *expertise*. Esse somatório dos valores da fórmula de *Ebbinghaus* (3.7) é apresentado na equação (3.8).

$$\bar{R} = \frac{\sum_{i=1}^{n} e^{-\frac{t_i}{s*n}}}{n} \tag{3.8}$$

Para simular o reforço do aprendizado nos conhecimentos de uma *expertise* diante da ocorrência de novas interações, é realizado um incremento no valor da Força de Memória (*s*) ao fazer multiplicação desta pela quantidade de interações (*n*) da respectiva *expertise*. Com isso, a taxa de decaimento da retenção de memória (*R*) é menor para os desenvolvedores que possuírem maior quantidade de interações.

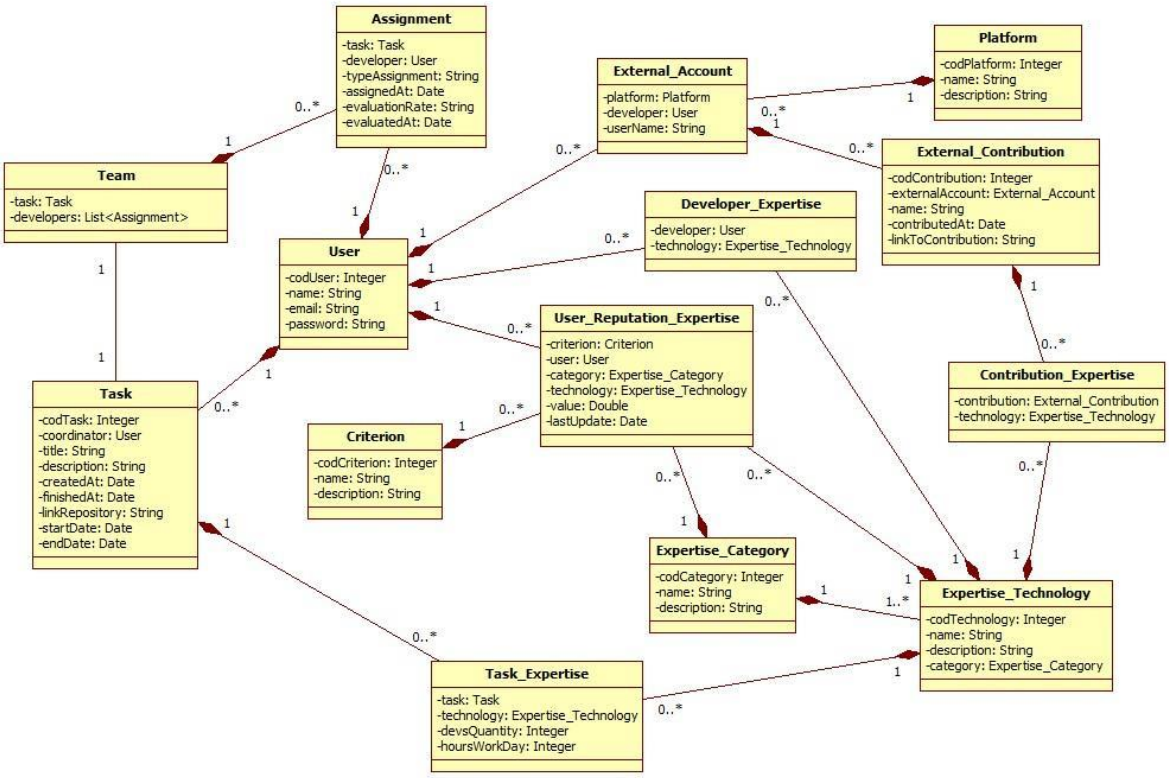
Com isso o critério de Esquecimento foi estabelecido para expressar o quão ocioso está o conhecimento do desenvolvedor em determinada *expertise*. Quanto maior estiver a retenção de memória nessa *expertise*, menor será o esquecimento desta. Porém quanto menor a retenção de memória, maior será o esquecimento. Com intuito de adequar o valor obtido da retenção de memória ao conceito de esquecimento, o critério de Esquecimento assume um valor inverso ao valor da Retenção de Memória da equação (3.8), sendo que valores próximos de 0 significam baixo esquecimento (alta retenção de memória) e valores próximos de 1 significam alto esquecimento (baixa retenção de memória).

3.4 PROJETO

Após apresentados os requisitos e a arquitetura da solução proposta, o projeto é descrito a seguir, começando com o diagrama de classe. Diagrama esse elaborado em conformidade com a arquitetura apresentada juntamente dos seus módulos e também de acordo com os requisitos funcionais definidos, e está ilustrado na Figura 3.5.

O sistema desenvolvido para a solução proposta tem por pretensão registrar as atribuições entre desenvolvedores e tarefas, para identificação das *expertises* as quais o desenvolvedor trabalhou. Essa identificação das *expertises* trabalhadas é complementada também com as contribuições externas das quais o desenvolvedor possui nas plataformas de apoio ao desenvolvimento de software (*GitHub*, *StackOverflow*, *TopCoder*). Os valores dos critérios de reputação e esquecimento são correlacionados com as *expertises* trabalhadas pelo usuário desenvolvedor nas atribuições e nas contribuições externas.

Figura 3.5 – Diagrama de classes da proposta desenvolvida.



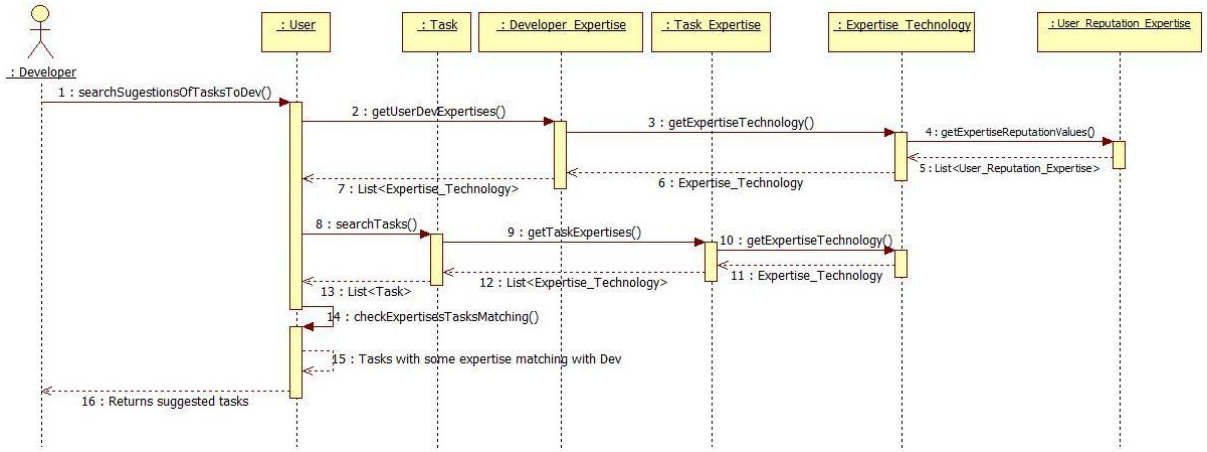
Fonte: Elaborado pelo autor (2020).

3.4.1 Abordagem de sugestão

A abordagem de sugestão contempla os módulos: *Expertise Matching* e *Decision Making* da arquitetura (Figura 3.2) para correspondência de *expertises* entre tarefas e usuários desenvolvedores e assim apresentar as sugestões relevantes conforme valores de reputação e esquecimento. A Figura 3.6 e a Figura 3.7 ilustram em diagramas de sequência como essas sugestões são obtidas na proposta desenvolvida considerando o diagrama de classe (Figura 3.5).

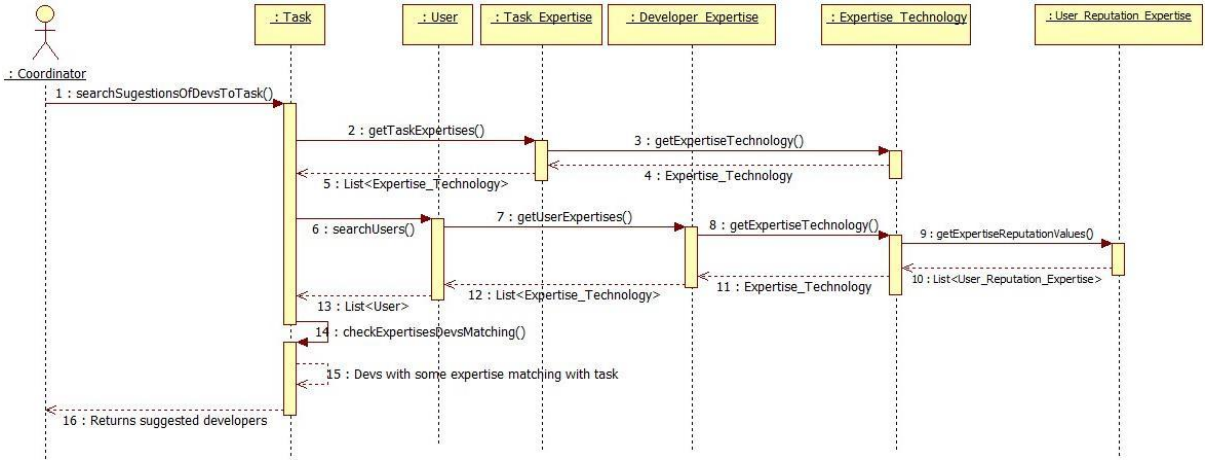
Os valores de reputação e esquecimento das *expertises* dos desenvolvedores são utilizados para estabelecer a ordenação de cada sugestão apresentada tanto na listagem de sugestões de tarefas (Figura 3.6) quanto para a listagem de sugestões de desenvolvedores (Figura 3.7). As sugestões que apresentam os maiores valores de reputação e os menores valores de esquecimento são posicionadas nas primeiras colocações da lista de sugestões, e estas variam de posição conforme são filtradas por esses valores separadamente.

Figura 3.6 – Diagrama de sequência da obtenção de sugestões de tarefas para um desenvolvedor.



Fonte: Elaborado pelo autor (2020).

Figura 3.7 – Diagrama de sequência da obtenção de sugestões de desenvolvedores para uma tarefa.



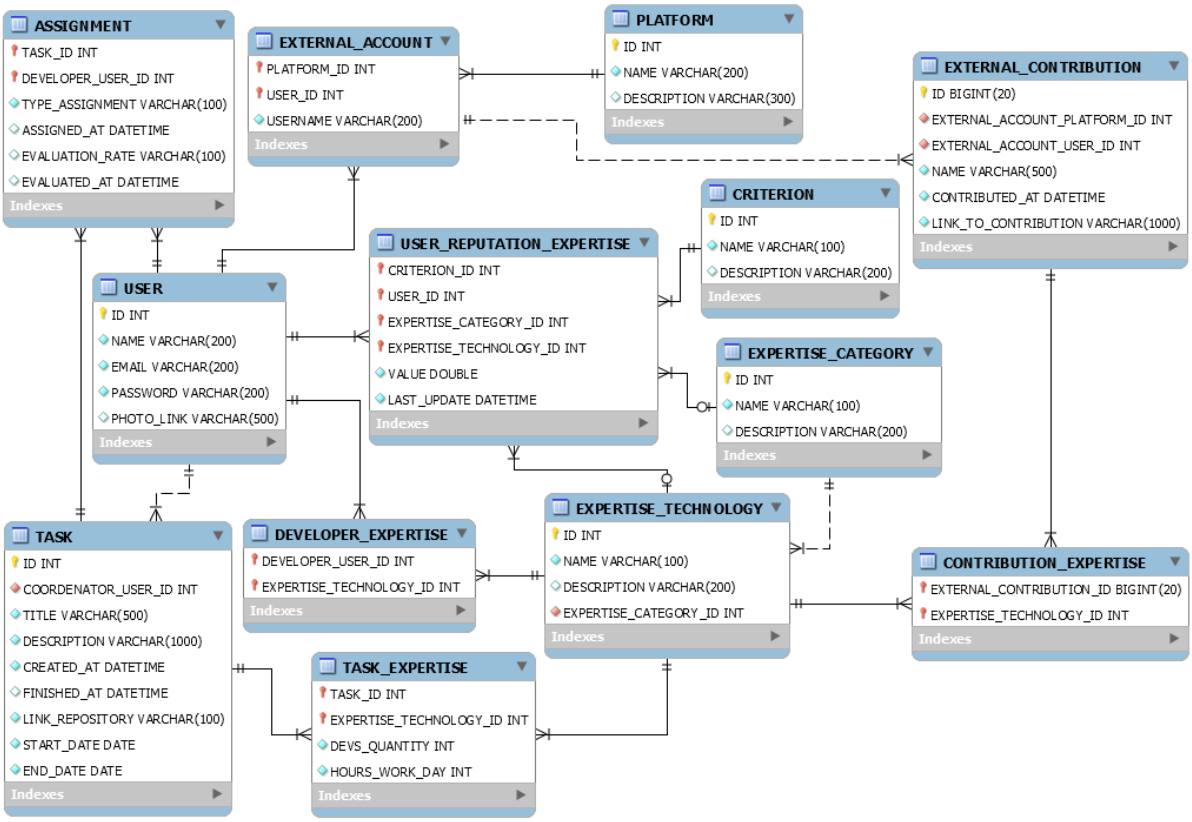
Fonte: Elaborado pelo autor (2020).

Um desenvolvedor ao receber sugestões de tarefas adequadas às suas *expertises* e desejando trabalhar em alguma dessas tarefas, tem a opção de solicitar ao coordenador participar na respectiva tarefa. Um coordenador ao receber sugestões de desenvolvedores para trabalharem em uma de suas tarefas tem a opção de convidar qualquer um deles para trabalhar na respectiva tarefa. Tanto o coordenador que recebe solicitações de desenvolvedores quanto o desenvolvedor que recebe convites de coordenadores, podem ambos escolher aceitar ou rejeitar a solicitação ou o convite respectivos. Quando as solicitações ou convites são aceitos, é formada a equipe entre o coordenador e os desenvolvedores atribuídos que vão trabalhar na tarefa referente. As subseções (3.5.3, 3.5.4, 3.5.5 e 3.5.6) apresentam mais detalhes desse processo de formação de equipes de desenvolvedores para as tarefas dos coordenadores.

3.4.2 Modelo do banco de dados

O banco de dados relacional da solução proposta foi elaborado em conformidade com os requisitos funcionais estabelecidos e seu projeto lógico se encontra apresentado na Figura 3.8.

Figura 3.8 – Projeto lógico do banco de dados.



Fonte: Elaborado pelo autor (2020).

As atribuições entre desenvolvedores (tabela *User*) e tarefas (tabela *Task*) são registradas na tabela *Assignment* das quais representam as contribuições dos desenvolvedores mediante suas *expertises* associadas às das tarefas (tabela *Task_Expertise*). As tabelas *External_Account* e *Platform* são responsáveis por registrar o nome de usuário das plataformas (*GitHub*, *StackOverflow*, *TopCoder*) que os desenvolvedores possuem registro de contribuições externas. E as APIs de cada uma dessas plataformas são utilizadas para retornar os registros dessas contribuições através desse nome de usuário. A tabela *External_Contribution* armazena essas contribuições externas, retornadas através das APIs das plataformas, e associa-as ao usuário desenvolvedor registrado no sistema. As *expertises* dessas contribuições são registradas através da tabela *Contribution_Expertise*, associando-as ao registro local de *expertises* nas tabelas *Expertise_Technology* e *Expertise_Category*. A

tabela *User_Reputation_Expertise* é responsável por registrar os valores dos critérios de reputação e esquecimento de cada *expertise* associada ao usuário.

3.5 IMPLEMENTAÇÃO

A solução proposta foi implementada através de um serviço Web que segue o padrão MVC (*Model-View-Controller*), cujo código-fonte desenvolvido se encontra disponível através deste link[7]. Esta implementação armazena no banco de dados relacional os registros de usuários e tarefas, ambos associados com *expertises*. Também há o registro dos valores de reputação e esquecimento que associam desenvolvedores às *expertises* trabalhadas. Esses valores são gerados com base no histórico de contribuições do desenvolvedor em cada *expertise* de conhecimentos tecnológicos aplicados para o desenvolvimento de software.

Foi obtida a listagem de categorias e suas tecnologias através do link[8] proveniente do trabalho de DE NEIRA *et al*. (2018) para uso como registro das *expertises* na base de dados, mais especificamente para as tabelas *Expertise_Technology* e *Expertise_Category* (Figura 3.8). Durante o desenvolvimento do presente trabalho essa listagem foi atualizada com novos registros de *expertises* de tecnologia adquiridos através da API[9] do *TopCoder*, que é por onde também os autores obtiveram a listagem original dessas tecnologias. E essas tecnologias foram categorizadas conforme a associação existente entre categorias e tecnologias na listagem original. A listagem atualizada de categorias com suas respectivas tecnologias associadas encontra-se disponível no Apêndice B.

As subseções a seguir detalham cada um dos seis subprocessos ilustrados resumidamente na Figura 3.3 do processo principal do sistema desenvolvido, e que foram desenvolvidos para atenderem aos requisitos funcionais definidos na subseção (3.1.1).

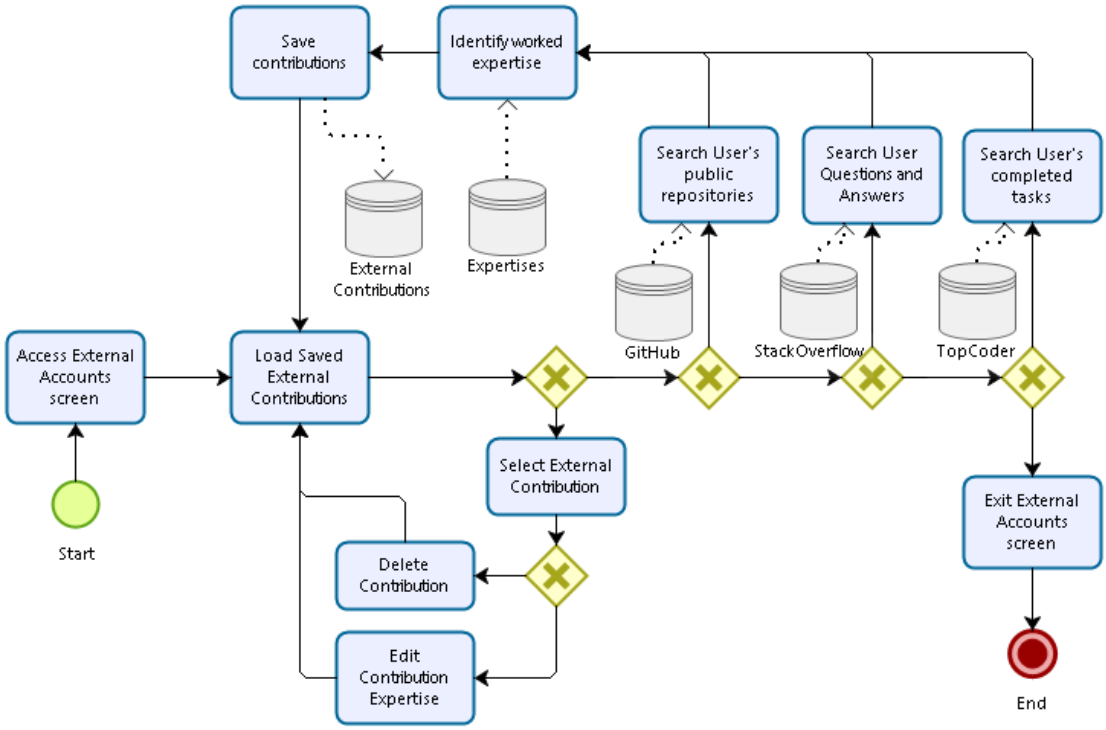
3.5.1 Formação do perfil de *expertises*

Na elaboração do perfil do desenvolvedor, tem-se a opção de registro de acesso às plataformas externas em que o respectivo desenvolvedor possua conta. Este, ao disponibilizar seu *username* do *GitHub*[10], seu *Id* do *StackOverflow*[11], ou seu *handle* do *TopCoder*[12] permite ao sistema acessar dados de suas respectivas contas disponíveis publicamente pelas próprias plataformas através de suas respectivas APIs. Com os dados disponíveis de contribuições históricas, associados a algum registro de *expertise* (por meio de *tags*, *skills*, ou nome de

[7] https://github.com/nathan2m/dissertacao/tree/main/projeto
[8] https://zenodo.org/record/1169411
[9] https://api.topcoder.com/v2/data/technologies
[10] https://developer.github.com/v3/
[11] https://api.stackexchange.com/
[12] https://tcapi.docs.apiary.io/

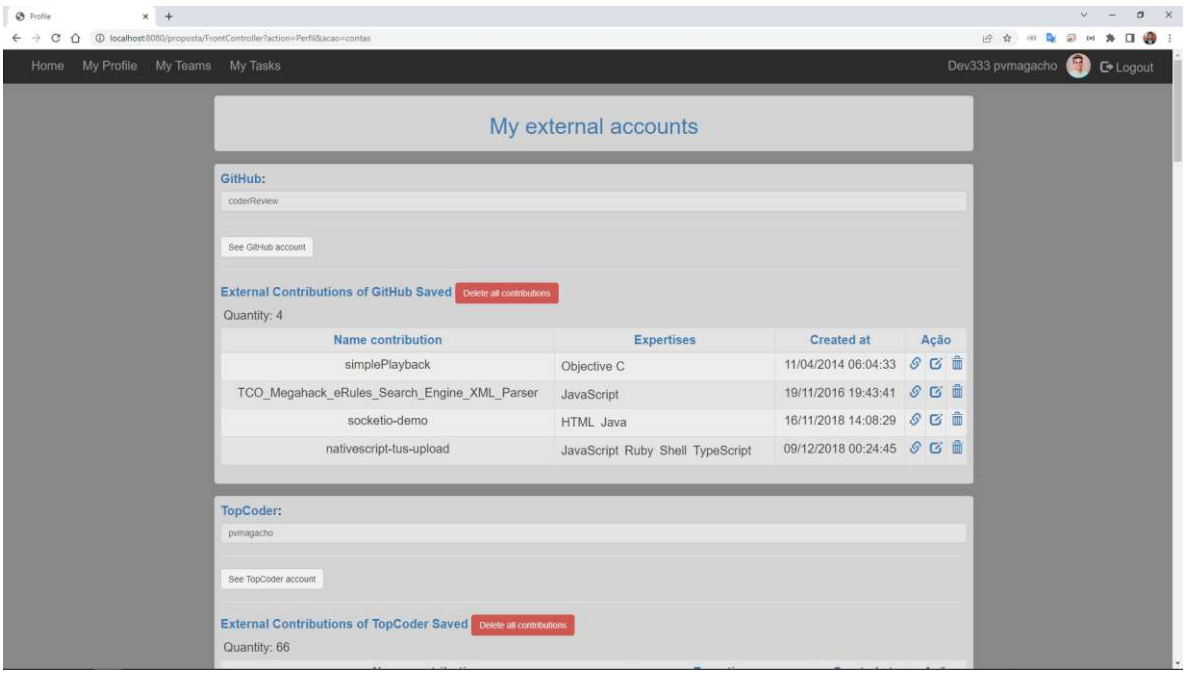
linguagens usadas nos repositórios), faz-se o salvamento dessas contribuições externas para registro de quais tecnologias (*expertises*) de desenvolvimento de software o desenvolvedor tem trabalhado.

Figura 3.9 – Diagrama BPMN do processo de formação do perfil de *expertises* do desenvolvedor.



Fonte: Elaborado pelo autor (2021).

Figura 3.10 – Tela por onde ocorrem as extrações das contribuições externas.



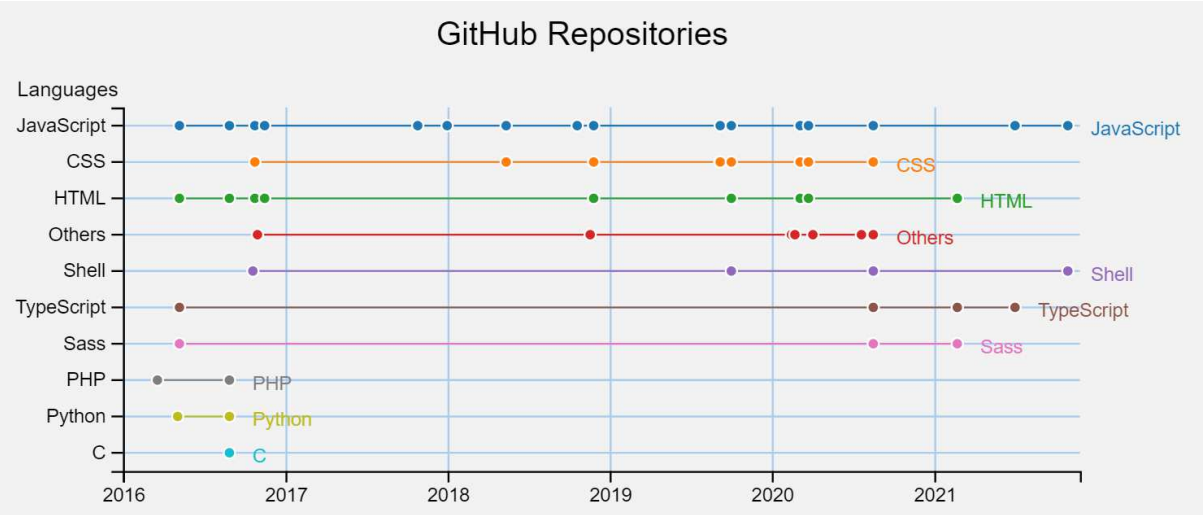
Fonte: Elaborado pelo autor (2021).

A Figura 3.9 resume o processo de busca dos dados de contribuições externas de um desenvolvedor em diferentes plataformas de apoio ao desenvolvimento de software, para com isso elaborar o perfil de conhecimentos em *expertises* do desenvolvedor. E a Figura 3.10 mostra a tela por onde o desenvolvedor executa esse processo de extração de suas contribuições externas das plataformas.

Com o objetivo de ajudar na visualização dos registros existentes de contribuições externas de um desenvolvedor e compará-lo com os demais, há a opção de exibir através de um gráfico essas contribuições efetuadas por *expertise* ao longo do tempo. O gráfico é gerado por meio de uma biblioteca[13] de criação de gráficos que permite a visualização de diferentes tipos de informação em possibilidades alternadas para facilitar a interpretação dos dados e auxiliar na tomada de decisão. Outros gráficos de visualização apresentados nas subseções seguintes também fazem uso dessa mesma biblioteca.

A Figura 3.11 mostra um exemplo deste gráfico no qual cada círculo preenchido representa cada contribuição de um desenvolvedor no *GitHub* e posicionados em ordem de data de criação da esquerda para a direita. Essas contribuições são repositórios públicos criados pelo respectivo desenvolvedor, e cada *expertise* é referente a esses repositórios. O mesmo gráfico pode ser utilizado para exibir as contribuições dos desenvolvedores em outras plataformas (*StackOverflow*, *TopCoder*) e também em tarefas locais no sistema.

Figura 3.11 – Gráfico exibindo as linguagens (*expertises*) utilizadas nos repositórios criados por um usuário do *GitHub*.



Fonte: Elaborado pelo autor (2021).

Com os dados das contribuições externas em *expertises*, salvos localmente no sistema da solução proposta, os valores de reputação e esquecimento podem ser gerados para

[13] http://d3js.org

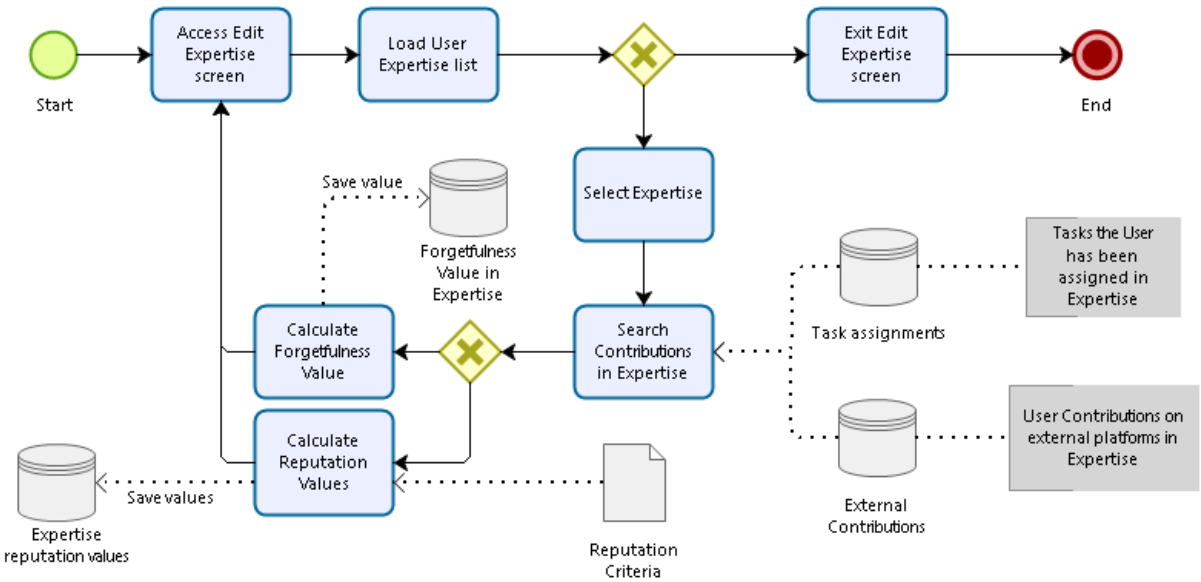
caracterizar melhor o estado atual de conhecimento do desenvolvedor nas *expertises* dos quais possui contribuições.

3.5.2 Geração dos valores de reputação e esquecimento

Os valores de cada um dos critérios de reputação e do critério de esquecimento são gerados com base nos dados de contribuições externas e das atribuições dos desenvolvedores em tarefas para cada *expertise* diferente de conhecimento em tecnologias de desenvolvimento de software. Esses valores devem ser atualizados conforme os desenvolvedores vão registrando no sistema novas contribuições de suas *expertises* advindas das suas respectivas contas nas plataformas (*GitHub*, *StackOverflow*, *TopCoder*) e também vêm sendo atribuídos em novas tarefas.

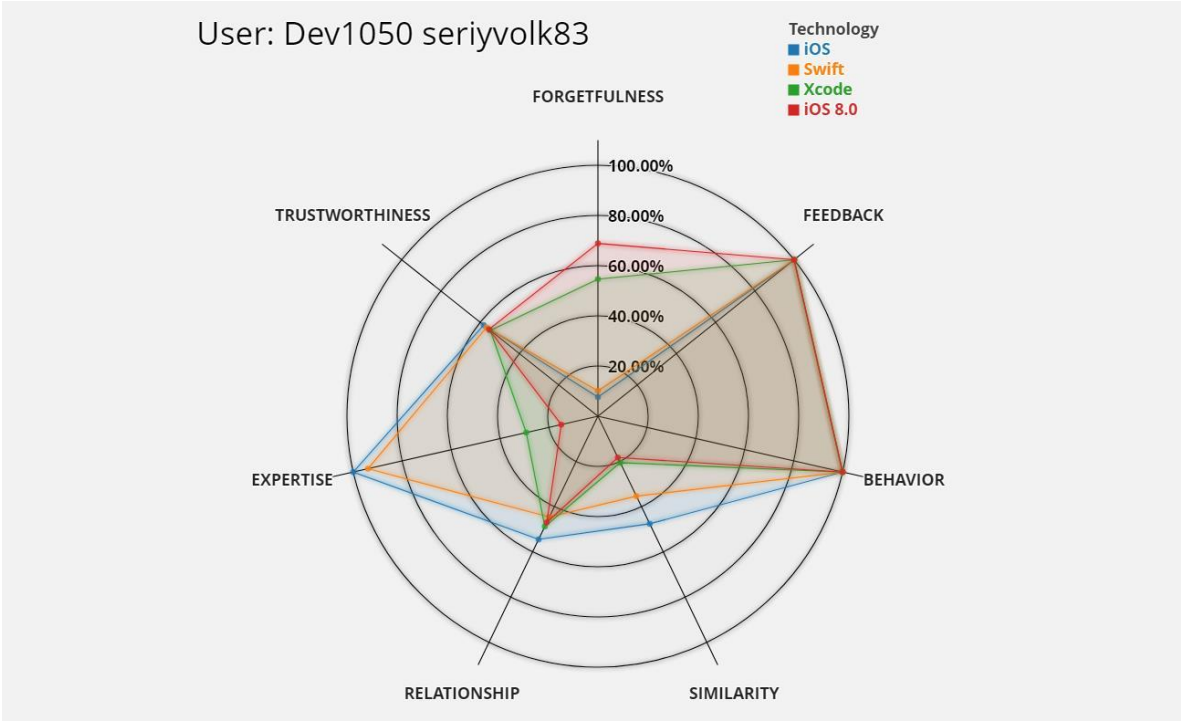
O desenvolvedor tem a opção de atualizar manualmente os valores de reputação e esquecimento em suas *expertises* às quais possui contribuições registradas e atribuições em tarefas. Esse processo é ilustrado na Figura 3.12 na qual o desenvolvedor acessa no sistema junto ao seu perfil de *expertises*. Ao escolher atualizar seus valores de reputação e esquecimento, seus respectivos dados de contribuições e atribuições são utilizados para gerar os novos valores dos critérios de reputação e esquecimento.

Figura 3.12 – Diagrama BPMN do processo de geração dos valores de reputação e esquecimento.



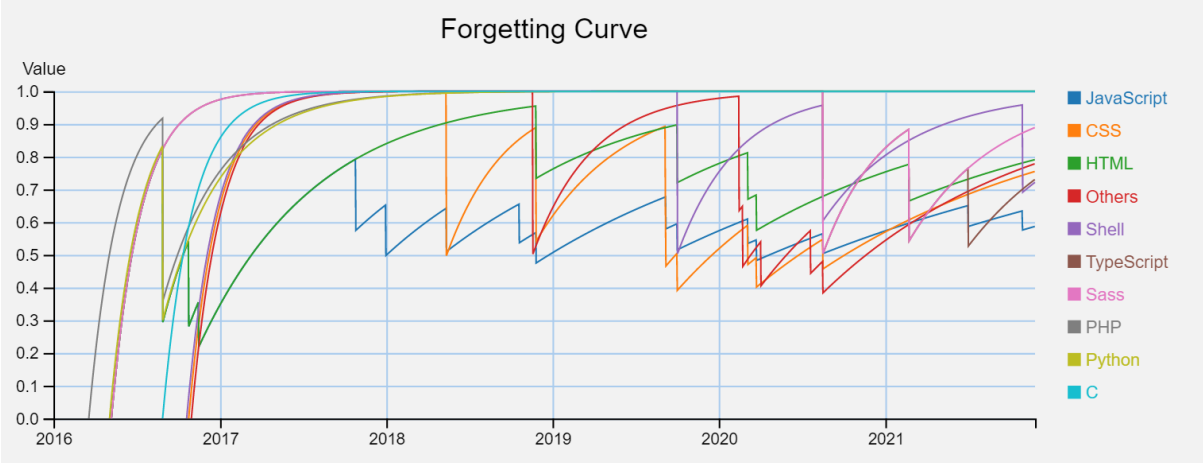
Fonte: Elaborado pelo autor (2021).

Figura 3.13 – Exemplo de gráfico de radar com valores de reputação e esquecimento de um desenvolvedor.



Fonte: Elaborado pelo autor (2021).

Figura 3.14 – Gráfico exibindo o aumento gradual do esquecimento nas *expertises*.



Fonte: Elaborado pelo autor (2021).

Os valores gerados de reputação e esquecimento são exibidos em um gráfico de radar (Figura 3.13) para facilitar a comparação dos valores de cada critério em cada *expertise* do perfil de um desenvolvedor. Nesse gráfico, os valores dos critérios de reputação por já estarem normalizados são exibidos variando entre 0.0 (0%) e 1.0 (100%). Conforme explicado na subseção (3.3.2) o valor do critério de Esquecimento é exibido de forma inversa ao valor da Retenção de Memória da equação (3.8), sendo que valores próximos de 0.0

significam baixo esquecimento (alta retenção de memória) e valores próximos de 1.0 significam alto esquecimento (baixa retenção de memória).

O gráfico da Figura 3.14 exemplifica a variação do valor de esquecimento conforme ao longo do tempo ocorrem as contribuições do desenvolvedor nas *expertises* em que este trabalha. No respectivo gráfico é observado o reforço do aprendizado nos conhecimentos aplicado pela equação (3.8) diante da ocorrência de novas interações em cada uma das *expertises* trabalhadas. O esquecimento aumenta conforme o desuso da *expertise*, e diminui diante da ocorrência de novas contribuições na respectiva *expertise*; evidenciando assim haver um uso ativo dos conhecimentos dessa expertise por parte do desenvolvedor.

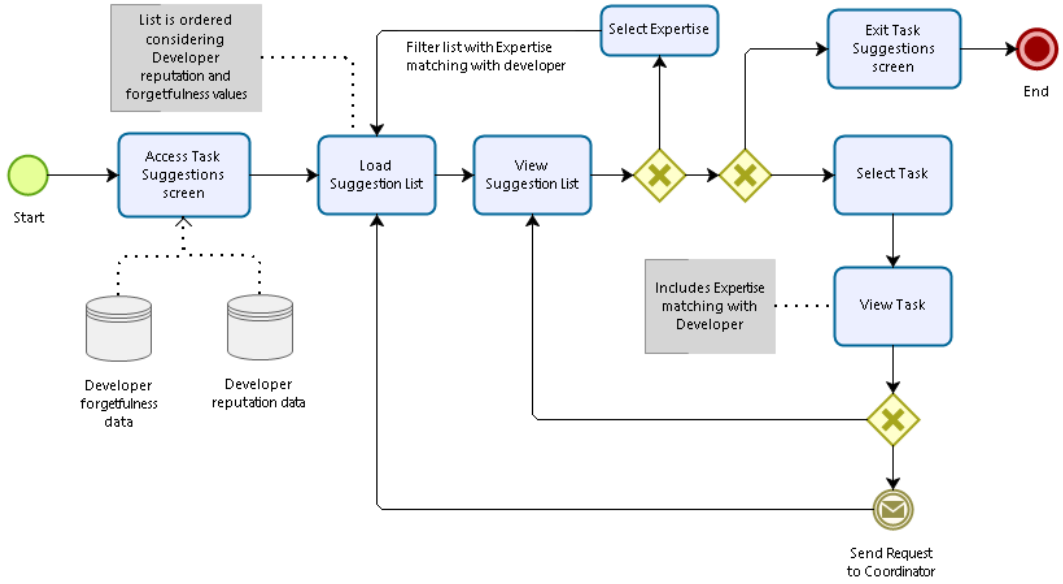
O gráfico da Figura 3.14 é uma aplicação do esquecimento sobre as contribuições exibidas no gráfico da Figura 3.11. E nesses gráficos nota-se que o desenvolvedor do *GitHub* possui uma quantidade maior de repositórios (contribuições) criados na linguagem (*expertise*) JavaScript. A cada nova interação (nova contribuição) efetuada, há um decréscimo na curva do esquecimento para exibir o reforço de memória ocorrido na *expertise*, e a ascensão natural do esquecimento passa a ser menor do que antes. Havendo poucas interações e estas sendo antigas, o conhecimento retido tende à zero por causa do esquecimento aplicado. Nesse exemplo do usuário do *GitHub*, este não possui mais relevância para com as linguagens C, Python e PHP por ter interagido apenas algumas vezes no passado e com isso haver alto esquecimento (baixa retenção de memória) nessas *expertises*.

3.5.3 Busca de sugestões de tarefas

Havendo o desenvolvedor formado o seu perfil de *expertises* com os registros de suas contribuições em desenvolvimento de software, e os valores de reputação e esquecimento gerados, tem-se a possibilidade dele buscar sugestões de tarefas. Tarefas essas que condizem com o estado atual de conhecimento do desenvolvedor, priorizando aquelas cujas *expertises* o desenvolvedor possua os maiores valores de reputação e os menores valores de esquecimento.

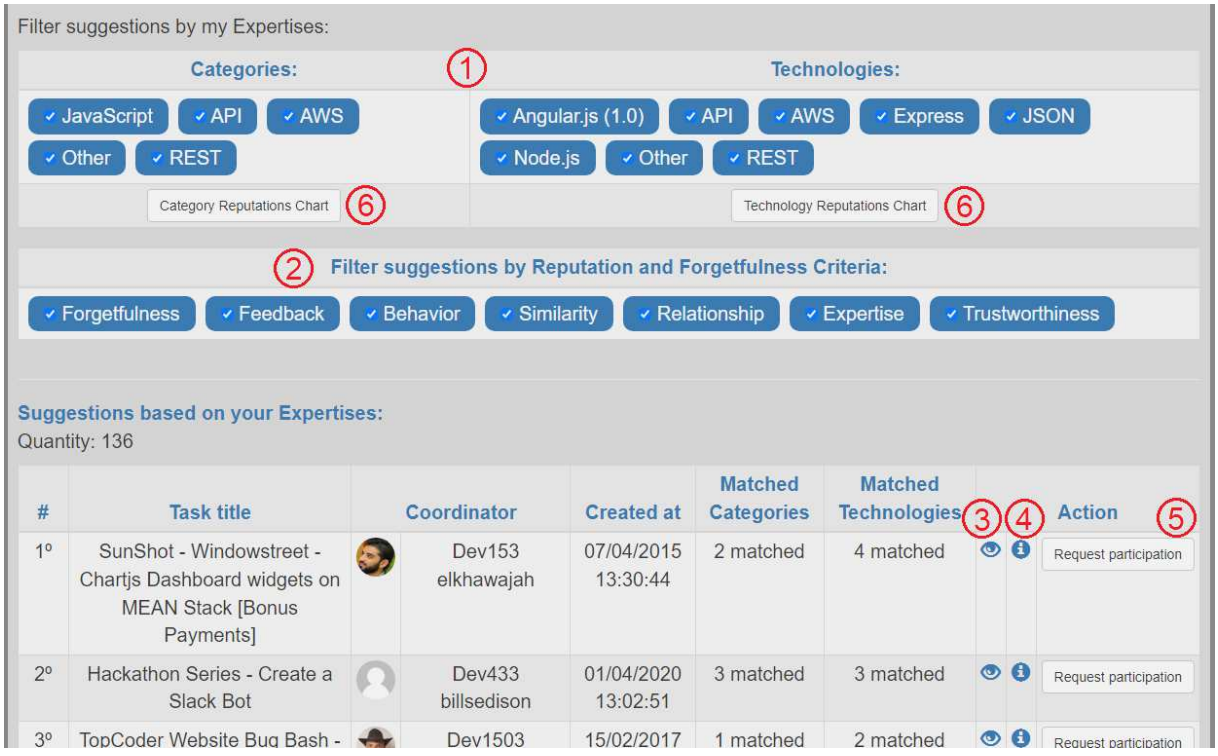
Essa busca das tarefas condizentes com os conhecimentos atuais do desenvolvedor segue o procedimento do diagrama de sequência da Figura 3.6 e seu processo está ilustrado na Figura 3.15. Este processo começa com a obtenção dos valores salvos de reputação e esquecimento do desenvolvedor em seus conhecimentos trabalhados, e segue com o carregamento das sugestões de tarefas com as mesmas *expertises* (categorias e tecnologias) registradas em seu perfil. Essa listagem de tarefas sugeridas é ordenada pelos maiores valores de reputação e menores valores de esquecimento, e também pelas maiores quantidades de *expertises* de categorias e tecnologias em comum com o desenvolvedor.

Figura 3.15 – Diagrama BPMN do processo de busca de sugestões de tarefas para desenvolvedor.



Fonte: Elaborado pelo autor (2021).

Figura 3.16 – Listagem de sugestões de tarefas para um desenvolvedor.



Fonte: Elaborado pelo autor (2021).

A Figura 3.16 exibe um exemplo dessa listagem de sugestões de tarefas para um desenvolvedor. Ao visualizar a listagem de tarefas sugeridas, o desenvolvedor tem a opção de filtrá-la por cada *expertise* (1), seja de Tecnologia ou de Categoria. Também há disponível a opção filtrar por cada critério de reputação e esquecimento (2). Cada tarefa dessa listagem é

acompanhada de uma opção para o desenvolvedor visualizar as informações da respectiva tarefa (3), visualizar também as *expertises* que têm em comum com a tarefa (4) e por fim um botão para solicitar ao coordenador participação na tarefa (5). Os botões referenciados pelo item (6) referem-se à opção de visualização por gráfico de radar (Figura 3.13) dos valores de reputação e esquecimento nas *expertises* do perfil do desenvolvedor.

Após o desenvolvedor visualizar as tarefas sugeridas e identificar aquelas que mais condizem com seu estado atual de conhecimento e depois solicitar participação aos coordenadores dessas tarefas, pode-se então prosseguir para o processo da entrada do desenvolvedor em equipe de tarefa.

3.5.4 Busca de sugestões de desenvolvedores

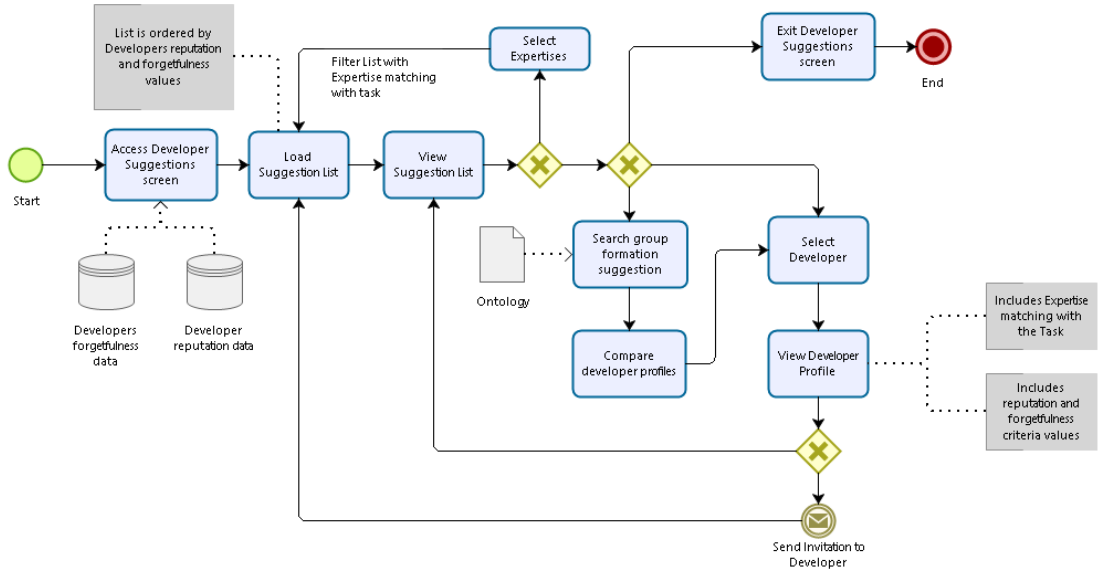
O usuário no papel de coordenador é aquele que elabora as tarefas para os desenvolvedores resolverem com seus conhecimentos técnicos (*expertises*). As tarefas criadas possuem título e descrição, juntamente com as *expertises* correlacionadas. O coordenador tem a possibilidade de buscar as sugestões de desenvolvedores a serem atribuídos para cada tarefa criada. Desenvolvedores esses que, em seus respectivos perfis, possuem registros de contribuições nos conhecimentos de desenvolvimento de software exigidos pela tarefa.

Essa busca de desenvolvedores que tenham alta reputação e baixo esquecimento em suas *expertises* segue o procedimento do diagrama de sequência da Figura 3.7 e seu processo está ilustrado na Figura 3.17. Esse processo começa com a busca pelos desenvolvedores que possuem em seus perfis as mesmas *expertises* que a tarefa exige, e segue com o carregamento dos valores de reputação e esquecimento dos respectivos desenvolvedores nesses conhecimentos. A listagem de desenvolvedores sugeridos é ordenada pelos maiores valores de reputação e menores valores de esquecimento, e também pela maior quantidade de *expertises* em comum com a tarefa.

A Figura 3.18 exibe um exemplo dessa listagem de sugestões de desenvolvedores para uma tarefa. Ao visualizar a listagem de desenvolvedores sugeridos, o coordenador tem a opção de filtrá-la por cada *expertise* (1), seja de Tecnologia ou de Categoria. Também há disponível a opção de filtrar por cada critério de reputação e esquecimento (2). Cada desenvolvedor dessa listagem é acompanhado de uma opção para o coordenador visualizar o respectivo perfil deste (3), visualizar também as *expertises* que tem em comum com a tarefa (4) e por fim um botão para convidar o desenvolvedor para participar na tarefa (5). O botão "gráfico" (6) refere-se à opção de visualização por gráfico de radar (Figura 3.13) dos valores de reputação e esquecimento nas *expertises* do desenvolvedor em comum com a tarefa. O

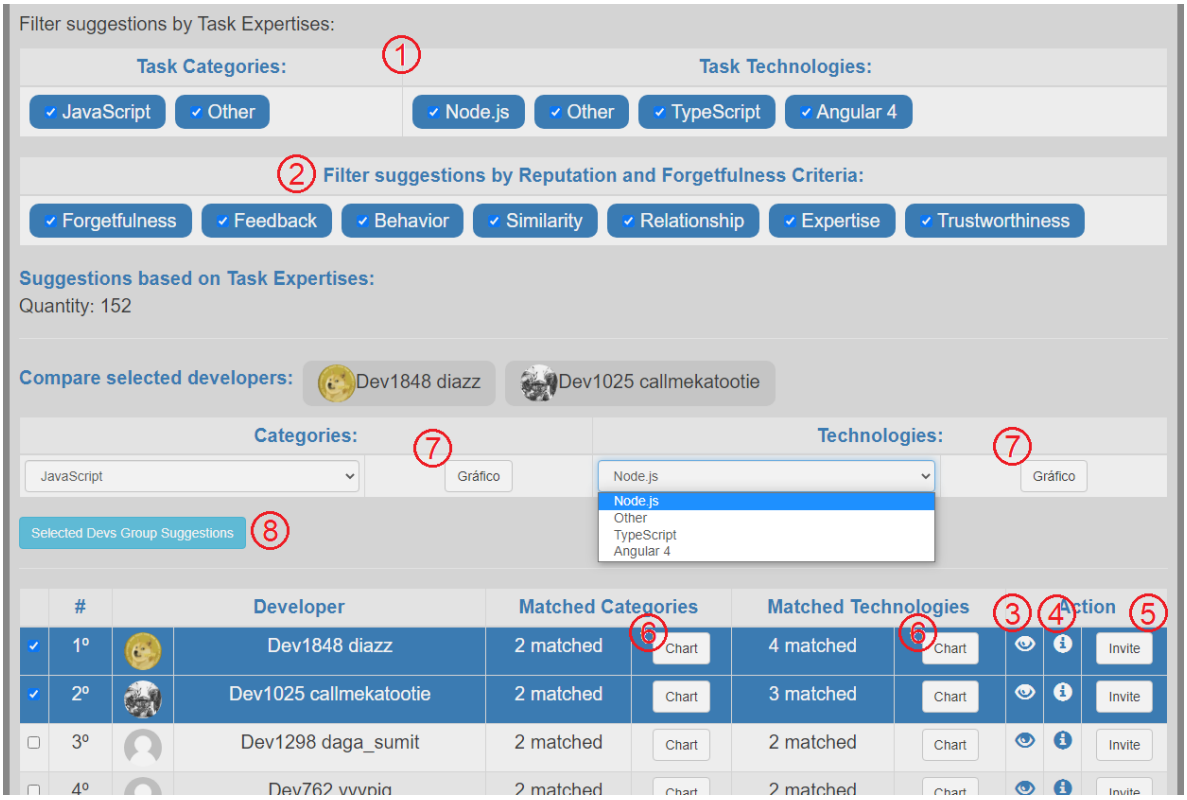
outro botão “gráfico” (7) é também um gráfico de radar, mas que serve para comparação dos valores de reputação e esquecimento entre desenvolvedores selecionados da lista para uma mesma *expertise*.

Figura 3.17 – Diagrama BPMN do processo de busca de sugestões de desenvolvedores para tarefa.



Fonte: Elaborado pelo autor (2021).

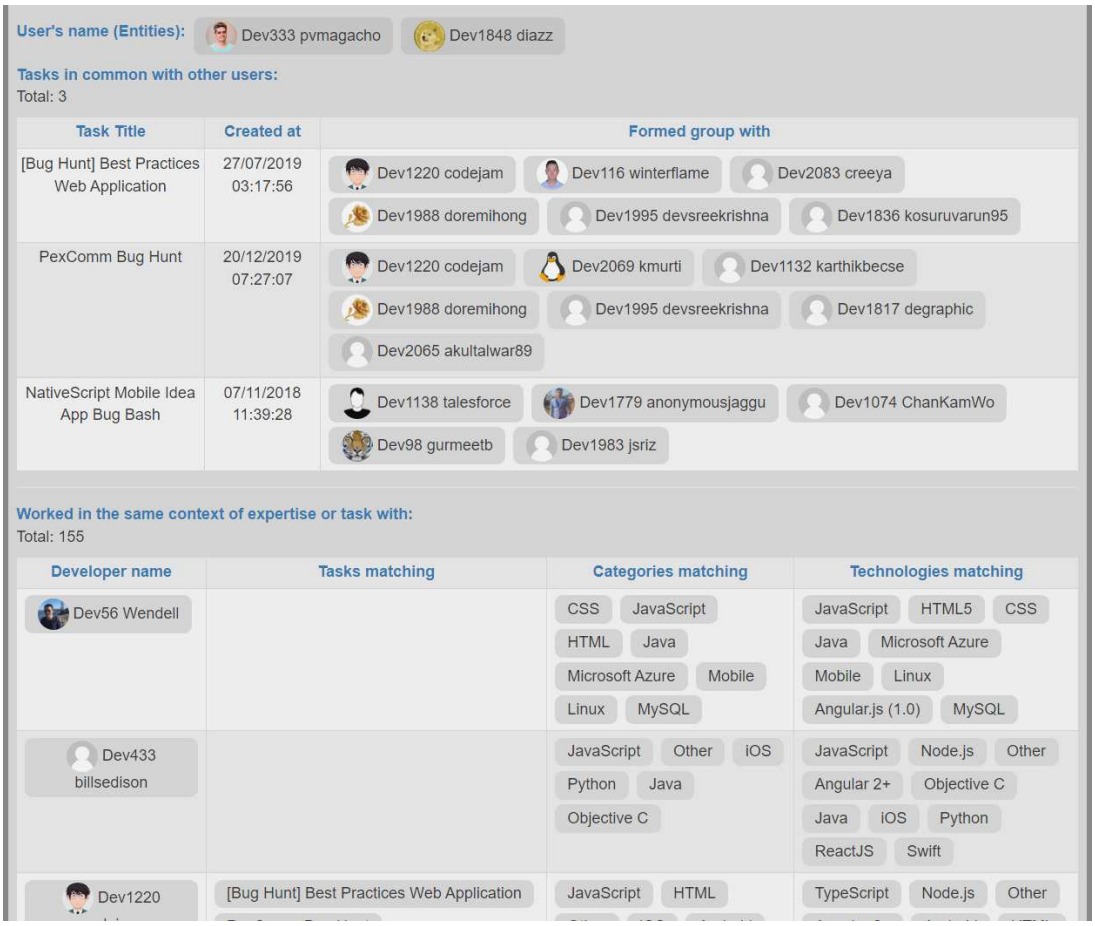
Figura 3.18 – Listagem de sugestões de desenvolvedores para uma tarefa.



Fonte: Elaborado pelo autor (2021).

O perfil do desenvolvedor contém informações quanto às suas tarefas trabalhadas, contribuições externas, contas nas plataformas de desenvolvimento de software (*GitHub*, *StackOverflow*, *TopCoder*) e os valores de reputação e esquecimento em suas *expertises*. A visualização dessas informações de contribuições nas *expertises* do desenvolvedor é complementada com o uso do gráfico de contribuições (Figura 3.11), o gráfico de radar (Figura 3.13) para os valores de reputação e esquecimento nas *expertises*, e o gráfico do esquecimento (Figura 3.14).

Figura 3.19 – Listagem de desenvolvedores sugeridos para formar grupo com os desenvolvedores analisados.



Fonte: Elaborado pelo autor (2021).

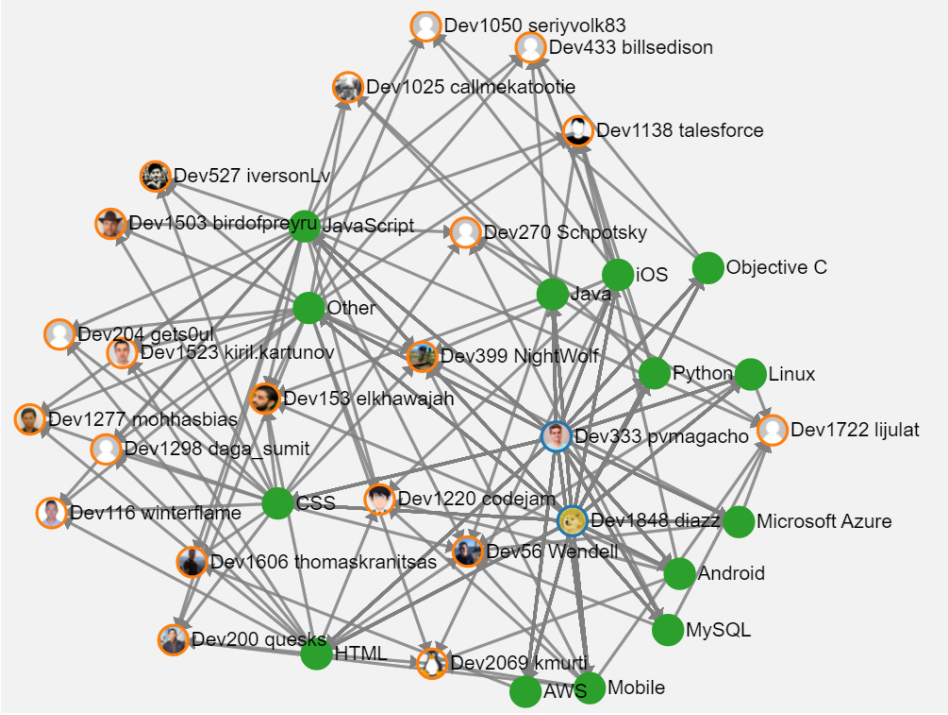
A ontologia *ArchiRiOnt* é utilizada para a busca de sugestões de grupos entre desenvolvedores que já tenham trabalhado nas mesmas tarefas no passado e também possuam as mesmas *expertises* (tecnologias e categorias) em seus perfis. Ainda na Figura 3.18, ao selecionar os desenvolvedores da lista (destacados em azul) e clicar no botão “Sugestões de Grupos” (8) o coordenador acessa uma nova página (Figura 3.19) que exibe uma lista de desenvolvedores que possuem as tarefas, categorias e tecnologias em comum com os desenvolvedores previamente selecionados. Essas tarefas em comum são listadas

separadamente para mostrar quais os desenvolvedores que trabalharam em cada uma dessas tarefas assim como os desenvolvedores sendo analisados.

Uma visualização de grafos é gerada para mostrar as relações entre os desenvolvedores selecionados (entidades) e os listados considerando as tarefas, categorias e tecnologias em comum entre as partes. Nessa visualização o foco é observar a quantidade de relações existentes através de tarefas, categorias e tecnologias em comum entre cada desenvolvedor analisado e cada desenvolvedor sugerido para formarem grupo um com o outro. A Figura 3.20 mostra um exemplo dessa visualização entre desenvolvedores analisados (nós azuis) e sugeridos (nós laranjas) através das categorias de *expertises* (nós verdes). Essa mesma visualização pode ser também utilizada para ver as tecnologias e as tarefas em comum entre os desenvolvedores.

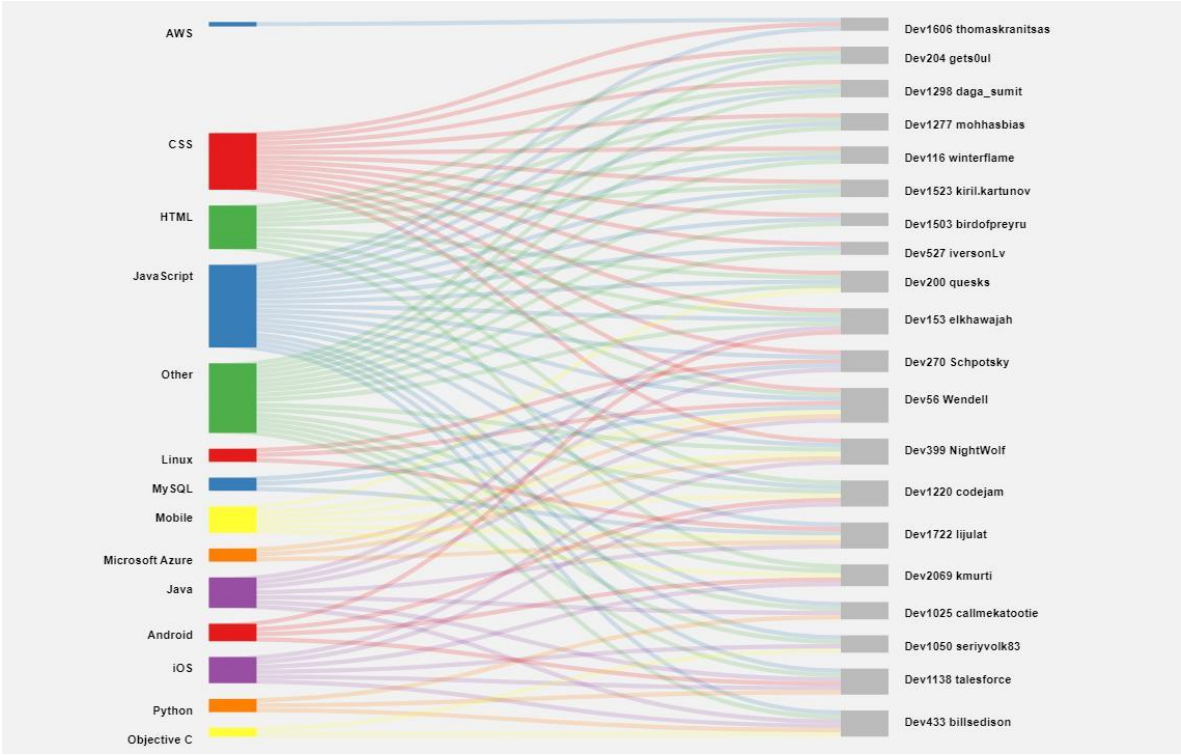
Um gráfico de *Sankey* é gerado para visualizar o agrupamento das relações existentes entre (tarefas, categorias, tecnologias) junto dos desenvolvedores analisados e sugeridos. A Figura 3.21 exibe um exemplo desse gráfico mostrando as relações entre categorias de *expertise* e os desenvolvedores, a fim de visualizar quais as *expertises* em comum entre os desenvolvedores. Esse mesmo gráfico pode ser também utilizado para visualizar as tecnologias e as tarefas em comum entre os desenvolvedores.

Figura 3.20 – Visualização por grafos das relações de categorias de *expertises* em comum entre os desenvolvedores analisados e sugeridos.



Fonte: Elaborado pelo autor (2021).

Figura 3.21 – Gráfico de *Sankey* para visualização das categorias de tecnologias (*expertises*) em comum entre desenvolvedores.



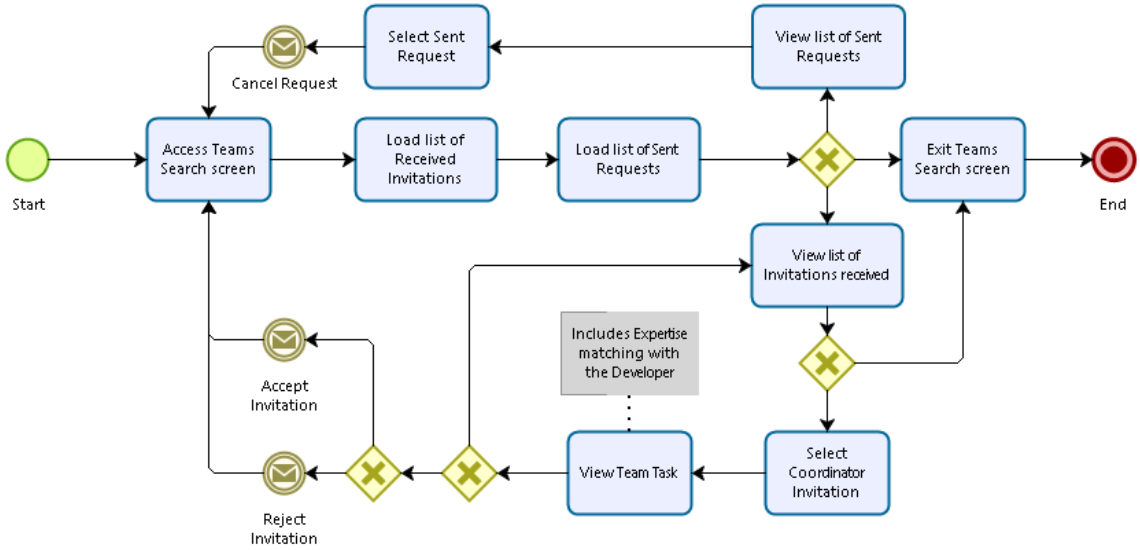
Fonte: Elaborado pelo autor (2021).

3.5.5 Desenvolvedor entrar em equipe

O desenvolvedor possui duas formas de entrar em uma equipe de uma tarefa de um coordenador. Seja o desenvolvedor solicitando ao coordenador participar em uma de suas tarefas, ou então o desenvolvedor sendo convidado pelo coordenador para desenvolver na tarefa em suas competências. Esse processo se encontra ilustrado na Figura 3.22 no qual o usuário desenvolvedor inicialmente visualiza os convites que recebeu de coordenadores para trabalhar em suas tarefas, e também a lista de solicitações que enviou para coordenadores de outras tarefas (Figura 3.23).

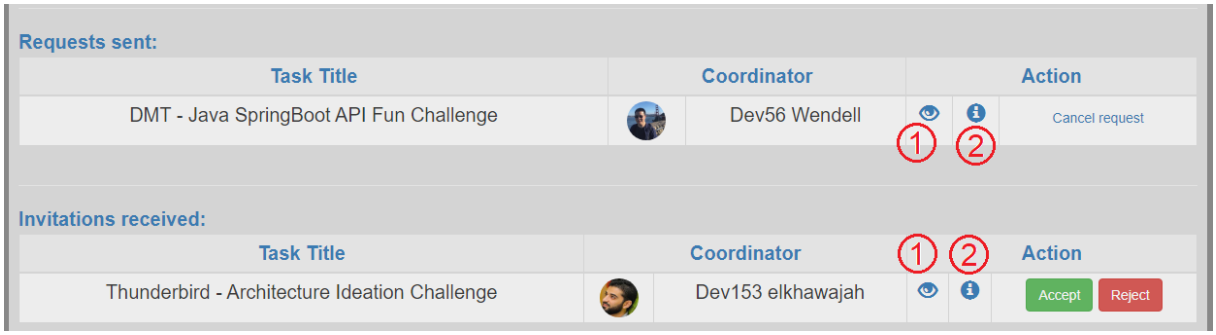
Para cada solicitação enviada o desenvolvedor tem a opção de cancelar o envio dessa, caso o coordenador ainda não tenha aceitado ou rejeitado a solicitação. Para cada convite recebido o desenvolvedor tem a opção de aceitar ou rejeitar o convite, caso o coordenador não tenha ainda cancelado o envio deste. Em ambas as listagens (Figura 3.23) o desenvolvedor tem a opção de através de um botão (1) visualizar os dados da tarefa (título, descrição, coordenador, *expertises* exigidas, etc...). Nessas duas listagens o desenvolvedor também tem a opção de, mediante outro botão (2), visualizar as *expertises* que possui em comum com a respectiva tarefa.

Figura 3.22 – Diagrama BPMN do processo de formação de equipe sob o ponto de vista do desenvolvedor.



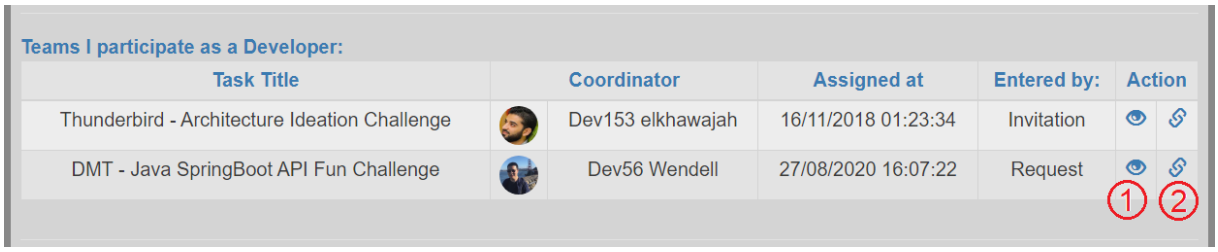
Fonte: Elaborado pelo autor (2021).

Figura 3.23 – Listagens de solicitações enviadas e de convites recebidos por um desenvolvedor.



Fonte: Elaborado pelo autor (2021).

Figura 3.24 – Listagem de tarefas em que um desenvolvedor participa como membro de equipe.



Fonte: Elaborado pelo autor (2021).

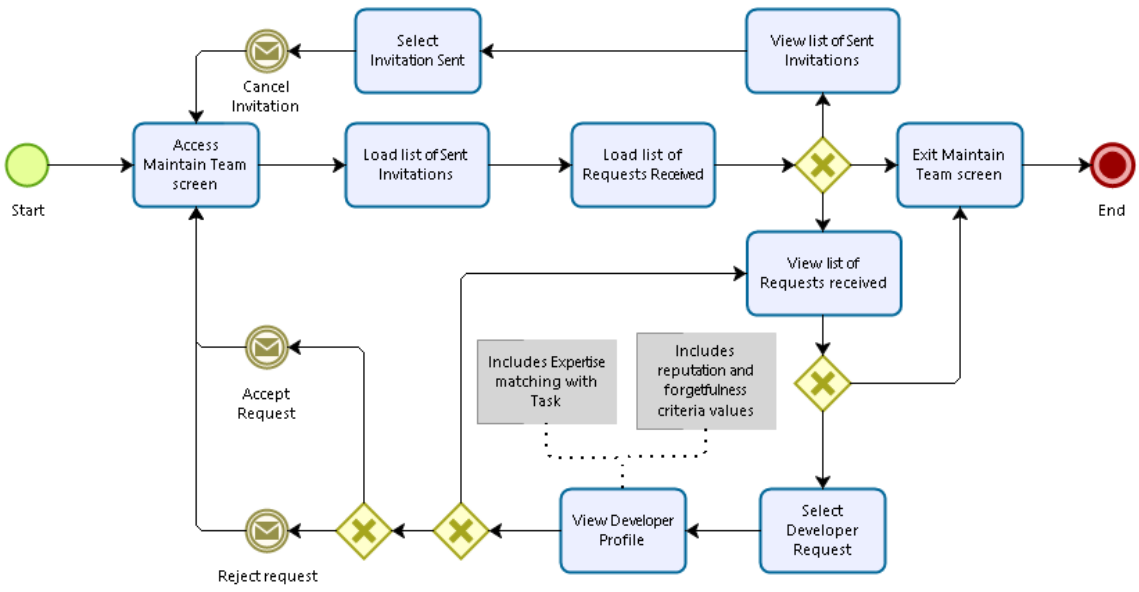
O desenvolvedor, ao aceitar os convites dos coordenadores ou então ter suas solicitações enviadas aceitas pelos coordenadores, passa a ser considerado como participante das equipes das respectivas tarefas. A Figura 3.24 é um exemplo de listagem das tarefas às quais o desenvolvedor foi aceito para participar como membro de equipe nelas. Nessa

listagem o desenvolvedor tem, através de um botão (1), a opção de visualizar os dados da tarefa (assim como na Figura 3.23) e também a opção, mediante o botão (2), de acessar o repositório de código fonte referente à respectiva tarefa. É nesse repositório que o desenvolvedor trabalha cumprindo com os objetivos da tarefa de desenvolvimento de software e exerce seus conhecimentos nas *expertises* exigidas pelo coordenador da tarefa. Cada tarefa listada é também acompanhada da data em que o respectivo desenvolvedor foi atribuído e a forma pelo qual entrou na equipe da tarefa (seja por convite do coordenador ou por solicitação ao coordenador). Caso o desenvolvedor queira sair da equipe, este deve comunicar ao coordenador solicitando que o remova da equipe.

3.5.6 Coordenador montar equipe de tarefa

Conforme já dito, o coordenador possui duas formas de atribuir desenvolvedores na equipe de suas tarefas. Seja o coordenador convidando o desenvolvedor para participar em uma de suas tarefas, ou então o coordenador recebendo solicitações de desenvolvedores dispostos a trabalharem em uma de suas tarefas. Esse processo se encontra ilustrado na Figura 3.25 no qual o usuário coordenador em uma de suas tarefas criadas visualiza as solicitações recebidas de desenvolvedores dispostos em trabalhar nessa tarefa, e também a lista dos convites que enviou para os desenvolvedores trabalharem na respectiva tarefa (Figura 3.26).

Figura 3.25 – Diagrama BPMN do processo de formação de equipe sob o ponto de vista do coordenador.



Fonte: Elaborado pelo autor (2021).

Para cada convite enviado o coordenador tem a opção de cancelar o envio desse, caso o desenvolvedor ainda não tenha aceitado ou rejeitado o convite. Para cada solicitação recebida

o coordenador tem a opção de aceitar ou rejeitar a solicitação, caso o desenvolvedor não tenha ainda cancelado o envio desta. Em ambas as listagens (Figura 3.26) o coordenador tem a opção de através de um botão (1) visualizar o perfil do respectivo desenvolvedor (nome, foto, *expertises* trabalhadas, etc...). Nessas duas listagens o coordenador também tem a opção de, mediante outro botão (2), visualizar as *expertises* que a tarefa possui em comum com o respectivo desenvolvedor.

Figura 3.26 – Listagens de convites enviados e de solicitações recebidas por um coordenador.

Fonte: Elaborado pelo autor (2021).

Figura 3.27 – Listagem de desenvolvedores participantes em uma tarefa de um coordenador.

Fonte: Elaborado pelo autor (2021).

A atribuição de desenvolvedores em uma tarefa permite a formação imediata da respectiva equipe e o coordenador desta já pode começar a gerenciá-la. O coordenador é responsável por incluir e excluir os participantes desenvolvedores e também por encerrar a tarefa, registrando-a como finalizada no banco de dados.

O coordenador em uma de suas tarefas que criou, ao aceitar as solicitações dos desenvolvedores ou então ter seus convites enviados aceitos pelos desenvolvedores, passa a considerar esses desenvolvedores como participantes na equipe de sua tarefa. A Figura 3.27 é um exemplo de listagem dos desenvolvedores aceitos a participarem como membros de equipe na tarefa. Nessa listagem o coordenador tem, através de um botão (1), a opção de visualizar o perfil do desenvolvedor (assim como na Figura 3.26) e também a opção, mediante o botão (2), visualizar as *expertises* às quais o desenvolvedor foi designado para trabalhar na respectiva tarefa. Cada desenvolvedor listado é também acompanhado da data de atribuição

na tarefa; a forma pelo qual entrou na equipe desta (seja mediante convite do coordenador ou por solicitação ao coordenador); e por fim uma opção de o coordenador remover o desenvolvedor da equipe.

3.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou uma solução proposta para apoiar a busca por desenvolvedores capacitados com conhecimentos técnicos para resolução de tarefas relacionadas ao Desenvolvimento Global de Software. Inicialmente, foram apresentados os requisitos e a arquitetura de projeto, e, em seguida, uma ontologia utilizada para apoio na formação de grupos conforme contextos de tarefas e *expertises*. Também foram apresentados os critérios de reputação e esquecimento com as respectivas fórmulas para geração de seus valores, e por fim, a implementação da solução com suas funcionalidades. O próximo capítulo mostra a avaliação da solução apresentada.

4 AVALIAÇÃO DA SOLUÇÃO

Este capítulo descreve a avaliação realizada sobre a solução proposta desenvolvida para uso da reputação e do esquecimento na sugestão de desenvolvedores mediante suas aptidões em tecnologias para desenvolvimento e manutenção de software. Uma breve introdução ao objetivo da avaliação é apresentada, seguindo do escopo definido, o planejamento detalhado, a execução da avaliação, e por fim, a análise e interpretação dos resultados.

4.1 INTRODUÇÃO DA AVALIAÇÃO

Existem dois cenários diferentes nos quais a solução desenvolvida pode ser avaliada. Um deles é o cenário de um projeto livre (aberto) nos quais os desenvolvedores se voluntariam para contribuir com o código fonte deste. O outro cenário diz respeito a um projeto restrito (fechado) a uma quantidade limitada de desenvolvedores, nos quais estes são atribuídos pelos coordenadores do projeto em tarefas conforme a especialidade (conhecimento técnico) de cada um.

A solução desenvolvida apresenta listagens de sugestões para atender aos dois cenários possíveis. Um deles para buscar desenvolvedores capacitados para determinada tarefa, e o outro para buscar tarefas condizentes com a capacidade atual de determinado desenvolvedor. Para a Avaliação, é considerado o contexto de projetos de software limitado aos usuários desenvolvedores que foram atribuídos nas tarefas de cada um desses projetos. Com isso a Avaliação tem por objetivo verificar, no contexto das tarefas de projeto de software, o quão equivalentes são as sugestões de desenvolvedores com as atribuições históricas ocorridas entre desenvolvedores e tarefas.

Essas listagens de sugestões de desenvolvedores são geradas considerando os valores dos critérios de reputação e esquecimento como forma de identificar o nível de capacidade dos desenvolvedores nos conhecimentos de tecnologias (*expertises*) em que trabalharam. Esses conhecimentos tecnológicos trabalhados pelos desenvolvedores podem ser identificados através de seu histórico de contribuições em: repositórios de código fonte no *GitHub*; perguntas e respostas no *StackOverflow*; tarefas de manutenção e desenvolvimento concluídas no *TopCoder*.

Pressupõe-se que o uso dessas listagens de sugestões seja capaz de ajudar coordenadores de tarefas a encontrarem desenvolvedores devidamente capacitados para trabalharem com os conhecimentos (*expertises*) tecnológicos exigidos para o desenvolvimento de software. E também ajudar desenvolvedores a encontrarem tarefas de desenvolvimento de

software relevantes para as suas capacidades de conhecimento (*expertises*) em tecnologias de desenvolvimento de software. Pretende-se então com a respectiva avaliação verificar a real utilidade da solução desenvolvida em identificar desenvolvedores capacitados para tarefas que exigem conhecimentos de desenvolvimento de software.

As etapas da Avaliação da Solução foram elaboradas seguindo as normas de WOHLIN *et al*. (2012). Começa primeiro com a definição do **Escopo** por meio de um arcabouço baseado no método GQM (BASILI *et al*., 1994), depois segue com o **Planejamento**, a **Execução da Avaliação**, e por fim **Análise e Interpretação dos Resultados**.

4.2 ESCOPO

O **objeto de estudo** da avaliação são as sugestões de desenvolvedores para cada tarefa de desenvolvimento de software analisada, considerando os valores dos critérios de reputação e esquecimento sobre as *expertises* trabalhadas pelos desenvolvedores.

O **propósito** da avaliação está em verificar a equivalência das sugestões de desenvolvedores com as atribuições históricas em cada tarefa analisada. Equivalência essa que diz respeito em verificar quais foram os desenvolvedores sugeridos que de fato foram atribuídos. Também verificar aqueles que foram sugeridos, mas não foram atribuídos na tarefa, e por fim aqueles que não foram sugeridos, mas por algum motivo foram atribuídos nessa tarefa.

O **foco** da avaliação está em analisar os valores dos critérios de reputação e esquecimento dos desenvolvedores, sob o **ponto de vista** das *expertises* exigidas em cada tarefa analisada dos quais esses desenvolvedores trabalharam.

O **contexto** abordado nessa avaliação são os projetos de desenvolvimento de software com suas contribuições de desenvolvimento recebidas de diversos desenvolvedores. Para essa avaliação tem-se por pretensão caracterizar as contribuições como sendo as tarefas de desenvolvimento resolvidas pelos desenvolvedores contribuintes dos projetos escolhidos para a avaliação.

Considerando as definições apresentadas, um sumário de Objetivo (*Goal*) foi então definido para corresponder à avaliação da solução proposta:

"**Analisar** (*Objeto do estudo*) as sugestões de desenvolvedor para tarefa de desenvolvimento de software, **com o propósito de** verificar a equivalência das sugestões com as atribuições reais, **com respeito à** (*Foco*) Reputação e Esquecimento, **sob o ponto de vista** das *expertises* necessárias para a tarefa **no contexto de** projetos de desenvolvimento de software".

4.3 PLANEJAMENTO

A etapa do planejamento da avaliação é composta pela escolha do contexto, formulação das hipóteses, seleção das variáveis, definição da amostragem, definição da organização do experimento (*experiment design*), instrumentação e validação; sendo esta última mencionada na subseção (4.5.1) de Análise e Interpretação dos Resultados.

4.3.1 Seleção do contexto

O contexto é caracterizado conforme quatro dimensões, de acordo com WOHLIN *et al*. (2012):

- Processo: on-line / off-line;
- Participantes: alunos / profissionais;
- Realidade: problema real / problema modelado;
- Generalidade: específico / geral;

A presente avaliação da solução proposta é em **processo** off-line, visto que são extraídos e trabalhados localmente os dados públicos reais de usuários e os dados dos projetos de software utilizados para a avaliação. Os **participantes** são usuários desenvolvedores de software que apresentam contribuições de desenvolvimento (código fonte, relato de *bugs*) dentro do respectivo projeto de software. O contexto desse estudo aborda um **problema real** visto que são caracterizadas as competências reais dos desenvolvedores participantes por meio do registro histórico de suas contribuições em *expertises* de desenvolvimento de software. O contexto tem caráter **específico**, pois em um universo amplo de desenvolvedores é considerado apenas aqueles que possuem contribuições nos respectivos projetos escolhidos para a Avaliação.

4.3.2 Formulação das hipóteses

Tem-se como hipótese geral que, as sugestões de desenvolvedores são capazes de ajudar coordenadores de tarefas a encontrarem os desenvolvedores mais condizentes, em termos de *expertises*, para as suas tarefas. E também de ajudar os desenvolvedores a encontrarem as tarefas de desenvolvimento de software adequadas aos conhecimentos (*expertises*) de programação dos quais trabalham. Para a elaboração das hipóteses, foi definida a pergunta de avaliação:

“Quão equivalentes são as sugestões de desenvolvedores com as atribuições reais de desenvolvedores nas tarefas, associando reputação e esquecimento, e seus respectivos valores?”

Por meio desta pergunta foram então definidas, a hipótese nula (H_0) e a hipótese alternativa (H_1):

H_0: O conjunto de desenvolvedores sugeridos é **equivalente** ao conjunto de desenvolvedores atribuídos nas tarefas.

H_1: O conjunto de desenvolvedores sugeridos **não é equivalente** ao conjunto de desenvolvedores atribuídos nas tarefas.

As métricas também foram definidas para ajudar na resposta da pergunta. Estas são compostas dos valores dos critérios de reputação e esquecimento de cada *expertise* em comum entre desenvolvedor e tarefa:

- (M1) Valor do critério Esquecimento.
- (M2) Valor do critério Opinião.
- (M3) Valor do critério Comportamento.
- (M4) Valor do critério Similaridade.
- (M5) Valor do critério Relacionamento.
- (M6) Valor do critério Expertise.
- (M7) Valor do critério Confiabilidade.

4.3.3 Seleção das variáveis

Após a definição das hipóteses, derivam-se as variáveis independentes e dependentes para o objetivo definido no escopo.

Para definição das **variáveis independentes** têm-se como fator as sugestões de desenvolvedores para as tarefas, e como tratamentos têm-se o uso e o não uso destas sugestões. A **variável dependente** é definida pela ordenação das sugestões de desenvolvedores, através dos valores das métricas (M1 a M7) de cada *expertise* em comum entre desenvolvedor e tarefa. Essa ordenação tem a opção de ser filtrada pelos valores de cada métrica separadamente para verificar a posição em que se encontrará cada desenvolvedor sugerido em comparação com os demais. A listagem de sugestões de desenvolvedores é ordenada do maior para o menor valor das métricas das *expertises* em comum de cada desenvolvedor com a tarefa analisada. Os desenvolvedores com os maiores valores das métricas de reputação (M2 a M7) e os menores valores na métrica de esquecimento (M1) nas *expertises* em comum com a tarefa analisada, estes são considerados como que sendo os mais aptos nessas *expertises* em relação aos demais. E, portanto são os que aparecem no topo da listagem de sugestões de desenvolvedores.

4.3.4 Seleção dos participantes

Os participantes selecionados para a respectiva avaliação são aqueles que possuem contribuições de desenvolvimento dentro dos projetos de software utilizados na avaliação. Contribuições essas que envolvem a participação destes participantes nas tarefas do projeto, desde a adição de novas funcionalidades no código fonte do projeto, até contribuições menores como, por exemplo, o relato de *bugs* no projeto.

4.3.5 Organização do experimento

Após a definição do objetivo do experimento para a avaliação e da escolha das variáveis independente e dependente e seleção dos participantes, agora é possível definir a organização do experimento (*Experiment Design*). Primeiramente seguem-se os princípios da experimentação geral na Engenharia de Software, que são a aleatoriedade, o agrupamento, o balanceamento.

Quanto à **aleatoriedade**, o uso da listagem de sugestões de desenvolvedores não é atribuído aleatoriamente para cada um dos participantes. Pois esse estudo tem como foco comparar os efeitos do uso e do não uso da solução desenvolvida sobre as contribuições históricas (caracterizadas como atribuições) que os participantes efetivaram em cada projeto de software. Os participantes não são escolhidos aleatoriamente, pois eles são usuários desenvolvedores que escolheram por si mesmos contribuírem nos projetos.

Quanto ao **agrupamento**, não faz parte do objetivo do estudo aplicar alguma abordagem sistemática de agrupamento sobre os participantes envolvidos. Pois o foco do estudo está em comparar as sugestões de desenvolvedores com as atribuições efetivadas historicamente entre desenvolvedores e tarefas.

Quanto ao **balanceamento**, pode-se dizer que a quantidade de participantes estará sempre balanceada relativamente ao projeto de software sendo usado para o experimento. Visto que, a comparação dos resultados do uso e do não uso das sugestões, é realizada sobre cada um dos mesmos participantes. Sabe-se que nenhum deles utilizou a solução desenvolvida. Portanto é feito uma simulação das sugestões que seriam apresentadas caso a solução proposta tivesse sido de fato utilizada.

O objetivo definido no escopo juntamente com suas hipóteses estabelecidas, variáveis e métricas derivadas, definiram o formato do experimento como sendo de **um fator e dois tratamentos**. Em relação às variáveis independentes para o respectivo objetivo, foi considerado como fator a própria listagem de sugestões, e os tratamentos o seu uso e o não uso. Mais detalhes se encontram na subseção (4.4.5).

4.3.6 Instrumentação

A instrumentação do experimento define os meios de execução e monitoramento do experimento, sem afetar o controle deste. Há três tipos de instrumentos para um experimento: Objetos, Diretrizes e Medições.

Objetos dizem respeito aos artefatos de documentos e código fonte desenvolvidos para o experimento. Para uso na execução do presente experimento, a solução desenvolvida foi adaptada para simular as sugestões de desenvolvedores baseado em dados históricos de antes de uma data específica. Para assim simular a busca das sugestões de desenvolvedores considerando suas contribuições de *expertises* anteriores à data de finalização de cada tarefa sendo testada.

Diretrizes são conjuntos de regras e guias necessárias a serem seguidas pelos participantes durante a participação deles no experimento (formulários ou entrevistas nos quais os participantes respondem como parte da coleta de dados do experimento). Porém o presente experimento envolve a utilização dos dados históricos dos desenvolvedores que contribuíram diretamente em projetos de software. E, portanto, não depende da ação direta desses participantes com uso de formulários para a coleta dos dados do experimento. Pois neste presente experimento, a coleta de dados é comparar as sugestões simuladas com as atribuições reais ocorridas entre desenvolvedor e tarefa.

A definição dos instrumentos de **Medições** tem correlação direta com a coleta dos dados. Visto que no caso deste experimento não é objetivo coletar dados dos participantes usando diretrizes especificas sendo seguidas, então a medição dos dados coletados se dá diretamente pela variável dependente definida para o objetivo estabelecido no escopo.

4.4 OPERAÇÃO DE EXECUÇÃO DA AVALIAÇÃO

O processo de execução da avaliação está dividido em: (1º) Preparo dos dados para a Avaliação, (2º) Execução da Avaliação e (3º) Geração dos resultados. De acordo com DE NEIRA *et al*. (2018) o desenvolvimento de projetos de software tem se mostrado eficaz utilizando o *crowdsourcing* de desenvolvedores distribuídos globalmente, cuja plataforma principal é o *TopCoder*. Com isso, optou-se pelos projetos trabalhados nesta plataforma para uso de seus respectivos dados na execução da Avaliação, considerando adaptações ao contexto da solução sendo avaliada. As tarefas no *TopCoder*, conhecidas como desafios (*challenges*), envolvem pessoas que são recompensadas em dinheiro quando suas soluções para o respectivo *challenge* são as melhores desenvolvidas. Segundo a documentação da

API[14] do *TopCoder*, essas tarefas desafios são classificadas em quatro grandes trilhas: Desenvolvimento; Design; Ciência de Dados; e QA (*Quality Assurance*). Para essa avaliação foi considerado escolher projetos no *TopCoder* cujos desafios (*challenges*) são da trilha "Desenvolvimento", pois são as que possuem as *expertises* que se adequam ao contexto de desenvolvimento de software, foco do presente trabalho.

4.4.1 Preparo dos dados

Nessa etapa, foi feito o preparo dos dados para a avaliação. Começou com a extração[15] dos *challenges* da trilha "Desenvolvimento" que foram criados entre Setembro de 2003 e Julho de 2021, totalizando em 19691 *challenges* finalizados recuperados. Os dados coletados desses *challenges* incluem informações dos desenvolvedores participantes; tecnologias (*expertises*) correlacionadas; datas de criação e finalização do *challenge*; usuário que criou o *challenge*; projeto associado; e outras informações técnicas. Desses *challenges* extraídos, 14209 estão distribuídos em um total de 2457 projetos. Os demais 5482 *challenges* são avulsos, não fazendo parte de nenhum projeto do *TopCoder*. A Tabela 4.1 mostra a quantidade de projetos existentes distribuídos em intervalos das quantidades de *challenges*, e a Tabela 4.2 mostra a quantidade de projetos por quantidades de usuários. A Figura 4.1 exibe em um gráfico de dispersão os projetos com suas quantidades de usuários e *challenges* envolvidos.

Tabela 4.1 – Quantidades de Projetos por quantidades de challenges.

Challenges	<10	10 – 20	20 – 40	40 – 60	60 – 80	80 – 100	>100
Quantidade de Projetos	2162	166	92	19	11	4	3

Fonte: Elaborado pelo autor (2021).

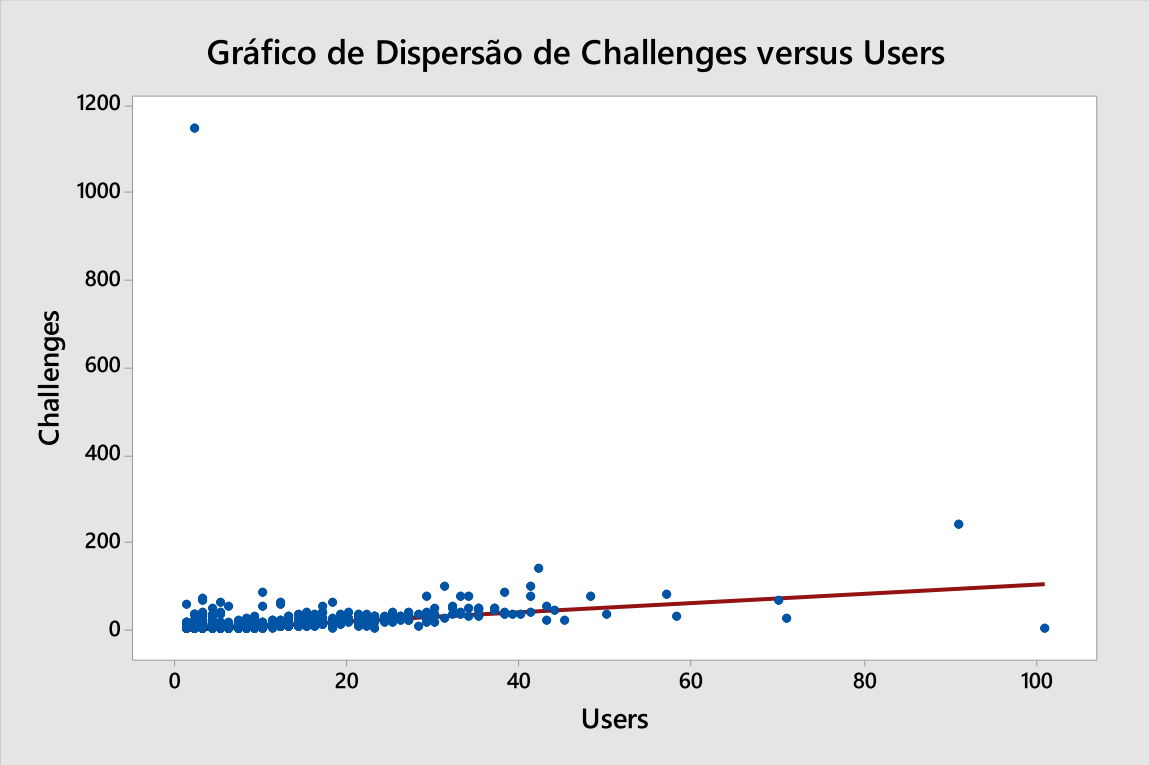
Tabela 4.2 – Quantidades de Projetos por quantidades de usuários.

Usuários	<10	10 – 20	20 – 30	30 – 40	40 – 60	>60
Quantidade de Projetos	2105	245	64	26	13	4

Fonte: Elaborado pelo autor (2021).

[14] http://api.topcoder-dev.com/v5/challenges/docs
[15] http://api.topcoder.com/v5/challenges?status=Completed&sortBy=created&sortOrder=desc

Figura 4.1 – Gráfico de dispersão entre usuários e *challenges* dos projetos.



Fonte: Elaborado pelo autor (2021).

Tabela 4.3 – Projetos no *TopCoder* e seus *challenges* associados.

Id do Projeto	**Quantidade de *challenges***	**Data dos *challenges***	**Quantidade de vencedores**	**Quantidade de criadores**
p_17469	238	13/06/2012 – 30/11/2014	86	6
p_18718	138	07/02/2014 – 02/02/2015	38	5
p_23783	98	24/12/2018 – 24/06/2021	39	3
p_23849	24	27/05/2019 – 26/03/2020	65	6
p_24917	28	03/08/2020 – 17/03/2021	56	2

Fonte: Elaborado pelo autor (2021).

Observando as tabelas e o gráfico de dispersão é possível perceber que a maioria dos projetos possuem pequenas quantidades de *challenges* e usuários envolvidos. Porém, para um melhor empenho do uso da solução proposta na Avaliação foram escolhidos projetos com maiores quantidades de *challenges* e de usuários envolvidos, exceto dois projetos identificados nos dois extremos das variáveis quantitativas do gráfico de dispersão. Pois um desses projetos, com 1144 *challenges*, possui apenas dois usuários envolvidos. O outro projeto, com 101 usuários, possui apenas um *challenge* envolvido. Considerando então os outros projetos com maior quantidade de usuários e maior quantidade de *challenges*, destes foram escolhidos **cinco** projetos para a avaliação. A Tabela 4.3 mostra a relação entre cada projeto escolhido e a quantidade de *challenges* associados como também o período de tempo

em que estes foram criados, a quantidade de usuários vencedores e a quantidade de usuários criadores desses *challenges*.

Para consideração de que um desenvolvedor participante de um *challenge* tenha de fato aptidão nas tecnologias (*expertises*) requisitadas, é preciso que este tenha sido considerado um dos vencedores do *challenge*. Pois somente se registrar para participar do *challenge* (sem ter enviado sua solução desenvolvida) não garante que o desenvolvedor tenha de fato aptidão nas *expertises* exigidas. O desenvolvedor ter enviado sua solução proposta para o *challenge* já é um indicador de alguma aptidão deste nas *expertises* exigidas. Porém, ainda não é garantido que o respectivo desenvolvedor tenha realmente trabalhado nesses conhecimentos técnicos. Pois sua solução desenvolvida não foi aceita por algum motivo de não atender às exigências de conhecimento das tecnologias requeridas. Já o desenvolvedor ter sido um dos vencedores do *challenge*, é uma indicação de que a sua solução foi verificada e aprovada (visto que há recompensa financeira para os vencedores). E, portanto, evidência de que esse desenvolvedor é de fato apto nas tecnologias (*expertises*) exigidas pelo *challenge*. Assim, consideram-se as tecnologias associadas aos *challenges* como sendo as *expertises* de cujos desenvolvedores vencedores demonstraram capacidade prática em trabalhar.

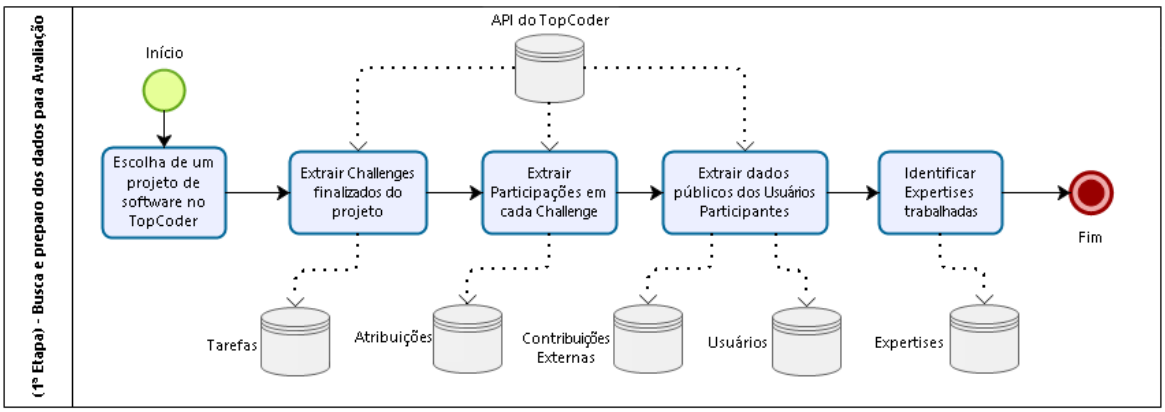
Identificar os conhecimentos técnicos trabalhados pelos desenvolvedores em tecnologias de desenvolvimento de software não é limitada somente à plataforma *TopCoder*. Esses mesmos desenvolvedores podem possuir contribuições em outras plataformas de apoio ao desenvolvimento de software, tais como o *GitHub* e o *StackOverflow*. Desenvolvedores podem disponibilizar em seus perfis no *TopCoder* o link de suas contas públicas do *GitHub* e/ou também do *StackOverflow*. Diante disso, foi então visto a possibilidade de se enriquecer os perfis de *expertises* desses desenvolvedores com informações de suas contribuições externas, advindas dessas plataformas diferentes do *TopCoder*. Realizou-se, através de um *webcrawler*[16], a extração dos links das contas no *GitHub* e *StackOverflow* que os usuários do *TopCoder* disponibilizam na página web de seus perfis da referida plataforma. Com esses links associados aos usuários do *TopCoder*, pôde então ser possível o acesso às suas contribuições em *expertises* em suas contas no *GitHub* e *StackOverflow*. Como contribuições externas em tecnologias de desenvolvimento de software do desenvolvedor, foram considerados seus repositórios públicos de código fonte criados no *GitHub* e suas perguntas e respostas efetuadas no *StackOverflow* sobre essas tecnologias. Através das APIs do *GitHub*[17]

[16] https://github.com/nathan2m/topcoder-webcrawler
[17] https://developer.github.com/v3/

e do *StackOverflow*[18] foram realizadas as extrações dessas contribuições externas juntamente com as informações das tecnologias (*expertises*) trabalhadas. Dos 2241 usuários identificados nos *challenges* extraídos do *TopCoder*, 189 deles possuem contas no *GitHub*, 42 possuem contas no *StackOverflow* e 26 possuem contas em ambas as plataformas. Os demais usuários não tinham registro em seus perfis suas respectivas contas no *GitHub* e/ou *StackOverflow*.

Dando prosseguimento ao processo do preparo dos dados, foi realizada uma correlação dos dados extraídos com o contexto da solução proposta sendo avaliada. Os *challenges* finalizados do tipo "Desenvolvimento" dos projetos escolhidos foram considerados como sendo as Tarefas trabalhadas, e os usuários que criaram esses *challenges* (Tarefas) foram caracterizados como sendo seus respectivos Coordenadores. Os usuários que foram vencedores nesses *challenges*, estes foram considerados como sendo os Desenvolvedores que trabalharam nas Tarefas. As tecnologias (*expertises*) associadas aos *challenges* e associadas às contribuições externas dos desenvolvedores vencedores foram correlacionadas com o contexto de *Expertise* Tecnologia e *Expertise* Categoria. (Por exemplo, TypeScript é uma *expertise* do tipo JavaScript. Nesse caso, JavaScript é a Categoria, e TypeScript é a Tecnologia). A Figura 4.2 ilustra o esquema de extração dos dados via a API do *TopCoder*.

Figura 4.2 – Diagrama BPMN do processo de preparo dos dados.



Fonte: Elaborado pelo autor (2021).

Após a correlação dos dados, estes foram inseridos no banco de dados da solução sendo avaliada. Desses dados correlacionados foram inseridos os registros de Tarefas trabalhadas (*challenges* finalizados em determinado projeto); os registros dos Usuários Coordenadores (aqueles que criaram os respectivos *challenges*) e Desenvolvedores (aqueles que foram vencedores dos *challenges*) como também suas contribuições externas (*challenges* de outros projetos no *TopCoder* e contribuições no *GitHub* e/ou *StackOverflow* quando disponíveis); os registros das correlações das *expertises* trabalhadas nas Tarefas e das *expertises* das

[18] https://api.stackexchange.com/

contribuições externas; e por fim, os registros das Atribuições dos Desenvolvedores às Tarefas.

4.4.2 Execução da avaliação

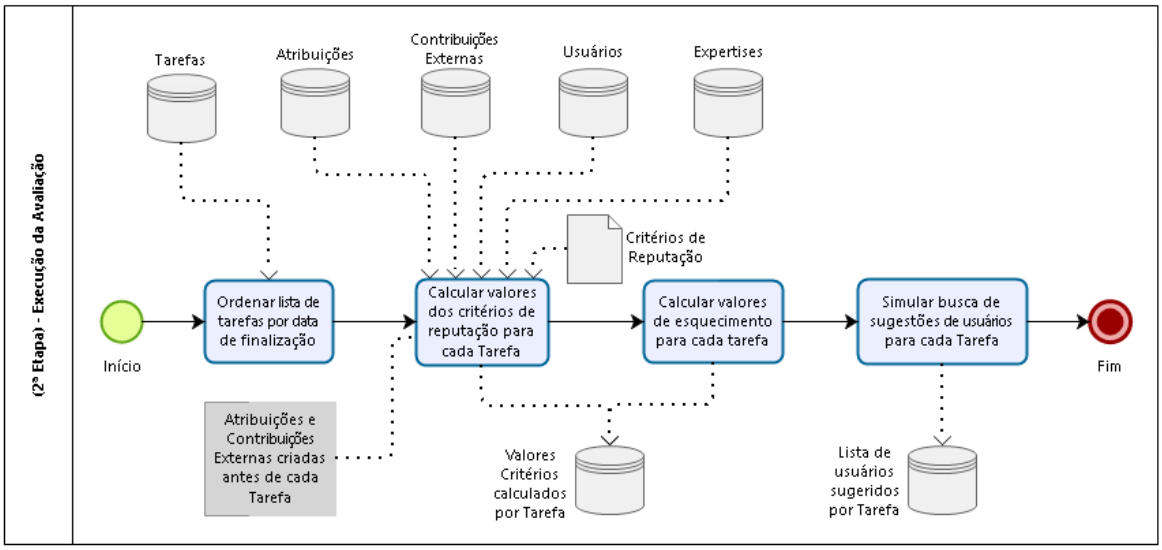
Para cada projeto analisado foi efetuada a etapa da execução da Avaliação, em suas respectivas tarefas e desenvolvedores associados, com o objetivo de registrar as sugestões de desenvolvedores listadas para cada uma das tarefas e, na etapa seguinte, comparar as listagens com as respectivas atribuições reais de cada tarefa. Primeiramente foram inseridos no banco de dados os registros dos usuários identificados como Coordenadores e Desenvolvedores. Em seguida, iniciou-se uma inserção sequencial de cada tarefa ordenada pela sua data de finalização (da mais antiga para a mais recente).

Para cada inserção de tarefa, foram feitas inserções dos registros das atribuições de desenvolvedores considerando aquelas com data de atribuição anterior à data de finalização da tarefa sendo analisada. Também foram feitas inserções dos registros das contribuições externas dos desenvolvedores com datas de contribuição anteriores à data de finalização da tarefa. Isso foi feito para que ao simular a obtenção das sugestões de desenvolvedores na respectiva tarefa, fossem evitadas sugestões de desenvolvedores com base em contribuições e atribuições de datas posteriores à data de finalização da tarefa. Assim, buscaram-se das contribuições externas e das atribuições, somente aquelas anteriores à data de finalização da tarefa sendo analisada. Lembrando que para cada inserção de tarefa e de contribuições externas, foram também inseridas as suas respectivas correlações com as *expertises* já registradas no banco de dados. E por fim, antes de aplicar a simulação de sugestões de desenvolvedores, fez-se o cálculo dos valores dos critérios de reputação e do esquecimento (baseado na data de finalização da tarefa, e nos registros de contribuições externas anteriores a essa data) e inseriu-se no banco de dados os valores obtidos para cada usuário existente no banco de dados.

Para finalizar o primeiro ciclo desta ação sequencial, foi executada a simulação da busca de sugestões de usuários desenvolvedores para a primeira tarefa sendo analisada. E assim, a listagem obtida foi salva para, na etapa seguinte da Avaliação, comparar as sugestões de desenvolvedores com as atribuições de desenvolvedores na respectiva tarefa antes da data de finalização da tarefa. Durante a presente etapa, a cada nova tarefa sendo analisada, os valores dos critérios de reputação e do esquecimento eram recalculados após haver nova inserção de registros de atribuições e contribuições externas. Após o término da sequência de inserção de todas as tarefas e todas as listagens de sugestões de desenvolvedores, devidamente registradas

em conformidade com a data de finalização da tarefa, é então que a Avaliação pôde prosseguir para a próxima etapa, isto é, depois de executar a presente etapa para cada projeto avaliado. A Figura 4.3 ilustra o esquema proposto nessa segunda etapa da avaliação.

Figura 4.3 – Diagrama BPMN do processo de execução da avaliação.



Fonte: Elaborado pelo autor (2021).

4.4.3 Geração dos resultados

Nessa etapa, os resultados da Avaliação foram gerados mediante a comparação das listagens de sugestões de desenvolvedores com os registros reais de atribuições dos desenvolvedores em cada uma das tarefas de cada um dos projetos selecionados. Para assim verificar a capacidade da solução proposta em apresentar os desenvolvedores atribuídos (mediante uso da reputação e do esquecimento) dentre as sugestões de desenvolvedores para tarefas dos projetos analisados. Esses resultados da comparação entre sugestões e atribuições foram gerados mediante um estudo de caso e um estudo experimental.

4.4.4 Estudo de caso

O estudo de caso foi realizado para averiguar os motivos pelos quais os desenvolvedores em uma tarefa são: sugeridos e atribuídos; sugeridos, porém não atribuídos; e atribuídos, porém não sugeridos. Para a primeira situação analisada (desenvolvedores sugeridos e atribuídos) tem-se uma tarefa[19] (*challenge*) do *TopCoder* e o usuário "*Dev1606 thomaskranitsas*" que nesse *challenge* foi o vencedor em 1º lugar. Porém, na solução proposta, considerando a data de 28/06/2017 correspondente à data de finalização da tarefa

[19] https://www.topcoder.com/challenges/e8bfb4ee-fbae-4314-84c1-d2673229b586

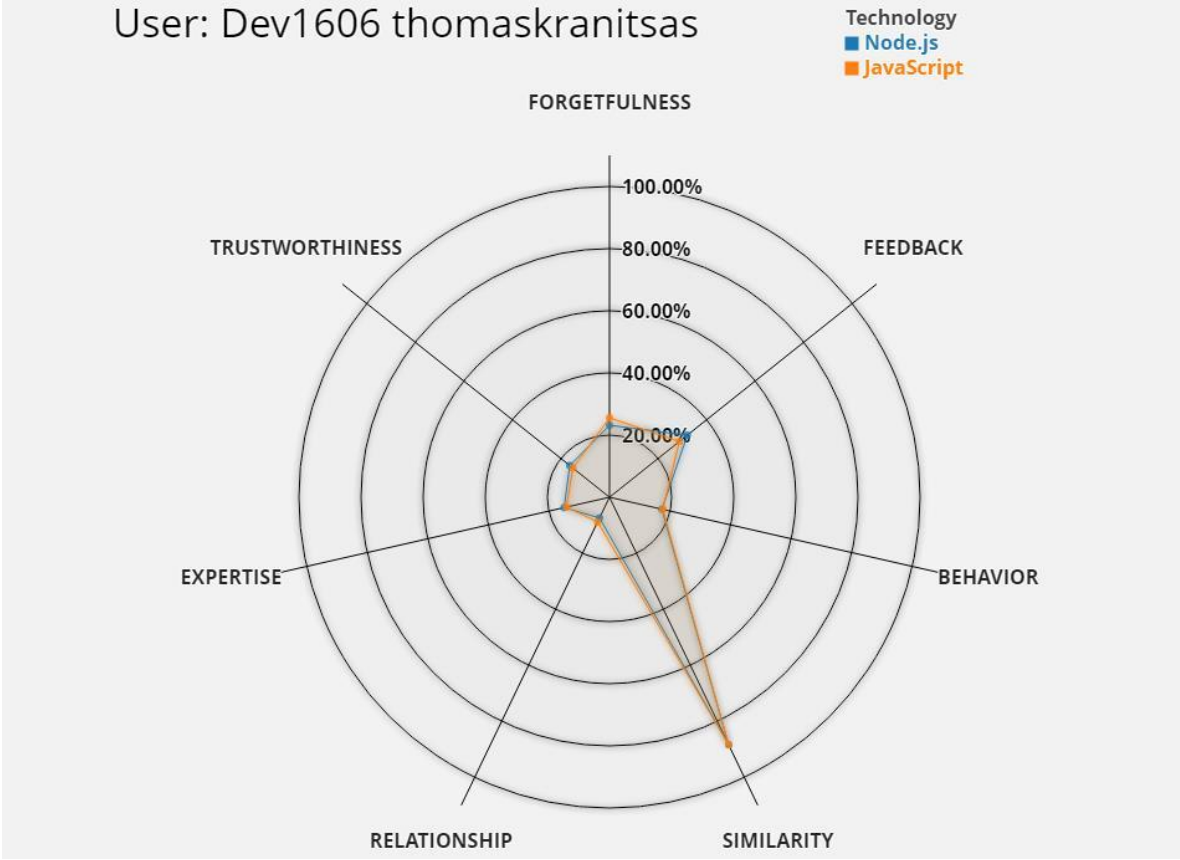
analisada, o Dev1606 se encontra como o 6º melhor colocado na lista de sugestões de desenvolvedores, conforme mostrado na Figura 4.4.

Figura 4.4 – Posição do Dev1606 na lista de sugestões.

#	Desenvolvedor		Contas Externas	Gráfico de Contribuições	Categorias (Ctgs) para a tarefa		Tecnologias (Tecs) para a tarefa	
1º	Dev1025 callmekatootie	Perfil	TopCoder StackOverflow	Contribuições	2 ctgs	Ctgs	4 tecs	Tecs
2º	Dev762 vvvpig	Perfil	TopCoder	Contribuições	2 ctgs	Ctgs	3 tecs	Tecs
3º	Dev1176 peakpado	Perfil	TopCoder	Contribuições	2 ctgs	Ctgs	3 tecs	Tecs
4º	Dev1396 shubhendus	Perfil	TopCoder GitHub	Contribuições	1 ctgs	Ctgs	2 tecs	Tecs
5º	Dev1262 ananthhh	Perfil	TopCoder GitHub StackOverflow	Contribuições	1 ctgs	Ctgs	2 tecs	Tecs
6º	Dev1606 thomaskranitsas	Perfil	TopCoder GitHub	Contribuições	1 ctgs	Ctgs	2 tecs	Tecs

Fonte: Elaborado pelo autor (2021).

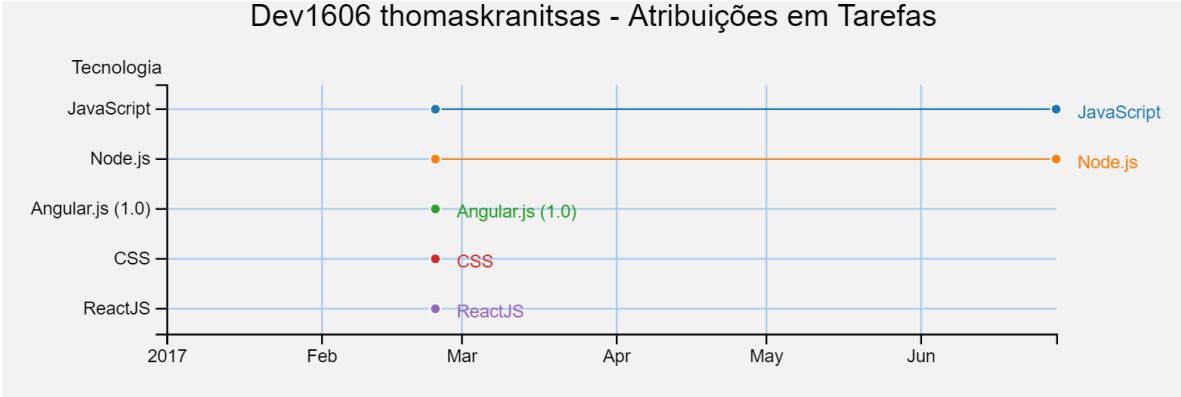
Figura 4.5 – Valores dos critérios de reputação e esquecimento para as *expertises* de Dev1606.



Fonte: Elaborado pelo autor (2021).

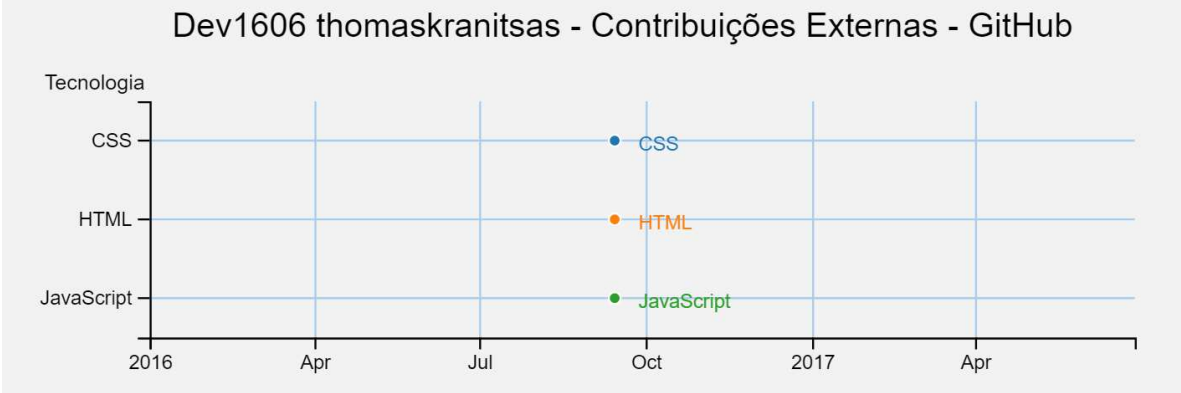
O Dev1606 possui duas *expertises* (*Node.js* e *JavaScript*) em comum com a tarefa analisada. Considerando as duas *expertises* e filtrando por cada critério de reputação e esquecimento, o Dev1606 fica nas seguintes posições na listagem de sugestões: *Forgetfulness* (4º), *Feedback* (9º), *Behavior* (7º), *Similarity* (4º), *Relatioship* (21º), *Expertise Rank* (10º), *Trustworthiness* (9º). Os valores desses critérios são mostrados na Figura 4.5. Considerando todos os critérios de reputação e esquecimento, e filtrando por cada *expertise* em comum com a tarefa, o Dev1606 fica nas seguintes posições na listagem de sugestões: *Node.js* (3º), *JavaScript* (6º).

Figura 4.6 – Atribuições do Dev1606 em tarefas.



Fonte: Elaborado pelo autor (2021).

Figura 4.7 – Contribuições Externas no *GitHub* do Dev1606.

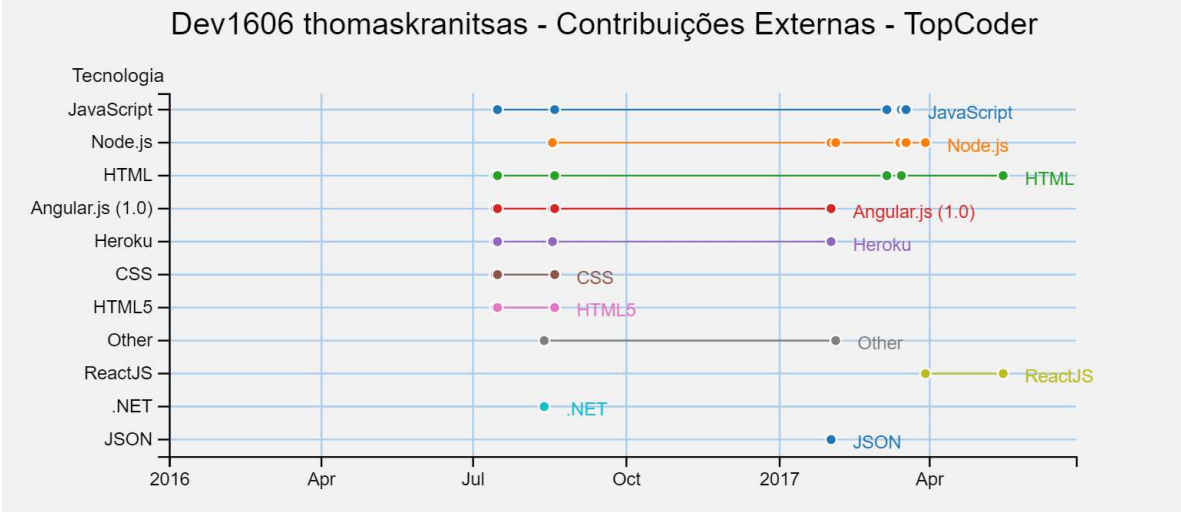


Fonte: Elaborado pelo autor (2021).

O Dev1606 possui 2 atribuições e 16 contribuições externas (15 no *TopCoder* e 1 no *GitHub*), ocorridas em até um ano antes da data de criação da tarefa (21/06/2017). Uma das atribuições é a própria tarefa analisada. A outra atribuição ocorreu cerca de 4 meses antes da tarefa analisada. E o perfil de *expertises* do Dev1606 apresentou correspondência com outras *expertises* da respectiva atribuição além daquelas correspondidas com a tarefa analisada. A Figura 4.6 mostra essas duas atribuições. A única contribuição no *GitHub* ocorreu 9 meses antes da tarefa analisada, conforme mostrada na Figura 4.7 (embora na figura pareçam ser 3

contribuições diferentes, nesse caso se trata de 3 *expertises* diferentes atreladas à uma única contribuição). E as demais contribuições externas (no *TopCoder*) ocorreram entre 11 meses e 2 meses antes da tarefa analisada, no qual a Figura 4.8 mostra. Essas contribuições externas possuíam em sua maioria pelo menos uma das *expertises* exigidas pela tarefa analisada.

Figura 4.8 – Contribuições Externas no *TopCoder* do Dev1606.



Fonte: Elaborado pelo autor (2021).

Após análise dos dados, verifica-se que "*Dev1606 thomaskranitsas*" foi sugerido e atribuído nessa tarefa por possuir uma atribuição em outra tarefa menos de um ano antes da tarefa analisada, e contribuições recentes (menos de um ano antes da tarefa analisada) em algumas das *expertises* (*Node.js* e *JavaScript*) exigidas pela respectiva tarefa. Constatou-se que nessas duas *expertises* o Dev1606 havia contribuído 9 vezes entre 11 meses e 5 meses antes da tarefa analisada. Embora a reputação seja relativamente baixa (exceto critério similaridade) para essas *expertises* (conforme mostrado na Figura 4.5) observou-se que o Dev1606 trabalhou cerca de 2 meses antes dessa tarefa em uma *expertise* (*ReactJS*) que faz parte da mesma categoria (*JavaScript*) das *expertises* (*Node.js* e *JavaScript*) exigidas pela tarefa analisada.

Para a segunda situação analisada (desenvolvedores sugeridos, porém não atribuídos) tem-se uma tarefa[20] (*challenge*) do *TopCoder* e o usuário "*Dev1025 callmekatootie*", do qual não foi atribuído nesse *challenge*. Porém, considerando a data de 28/06/2017, correspondente à data de finalização da tarefa analisada, o Dev1025 se encontra como o 1º melhor colocado na lista de sugestões de desenvolvedores, conforme mostrado na Figura 4.9. O Dev1025 possui quatro *expertises* (*Angular 2+*, *IBM Watson*, *JavaScript* e *Node.js*) em comum com a

[20] https://www.topcoder.com/challenges/e8bfb4ee-fbae-4314-84c1-d2673229b586

tarefa analisada. Considerando as quatro *expertises* e filtrando por cada critério de reputação e esquecimento, o Dev1025 fica nas seguintes posições na listagem de sugestões:

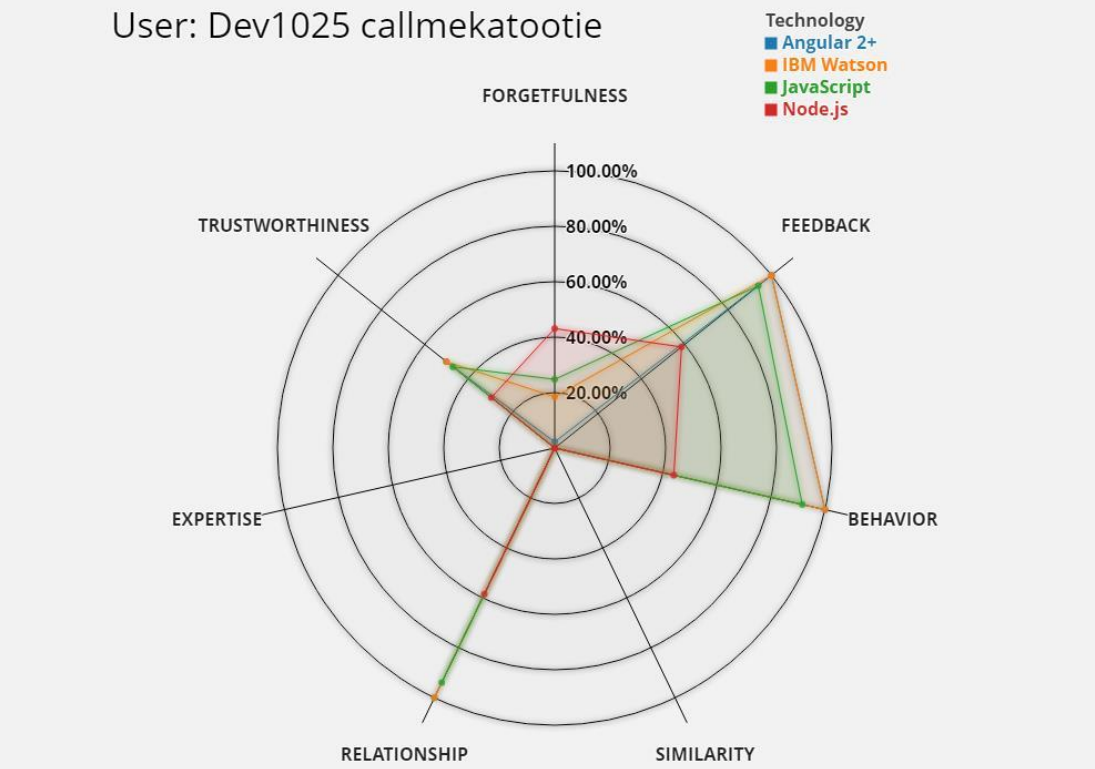
Forgetfulness (1º), *Feedback* (1º), *Behavior* (1º), *Similarity* (42º), *Relationship* (1º), *Expertise Rank* (42º), *Trustworthiness* (1º). Os valores desses critérios são mostrados na Figura 4.10.

Figura 4.9 – Posição do Dev1025 na lista de sugestões.

#	Desenvolvedor		Contas Externas	Gráfico de Contribuições	Categorias (Ctgs) para a tarefa		Tecnologias (Tecs) para a tarefa	
1º	Dev1025 callmekatootie	Perfil	TopCoder StackOverflow	Contribuições	2 ctgs	Ctgs	4 tecs	Tecs
2º	Dev762 vvvpig	Perfil	TopCoder	Contribuições	2 ctgs	Ctgs	3 tecs	Tecs
3º	Dev1176 peakpado	Perfil	TopCoder	Contribuições	2 ctgs	Ctgs	3 tecs	Tecs
4º	Dev1396 shubhendus	Perfil	TopCoder GitHub	Contribuições	1 ctgs	Ctgs	2 tecs	Tecs
5º	Dev1262 ananthhh	Perfil	TopCoder GitHub StackOverflow	Contribuições	1 ctgs	Ctgs	2 tecs	Tecs
6º	Dev1606 thomaskranitsas	Perfil	TopCoder GitHub	Contribuições	1 ctgs	Ctgs	2 tecs	Tecs

Fonte: Elaborado pelo autor (2021).

Figura 4.10 – Valores dos critérios de reputação e esquecimento para as *expertises* de Dev1025.

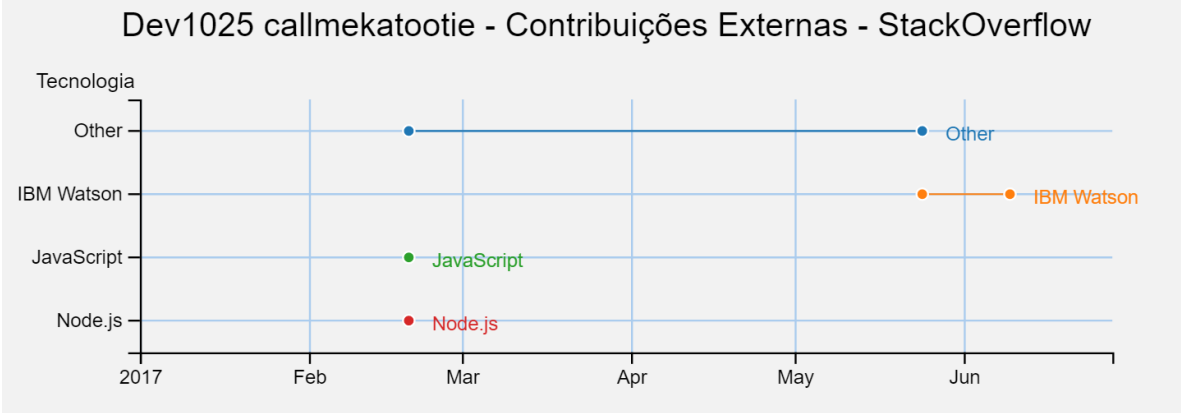


Fonte: Elaborado pelo autor (2021).

Considerando todos os critérios de reputação e esquecimento, e filtrando por cada *expertise* em comum com a tarefa, o Dev1025 fica nas seguintes posições na listagem de sugestões: *Angular 2+* (1º), *IBM Watson* (1º), *Node.js* (4º), *JavaScript* (2º).

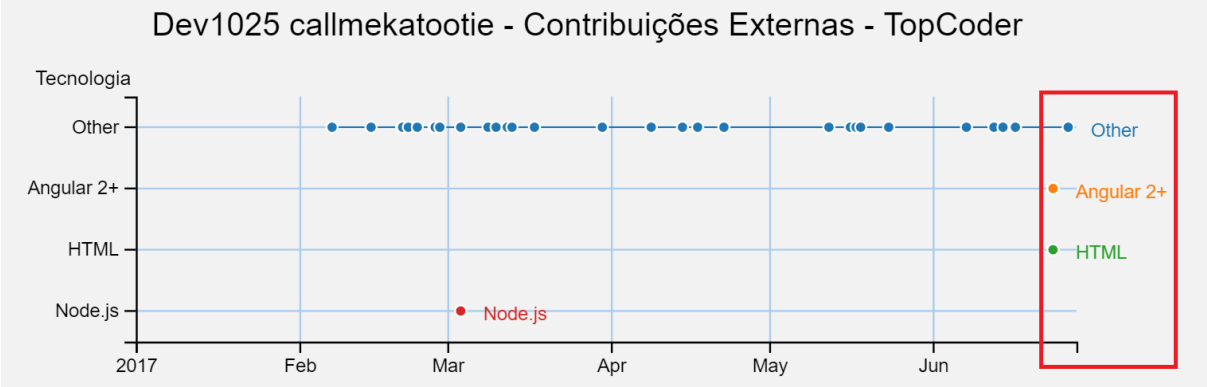
O Dev1025 não possui nenhuma atribuição em outras tarefas em até um ano antes da data de criação da tarefa (21/06/2017). Porém, durante esse período de tempo possui 45 contribuições externas (42 no *TopCoder* e 3 no *StackOverflow*). As 3 contribuições no *StackOverflow* ocorreram entre 5 meses e menos de 1 mês antes da tarefa analisada, conforme mostrada na Figura 4.11.

Figura 4.11 – Contribuições Externas no *StackOverflow* do Dev1025.



Fonte: Elaborado pelo autor (2021).

Figura 4.12 – Contribuições Externas no *TopCoder* do Dev1025.



Fonte: Elaborado pelo autor (2021).

As demais contribuições (*TopCoder*) ocorreram também entre 5 meses e menos de 1 mês antes da tarefa analisada, sendo que 2 ocorreram entre a data de criação e finalização da respectiva tarefa. Na Figura 4.12, estão destacadas dentro de um retângulo vermelho as 2 contribuições que ocorreram entre as datas de criação e finalização da tarefa analisada. Em uma dessas contribuições o Dev1025 trabalhou com *Angular 2+* e *HTML*, e a outra contribuição foi em uma *expertise* não identificada (*Other*). Essas contribuições externas

possuíam em sua maioria pelo menos uma das *expertises* exigidas pela tarefa analisada. Porém a maioria dessas contribuições no *TopCoder* não tiveram suas *expertises* identificadas, sendo assim rotuladas como "*Other*".

Figura 4.13 – Lista de Sugestões filtrada pela *expertise* "IBM Watson".

Fonte: Elaborado pelo autor (2021).

Figura 4.14 – Lista de Sugestões filtrada pela *expertise* "Angular 2+".

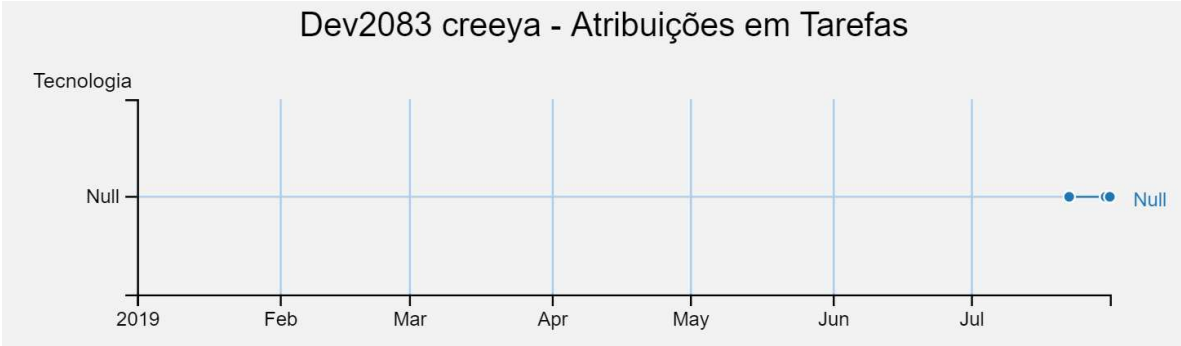
Fonte: Elaborado pelo autor (2021).

Após análise dos dados, evidências foram observadas para justificar o porquê de "*Dev1025 callmekatootie*" ter sido sugerido, mas não atribuído na tarefa analisada. Uma das evidências é a de que durante o período ativo da respectiva tarefa (entre a data de criação e de

finalização) o Dev1025 esteve contribuindo em outras tarefas (*challenges* no *TopCoder*), conforme Figura 4.12. Outra evidência, conforme a Figura 4.10 são os baixos valores de reputação e alto valor de esquecimento do Dev1025 para uma de suas *expertises* (*Node.js*) em comum com a tarefa. E embora nas outras *expertises* (*IBM Watson*, *JavaScript* e *Angular 2+*) os valores de reputação sejam altos (com exceção dos critérios Expertise Rank e Similaridade) e o valor do esquecimento baixo, verifica-se que ao filtrar a listagem de sugestões por 2 dessas *expertises*, aparece apenas o Dev1025 de sugestão (na Figura 4.13), e mais outro desenvolvedor (na Figura 4.14). Esse desenvolvedor ser o único sugerido na *expertise* filtrada não significa que ele seja o melhor nessa *expertise*, pois não apareceram outros desenvolvedores para serem comparados.

Para a última situação analisada (desenvolvedores atribuídos, porém não sugeridos) tem-se uma tarefa[21] (*challenge*) do *TopCoder* e o usuário "*Dev2083 creeya*". Considerando a data de 31/07/2019, correspondente à data de finalização da tarefa analisada, não foi constatada nenhuma *expertise* em comum do Dev2083 com a respectiva tarefa. Em até um ano antes da criação da tarefa, o Dev2083 não possui nenhuma contribuição externa, mas possui 3 atribuições registradas, conforme mostra a Figura 4.15. A atribuição mais recente (mais a direita) que aparece no gráfico é a própria tarefa analisada.

Figura 4.15 – Atribuições do Dev2083 em tarefas.



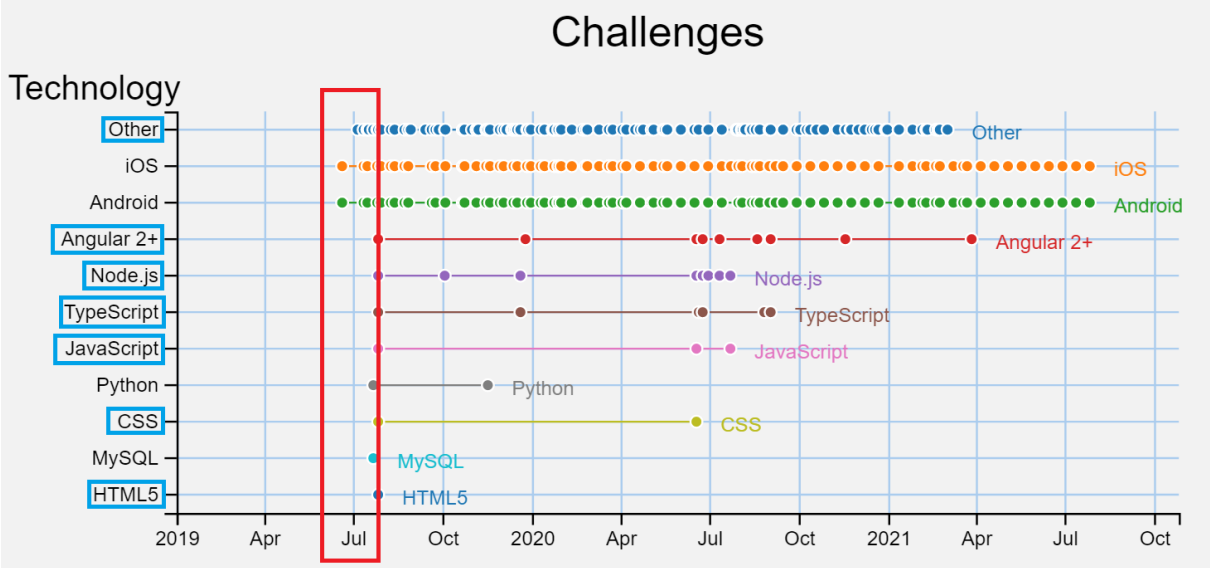
Fonte: Elaborado pelo autor (2021).

Para procurar evidências que justificam o porquê de o Dev2083 não ter sido sugerido, mas mesmo assim ter sido atribuído na tarefa analisada, fez-se uma busca de suas contribuições no *TopCoder* via API da própria plataforma. E verificou-se que o Dev2083 possui contribuições no *TopCoder* antes da data de criação (27/07/2019) da tarefa analisada. Conforme destacado em um retângulo vermelho na Figura 4.16, antes da data de criação da tarefa analisada, há *challenges* dos quais o Dev2083 participou. Porém vê-se também que desses *challenges* em nenhum deles o Dev2083 trabalhou nas *expertises* exigidas pela tarefa

[21] https://www.topcoder.com/challenges/7d9c729c-64dc-4bda-b175-bca48e1ceded

analisada (destacadas em azul), exceto para *expertises* não identificadas (rotuladas como "*Other*"). Embora se tenha atestado que o Dev2083 não possui contribuições nas *expertises* exigidas pela tarefa analisada, outros motivos podem ter levado à sua atribuição nessa tarefa. Porém, existe a hipótese de que ele possa ter sido atribuído por ter contribuições nessas *expertises* em outras plataformas não exploradas neste trabalho (plataformas diferentes do *GitHub*, *StackOverflow* e *TopCoder*).

Figura 4.16 – Contribuições do Dev2083 no *TopCoder*.



Fonte: Elaborado pelo autor (2021).

4.4.5 Estudo experimental

Considerando que as sugestões de desenvolvedores com os maiores valores de reputação e os menores valores de esquecimento são posicionadas no topo da listagem, agrupou-se então a listagem de sugestões em grupos contendo as 10, 20 e 30 primeiras sugestões listadas (Top10, Top20 e Top30). Isso para comparar entre esses grupos as quantidades dessas sugestões que compõem os desenvolvedores que foram de fato atribuídos nas tarefas respectivas. Quantidades essas de desenvolvedores que estão presentes tanto nas sugestões quanto nas atribuições, e com isso foram agrupadas em um grupo à parte nomeado interseção. A quantidade de interseção é sempre menor ou igual à quantidade de atribuições. Quando a quantidade de interseção é igual à quantidade de atribuições para determinada tarefa, isso indica que a solução proposta alcança o objetivo de sugerir desenvolvedores equivalentes em *expertises* àqueles que foram realmente atribuídos nessa tarefa. E caso essa igualdade já ocorra nas sugestões que estão nas primeiras posições na lista (ou seja, que contém os maiores valores de reputação e menores valores de esquecimento), então significa

que a reputação e o esquecimento são efetivos em apresentar sugestões de desenvolvedores equivalentes aos atribuídos. Para cada projeto analisado foi gerado uma tabela com as quantidades de sugestões, atribuições e interseções em cada tarefa do respectivo projeto, conforme exemplo parcial mostrado na Tabela 4.4. O restante dessa tabela como também as demais tabelas geradas para os outros projetos, estão disponíveis através deste link[22]. As quantidades de atribuições e interseções dessas tabelas foram utilizadas no estudo experimental explicado a seguir.

Tabela 4.4 – Quantidades de desenvolvedores em cada tarefa do projeto p_24917

Tarefa	**Tipo**	**Top10**	**Top20**	**Top30**
...	...	...	...	...
tarefa11	sugestões	10	20	21
tarefa11	atribuições	4	4	4
tarefa11	interseção	1	1	1
tarefa12	sugestões	10	20	21
tarefa12	atribuições	5	5	5
tarefa12	interseção	0	1	2
tarefa13	sugestões	10	20	21
tarefa13	atribuições	5	5	5
tarefa13	interseção	0	0	0
tarefa14	sugestões	10	20	21
tarefa14	atribuições	5	5	5
tarefa14	interseção	1	2	2
tarefa15	sugestões	10	20	21
tarefa15	atribuições	4	4	4
tarefa15	interseção	2	2	3
tarefa16	sugestões	10	20	23
tarefa16	atribuições	1	1	1
tarefa16	interseção	0	0	0
tarefa17	sugestões	10	20	23
tarefa17	atribuições	1	1	1
tarefa17	interseção	1	1	1
...	...	...	...	...

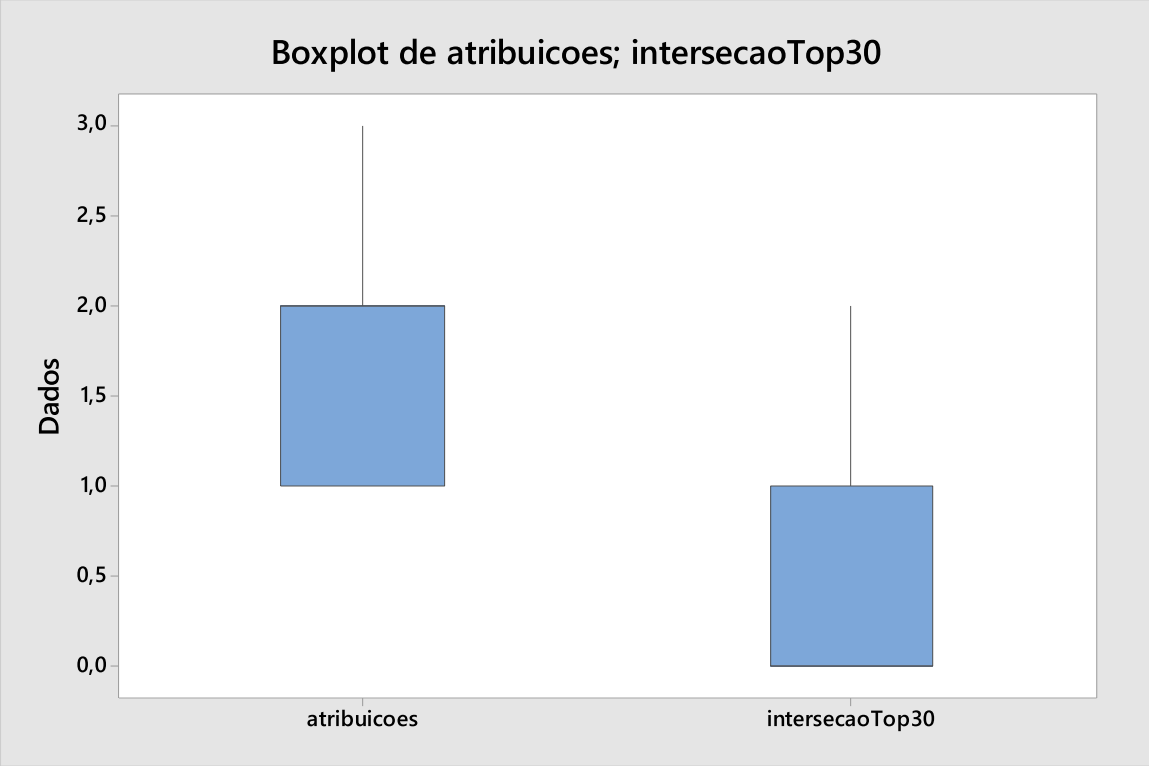
Fonte: Elaborado pelo autor (2022).

A pergunta de avaliação definida na elaboração das hipóteses diz respeito a quão equivalentes são as sugestões de desenvolvedores com as atribuições reais de desenvolvedores nas tarefas, mediante valores de reputação e esquecimento. Para responder a tal pergunta foi realizado um estudo experimental em experimentos com **design de um fator e dois tratamentos**, sendo utilizada a ferramenta MiniTab[23] para tal propósito. Em cada

[22] https://github.com/nathan2m/dissertacao/tree/main/avaliacao
[23] https://www.minitab.com/pt-br/

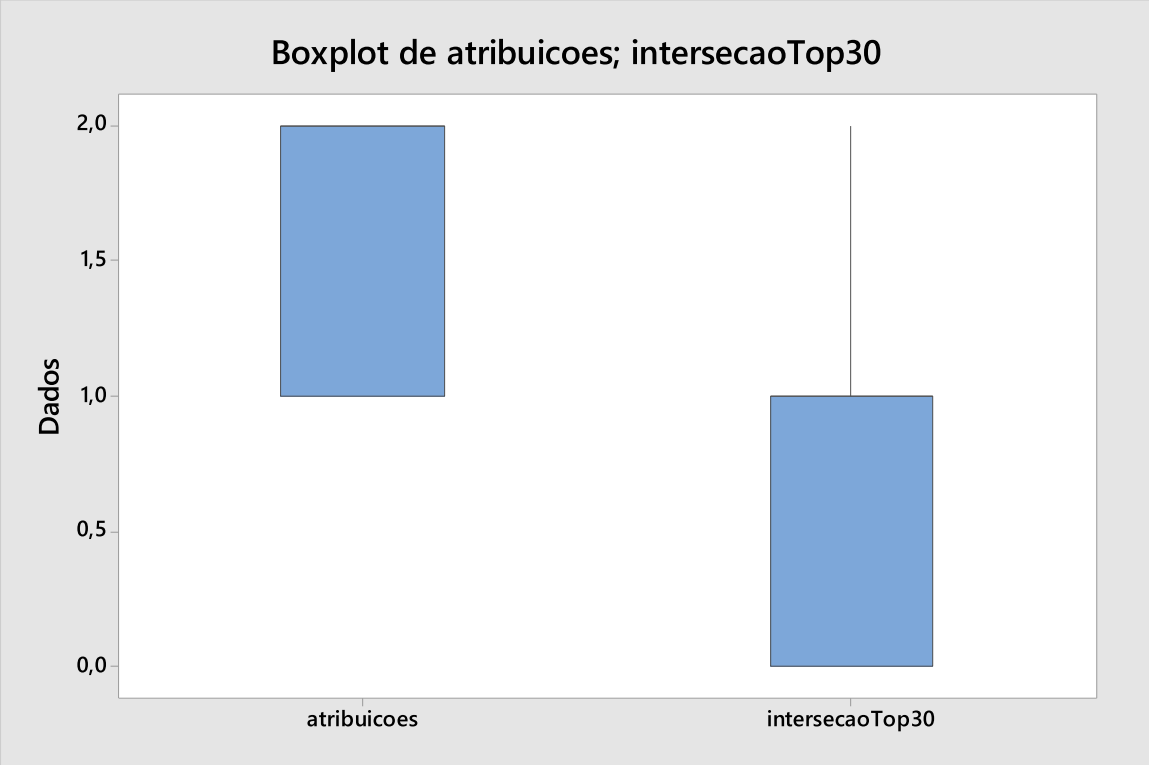
experimento, o fator analisado foi cada grupo contendo respectivamente os 10, 20 e 30 primeiros desenvolvedores sugeridos para as tarefas em cada projeto analisado. Os dois tratamentos foram o uso e o não uso das sugestões, sendo o grupo interseção (respectivo a cada grupo de 10, 20 e 30) representando o uso das sugestões, e as atribuições representando o não uso das sugestões. As figuras a seguir (Figura 4.17, Figura 4.18, Figura 4.19, Figura 4.20 e Figura 4.21) mostram na forma de *boxplot* a distribuição empírica dos dados das quantidades de atribuições e interseções (no grupo top 30) nas tarefas em cada projeto analisado. Os demais grupos de interseção (top 20 e top 10) apresentaram valores iguais ou menores que o grupo top 30.

Figura 4.17 – *Boxplot* das variáveis Atribuições e Interseção para projeto p_17469.



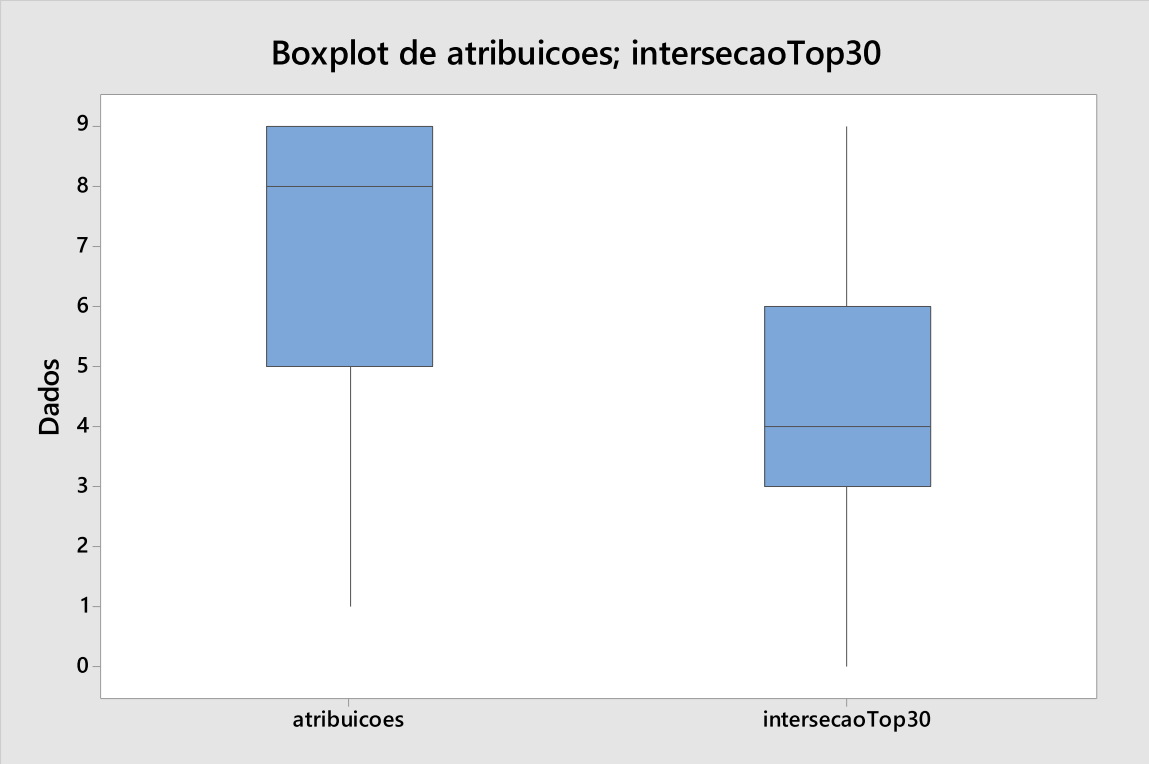
Fonte: Elaborado pelo autor (2022).

Figura 4.18 – *Boxplot* das variáveis Atribuições e Interseção para projeto p_18718.



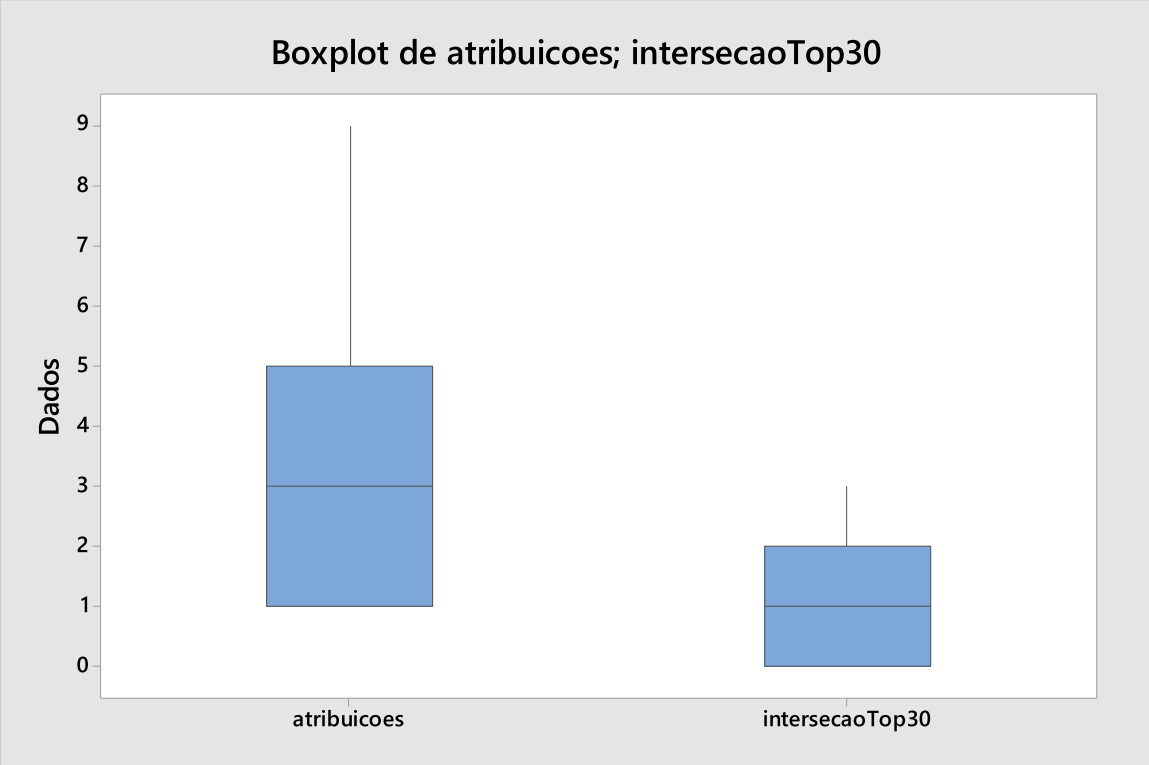
Fonte: Elaborado pelo autor (2022).

Figura 4.19 – *Boxplot* das variáveis Atribuições e Interseção para projeto p_23783.



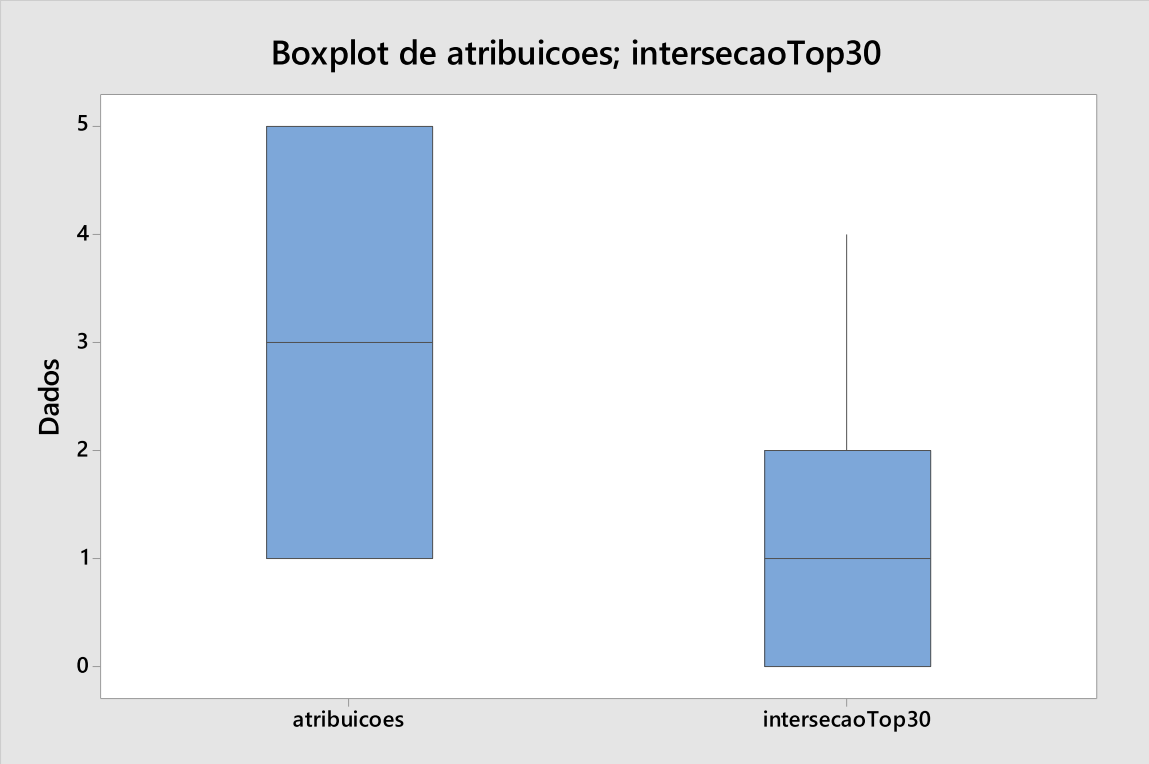
Fonte: Elaborado pelo autor (2022).

Figura 4.20 – *Boxplot* das variáveis Atribuições e Interseção para projeto p_23849.



Fonte: Elaborado pelo autor (2022).

Figura 4.21 – *Boxplot* das variáveis Atribuições e Interseção para projeto p_24917.



Fonte: Elaborado pelo autor (2022).

Pelos gráficos *boxplot* apresentados, é possível notar haver uma aparente diferença entre as médias de atribuições com cada interseção para os projetos utilizados na presente avaliação. No intuito de verificar a existência de diferenças estatisticamente significativas entre as amostras dos tratamentos (quantidades das atribuições e interseções) de cada fator analisado (sugestões de desenvolvedores nas tarefas em cada projeto), aplicou-se o uso de testes para comparação das médias entre as amostras dos tratamentos. Esses testes podem ser paramétricos ou não paramétricos. A utilização de um teste paramétrico exige com que as amostras obtidas sejam de distribuição normal e homocedástica. Caso contrário, é utilizado um teste não paramétrico. Dois métodos estatísticos podem ser utilizados para testar a normalidade das amostras. Seguindo as convenções apresentadas por SIRQUEIRA *et al*. (2020), utiliza-se o teste de **Kolmogorov-Smirnov** para amostras com mais de 30 valores, e o teste de **Shapiro-Wilk** para amostras com 30 ou menos valores. Para este teste foi estabelecido um nível de significância de (0,05) e as hipóteses:

- H_0 (hipótese nula): As amostras apresentam distribuição normal.
- H_1 (hipótese alternativa): As amostras não apresentam distribuição normal.

Cada um dos experimentos executados continha a quantidade de amostras maior do que 30 (somando as amostras de atribuições com interseção) e, portanto neles foi utilizado o teste de **Kolmogorov-Smirnov** para testar a normalidade. Todos esses experimentos apresentaram *p-value* abaixo do nível de significância estabelecido, mostrando assim haver indícios de que a hipótese nula deve ser rejeitada e ser aceita a alternativa de que seus dados **não apresentam distribuição normal**. A Tabela 4.5 exibe os *p-values* dos testes de normalidade obtidos para cada experimento nas comparações entre atribuições e interseções do top 30. As demais comparações (com top 20 e top 10) apresentaram também os *p-values* menores que 0,010.

Tabela 4.5 – *P-values* obtidos dos testes de normalidade.

Id do Projeto	**Amostras das variáveis: atribuições e interseções**	***P-value* das comparações: Atribuições X Interseção Top30**
p_17469	468	< 0,010
p_18718	272	< 0,010
p_23783	194	< 0,010
p_23849	48	< 0,010
p_24917	54	< 0,010

Fonte: Elaborado pelo autor (2022).

Como nenhum dos experimentos apresentou distribuição normal das amostras, não foi preciso verificar a homocedasticidade destas. Portanto prosseguiu-se para o uso do teste não

paramétrico de **Mann-Whitney** para avaliar a diferença entre as médias das quantidades de atribuições e interseções. Foi estabelecido um nível de significância em (0,05) e as hipóteses:

- H_0 (hipótese nula): Não há diferença entre as médias.
- H_1 (hipótese alternativa): Há diferença entre as médias.

Todos os experimentos executados apresentaram *p-value* abaixo do nível de significância estabelecido. Nesta situação, há indícios para rejeitar a hipótese nula, aceitando a situação de que os dados **são estatisticamente diferentes**. A Tabela 4.6 exibe os *p-values* do teste de média obtidos para cada experimento nas comparações entre atribuições e interseções do top 30. As demais comparações (com top 20 e top 10) apresentaram *p-values* no valor de 0,000. As tabelas de dados dos experimentos como também os resultados mais detalhados dos testes estatísticos se encontram disponíveis através deste link[24].

Tabela 4.6 – *P-values* obtidos dos testes de médias.

Id do Projeto	**Amostras da variável: atribuições**	**Amostras da variável: interseções**	***P-value* das comparações: Atribuições X Interseção Top30**
p_17469	234	234	0,000
p_18718	136	136	0,000
p_23783	97	97	0,000
p_23849	24	24	0,002
p_24917	27	27	0,001

Fonte: Elaborado pelo autor (2022).

4.5 ANÁLISE E INTERPRETAÇÃO DOS RESULTADOS

Considerando os resultados dos testes estatísticos obtidos no estudo experimental, não foram encontradas evidências para a aceitação da hipótese (H_0) de que a solução proposta sugere, de forma geral, desenvolvedores equivalentes em *expertises* (considerando reputação e esquecimento) aos desenvolvedores atribuídos originalmente às tarefas dos projetos analisados. Porém, segundo o estudo de caso, há evidências sim para a aceitação da hipótese (H_0). Respondendo à pergunta de avaliação segundo o estudo experimental aceitou-se a hipótese (H_1) de que "o conjunto de desenvolvedores sugeridos **não é equivalente** ao conjunto de desenvolvedores atribuídos nas tarefas". Uma explicação para os *p-values* terem ficado muito abaixo do nível de significância estabelecido se deve às poucas quantidades de desenvolvedores atribuídos em cada tarefa de cada projeto. Pois essas pequenas quantidades ao serem comparadas com as pequenas quantidades de interseções, qualquer diferença que houvesse entre esses dois grupos culminaria em uma grande diferença percentual relativa. Por exemplo, em uma tarefa havendo 2 atribuições e 1 interseção, a diferença percentual entre as

[24] https://github.com/nathan2m/dissertacao/tree/main/avaliacao

duas quantidades é de 50%. Já em uma tarefa ao qual houvesse 10 atribuições e 9 interseções, essa diferença percentual é de 10%. Nota-se que em ambos os exemplos, a diferença absoluta entre atribuições e interseções é a mesma (1 de diferença). Portanto, são necessários outros projetos a serem avaliados dos quais contenham maiores quantidades de desenvolvedores atribuídos em cada tarefa, para melhor precisão dos *p-values*.

A presunção inicial desta pesquisa é a de que as *expertises* tecnológicas dos desenvolvedores são motivo destes serem atribuídos em tarefas pelos respectivos coordenadores. E isso com a utilização da reputação e do esquecimento para estimar o quanto desse conhecimento esses desenvolvedores possuem nessas tecnologias para desenvolvimento de software. Porém, notaram-se desenvolvedores atribuídos que não foram sugeridos pela solução proposta, devido às *expertises* destes não terem sido identificadas. Também foram notados desenvolvedores no grupo interseção (desenvolvedores atribuídos que foram sugeridos) que não se encontravam entre as 10 primeiras posições na listagem das sugestões. Para cada projeto avaliado foi gerado uma tabela com o percentual de tarefas que cada desenvolvedor participante do projeto aparece nos grupos atribuições, sugestões e interseções, conforme mostrado nas tabelas a seguir para cada projeto analisado. Nessas tabelas são mostrados os desenvolvedores que foram sugeridos em pelo menos uma tarefa daquelas ao qual foi atribuído. Os demais desenvolvedores não exibidos nas tabelas são aqueles que não foram sugeridos em nenhuma das tarefas dos projetos às quais foram atribuídos. Esses dados restantes estão disponíveis no link[25] com os dados do estudo experimental.

O projeto **p_17469** possui 234 tarefas (*challenges*) criadas entre 13/06/2012 e 30/11/2014, e possui 85 desenvolvedores que foram atribuídos e/ou sugeridos em pelo menos uma dessas tarefas do projeto. Desses desenvolvedores, 14 (16,47%) foram sugeridos pela solução proposta nas respectivas tarefas em que estes foram de fato atribuídos. E desses, 11 (12,94%) desenvolvedores se encontravam entre os dez primeiros sugeridos (Top 10) respectivamente em cada uma das tarefas em que foram atribuídos. Porém, houve tarefas em que a solução proposta não conseguiu sugerir esses desenvolvedores nelas atribuídos. Dos 14 desenvolvedores sugeridos e atribuídos, 11 deles a solução proposta os sugeriu em 100% das tarefas que foram atribuídos. E dos 11 desenvolvedores atribuídos e que se encontravam entre os dez primeiros sugeridos, 8 (9,41%) deles a solução proposta os sugeriu em 100% das tarefas que foram atribuídos. A Tabela 4.7 mostra mais detalhes do percentual de tarefas em que esses 14 desenvolvedores foram atribuídos e sugeridos. Os 71 (83,53%) desenvolvedores

[25] https://github.com/nathan2m/dissertacao/tree/main/avaliacao

restantes do projeto compõem aqueles que foram sugeridos em tarefas dos quais não foram atribuídos, e também aqueles que foram atribuídos em tarefas dos quais não foram sugeridos pela solução proposta.

Tabela 4.7 – Percentual de tarefas do projeto p_17469 cujos desenvolvedores foram sugeridos e atribuídos.

Devs	Percentual de tarefas atribuído	Percentual de tarefas sugerido (top 10) e atribuído	Percentual de tarefas sugerido e atribuído	Percentual Interseção (Top 10) / Atribuição	Percentual Interseção / Atribuição
dev11	0,427%	0,427%	0,427%	100,000%	100,000%
dev14	0,427%	0,427%	0,427%	100,000%	100,000%
dev25	0,427%	0,427%	0,427%	100,000%	100,000%
dev38	16,239%	4,274%	4,274%	26,316%	26,316%
dev40	26,496%	14,530%	14,530%	54,839%	54,839%
dev41	1,282%	1,282%	1,282%	100,000%	100,000%
dev44	0,427%	0,427%	0,427%	100,000%	100,000%
dev51	1,282%	1,282%	1,282%	100,000%	100,000%
dev55	0,427%	0,000%	0,427%	0,000%	100,000%
dev57	0,855%	0,000%	0,855%	0,000%	100,000%
dev59	0,855%	0,427%	0,427%	50,000%	50,000%
dev66	5,983%	5,983%	5,983%	100,000%	100,000%
dev79	0,427%	0,427%	0,427%	100,000%	100,000%
dev85	0,855%	0,000%	0,855%	0,000%	100,000%

Fonte: Elaborado pelo autor (2022).

O projeto **p_18718** possui 136 tarefas (*challenges*) criadas entre 07/02/2014 e 02/02/2015, e possui 38 desenvolvedores que foram atribuídos e/ou sugeridos em pelo menos uma dessas tarefas do projeto. Desses desenvolvedores, 22 (57,89%) foram sugeridos pela solução proposta nas respectivas tarefas em que estes foram de fato atribuídos. E desses, 15 (39,47%) desenvolvedores se encontravam entre os dez primeiros sugeridos (Top 10) respectivamente em cada uma das tarefas em que foram atribuídos. Porém, houve tarefas em que a solução proposta não conseguiu sugerir esses desenvolvedores nelas atribuídos. Dos 22 desenvolvedores sugeridos e atribuídos, 17 (44,74%) deles a solução proposta os sugeriu em 100% das tarefas que foram atribuídos. E dos 15 desenvolvedores atribuídos e que se encontravam entre os dez primeiros sugeridos, 10 (26,32%) deles a solução proposta os sugeriu em 100% das tarefas que foram atribuídos. A Tabela 4.8 mostra mais detalhes do percentual de tarefas em que esses 22 desenvolvedores foram atribuídos e sugeridos. Os 16 (42,11%) desenvolvedores restantes do projeto compõem aqueles que foram sugeridos em tarefas dos quais não foram atribuídos, e também aqueles que foram atribuídos em tarefas dos quais não foram sugeridos pela solução proposta.

Tabela 4.8 – Percentual de tarefas do projeto p_18718 cujos desenvolvedores foram sugeridos e atribuídos.

Devs	Percentual de tarefas atribuído	Percentual de tarefas sugerido (top 10) e atribuído	Percentual de tarefas sugerido e atribuído	Percentual Interseção (Top 10) / Atribuição	Percentual Interseção / Atribuição
dev2	1,471%	0,000%	0,735%	0,000%	50,000%
dev4	12,500%	3,676%	3,676%	29,412%	29,412%
dev9	30,882%	18,382%	18,382%	59,524%	59,524%
dev10	0,735%	0,735%	0,735%	100,000%	100,000%
dev11	0,735%	0,000%	0,735%	0,000%	100,000%
dev12	4,412%	4,412%	4,412%	100,000%	100,000%
dev15	0,735%	0,000%	0,735%	0,000%	100,000%
dev16	2,206%	2,206%	2,206%	100,000%	100,000%
dev17	0,735%	0,000%	0,735%	0,000%	100,000%
dev19	6,618%	0,000%	5,882%	0,000%	88,889%
dev21	2,206%	2,206%	2,206%	100,000%	100,000%
dev22	2,206%	1,471%	2,206%	66,667%	100,000%
dev24	1,471%	1,471%	1,471%	100,000%	100,000%
dev25	1,471%	1,471%	1,471%	100,000%	100,000%
dev26	0,735%	0,735%	0,735%	100,000%	100,000%
dev29	0,735%	0,000%	0,735%	0,000%	100,000%
dev30	0,735%	0,000%	0,735%	0,000%	100,000%
dev32	17,647%	0,735%	0,735%	4,167%	4,167%
dev33	2,206%	2,206%	2,206%	100,000%	100,000%
dev35	22,059%	22,059%	22,059%	100,000%	100,000%
dev36	8,824%	8,088%	8,824%	91,667%	100,000%
dev37	3,676%	3,676%	3,676%	100,000%	100,000%

Fonte: Elaborado pelo autor (2022).

O projeto **p_23783** possui 97 tarefas (*challenges*) criadas entre 24/12/2018 e 24/06/2021, e possui 39 desenvolvedores que foram atribuídos e/ou sugeridos em pelo menos uma dessas tarefas do projeto. Desses desenvolvedores, 25 (64,1%) foram sugeridos pela solução proposta nas respectivas tarefas em que estes foram de fato atribuídos. E desses, 16 (41,03%) desenvolvedores se encontravam entre os dez primeiros sugeridos (Top 10) respectivamente em cada uma das tarefas em que foram atribuídos. Porém, houve tarefas em que a solução proposta não conseguiu sugerir esses desenvolvedores nelas atribuídos. Dos 25 desenvolvedores sugeridos e atribuídos, 15 (38,46%) deles a solução proposta os sugeriu em 100% das tarefas em que foram atribuídos. E dos 16 desenvolvedores atribuídos e que se encontravam entre os dez primeiros sugeridos, 4 (10,26%) deles a solução proposta os sugeriu em 100% das tarefas que foram atribuídos. A Tabela 4.9 mostra mais detalhes do percentual de tarefas em que esses 25 desenvolvedores foram atribuídos e sugeridos. Os 14 (35,9%)

desenvolvedores restantes do projeto compõem aqueles que foram sugeridos em tarefas dos quais não foram atribuídos, e também aqueles que foram atribuídos em tarefas dos quais não foram sugeridos pela solução proposta.

Tabela 4.9 – Percentual de tarefas do projeto p_23783 cujos desenvolvedores foram sugeridos e atribuídos.

Devs	Percentual de tarefas atribuído	Percentual de tarefas sugerido (top 10) e atribuído	Percentual de tarefas sugerido e atribuído	Percentual Interseção (Top 10) / Atribuição	Percentual Interseção / Atribuição
dev1	1,031%	1,031%	1,031%	100,000%	100,000%
dev3	1,031%	1,031%	1,031%	100,000%	100,000%
dev4	9,278%	1,031%	9,278%	11,111%	100,000%
dev5	28,866%	28,866%	28,866%	100,000%	100,000%
dev6	8,247%	3,093%	5,155%	37,500%	62,500%
dev9	4,124%	0,000%	4,124%	0,000%	100,000%
dev10	36,082%	31,959%	31,959%	88,571%	88,571%
dev11	36,082%	36,082%	36,082%	100,000%	100,000%
dev13	13,402%	0,000%	13,402%	0,000%	100,000%
dev15	3,093%	0,000%	3,093%	0,000%	100,000%
dev16	22,680%	20,619%	20,619%	90,909%	90,909%
dev18	12,371%	11,340%	11,340%	91,667%	91,667%
dev19	24,742%	3,093%	20,619%	12,500%	83,333%
dev21	3,093%	0,000%	3,093%	0,000%	100,000%
dev22	27,835%	13,402%	27,835%	48,148%	100,000%
dev23	73,196%	69,072%	69,072%	94,366%	94,366%
dev24	3,093%	0,000%	3,093%	0,000%	100,000%
dev25	1,031%	0,000%	1,031%	0,000%	100,000%
dev28	24,742%	1,031%	24,742%	4,167%	100,000%
dev29	37,113%	22,680%	22,680%	61,111%	61,111%
dev31	48,454%	26,804%	40,206%	55,319%	82,979%
dev32	13,402%	0,000%	13,402%	0,000%	100,000%
dev33	58,763%	44,330%	49,485%	75,439%	84,211%
dev34	13,402%	0,000%	12,371%	0,000%	92,308%
dev37	1,031%	0,000%	1,031%	0,000%	100,000%

Fonte: Elaborado pelo autor (2022).

O projeto **p_23849** possui 24 tarefas (*challenges*) criadas entre 27/05/2019 e 26/03/2020, e possui 68 desenvolvedores que foram atribuídos e/ou sugeridos em pelo menos uma dessas tarefas do projeto. Desses desenvolvedores, 23 (33,82%) foram sugeridos pela solução proposta nas respectivas tarefas em que estes foram de fato atribuídos. E desses, 10 (14,71%) desenvolvedores se encontravam entre os dez primeiros sugeridos (Top 10) respectivamente em cada uma das tarefas em que foram atribuídos. Porém, houve tarefas em

que a solução proposta não conseguiu sugerir esses desenvolvedores nelas atribuídos. Dos 23 desenvolvedores sugeridos e atribuídos, 22 (32,35%) deles a solução proposta os sugeriu em 100% das tarefas que foram atribuídos. E dos 10 desenvolvedores atribuídos e que se encontravam entre os dez primeiros sugeridos, 6 (8,82%) deles a solução proposta os sugeriu em 100% das tarefas que foram atribuídos. A Tabela 4.10 mostra mais detalhes do percentual de tarefas em que esses 23 desenvolvedores foram atribuídos e sugeridos. Os 45 (66,18%) desenvolvedores restantes do projeto compõem aqueles que foram sugeridos em tarefas dos quais não foram atribuídos, e também aqueles que foram atribuídos em tarefas dos quais não foram sugeridos pela solução proposta.

Tabela 4.10 – Percentual de tarefas do projeto p_23849 cujos desenvolvedores foram sugeridos e atribuídos.

Devs	**Percentual de tarefas atribuído**	**Percentual de tarefas sugerido (top 10) e atribuído**	**Percentual de tarefas sugerido e atribuído**	**Percentual Interseção (Top 10) / Atribuição**	**Percentual Interseção / Atribuição**
dev1	4,167%	4,167%	4,167%	100,000%	100,000%
dev3	4,167%	0,000%	4,167%	0,000%	100,000%
dev4	4,167%	0,000%	4,167%	0,000%	100,000%
dev6	4,167%	4,167%	4,167%	100,000%	100,000%
dev8	8,333%	4,167%	4,167%	50,000%	50,000%
dev9	4,167%	0,000%	4,167%	0,000%	100,000%
dev10	4,167%	4,167%	4,167%	100,000%	100,000%
dev13	4,167%	0,000%	4,167%	0,000%	100,000%
dev14	4,167%	0,000%	4,167%	0,000%	100,000%
dev15	8,333%	4,167%	8,333%	50,000%	100,000%
dev16	4,167%	4,167%	4,167%	100,000%	100,000%
dev20	4,167%	0,000%	4,167%	0,000%	100,000%
dev24	4,167%	0,000%	4,167%	0,000%	100,000%
dev25	4,167%	0,000%	4,167%	0,000%	100,000%
dev27	4,167%	4,167%	4,167%	100,000%	100,000%
dev29	4,167%	0,000%	4,167%	0,000%	100,000%
dev34	4,167%	4,167%	4,167%	100,000%	100,000%
dev41	4,167%	0,000%	4,167%	0,000%	100,000%
dev44	8,333%	4,167%	8,333%	50,000%	100,000%
dev50	4,167%	0,000%	4,167%	0,000%	100,000%
dev56	4,167%	0,000%	4,167%	0,000%	100,000%
dev61	4,167%	0,000%	4,167%	0,000%	100,000%
dev62	12,500%	8,333%	12,500%	66,667%	100,000%

Fonte: Elaborado pelo autor (2022).

O projeto **p_24917** possui 27 tarefas (*challenges*) criadas entre 03/08/2020 e 17/03/2021, e possui 54 desenvolvedores que foram atribuídos e/ou sugeridos em pelo menos

uma dessas tarefas do projeto. Desses desenvolvedores, 20 (37,04%) foram sugeridos pela solução proposta nas respectivas tarefas em que estes foram de fato atribuídos. E desses, 7 (12,96%) desenvolvedores se encontravam entre os dez primeiros sugeridos (Top 10) respectivamente em cada uma das tarefas em que foram atribuídos. Porém, houve tarefas em que a solução proposta não conseguiu sugerir esses desenvolvedores nelas atribuídos. Exceto que dos 20 desenvolvedores sugeridos e atribuídos, todos eles a solução proposta os sugeriu em 100% das tarefas que foram atribuídos. Mas dos 7 desenvolvedores atribuídos e que se encontravam entre os dez primeiros sugeridos, 6 (11,11%) deles a solução proposta os sugeriu em 100% das tarefas que foram atribuídos. A Tabela 4.11 mostra mais detalhes do percentual de tarefas em que esses 20 desenvolvedores foram atribuídos e sugeridos. Os 34 (62,96%) desenvolvedores restantes do projeto compõem aqueles que foram sugeridos em tarefas dos quais não foram atribuídos, e também aqueles que foram atribuídos em tarefas dos quais não foram sugeridos pela solução proposta.

Tabela 4.11 – Percentual de tarefas do projeto p_24917 cujos desenvolvedores foram sugeridos e atribuídos.

Devs	**Percentual de tarefas atribuído**	**Percentual de tarefas sugerido (top 10) e atribuído**	**Percentual de tarefas sugerido e atribuído**	**Percentual Interseção (Top 10) / Atribuição**	**Percentual Interseção / Atribuição**
dev1	3,704%	0,000%	3,704%	0,000%	100,000%
dev2	3,704%	0,000%	3,704%	0,000%	100,000%
dev3	3,704%	0,000%	3,704%	0,000%	100,000%
dev7	3,704%	0,000%	3,704%	0,000%	100,000%
dev8	7,407%	7,407%	7,407%	100,000%	100,000%
dev9	3,704%	0,000%	3,704%	0,000%	100,000%
dev10	3,704%	0,000%	3,704%	0,000%	100,000%
dev13	3,704%	0,000%	3,704%	0,000%	100,000%
dev14	11,111%	3,704%	11,111%	33,333%	100,000%
dev17	11,111%	11,111%	11,111%	100,000%	100,000%
dev18	3,704%	0,000%	3,704%	0,000%	100,000%
dev19	3,704%	0,000%	3,704%	0,000%	100,000%
dev22	3,704%	0,000%	3,704%	0,000%	100,000%
dev23	7,407%	7,407%	7,407%	100,000%	100,000%
dev25	7,407%	7,407%	7,407%	100,000%	100,000%
dev28	11,111%	11,111%	11,111%	100,000%	100,000%
dev29	3,704%	0,000%	3,704%	0,000%	100,000%
dev30	3,704%	0,000%	3,704%	0,000%	100,000%
dev31	14,815%	14,815%	14,815%	100,000%	100,000%
dev35	3,704%	0,000%	3,704%	0,000%	100,000%

Fonte: Elaborado pelo autor (2022).

A Tabela 4.12 resume os percentuais das quantidades de desenvolvedores atribuídos que a solução proposta sugeriu para pelo menos uma tarefa (**>0% - 100%**) em que o respectivo desenvolvedor foi atribuído no projeto. Também mostra a quantidade percentual dos desenvolvedores que foram sugeridos pela solução proposta em todas as tarefas dos quais foram atribuídos (**100%**) qualquer que fossem suas posições na lista de sugestões. São também exibidas as quantidades percentuais desses desenvolvedores que se encontravam no Top 10 da listagem de sugestões para as tarefas dos quais foram atribuídos, considerando pelo menos uma tarefa (**(Top 10) >0% - 100%**) e todas essas tarefas (**(Top 10) 100%**). E por fim também exibida, para cada projeto, a quantidade percentual dos desenvolvedores que não foram sugeridos em nenhuma das tarefas aos quais foram atribuídos (**0%**).

Tabela 4.12 – Resumo das quantidades de desenvolvedores atribuídos e sugeridos em cada projeto.

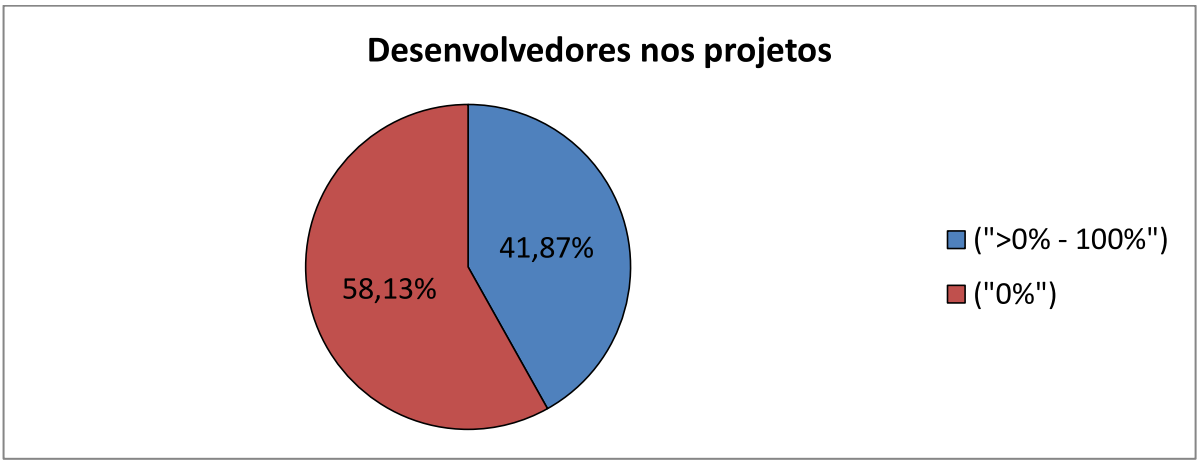
Id do Projeto	Tarefas	Devs	>0% - 100%	100%	(Top 10) >0% - 100%	(Top 10) 100%	0%
p_17469	234	85	14 (16,47%)	11 (12,94%)	11 (12,94%)	8 (9,41%)	71 (83,53%)
p_18718	136	38	22 (57,89%)	17 (44,74%)	15 (39,47%)	10 (26,32%)	16 (42,11%)
p_23783	97	39	25 (64,1%)	15 (38,46%)	16 (41,03%)	4 (10,26%)	14 (35,9%)
p_23849	24	68	23 (33,82%)	22 (32,35%)	10 (14,71%)	6 (8,82%)	45 (66,18%)
p_24917	27	54	20 (37,04%)	20 (37,04%)	7 (12,96%)	6 (11,11%)	34 (62,96%)

Fonte: Elaborado pelo autor (2022).

A pergunta de pesquisa definida no capítulo de introdução da presente dissertação, diz respeito em “como o suporte à reputação e ao esquecimento apoiam a seleção de desenvolvedores para tarefas de manutenção de software”. A solução proposta busca responder à pergunta utilizando valores dos critérios de reputação e do esquecimento para avaliar o conhecimento do desenvolvedor em *expertises* de tecnologias de desenvolvimento de software. Isso para apoiar a seleção desses desenvolvedores em tarefas de manutenção e desenvolvimento de software apresentando-os em uma lista de sugestões dos desenvolvedores; sendo estes possuindo as maiores quantidades de *expertises* em comum com a tarefa, e nessas *expertises* com os maiores valores de reputação e menores valores de esquecimento. Conforme pode ser observado nos resultados da Tabela 4.12, dos desenvolvedores atribuídos nas tarefas dos projetos analisados, muitos deles não foram sugeridos em nenhum momento pela solução proposta (em média 58,13% dos desenvolvedores dos projetos, conforme mostra o Gráfico 4.1). Isso porque para esses desenvolvedores não foram identificadas as *expertises* dos quais eles trabalharam em outras

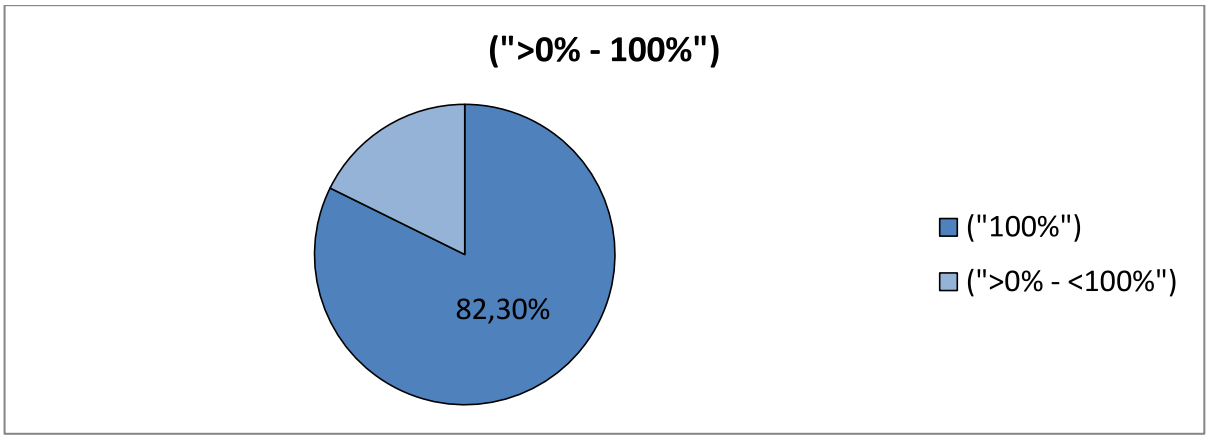
contribuições de desenvolvimento de software. E não tendo essas *expertises* identificadas, não é possível aplicar valores de reputação e esquecimento nesses desenvolvedores.

Gráfico 4.1 – Percentual médio de desenvolvedores não sugeridos e de desenvolvedores sugeridos nas tarefas aos quais foram atribuídos.



Fonte: Elaborado pelo autor (2022).

Gráfico 4.2 – Percentual médio de desenvolvedores sugeridos em 100% das tarefas em que foram atribuídos.

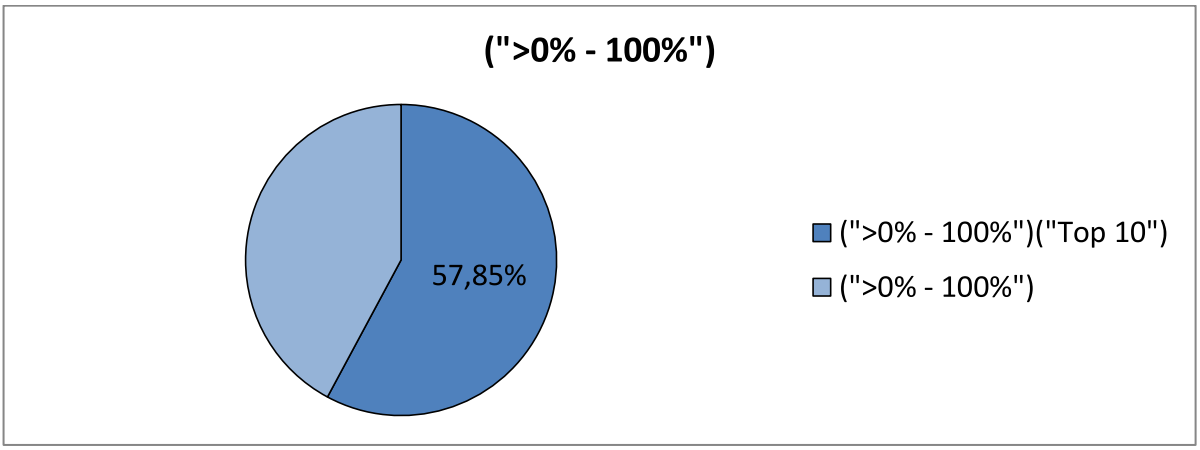


Fonte: Elaborado pelo autor (2022).

Porém, para aqueles desenvolvedores atribuídos que a solução proposta sugeriu-os em pelo menos uma das tarefas nos projetos (em média 41,87% dos desenvolvedores dos projetos), nota-se que muitos destes (média de 82,3% relativa aos 41,87%) foram sugeridos em todas as tarefas aos quais foram atribuídos, conforme mostra o Gráfico 4.2. Também, o uso dos valores de reputação e esquecimento nas *expertises* dos desenvolvedores mostrou apoiar a seleção de desenvolvedores ao notar que parte daqueles que foram sugeridos (média de 57,85% relativa aos 41,87%), em pelo menos uma tarefa das quais atribuídos, estes estavam presentes entre os dez primeiros nas listagens de sugestões, conforme mostrado no Gráfico 4.3. E também houve esse apoio para aqueles que foram sugeridos no top 10 em todas

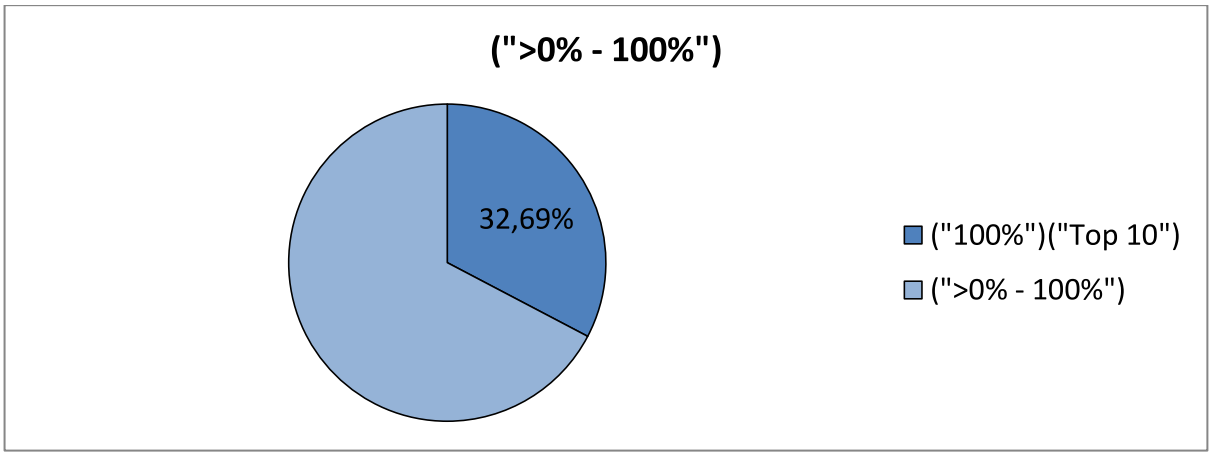
as tarefas dos quais foram atribuídos (média de 32,69% relativa aos 41,87%) conforma mostrado no Gráfico 4.4.

Gráfico 4.3 – Percentual médio de desenvolvedores sugeridos em pelo menos uma das tarefas aos quais foram atribuídos estando presentes no Top 10 das sugestões.



Fonte: Elaborado pelo autor (2022).

Gráfico 4.4 – Percentual médio de desenvolvedores sugeridos em 100% das tarefas aos quais foram atribuídos estando presentes no Top 10 das sugestões.



Fonte: Elaborado pelo autor (2022).

4.5.1 Ameaças à validade

Uma abordagem fundamental da avaliação é analisar o quão válidos são os resultados apresentados. E com isso certas ameaças podem comprometer a validade destes resultados. Essas ameaças estão relatadas de acordo com as normas de WOHLIN *et al*. (2012) e são descritas a seguir.

Ameaças à **Validade Externa** definem condições que limitam a generalização dos resultados desta avaliação. Uma limitação é a de que os projetos do *TopCoder* escolhidos para serem avaliados não representam as características da maioria dos projetos encontrados no

TopCoder. Pois esses projetos em sua maioria possuem poucos desenvolvedores vencedores e poucos *challenges* para alimentar a solução proposta com dados de contribuições dos desenvolvedores. Para mitigar este problema, a solução proposta dispõe do recurso de o desenvolvedor alimentar seu perfil de *expertises* com contribuições de desenvolvimento em outras plataformas de apoio ao desenvolvimento de software (*StackOverflow*, *GitHub*) além do *TopCoder*. No entanto, uma minoria de usuários do *TopCoder* dispunham o acesso às suas contribuições externas. Outra limitação observada refere-se aos dados utilizados nesta avaliação estarem viesados, uma vez que as amostras de desenvolvedores possuem grupos específicos que optaram por participarem nos respectivos projetos avaliados. Logo, existe um problema de auto seleção, visto que o ideal seria também considerar o contrafactual, que são desenvolvedores que poderiam ter participado, mas não participaram dos projetos avaliados. Portanto, novos estudos são necessários envolvendo também esses outros desenvolvedores nos projetos de desenvolvimento e manutenção de software; isso para chegar mais próximo de uma generalização dos resultados da avaliação.

Ameaças à **Validade Interna** dizem respeito ao conhecimento sobre todos os fatores que possam influenciar o objeto de estudo da presente avaliação. Nesse caso, o fator abordado é a listagem de sugestões de desenvolvedores nas tarefas (*challenges*) dos projetos avaliados do *TopCoder*. A premissa dessa pesquisa é a de que desenvolvedores são atribuídos em tarefas de manutenção e desenvolvimento de software através de suas aptidões em *expertises* das tecnologias utilizadas nessas tarefas, e com o uso da reputação e do esquecimento para medir o quanto de conhecimento têm esses desenvolvedores nas ditas *expertises*. Porém os resultados da avaliação mostraram uma maioria de desenvolvedores que foram historicamente atribuídos sem serem identificados pelas *expertises* para que estes pudessem ser sugeridos. Visto que se não tiver dados de *expertises* ligadas às contribuições dos desenvolvedores, então não é possível a utilização da solução desenvolvida. Para mitigar esse problema, a solução proposta dispõe do recurso de sugestão de formação de grupos, do qual além de buscar os desenvolvedores com *expertises* em comum com a tarefa, busca também desenvolvedores que foram atribuídos nas mesmas tarefas anteriores que os desenvolvedores atribuídos na tarefa atual. Porém novos estudos são necessários para averiguar os outros motivos de os desenvolvedores serem atribuídos em tarefas de desenvolvimento e manutenção de software. Outra ameaça à validade interna observada é em relação à utilização da solução proposta para processar maiores quantidades de dados de contribuições dos desenvolvedores em suas *expertises*, sendo isso uma limitação em ambientes computacionais convencionais. Para mitigar esse problema, um experimento dessa solução proposta foi executado em um

ambiente computacional de alto desempenho (MAGALHÃES *et al*., 2020). Isso para atestar a possibilidade de utilizar sistemas computacionais distribuídos para processamento de maiores quantidades de dados de desenvolvedores em suas contribuições nas *expertises* e em mais plataformas de apoio ao desenvolvimento de software.

4.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo foi apresentada a avaliação da funcionalidade de apresentar sugestões de desenvolvedores com conhecimentos em *expertises* (tecnologias) para tarefas de desenvolvimento e manutenção de software. Funcionalidade essa sendo parte principal da solução proposta desenvolvida para responder a como o suporte à reputação e ao esquecimento apoiam a seleção de desenvolvedores para tarefas de manutenção de software. A avaliação foi realizada utilizando dados históricos de atribuições de desenvolvedores nas tarefas de projetos públicos da plataforma *TopCoder*, sendo essas atribuições utilizadas para comparar com as sugestões de desenvolvedores simuladas nessas mesmas tarefas. Considerando o estudo experimental, não foi possível confirmar a hipótese de que a solução proposta sugere os mesmos desenvolvedores atribuídos, pois os desenvolvedores atribuídos em sua maioria, não foram sugeridos. Porém, com a realização do estudo de caso, e a análise dos dados depois do estudo experimental, foram encontradas evidências de que a solução proposta consegue sim sugerir desenvolvedores equivalentes aos atribuídos, mediante comparação de suas *expertises* com aquelas exigidas pelas tarefas.

5 CONSIDERAÇÕES FINAIS

A busca de desenvolvedores capacitados em *expertises* de tecnologias para realizarem tarefas de desenvolvimento e manutenção de software é o foco deste trabalho para apresentar uma solução que venha a suprir a demanda crescente por esses profissionais. Essa busca de desenvolvedores envolve o uso dos conceitos de reputação e esquecimento para os gerentes de projetos estabelecerem confiança nas habilidades de programação dos desenvolvedores. Este trabalho começa apresentando uma fundamentação teórica com os principais conceitos envolvendo o apoio à seleção de desenvolvedores para tarefas de manutenção em projetos de software. Primeiramente, foi apresentado o conceito da reputação como uma forma de, o gerente de projetos, confiar no desenvolvedor em suas habilidades com tecnologias de desenvolvimento de software. Em seguida, apresentou-se o conceito do esquecimento ser algo importante para atestar que o desenvolvedor precisa estar utilizando ativamente seus conhecimentos para o desenvolvimento e manutenção de software. Por fim, foi apresentado o conceito do *crowdsourcing* como um modelo de negócio utilizado no recrutamento de programadores distribuídos globalmente, e de diversas especialidades.

Considerando as dificuldades em encontrar desenvolvedores com a capacidade necessária para tarefas de software, este trabalho propôs uma solução para apoiar a seleção desses profissionais por meio da reputação e do esquecimento destes nos conhecimentos de tecnologias em projetos de software. Em cada tarefa de software são listadas sugestões de desenvolvedores considerando reputação e esquecimento nos conhecimentos exigidos pela tarefa. Os perfis de *expertises* dos desenvolvedores podem ser alimentados com suas contribuições históricas nesses conhecimentos em diferentes plataformas de apoio ao desenvolvimento de software. A pergunta de pesquisa definida neste trabalho diz respeito em como o suporte à reputação e ao esquecimento apoiam a seleção de desenvolvedores para tarefas de manutenção de software. Para responder à pergunta, uma avaliação foi realizada com intuito de verificar equivalência entre as sugestões de desenvolvedores com as atribuições históricas destes nas tarefas de projetos de software. Foram encontradas evidências a favor do uso da reputação e do esquecimento para o apoio na seleção de desenvolvedores em tarefas de manutenção de software.

5.1 CONTRIBUIÇÕES

O presente trabalho apresentou também contribuições que vão além da busca de sugestões de desenvolvedores aptos em conhecimentos para tarefas de projetos de software. Essas contribuições são:

- Implantação de um serviço web com cadastro de usuários para atuarem como desenvolvedores e/ou coordenadores nas tarefas de manutenção e desenvolvimento de software.
- Junção dos critérios de reputação com o critério do esquecimento para a seleção de especialistas e hiperespecialistas em *expertises* de tecnologias para desenvolvimento de software.
- Além das sugestões de desenvolvedores em cada tarefa, também foi realizada a implantação das sugestões de tarefas para cada desenvolvedor.
- Implantação das sugestões de grupos de desenvolvedores, por meio do uso da ontologia *ArchiRiOnt*, considerando *expertises* e tarefas trabalhadas em comum entre esses desenvolvedores.
- Relação das plataformas *GitHub*, *StackOverflow* e *TopCoder* para usuários desenvolvedores exportarem, na ferramenta da solução proposta, suas contribuições dessas plataformas para formar seu perfil de *expertises*.
- Desenvolvimento de visualizações gráficas para apoio na escolha de desenvolvedores:
    - Gráfico com as contribuições do desenvolvedor por *expertise*.
    - Gráfico de radar com os valores dos critérios de reputação e esquecimento.
    - Gráfico de visualização da evolução do valor de esquecimento ao longo do tempo para cada *expertise*.
    - Visualização, em forma de grafos, dos desenvolvedores com *expertises* e tarefas em comum entre eles.
    - Gráfico de *Sankey* para visualização das *expertises* e tarefas em comum entre desenvolvedores.

5.2 TRABALHOS FUTUROS

Como trabalhos futuros, considera-se a incorporação de outras plataformas de apoio ao desenvolvimento de software, além daquelas já utilizadas, para possibilitar maior enriquecimento dos perfis de *expertises* dos desenvolvedores com contribuições externas. Outra possibilidade de trabalho futuro é a realização de novas avaliações das funcionalidades desenvolvidas na solução proposta, utilizando outros conjuntos de projetos. Por fim, também é considerado executar uma avaliação em contexto real de utilização da solução proposta.

REFERÊNCIAS

AL-ANI, Ban et al. An understanding of the role of trust in knowledge seeking and acceptance practices in distributed development teams. In: **2011 IEEE Sixth International Conference on Global Software Engineering**. IEEE, 2011. p. 25-34.

AL-ANI, Ban; REDMILES, David. In strangers we trust? Findings of an empirical study of distributed teams. In: **2009 Fourth IEEE International Conference on Global Software Engineering**. IEEE, 2009. p. 121-130.

ARTZ, Donovan; GIL, Yolanda. A survey of trust in computer science and the semantic web. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 5, n. 2, p. 58-71, 2007.

BADASHIAN, Ali Sajedi. Realistic bug triaging. In: **Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on**. IEEE, 2016. p. 847-850.

BADASHIAN, Ali Sajedi; HINDLE, Abram; STROULIA, Eleni. Crowdsourced bug triaging: Leveraging q&a platforms for bug assignment. In: **International Conference on Fundamental Approaches to Software Engineering**. Springer, Berlin, Heidelberg, 2016. p. 231-248.

BASILI, Victor R.; CALDIERA, Gianluigi; ROMBACH, H. Dieter. The goal question metric approach. **Encyclopedia of software engineering**, p. 528-532, 1994.

BEGEL, Andrew; BOSCH, Jan; STOREY, Margaret-Anne. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. **IEEE Software**, v. 30, n. 1, p. 52-66, 2013.

CALEFATO, Fabio; LANUBILE, Filippo; NOVIELLI, Nicole. A preliminary analysis on the effects of propensity to trust in distributed software development. In: **2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)**. IEEE, 2017. p. 56-60.

CAVERLEE, James; LIU, Ling; WEBB, Steve. The SocialTrust framework for trusted social information management: Architecture and algorithms. **Information Sciences**, v. 180, n. 1, p. 95-112, 2010.

CRUZ NAVEA, Pablo. Improving Recommender Systems Using Knowledge Obsolescence as a Predictor of Trust. In: **XXIV Concurso Latinoamericano de Tesis de Maestría (CLTM-CLAI)-JAIIO 46 (Córdoba, 2017).** 2017.

DE NEIRA, Anderson Bergamini; STEINMACHER, Igor; WIESE, Igor Scaliante. Characterizing the hyperspecialists in the context of crowdsourcing software development. **Journal of the Brazilian Computer Society**, v. 24, n. 1, p. 17, 2018.

EBBINGHAUS, Hermann. **Über das gedächtnis: untersuchungen zur experimentellen psychologie**. Duncker & Humblot, 1885.

ERLIKH, Len. Leveraging legacy system dollars for e-business. **IT professional**, v. 2, n. 3, p. 17-23, 2000.

GOYAL, Anjali; SARDANA, Neetu. Machine Learning or Information Retrieval Techniques for Bug Triaging: Which is better?. **e-Informatica Software Engineering Journal**, v. 11, n. 1, 2017.

GUERCIO, Hugo et al. Complex Network Analysis in a Software Ecosystem: Studying the Eclipse Community. In: **2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design (CSCWD)**. IEEE, 2018. p. 618-623.

HACKER, Janine Viol et al. Trust in Virtual Teams: A Multidisciplinary Review and Integration. **Australasian Journal of Information Systems**, v. 23, 2019.

HATTORI, Lile Palma; LANZA, Michele; ROBBES, Romain. Refining code ownership with synchronous changes. **Empirical Software Engineering**, v. 17, n. 4-5, p. 467-499, 2012.

ILLAHI, Inam et al. An Empirical Study on Competitive Crowdsource Software Development: Motivating and Inhibiting Factors. **IEEE Access**, v. 7, p. 62042-62057, 2019.

KANG, Keumseok; HAHN, Jungpil. Learning and forgetting curves in software development: Does type of knowledge matter?. **ICIS 2009 Proceedings**, p. 194, 2009.

KHATUN, Afrina; SAKIB, Kazi. A bug assignment technique based on bug fixing expertise and source commit recency of developers. In: **Computer and Information Technology (ICCIT), 2016 19th International Conference on**. IEEE, 2016. p. 592-597.

KINTAB, Ghadeer A.; ROY, Chanchal K.; MCCALLA, Gordon I. Recommending software experts using code similarity and social heuristics. In: **Proceedings of 24th Annual International Conference on Computer Science and Software Engineering**. IBM Corp., 2014. p. 4-18.

KITCHENHAM, B.; CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering. Version 2.3 University of keele (software engineering group, school of computer science and mathematics) and Durham. **Department of Conputer Science, UK**, 2007.

KRÜGER, Jacob et al. Do you remember this source code?. In: **2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)**. IEEE, 2018. p. 764-775.

KRÜGER, Jacob et al. Understanding How Programmers Forget. **Software Engineering and Software Management 2019**, 2019.

LATOZA, Thomas D.; VAN DER HOEK, Andre. Crowdsourcing in software engineering: Models, motivations, and challenges. **IEEE software**, v. 33, n. 1, p. 74-80, 2016.

LEHMAN, Meir M. Programs, life cycles, and laws of software evolution. **Proceedings of the IEEE**, v. 68, n. 9, p. 1060-1076, 1980.

LÉLIS, Cláudio Augusto S. et al. ArchiRI-uma arquitetura baseada em ontologias para a troca de informações de reputação. In: **Anais do XII Simpósio Brasileiro de Sistemas de Informação**. SBC, 2016. p. 060-067.

LÉLIS, C.A.S.: Um modelo dinâmico de reputação para apoiar a manutenção colaborativa de software. Dissertação (Mestrado em Ciência da Computação), Universidade Federal de Juiz de Fora (UFJF), Juiz de Fora - MG (2017).

MAGALHÃES, Nathan et al. Suporte às atividades de manutenção de software em bases de dados abertas e distribuídas. In: **Anais do XXI Simpósio em Sistemas Computacionais de Alto Desempenho**. SBC, 2020. p. 227-238.

MAO, Ke et al. Developer recommendation for crowdsourced software development tasks. In: **Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on**. IEEE, 2015. p. 347-356.

MEYERSON, Debra et al. Swift trust and temporary groups. **Trust in organizations: Frontiers of theory and research**, v. 166, p. 195, 1996.

MIGUEL, Marcos Alexandre et al. A framework to support effort estimation on software maintenance and evolution activities. In: **Proceedings of the XII Brazilian Symposium on Information Systems on Brazilian Symposium on Information Systems: Information Systems in the Cloud Computing Era-Volume 1**. Brazilian Computer Society, 2016. p. 31.

MOHAN, Devina et al. Visheshagya: Time based expertise model for bug report assignment. In: **Contemporary Computing (IC3), 2016 Ninth International Conference on**. IEEE, 2016. p. 1-6.

OLIVEIRA JR, Marcio et al. Recommending External Developers to Software Projects based on Historical Analysis of Previous Contributions. In: **Proceedings of the XXXIII Brazilian Symposium on Software Engineering**. 2019. p. 417-426.

PFLEEGER, Shari Lawrence. **Engenharia de software: teoria e prática**. Prentice Hall, 2004.

QU, Xiangli; ZHONG, Jingwei; YANG, Xuejun. Towards reliable and trustworthy cooperation in Grid: A pre-evaluating set based trust model. In: **International Conference on Computational Science and Its Applications**. Springer, Berlin, Heidelberg, 2006. p. 224-235.

SCHETTINO, Vinicius et al. Towards community and expert detection in open source global development. In: **2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)**. IEEE, 2019. p. 350-355.

SHOKRIPOUR, Ramin et al. A time-based approach to automatic bug report assignment. **Journal of Systems and Software**, v. 102, p. 109-122, 2015.

SIRQUEIRA, Tassio Ferenzini Martins et al. **Aplicação de Métodos Estatísticos em Engenharia de Software: Teoria e Prática**. In *Engenharia no Século XXI*. Volume 18, Editora Poisson, 2020, pages 228-246.

SOTO, Juan Pablo; VIZCAÍNO, Aurora; PIATTINI, Mario. Fostering Knowledge Reuse in Communities of Practice by Using a Trust Model and Agents. **International Journal of Information Technology & Decision Making**, v. 16, n. 05, p. 1409-1439, 2017.

STEINMACHER, Igor; TREUDE, Christoph; GEROSA, Marco Aurélio. Let me in: Guidelines for the successful onboarding of newcomers to open source projects. **IEEE Software**, v. 36, n. 4, p. 41-49, 2018.

SUN, Xiaobing et al. Effectiveness of exploring historical commits for developer recommendation: an empirical study. **Frontiers of Computer Science**, v. 12, n. 3, p. 528-544, 2018.

TAGLIAFERRI, Mirko; ALDINI, Alessandro. A Taxonomy of Computational Models for Trust Computing in Decision-Making Procedures. In: **European Conference on Cyber Warfare and Security**. Academic Conferences International Limited, 2018. p. 571-XVI.

TAVARES, Jacimar Fernandes et al. Uma Infraestrutura baseada em Múltiplas Visões Interativas para Apoiar Evolução de Software. **iSys-Revista Brasileira de Sistemas de Informação**, v. 8, n. 1, p. 65-101, 2015.

TRAINER, Erik H.; REDMILES, David F. Bridging the gap between awareness and trust in globally distributed software teams. **Journal of Systems and Software**, v. 144, p. 328-341, 2018.

WEI, Qingjie; LIU, Jiao; CHEN, Jun. A Method for Recommending Bug Fixer Using Community Q&A Information. In: **MATEC Web of Conferences**. EDP Sciences, 2018. p. 03031.

WOHLIN, Claes et al. **Experimentation in software engineering**. Springer Science & Business Media, 2012.

XIE, Xinqiang; YANG, Xiaochun; WANG, Bin. SoftRec: Multi-Relationship Fused Software Developer Recommendation. **Applied Sciences**, v. 10, n. 12, p. 4333, 2020.

YANG, Hui et al. Dr_psf: Enhancing developer recommendation by leveraging personalized source-code files. In: **Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual**. IEEE, 2016. p. 239-244.

YANG, Cheng et al. RevRec: A two-layer reviewer recommendation algorithm in pull-based development model. **Journal of Central South University**, v. 25, n. 5, p. 1129-1143, 2018.

YE, Yunwen; YAMAMOTO, Yasuhiro; NAKAKOJI, Kumiyo. A socio-technical framework for supporting programmers. In: **Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering**. ACM, 2007. p. 351-360.

ZANATTA, Alexandre Lazaretti et al. Barriers faced by newcomers to software-crowdsourcing projects. **IEEE Software**, v. 34, n. 2, p. 37-43, 2017.

ZHANG, Jun; ACKERMAN, Mark S.; ADAMIC, Lada. Expertise networks in online communities: structure and algorithms. In: **Proceedings of the 16th international conference on World Wide Web**. 2007. p. 221-230.

ZHOU, Cheng; KUTTAL, Sandeep Kaur; AHMED, Iftekhar. What Makes a Good Developer? An Empirical Study of Developers' Technical and Social Competencies. In: **2018**

IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 2018. p. 319-321.

ZHU, Jiangang; SHEN, Beijun; HU, Fanghuai. A learning to rank framework for developer recommendation in software crowdsourcing. In: **Software Engineering Conference (APSEC), 2015 Asia-Pacific**. IEEE, 2015. p. 285-292.

APÊNDICE A – Mapeamento Sistemático da Literatura

Segundo KITCHENHAM e CHARTERS (2007) o mapeamento sistemático busca avaliar a área de interesse, respondendo questões gerais de pesquisa. O intuito é de detectar as principais abordagens presentes na literatura para tratar a área de interesse. Através de um protocolo pré-definido, este tipo de estudo busca seguir passos metodológicos precisos e rigorosos para a seleção e análise de documentos relevantes. O presente trabalho seguiu esse processo e, para isso, foi utilizada uma ferramenta nomeada Parsifal[26]. Essa é uma ferramenta online concebida para apoiar pesquisadores na realização de revisões sistemáticas da literatura no contexto da Engenharia de Software.

A.1 PLANEJAMENTO

A etapa de planejamento visa definir os objetivos a serem atingidos pela pesquisa e pelo protocolo do mapeamento. Seguindo as diretrizes de KITCHENHAM e CHARTERS (2007) para definir um protocolo de mapeamento, o primeiro passo é especificar o objetivo do mapeamento para que as questões de mapeamento possam ser definidas. O objetivo deste mapeamento foi definido como:

"Buscar **conhecer** o estado-da-arte de estudos primários sobre abordagens (técnicas, modelos, ferramentas, frameworks, métodos) que **recomendem (sugerem)** desenvolvedores adequados para tarefas de manutenção **relacionadas** com relatórios de erros (*bug reports*) e solicitações de mudanças (*change requests*), no **contexto** da manutenção, desenvolvimento e evolução de software."

A.1.1 Questões de mapeamento

As questões de mapeamento foram então elaboradas a partir da definição acima, para assim alcançar o objetivo do mapeamento. A Tabela A.1 mostra as respectivas questões e as suas motivações.

A.1.2 PICOC

Com o objetivo e as questões de mapeamento definidos, o segundo passo então é a definição dos parâmetros de: *Population*, *Intervention*, *Comparison*, *Outcome*, *Context*, que juntos, recebem o nome de PICOC (WOHLIN *et al*., 2012). Para este mapeamento, é apresentado na Tabela A.2 o PICOC definido juntamente com as respectivas descrições.

[26] https://parsif.al/

Tabela A.1 – Questões de Mapeamento.

Questão de Mapeamento	Motivação
QM1 – Qual é a quantidade de artigos publicados sobre o tema abordado ao longo dos anos?	Identificar o número de publicações relevantes para com o tema abordado e conhecer a distribuição das publicações ao longo dos anos.
QM2 – Quem são os autores relevantes para com o tema abordado?	Buscar identificar os autores de maior proeminência no tema abordado para obter possíveis fontes de informação sobre o assunto.
QM3 – Quais são os meios de publicação (simpósios, revistas, etc.) relevantes para o tema abordado?	Para planejar os trabalhos sobre o tema abordado como objetivo de atender aos requisitos estabelecidos dos meios de publicação relevantes.
QM4 – Quais as soluções desenvolvidas para apoio à atribuição de desenvolvedores em tarefas de manutenção?	Conhecer as soluções já propostas no tema abordado no intuito de saber quais são as perspectivas para com a elaboração de uma nova solução.

Fonte: Elaborado pelo autor (2018).

Tabela A.2 – Definição do PICOC.

PICOC	Descrição	Termos derivados
Population	Tarefas de manutenção de software relacionadas com relatório de bugs, solicitações de mudança, rastreamento e repositório de bugs.	*bug report*, *change request*, *bug tracking, bug repository*
Intervention	Recomendação ou atribuição de desenvolvedores mantenedores para com tarefas de manutenção.	*developer recommendation, developer assignment, fixer recommendation, bug assignment*
Comparison	–	–
Outcome	Abordagens (técnicas, modelos, ferramentas, frameworks, métodos) sobre recomendação de desenvolvedores.	*approach, technique, model, tool, framework, method*
Context	Manutenção, Desenvolvimento e Evolução de Software.	*software maintenance, software development, software evolution*

Fonte: Elaborado pelo autor (2018).

A.1.3 *String* de busca

Para definir a *string* de busca, foram utilizados os itens que compõem o PICOC no intuito de retirar as palavras e expressões mais relevantes para com o tema abordado. Esta atividade tem por finalidade encontrar o melhor conjunto de palavras, as quais são

relacionadas por operadores lógicos. A Tabela A.3 exibe as palavras chaves definidas e seus respectivos sinônimos (caso existente), organizadas conforme os itens do PICOC.

Tabela A.3 – Palavras chaves definidas.

Palavra chave	Sinônimos	PICOC relativo
bug report	–	**Population**
bug repository *bug tracking*	–	
change request	–	
bug assignment	–	**Intervention**
developer recommendation	*developer assignment* *fixer recommendation*	
approach *framework* *method*	–	**Outcome**
model *technique* *tool*	–	
software maintenance *software development* *software evolution*	–	**Context**

Fonte: Elaborado pelo autor (2018).

Este conjunto de palavras chaves forma a *string* de busca, que tem por objetivo fazer as bases fornecerem estudos mais aderentes ao contexto da pesquisa. A *string* final foi definida após a incorporação de termos diferentes encontrados durante os resultados dos testes pilotos realizados com versões prévias da *string*. Como resultado, foi gerada a *string* a seguir:

("*bug report*" **OR** "*change request*" **OR** "*bug tracking*" **OR** "*bug repository*") **AND** ("*developer recommendation*" **OR** "*developer assignment*" **OR** "*fixer recommendation*" **OR** "*bug assignment*") **AND** ("*approach*" **OR** "*technique*" **OR** "*model*" **OR** "*tool*" **OR** "*framework*" **OR** "*method*") **AND** ("*software maintenance*" **OR** "*software development*" **OR** "*software evolution*")

A.1.4 Critérios de inclusão e exclusão

Com a *string* pronta, o próximo passo foi definir os critérios de inclusão e exclusão dos artigos que serão pesquisados. Estes têm por objetivo avaliar a relevância das publicações encontradas para determinar sua inclusão ou não nos resultados do mapeamento. Os critérios são postos em prática durante as filtragens realizadas na etapa de condução do mapeamento, descrita na subseção A.2.

Critérios de inclusão:

- Publicação aborda sobre escolha de desenvolvedores para tarefas de manutenção.
- Publicação não é excluída por nenhum critério de exclusão.

Critérios de exclusão:

- Publicação não está escrita na língua inglesa.
- Publicação não possui um resumo.
- Publicação não é do tipo artigo completo.
- Publicação não é um estudo primário.
- Publicação não disponibiliza publicamente o texto completo.
- Publicação não aborda sobre recomendação de desenvolvedores.

Para justificar a escolha dos critérios, alguns motivos são citados. Foram escolhidas publicações de língua inglesa para obter resultados que sejam relevantes e passíveis de utilização pela comunidade científica internacional. A presença de um resumo da publicação é importante para avaliação rápida e sucinta do trabalho. Publicações que não são do tipo artigo podem incluir diversos trabalhos como relatórios técnicos, anais de eventos, capítulos de livros e outras publicações que não possuem o foco diretamente na área pesquisada. Da mesma forma, estudos não primários, como mapeamentos sistemáticos, por estes não serem o objetivo de pesquisa deste trabalho não contribuiriam com a pesquisa de maneira efetiva. Por fim, foi definida a presença de artigos públicos pela necessidade de avaliação dos seus conteúdos para inclusão nesta pesquisa. Os demais critérios são relativos ao tema abordado.

A.1.5 Bases de pesquisa

O último passo da etapa de planejamento é a seleção das bases de pesquisa a serem efetuadas as buscas com a *string* definida. Foram selecionadas bases que em seu acervo estivessem presentes publicações da área de ciência da computação. Outro critério de escolha é em relação à presença de uma opção avançada para pesquisas, devido ao motivo da *string* de busca ter de ser aplicada para o mapeamento. Por fim, todas as bases selecionadas devem ser compatíveis com a ferramenta de apoio escolhida. As bases também foram escolhidas mediante sugestões propostas por KITCHENHAM e CHARTERS (2007).

Bases escolhidas:

- IEEE Digital Library (http://ieeexplore.ieee.org)
- Science@Direct (http://www.sciencedirect.com)
- Scopus (http://www.scopus.com)
- Springer Link (http://link.springer.com)

- ACM Digital Library (http://dl.acm.org/)

A.2 CONDUÇÃO

A primeira atividade executada na condução do mapeamento é a obtenção dos estudos através dos mecanismos de busca automática, presentes nas bases de pesquisa. Para isso, a *string* de busca foi executada em cada base escolhida, e os resultados obtidos foram exportados no formato *bibitex*, que foram importados para a ferramenta Parsifal. A base *Springer Link* disponibiliza a exportação dos resultados somente no formato *csv*.

Como o Parsifal tem suporte de inserção dos resultados apenas no formato *bibitex*, fez-se necessário utilizar a ferramenta Zotero[27] para gerar um arquivo *bibitex* dos resultados presentes no arquivo *csv*. Esta ferramenta dispõe de uma funcionalidade para listagem das publicações, através da busca de seus metadados mediante o uso do identificador único *Digital Object Identifier* (DOI) de cada publicação, disponível no arquivo *csv*. O Zotero gera um arquivo no formato *bibitex*, com esses metadados obtidos através do DOI de cada publicação.

As buscas foram realizadas entre **outubro e dezembro de 2018**. O número total de publicações retornadas foi de 372. Foram identificadas 97 duplicações, pertencentes em sua maioria à base *Scopus*, e as demais, à base ACM, que indexaram as publicações das outras bases escolhidas. As duplicatas foram removidas de ambas as bases antes do início das filtragens.

Para aplicação da primeira filtragem foram utilizados os critérios de inclusão e exclusão, definidos na etapa do planejamento, que não fossem específicos ao tema abordado, sendo estes nomeados critérios básicos. Os demais critérios, relacionados com o tema, foram utilizados na segunda filtragem, que consiste na leitura dos títulos e resumos para identificar as publicações relevantes com o tema.

Após a remoção das 97 duplicações, foi realizada a primeira filtragem sobre as 275 publicações únicas. Destas, foram rejeitadas 28 considerando os critérios básicos, restando então 247 artigos únicos. Aplicando a segunda filtragem, foram rejeitados 161 artigos que não correspondiam com o tema abordado, restando então 86 artigos relevantes para serem lidos e responderem as questões de mapeamento. A Tabela A.4 mostra para cada base as quantidades de artigos em cada etapa da condução.

[27] https://www.zotero.org/

Tabela A.4 – Quantidades de artigos por base em cada etapa.

Base	Publicações retornadas	Após remoção de duplicações	Após 1ª filtragem	Após 2ª filtragem
ACM	25	8	8	7
IEEE	90	90	87	36
Science	22	22	20	6
Scopus	194	114	97	27
Springer	41	41	35	10
Total	**372**	**275**	**247**	**86**

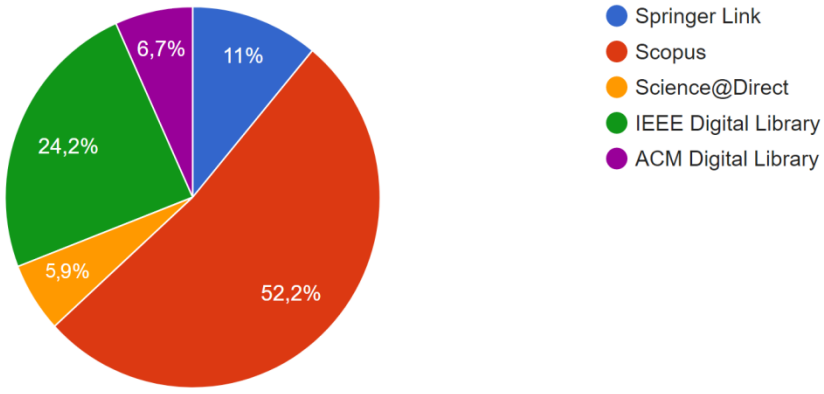
Fonte: Elaborado pelo autor (2018).

A condução do mapeamento visa capturar, selecionar e filtrar os artigos que se adequem à pesquisa, caracterizados pelos critérios de inclusão e exclusão. Obtidos os artigos, é possível efetuar uma avaliação do panorama do tema abordado, analisando os dados destes. Esta atividade é realizada na terceira etapa do mapeamento, o relatório geral.

A.3 RELATÓRIO GERAL

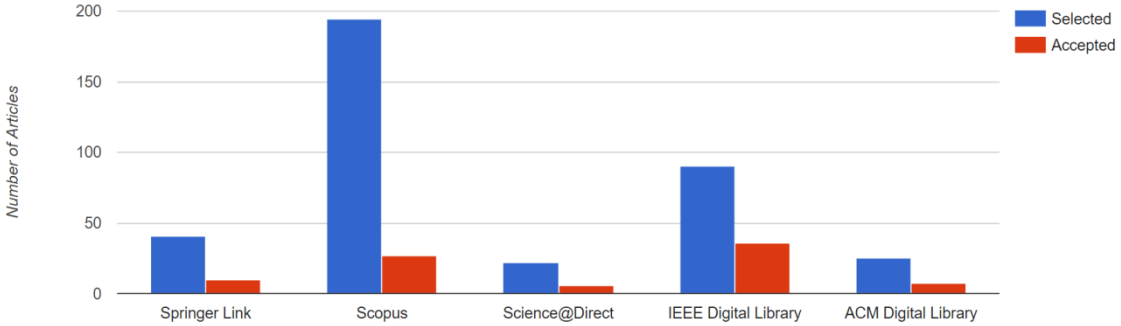
Para representar o cenário geral do mapeamento podem ser descritas algumas informações importantes. Primeiramente, pode-se analisar a distribuição do número de publicações por base para demonstrar a principal base de interesse da área, onde a maioria dos estudos está indexada. O Gráfico A.1 mostra a distribuição dos estudos obtidos por base, antes de serem realizadas as filtragens. O Gráfico A.2 mostra a distribuição dos estudos aceitos em cada base até a segunda filtragem.

Gráfico A.1 – Distribuição dos artigos por base.



Fonte: Elaborado pelo autor com uso do Parsifal (2018).

Gráfico A.2 – Distribuição de artigos aceitos por base.



Fonte: Elaborado pelo autor com uso do Parsifal (2018).

Comparando esses gráficos, algumas observações podem ser destacadas. Primeiramente, a base que apresentou maior relevância na busca inicial, foi a *Scopus*, por esta ter tido o maior número de resultados. Em contrapartida, nesta base foi encontrado o maior número de falsos positivos (artigos encontrados pela *string* de busca, mas que não foram aceitos durante o processo). A base IEEE por sua vez se destacou como sendo a que obteve o maior número de artigos aceitos, tornando-a mais relevante para com os resultados finais.

Outra observação que pode ser feita, é em relação ao menor número de artigos encontrados. Neste quesito, a base Science@Direct obteve o menor número de estudos obtidos na busca inicial. Esta base juntamente com a ACM Digital Library, tornaram-se as bases com o menor número de artigos aceitos.

Apesar de a base *Scopus* ter obtido a segunda maior quantidade de artigos aceitos, foi constatado que estes são indexados de outras bases, principalmente da IEEE. E visto que a base IEEE se tornou a base com mais artigos aceitos, embora seja a que retornou a segunda maior quantidade na busca inicial, conclui-se que ambas as bases são as mais relevantes para a área do tema abordado, com mais destaque para a IEEE. É intrigante notar que usando a mesma *string* de busca nas bases *Scopus* e IEEE, esta omitiu alguns resultados no qual a *Scopus* retornou indexado desta, pois caso contrário, teriam sido identificadas como duplicações. O mesmo fato também ocorreu entre a base ACM para com a IEEE.

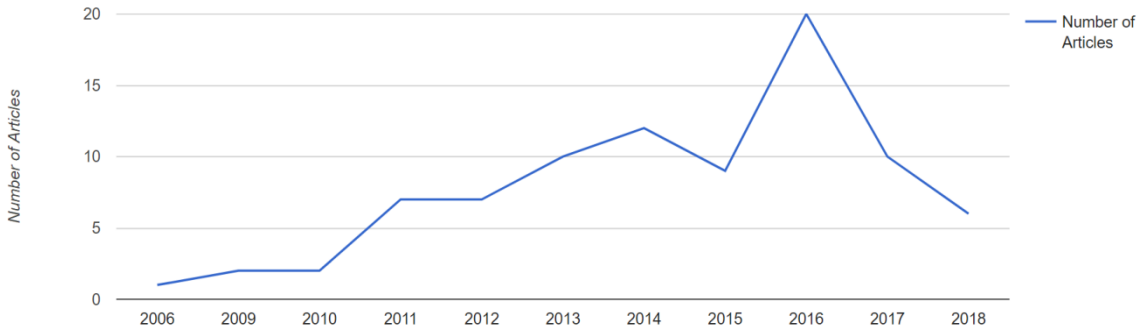
A.4 RESULTADOS OBTIDOS

Com o processo de busca concluído e, apresentadas as análises do relatório de mapeamento, os resultados de pesquisa obtidos podem ser apresentados. Esses resultados são as respostas para as questões de mapeamento apresentadas no protocolo. Esta subseção visa apresentar as respostas para cada uma das questões de mapeamento através dos conteúdos

relevantes extraídos da seleção final dos 86 artigos aprovados. Mais detalhes técnicos em relação aos artigos se encontram disponíveis através deste link[28].

A.4.1 QM1 – Qual é a quantidade de artigos publicados sobre o tema abordado ao longo dos anos?

Gráfico A.3 – Quantidade de artigos por ano de publicação.



Fonte: Elaborado pelo autor com uso do Parsifal (2018).

Os artigos publicados possuem uma distribuição estável entre 2006 e 2010, com uma ascendência até 2014 e seguida de uma ligeira queda no número de novas publicações em 2015. Em 2016 houve um aumento significativo na quantidade de novas contribuições para com o tema abordado, sendo seguido por uma queda repentina de mesma intensidade em 2017. Até o fechamento da escrita deste mapeamento evidenciou-se um decréscimo de publicações em 2018. O Gráfico A.3 apresenta a distribuição das publicações ao decorrer dos anos, dos artigos relacionados com o tema abordado e escolhidos na última filtragem.

A.4.2 QM2 – Quem são os autores relevantes para com o tema abordado?

Autores e coautores são ambos importantes para a elaboração das publicações relevantes a um tema de pesquisa. Com isso, optou-se por classificar a relevância dos autores com base na quantidade de publicações (retornadas neste mapeamento) em que estes são autores primários (no qual o nome pessoal aparece primeiro na publicação) ou são autores secundários (coautores). Para cada quantidade de publicações em que o autor é primário foi atribuído peso (2) ao valor; e para quando o autor é secundário, peso (1). Os pesos atribuídos foram somados para classificar cada autor por relevância. O cálculo da relevância do autor ocorre mediante o dobro da quantidade de publicações em que o autor é primário, ser somado com a quantidade de publicações em que o autor é secundário, gerando assim o resultado que indica a relevância do autor. A Tabela A.5 apresenta a lista dos autores por ordem decrescente da soma dos pesos atribuídos.

[28] https://github.com/nathan2m/dissertacao/tree/main/mapeamento

Foi identificado um total de 164 autores distintos, sendo 53 deles com publicações em que é autor primário em pelo menos uma destas, e os 111 restantes que possuem publicações sendo apenas autores secundários. Devido ao excesso de autores com poucas quantidades de publicações, optou-se por exibir aqueles que apresentaram somatório dos pesos (relevância) igual ou superior a seis (6). Os demais autores com suas respectivas quantidades de publicações estão disponíveis por meio deste link[29].

Tabela A.5 – Principais autores.

#	Nome do autor	Publicações como primário	Publicações como secundário	Total	Soma dos pesos
1	Tao Zhang	6	2	8	14
2	Wen Zhang	5	4	9	14
3	Ramin Shokripour	4	1	5	9
4	Huzefa Kagdi	3	3	6	9
5	Xin Xia	3	3	6	9
6	Qing Wang	0	9	9	9
7	John Anvik	1	6	7	8
8	Geunseok Yang	2	3	5	7
9	Song Wang	2	3	5	7
10	Byungjeong Lee	0	7	7	7
11	Ali Sajedi Badashian	3	0	3	6
12	Meera Sharma	3	0	3	6
13	V. Akila	3	0	3	6
14	Yguaratã C. Cavalcanti	3	0	3	6
15	Hui Yang	2	2	4	6
16	Xiaobing Sun	2	2	4	6
17	Sima Zamani	1	4	5	6

Fonte: Elaborado pelo autor (2018).

A.4.3 QM3 – Quais são os meios de publicação (simpósios, revistas, etc.) relevantes para o tema abordado?

A identificação dos meios de publicação para com o tema abordado é importante, no que se diz respeito, à busca de outras publicações que sejam referentes ao tema abordado, em um futuro trabalho. Os números de artigos identificados por cada meio de publicação estão listados em ordem decrescente de quantidade, divididos entre periódicos (Tabela A.6) e conferências (Tabela A.7). Por conta do excesso de meios de publicação que continham apenas uma publicação, optou-se por apresentar as conferências e periódicos que tivessem

[29] https://github.com/nathan2m/dissertacao/tree/main/mapeamento

pelo menos dois artigos aceitos de acordo com o protocolo descrito na subseção A.1. Em ambas as tabelas os demais meios de publicação generalizados como “outros” contém um artigo cada. Os demais meios de publicação se encontram disponíveis através deste link[30].

Tabela A.6 – Quantidade de artigos encontrados em cada periódico.

Nome do periódico	Artigos
Journal of Systems and Software	4
Journal of Software: Evolution and Process	3
Knowledge and Information Systems	2
IET Software	2
International Journal of Software Engineering and Knowledge Engineering	2
Chinese Journal of Electronics	2
Science China Information Sciences	2
Outros	14

Fonte: Elaborado pelo autor (2018).

Tabela A.7 – Quantidade de artigos encontrados em cada conferência.

Nome da conferência / simpósio	Artigos
(ICSE) International Conference on Software Engineering	5
(MSR) International Working Conference on Mining Software Repositories	5
(COMPSAC) Annual Computer Software and Applications Conference	4
(ICSME) International Conference on Software Maintenance and Evolution	4
(ASE) International Conference on Automated Software Engineering	3
(ICPC) International Conference on Program Comprehension	3
(APSEC) Asia-Pacific Software Engineering Conference	3
(SAC) Symposium on Applied Computing	2
(ESEM) International Symposium on Empirical Software Engineering and Measurement	2
(SEKE) International Conference on Software Engineering and Knowledge Engineering	2
(PROMISE) International Conference on Predictive Models in Software Engineering	2
Outros	20

Fonte: Elaborado pelo autor (2018).

A.4.4 QM4 – Quais as soluções desenvolvidas para apoio à atribuição de desenvolvedores em tarefas de manutenção?

Saber quais são as abordagens que já foram desenvolvidas (ferramentas, frameworks e técnicas) é importante, pois se deseja conhecer as estratégias e ideias criadas para com o tema abordado. Os resultados obtidos no mapeamento mostraram abordagens que sugerem, para

[30] https://github.com/nathan2m/dissertacao/tree/main/mapeamento

tarefas de manutenção, desenvolvedores que possuem histórico de contribuições dos seus conhecimentos em plataformas de apoio ao desenvolvimento de software (*GitHub*, *StackOverflow*, *TopCoder*). Outras abordagens encontradas visam verificar o estabelecimento de confiança dos desenvolvedores, para uns com os outros, em comunidades de desenvolvimento de projetos de softwares públicos. Essa confiança é construída através das contribuições em conhecimentos técnicos de tecnologias utilizadas nos softwares desenvolvidos. E com isso, esses desenvolvedores mostram ter boa reputação para com a comunidade do qual contribuem com programação. Também foram encontradas abordagens que, ao buscar as contribuições históricas dos desenvolvedores em conhecimentos técnicos para o desenvolvimento de software, essas abordagens têm como foco verificar o tempo que se passou desde a última contribuição e o momento da verificação dessas contribuições. Isto é, consideram que o conhecimento do desenvolvedor tende ao esquecimento quando não utilizado por algum tempo. E dessa forma, essas abordagens evitam sugerir desenvolvedores que já não estejam mais capacitados nas expertises aos quais trabalharam há muito tempo.

A.5 AMEAÇAS À VALIDADE DO MAPEAMENTO

Como ameaças à validade deste mapeamento, pode-se citar a diferenciação entre os motores de busca das bases de pesquisa. Pois foi constatado que a base *Scopus* retornou publicações indexadas de outras bases como IEEE e ACM, nos quais ao executar a mesma *string* de busca nelas, não apresentaram retorno dessas mesmas publicações. Isso poderia culminar em um desfalque de publicações relevantes, caso a busca fosse realizada somente fora da base *Scopus*. Um motor de busca mais genérico apresenta a vantagem de indexar publicações não listadas por um motor mais robusto das bases específicas. Há, porém, a desvantagem de aparecer uma maior quantidade de falsos positivos com um motor de busca mais genérico, fato este observado durante a filtragem na base *Scopus*, que teve a maior quantidade de falsos positivos identificados e removidos. As fontes escolhidas para a busca também podem ser uma ameaça à validade deste mapeamento, uma vez que não foram consideradas todas as bases existentes. Porém, as bases selecionadas foram consideradas como suficientes para obter uma grande representação da área pesquisada.

APÊNDICE B – Tabela de Tecnologias e Categorias

Tecnologias	Categorias
.NET	.NET
.NET 2.0	.NET
.NET 3.0	.NET
.NET 3.5	.NET
.NET 4.0	.NET
.NET System.Addins	.NET
1C Enterprise	1C Enterprise
ABAP	ABAP
Actian Data Analytics	Actian Data
Actian Data Integration	Actian Data
Actian Data Management	Actian Data
Actian DataConnect – Data Integration	Actian Data
Actian Ingres – Relational Database	Actian Data
Actian PSQL – Embeddable Database	Actian Data
Actian Vector – SMP Analytics Database	Actian Data
Actian Versant – NoSQL Object Database	Actian Data
Actian X – Hybrid Database	Actian Data
ActionScript	ActionScript
Active Directory	Active Directory
Activity Diagrams (TCUML)	UML
Ada	Ada
ADO.NET	.NET
AGS Script	AGS Script
AI	AI
AJAX	JavaScript
Amazon QLDB	Amazon QLDB
Android	Android
Android 2.0	Android
Android 2.1	Android
Android 2.2	Android
Angular 11	JavaScript
Angular 2+	JavaScript
Angular 4	JavaScript
Angular 8	JavaScript
Angular.js (1.0)	JavaScript
Ant	Java
ANTLR	ANTLR
Apache Camel	Apache Camel
Apache Cordova	Apache Cordova
Apache Derby	Apache Derby
Apache Kafka	Apache Kafka
ApacheConf	ApacheConf

Apex	Salesforce
API	API
API Testing	API
API Blueprint	API
Appium	Appium
AppleScript	AppleScript
Applet	Java
Arduino	C++
ASP	ASP
ASP.NET	.NET
ASP.NET AJAX	.NET
ASP.NET Core	.NET
ASP.NET Web API	.NET
ASP.NET Web Parts	.NET
AspectJ	AspectJ
Assembly	Assembly
AutoHotkey	AutoHotkey
Awk	Awk
AWS	AWS
AWS Lambda	AWS
Backbone.js	JavaScript
Bash	Shell
Batchfile	Batchfile
Beanstalk	AWS
BitBake	BitBake
Blackberry SDK	Blackberry SDK
Blade	PHP
Blockchain	Blockchain
Boo	Boo
Bootstrap	Bootstrap
Brainfuck	Brainfuck
Brightscript	Brightscript
C	C
C#	.NET
C++	C++
Calabash	Calabash
Castor	Java
Chatter	Salesforce
Cisco	Cisco
Clean	Clean
ClickOnce	.NET
CLIPS	CLIPS
Clojure	Clojure
Cloud Foundry	Cloud Foundry
CMake	CMake

COBOL	COBOL
CoffeeScript	CoffeeScript
Cognitive	Cognitive
ColdFusion	ColdFusion
COM	COM
COM+	COM
Commerce Server 2009	Commerce Server 2009
Common Lisp	Common Lisp
Coq	Coq
Crystal	Crystal
CSON	CoffeeScript
CSS	CSS
Cuda	C++
Custom Tag	Java
D	D
D3.JS	JavaScript
Dart	Dart
Data Science	Data Science
Delphi	Pascal
Diff	Diff
DIGITAL Command Language	DIGITAL Command Language
Django	Python
Docker	Docker
Dockerfile	Docker
Dojo	Dojo
DOT	Graphviz (DOT)
Drools	Java
Dropwizard	Java
DTrace	DTrace
Dynamodb	NoSQL
E	E
Eagle	XML
Eclipse Plugin	Java
EJB	Java
EJB 3	Java
Elasticsearch	Elasticsearch
Elixir	Elixir
Elm	Elm
Emacs Lisp	Common Lisp
Entity-Framework	.NET
Erlang	Erlang
Ethereum	Ethereum
Express	JavaScript
F#	.NET
Factor	Factor

Flash	Flash
Flex	Flex
Flutter	Dart
Force.com	Salesforce
Force.com Sites	Salesforce
Fortran	Fortran
Frege	Haskell
Game Maker Language	Game Maker Language
GCC Machine Description	Common Lisp
GDScript	GDScript
Gherkin	Gherkin
Git	Git
Github	Git
Gitlab	Git
GLSL	GLSL
Gnuplot	Gnuplot
Go	Go
Golang	Go
Google	Google
Google API	Google API
Google App Engine	Google App Engine
Google Dart	Dart
Gosu	Gosu
Gradle	Java
Graphviz (DOT)	Graphviz (DOT)
Groff	Roff
Groovy	Groovy
Gulp	JavaScript
Hack	PHP
Hadoop	Java
Handlebars	Handlebars
Harbour	Harbour
Haskell	Haskell
Haxe	Haxe
HCL	Ruby
Heroku	Java
Hibernate	Java
HLSL	HLSL
HPE Haven OnDemand	HPE Haven OnDemand
HTML	HTML
HTML5	HTML
HTTP	HTTP
Hyperledger Fabric	Hyperledger Fabric
iBATIS/MyBatis	iBATIS/MyBatis
IBM AIX	IBM

IBM Bluemix	IBM
IBM Cloud	IBM
IBM Cognitive	IBM
IBM COGNOS	IBM
IBM DB2	IBM
IBM Lotus Domino	IBM
IBM Lotus Notes	IBM
IBM PL/1	Java
IBM Rational Application Developer	Java
IBM Rational Data Architect	Java
IBM Rational Software Architect	Java
IBM Rational Team Concert	IBM
IBM Watson	IBM
IBM WebSphere Application Server	Java
IBM WebSphere MQ	Java
IGOR Pro	IGOR Pro
IIS	.NET
Illustrator	Illustrator
Inform 7	Inform 7
Inno Setup	Inno Setup
Ionic	Ionic
iOS	iOS
iOS 4.0	iOS
iOS 5.0	iOS
iOS 6.0	iOS
iOS 8.0	iOS
J2EE	Java
J2ME	Java
Jabber	Java
Java	Java
Java Application	Java
JavaBean	Java
JavaScript	JavaScript
JBoss Seam	Java
JDBC	Java
Jekyll	Ruby
JFace	Java
JIRA	Java
JMS	Java
JPA	Java
jQuery	JavaScript
jQuery Mobile	JavaScript
JSF	Java
JSON	JavaScript
JSONiq	JavaScript

JSP	Java
Julia	Julia
JUnit	Java
Jupyter Notebook	JavaScript
KiCad	Common Lisp
Kotlin	Java
Kubernetes	Kubernetes
LDAP	LDAP
Lex	Lex
Lightning	Lightning
Linux	Linux
Liquid	Liquid
LiveScript	LiveScript
LLVM	LLVM
Logos	Logos
LSL	LSL
Lua	Lua
M4	M4
Makefile	CMake
Markdown	Markdown
Mathematica	Mathematica
MATLAB	MATLAB
Maven	Java
Max	JavaScript
MAXScript	MAXScript
Meteor.js	JavaScript
Microsoft Azure	Microsoft Azure
Microsoft SilverLight	Microsoft SilverLight
MIDP 2.0	MIDP 2.0
Mobile	Mobile
MongoDB	NoSQL
MoonScript	MoonScript
MQL4	MQL5
MSMQ	MSMQ
MySQL	MySQL
Neo4J	Neo4J
nesC	nesC
NetLogo	Common Lisp
Nginx	Nginx
Nim	Nim
Nimrod	Nim
Nix	Nix
Node.js	JavaScript
NoSQL	NoSQL
Npm	JavaScript

NSIS	NSIS
Objective C	Objective C
Objective-C++	Objective C
Objective-J	Objective-J
OCaml	OCaml
Opa	Opa
OpenEdge ABL	OpenEdge ABL
OpenSCAD	OpenSCAD
Oracle 10g	Oracle Xg
Oracle 9i	Oracle Xg
OSX	OSX
Other	Other
Oxygene	Oxygene
Oz	Oz
Pascal	Pascal
PAWN	PAWN
Perl	Perl
PhoneGap	PhoneGap
Photoshop	Photoshop
PHP	PHP
Pike	Pike
PL/SQL	Oracle Xg
Play! Framework	Java
PLpgSQL	SQL
Pony	Pony
PostgreSQL	PostgreSQL
Postman	API
PostScript	PostScript
Power BI	Power BI
PowerShell	PowerShell
Predix	Predix
Processing	Processing
Prolog	Prolog
Protocol Buffer	Protocol Buffer
Protractor	JavaScript
Pug	Pug
Puppet	Puppet
PureScript	Haskell
Pure Data	Pure Data
Python	Python
QMake	QMake
QML	QML
R	R
Racket	Racket
Ragel	Ragel

Ragel in Ruby Host	Ragel
React Native	JavaScript
ReactJS	JavaScript
Reason	Rust
Redis	Redis
Regex	Regex
Remoting	.NET
REST	REST
RMI	Java
RobotFramework	RobotFramework
Roff	Roff
Rouge	Clojure
Ruby	Ruby
Ruby on Rails	Ruby on Rails
Rust	Rust
Salesforce	Salesforce
Salesforce.com	Salesforce
SaltStack	YAML
SAP	SAP
Sass	Sass
Scala	Scala
Scheme	Scheme
Scilab	Scilab
SCSS	CSS
Selenium	Selenium
Sencha Touch 2	JavaScript
Servlet	Java
SFDC Mobile	Salesforce
ShaderLab	ShaderLab
Sharepoint 3.0	Sharepoint 3.0
Shell	Shell
Siebel	Siebel
Smalltalk	Smalltalk
Smarty	Smarty
SoapUI	API
Solidity	Solidity
SourcePawn	PAWN
Spring	Java
SQF	SQF
SQL	SQL
SQL Server	SQL Server
SQL Server 2000	SQL Server
SQL Server 2008	SQL Server
SQLPL	SQL
Squirrel	Squirrel

SRecode Template	Common Lisp
SSIS	SSIS
Stan	Stan
Standard ML	Standard ML
Starlark	Python
Struts	Java
Stylus	Stylus
Svelte	HTML
Swagger	API
Swift	iOS
Swing	Java
SWT	Java
Tcl	Tcl
TeX	TeX
Thrift	Thrift
Titanium	JavaScript
TSQL	SQL
tvOS	tvOS
Twitter Bootstrap	Bootstrap
TypeScript	JavaScript
UML	UML
Unity3D	Unity3D
UnrealScript	Java
Use Case Diagrams (TCUML)	UML
V	Go
Vala	Vala
VB	.NET
VB.NET	.NET
VBA	.NET
Verilog	Verilog
Vertica	Vertica
VHDL	VHDL
Vim	Vim
VimL	Vim
Vim script	Vim
Vim Snippet	Vim
Visual-Studio	Visual-Studio
Visualforce	Salesforce
Visual Basic	.NET
VSCode	VSCode
Vue	Vue
Vuejs	JavaScript
Web Application	Web Application
Web Services	Web Services
WebAssembly	Common Lisp

Web Ontology Language	XML
Windows	Windows
Windows Communication Foundation	.NET
Windows Server	Windows Server
Windows Server 2003	Windows Server
Windows Workflow Foundation	.NET
WinForms Controls	.NET
wisp	Clojure
Word/Rich Text	Word/Rich Text
Wordpress	Wordpress
WPF	.NET
Xamarin	Objective C
XAML	XAML
Xcode	Xcode
XML	XML
XS	XS
XSL	XSL
XSLT	XML
XUL	XUL
Yacc	Yacc
YAML	YAML
YASnippet	YASnippet
Zig	Zig