UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

PAULO CEZAR DE OLIVEIRA

DAPROG - DOJO DE APRENDIZAGEM DE PROGRAMAÇÃO DE COMPUTADORES

DISSERTAÇÃO

CURITIBA

2019

PAULO CEZAR DE OLIVEIRA

DAPROG - DOJO DE APRENDIZAGEM DE PROGRAMAÇÃO DE COMPUTADORES

Dissertação apresentada ao Programa de Pós-graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de "Mestre em Computação Aplicada" – Área de Concentração: Engenharia de Sistemas Computacionais.

Orientador: Adolfo Gustavo Serra Seca Neto

Co-orientadora: Maria Cláudia Figueiredo Pereira Emer

CURITIBA

2019

Dados Internacionais de Catalogação na Publicação

Oliveira, Paulo Cezar de
DAPROG [recurso eletrônico] : Dojo de Aprendizagem de Programação de Computadores / Paulo Cezar de Oliveira.-- 2019.
1 arquivo texto (121 f.) : PDF ; 4,17 MB.

Modo de acesso: World Wide Web
Título extraído da tela de título (visualizado em 26 set. 2019)
Texto em português com resumo em inglês
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada, Curitiba, 2019
Bibliografia: f. 82-87

1. Computação - Dissertações. 2. Computadores - Programação. 3. Software - Desenvolvimento. 4. Dojo de programação. 5. Informática na educação. 6. Informática - Estudo e ensino (Superior). 7. Software livre - Estudo e ensino (Superior). I. Seca Neto, Adolfo Gustavo Serra. II. Emer, Maria Cláudia Figueiredo Pereira. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD: Ed. 22 – 621.39

Biblioteca Central da UTFPR, Câmpus Curitiba
Bibliotecário: Adriano Lopes CRB-9/1429

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação

CAMPUS CURITIBA

ATA DA DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 81

DISSERTAÇÃO PARA OBTENÇÃO DO TÍTULO DE MESTRE EM COMPUTAÇÃO APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM: COMPUTAÇÃO APLICADA
ÁREA DE CONCENTRAÇÃO: ENGENHARIA DE SISTEMAS COMPUTACIONAIS
LINHA DE PESQUISA: ENGENHARIA DE SOFTWARE

No dia 28 de junho de 2019 às 9h30min reuniu-se na Sala B-204 da Sede Centro a banca examinadora composta pelos pesquisadores indicados a seguir, para examinar a dissertação de mestrado do candidato Paulo Cezar de Oliveira, intitulada: Dojo de Aprendizagem de Programação de Computadores. .

Orientador: **Adolfo Gustavo Serra Seca Neto**

Após a apresentação, o candidato foi arguido pelos examinadores que, em seguida à manifestação dos presentes, consideraram o trabalho de pesquisa: (X) Aprovado. ( ) Aprovado com restrições. Revisor indicado para verificação: _____. ( ) Reprovado.

Observações:
ACatar as recomendações de banca.

Nada mais havendo a tratar, a sessão foi encerrada às __h__, dela sendo lavrado a presente ata, que segue assinada pela Banca Examinadora e pelo Candidato.

O candidato está ciente que a concessão do referido título está condicionada à: (a) satisfação dos requisitos solicitados pela Banca Examinadora; (b) entrega da dissertação em conformidade com as normas exigidas pela UTFPR; (c) atendimento ao requisito de publicação estabelecido nas normas do Programa; e (d) entrega da documentação necessária para elaboração do Diploma. A Banca Examinadora determina um **prazo máximo de** 60 **dias**, considerando os prazos máximos definidos no Regulamento Geral do Programa, para o cumprimento dos requisitos (desconsiderar caso reprovado), sob pena de, não o fazendo, ser desvinculado do Programa sem o Título de Mestre.

Profa. Dra. Maria Claudia F. Pereira Emer – Presidente – UTFPR

Prof. Dr. Laudelino Cordeiro Bastos – UTFPR

Prof. Dr. Robinson Vida Noronha – UTFPR

Prof. Dr. Roberto Pereira – UFPR

Assinatura do Candidato:

Reservado à Coordenação

DECLARAÇÃO PARA A OBTENÇÃO DO TÍTULO DE MESTRE

A Coordenação do Programa declara que foram cumpridos todos os requisitos exigidos pelo Programa de Pós-Graduação para a obtenção do título de Mestre.

Curitiba, 28 de junho de 2019.

Carimbo e Assinatura do(a) Coordenador(a) do Programa

Prof. Dr. Adolfo Gustavo Serra S. Neto
Coordenador(a) Adjunto(a) do PPGCA
UTFPR - Câmpus Curitiba

Dedico este trabalho a Gleice, minha esposa, amiga e companheira de lutas e conquistas. Ao Mateus e ao Thomas, filhos e amigos que torceram por este projeto. Aos demais familiares, aqui representados por Dona Lourdes, mãe e conselheira.

AGRADECIMENTOS

Aos Professores Adolfo e Maria Cláudia, por suas orientações e zelo pelo bom andamento deste trabalho. Aos amigos pelo incentivo e ajuda no aprimoramento dos questionários. À UTFPR, por propiciar as condições adequadas de estudo e pesquisa.

Bendito seja Deus, que não rejeitou a minha oração, nem desviou de mim sua misericórdia. Salmo 66:20.

RESUMO

OLIVEIRA, Paulo Cezar de. DAPROG - DOJO DE APRENDIZAGEM DE PROGRAMAÇÃO DE COMPUTADORES. 121 f. Dissertação – Programa de Pós-graduação em Computação Aplicada, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

Desenvolvedores de software têm promovido encontros presenciais para discutir, programar e compartilhar conhecimento. Em muitos desses encontros, os propósitos variam entre diversão, cumprir desafios e o desenvolvimento profissional de software. Dentre estes encontros, *Coding Dojos* se destacam pelo interesse de seus participantes em aprender novas práticas de desenvolvimento de software além de aprender novas linguagens de programação. Porém, *Coding Dojos*, em qualquer de seus estilos, quando empregados para o ensino em sala de aula pode não ser tão motivador e efetivo para o aprendizado dos alunos. Neste trabalho apresentamos o DAPROG (Dojo de Aprendizagem de Programação), uma proposta que pretende adaptar Coding Dojos para o ambiente acadêmico, e avaliamos se a participação dos alunos no DAPROG apresenta resultados efetivos e motivadores para o aprendizado de novas linguagens de programação. Este trabalho foi realizado em duas etapas. Na primeira, participamos de encontros de *Coding Dojos* realizados em instituição de ensino, evento de software livre e em uma empresa de tecnologia com o intuito de nos envolvermos nas atividades. Em cada encontro aplicamos questionários para saber se o *Coding Dojo* possibilita alguma aprendizagem. Constatamos nesta pesquisa que, para os participantes daqueles *Coding Dojos*, 78,5% aprenderam algo e 81% dos mesmos puderam compreender melhor a linguagem de programação usada naquele *Coding Dojo*. Em outro momento, apresentamos a 53 alunos de cursos de graduação nas áreas de Sistemas de Informação e Engenharia da Computação o Dojo de Aprendizagem de Programação. E os resultados foram: 72,2% afirmaram que gostaram da atividade proposta e 74,1% dos mesmos gostaram daquela metodologia de ensino. E o mesmo percentual de 74,1% dos respondentes afirmaram que seu nível de conhecimento em determinada linguagem teve boa melhoria. Os resultados apresentados neste trabalho indicam que houve uma boa aceitação da atividade proposta para os alunos daqueles estudos de caso. Assim podemos observar que, ao envolver os discentes tanto no processo de aprendizado, quanto no processo de compartilhar seu conhecimento, as aulas podem ficar mais atrativas e alcançar um maior número de alunos na sala de aula.

Palavras-chave: Aprendizado, Dojo de programação, Metodologia de ensino, Programação pareada, Retrospectiva, Programação, Aprendizado colaborativo

ABSTRACT

OLIVEIRA, Paulo Cezar de. COMPUTER PROGRAMMING LEARNING DOJO. 121 f. Dissertação – Programa de Pós-graduação em Computação Aplicada, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

Software developers have organized face-to-face meetings to discuss, program and share knowledge. In many of these meetings, the purposes range from fun, facing challenges, and professional software development. Among these meetings, Coding Dojos are distinguished by the interest of its participants in learning new software development practices and learning new programming languages. However, Coding Dojos, in any of its styles, when employed for teaching in the classroom may not be as motivating and effective for student learning. In this work, we present the DAPROG (Computer Programming Learning Dojo), a proposal that intends to adapt Coding Dojos to the academic environment and evaluate if the participation of the students in DAPROG presents effective and motivating results for learning of new programming languages. This work was carried out in two stages. In the first one, we participated in Coding Dojo meetings held in a teaching institution, a free software event and in a technology company with the intention of getting involved in the activities. At each meeting, we applied questionnaires to find out if Coding Dojo allows any learning. We found that 78.5% participants learned something and 81% of them could better understand the programming language used in that Coding Dojo. In another moment, we presented to the 53 undergraduate students in the areas of Information Systems and Computer Engineering the Computer Programming Learning Dojo. And the results were: 72.2% stated that they liked the proposed activity and 74.1% of them liked its that teaching methodology. And the same percentage of 74.1% of the respondents stated that their level of knowledge in a given language presented good improvement. The results presented in this study show that there was a good acceptance of the activity proposed to the students of those case studies. Therefore, we can observe that by involving students in both the learning process and the process of imparting their knowledge, the classes can become increasingly attractive and reach a more considerable number of students in the classroom.

Keywords: Learning, Coding Dojo, Learning metodology, Pair programming, Retrospective, Programming, Collaborative learning

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

DAPROG	Dojo de Aprendizagem de Programação
XP	*eXtreme Programming*
PP	Programação Pareada
TDD	*Test Driven Development*
NSR	Não Se Repita (Princípio definido por Hunt e Thomas)
GDCR	*Global Day of Coderetreat*
GT	Grupo de Testes
UTFPR	Universidade Tecnológica Federal do Paraná
FTSL	Fórum de Tecnologia em Software Livre
TI	Tecnologia da Informação
BSI	Bacharelado em Sistemas de Informação

SUMÁRIO

1 INTRODUÇÃO

Há alguns anos o desenvolvimento de software deixou de ser uma atividade individual para se tornar compartilhada e geograficamente distribuída (MCDOWELL et al., 2002). Desenvolvedores cooperam uns com os outros, muitas vezes, pelo interesse em transmitir seu conhecimento. Isto pode levar ao aprendizado e formar uma cultura colaborativa entre os participantes. Por conta disso, ao longo do tempo, diversos canais de comunicação social surgiram para dar suporte a essa comunidade que se forma (STOREY et al., 2017).

Duas tendências que esta cooperação tem despertado em grupos de desenvolvimento de software são a colaboração mútua no aprendizado e as trocas de experiências em encontros de programadores. Esses encontros têm as mais variadas configurações. Porém, seu propósito principal é permitir que programadores se reúnam para programar, superar desafios e compartilhar conhecimento. Em alguns desses encontros, os desenvolvedores procuram empregar práticas advindas da comunidade ágil de desenvolvimento de software permitindo uma maior interação entre seus praticantes (SATO et al., 2008), (LUZ; NETO, 2012), (BACHE, 2013).

Na colaboração mútua podemos destacar o interesse pelo aprendizado de novos métodos de desenvolvimento de software, bem como o interesse em aprender novas linguagens de programação (BOSSAVIT; GAILLOT, 2005). Em muitos desses encontros, os propósitos variam entre diversão (ESTÁCIO et al., 2015b), cumprir desafios (PARSONS et al., 2014) e objetivos profissionais (ZUILL, 2014).

1.1 MOTIVAÇÃO

Com a diversidade de perfis e interesses nos aprendizes de programação de computadores, seja qual for o ambiente, é importante buscar novas estratégias para um processo de ensino que atinja o maior público possível, contribuindo para o desenvolvimento de habilidades e competências estabelecidas por uma instituição de ensino aos seus alunos, por exemplo, autonomia no aprendizado, trabalho colaborativo e capacidade de resolução de

problemas (RODRIGUES et al., 2017).

Este trabalho teve como motivações em primeiro momento entender se é possível aplicar práticas colaborativas de programação em sala de aula. Uma vez entendido que os perfis dos participantes destas práticas colaborativas são diferentes do perfil de um aluno de cursos de programação, desenvolvemos uma proposta de atividade de ensino que, dadas a diversidade e os interesses dos aprendizes de programação de computadores, fosse possível atender a um maior número de pessoas.

1.2 OBJETIVOS

Ao estudarem sobre o ensino de programação de computadores, alguns pesquisadores concentram-se nas práticas colaborativas de programação como por exemplo, a programação em pares. Isto tem se mostrado eficiente conforme estudos realizados por (WILLIAMS et al., 2008), (KONGCHAROEN; HWANG, 2015) e (NAWAHDAH et al., 2015).

1.2.1 OBJETIVO GERAL

Os objetivos gerais deste trabalho são propor a atividade de ensino de programação de computadores nominada de DAPROG. E avaliar se esta atividade motiva o interesse pelo aprendizado de novas linguagens de programação, permitindo que a troca de experiências entre os alunos da graduação de cursos de computação os leve a perceber outras maneiras de alcançar um resultado.

1.2.2 OBJETIVOS ESPECÍFICOS

Em acordo com o objetivo geral foram determinados os seguintes objetivos específicos:

- Caracterizar a percepção dos participantes de encontros colaborativos de desenvolvimento de software sobre o uso da prática colaborativa no aprendizado de programação de computadores.
- Testar se a participação na atividade Dojo de Aprendizagem de Programação resulta na auto percepção de aprendizagem dos alunos de graduação que cursam programação de computadores.
- Avaliar se a participação na atividade Dojo de Aprendizagem de Programação motiva a interação dos alunos de graduação nas aulas de programação de computadores.

1.3 QUESTÕES DE PESQUISA

Para que possamos obter respostas que nos permitam entender melhor a interação entre os participantes e ter um vislumbre mais amplo do DAPROG, elaboramos as seguintes perguntas de pesquisa que nortearam nossos estudos:

RQ01: DAPROG facilita o aprendizado de novas linguagens de programação?

RQ02: DAPROG possibilita a troca de experiências entre os alunos?

RQ03: DAPROG facilita a compreensão da programação em pares?

RQ04: DAPROG é útil para o ensino de programação para uma sala de aula com um número maior do que 20 alunos?

1.4 ESTRUTURA DO TRABALHO

Este trabalho organiza-se da seguinte forma: A fundamentação teórica é apresentada no capítulo 2. O capítulo 3 apresenta o Estado da Arte, evidenciando trabalhos relacionados ao tema. No capítulo 4 está descrita a Metodologia de Pesquisa proposta detalhadamente neste trabalho. O capítulo 5 descreve a Proposta e o detalhamento do DAPROG. No capítulo 6 descrevemos os Estudos de Casos Coding Dojos e DAPROG, este último realizado para testar a proposta deste trabalho. No capítulo 7 apresentamos as lições extraídas por meio das respostas dadas nas retrospectivas do *Coding Dojo* Elixir e DAPROG. Por fim, o capítulo 8 conclui o documento apresentando as conclusões gerais deste trabalho e propõe possibilidades de diferentes abordagens para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo descreve cada um dos conceitos importantes para as práticas colaborativas de desenvolvimento de software apresentadas no capítulo 3.

2.1 PRÁTICAS ÁGEIS DE DESENVOLVIMENTO DE SOFTWARE

Para Diebold e Zehler (2016), Práticas Ágeis podem ser definidas como: tarefas, atividades, diretrizes ou algum aspecto técnico centrado em pelo menos um dos valores do manifesto ágil. Para eles o exemplo mais comum é o conjunto de 12 práticas de desenvolvimento de software do método *eXtreme Programming* - XP. Dentre outros exemplos de Práticas Ágeis, podemos citar o Design Simples, a Integração Contínua, a Programação em Pares (ou Programação Pareada), os Encontros Diários e a Refatoração (DIEBOLD; ZEHLER, 2016).

De acordo com a *Agile Alliance*[1], o termo "desenvolvimento ágil de software" abrange um conjunto de métodos e práticas baseados nos valores e princípios estabelecidos no manifesto ágil. Cohn (2006) afirma que processos ágeis de desenvolvimento de software são altamente iterativos e incrementais e fornecem software codificado e testado ao final de cada iteração. Ainda de acordo com Cohn (2006), para que uma equipe ágil funcione bem é preciso observar o seguinte:

- Trabalhar como um time.
- Trabalhar em iterações curtas.
- Entregas a cada iteração.
- Foco nas prioridades do negócio.
- Inspecionar e adaptar.

[1] https://www.agilealliance.org/agile101/

2.1.1 PROGRAMAÇÃO PAREADA

Programação Pareada (PP) é uma atividade colaborativa que consiste em dois programadores trabalhando juntos (COCKBURN; WILLIAMS, 2000), em um mesmo computador e com um único teclado e mouse trocando experiências e sugestões sobre o processo de desenvolvimento (BECK; ANDRES, 2005), (ROBERT; MICAH, 2006), (KONGCHAROEN; HWANG, 2015).

Em PP existem papéis distintos porém mutuamente colaborativos. O piloto assume o teclado e aplica as ideias ao código e o copiloto acompanha todo o processo de desenvolvimento orientando e dando sugestões sobre como melhorar o código (WILLIAMS; UPCHURCH, 2001). Essas funções não são permanentes; elas podem ser trocadas entre os envolvidos a qualquer momento de acordo com o entendimento de ambos (PLONKA et al., 2011).

Williams e Upchurch (2001) afirmam que, à primeira vista, programação pareada aparenta ter uma falsa impressão de gastos dobrados com duas pessoas trabalhando no mesmo código, já que ambos se concentram em uma mesma tarefa. Entretanto em estudos realizados, essa ideia foi refutada, pois as duplas de programação geraram uma economia de tempo significativamente maior do que programadores individuais.

Ao final do desenvolvimento em Programação Pareada, mais pessoas estão familiarizadas com o código, além de reduzir o tempo decorrido no processo de desenvolvimento. A troca de conhecimento traz maior satisfação com o trabalho, e um código desenvolvido por um indivíduo é mais propenso a erros do que um código desenvolvido por pares (FRONZA et al., 2011), (ZIERIS; PRECHELT, 2014), (NOSEK, 1998).

Em outras palavras, enquanto um programador não percebe pequenos erros, uma segunda pessoa poderá facilmente percebê-los e sugerir melhorias no código (BECK, 2002), inclusive detectando com uma maior rapidez a solução de problemas, por vezes, insolúveis (WILLIAMS; UPCHURCH, 2001).

A programação pareada é uma boa maneira de aumentar a produtividade, pois pessoas que trabalham em pares acreditam que a experiência é mais divertida do que trabalhar sozinhos (COCKBURN; WILLIAMS, 2000).

2.1.2 PASSOS DE BEBÊ

Conhecido e utilizado por praticantes de TDD (*Test Driven Development*) (Seção 2.1.5), a prática Passos de Bebê garante que desenvolvedores de software façam a codificação

dos testes em pequenos passos permitindo que todos os envolvidos na codificação entendam a solução garantindo ainda que uma solução mínima seja criada (RODRIGUES et al., 2017).

Seu objetivo é fazer mudanças sutis no desenvolvimento do código de forma que, além de obter uma melhor solução, também seja possível que outros compreendam todo o processo de desenvolvimento, sendo inclusive, possível sua replicação posterior (LUZ; NETO, 2012), (BECK, 2002).

A intenção da prática Passos de Bebê é possibilitar ao desenvolvedor que escreva um código que ofereça maior simplicidade e facilidade de compreensão. Portanto, ele deve buscar a solução mais simples, e não a modificação mais simples do código (ANICHE, 2012).

2.1.3 TESTE DE UNIDADE

De acordo com o Vocabulário de Sistemas e Engenharia de Software do ISO/IEC/IEEE 24765 segunda edição (2017), Teste de Unidade é um teste de rotinas e módulos individuais realizados por um desenvolvedor ou por um testador independente possibilitando que não hajam erros de análise ou programação (RUNESON, 2006).

Frequentemente é o primeiro nível de testes de um sistema de software, sua motivação é o fato de que os custos de encontrar e corrigir os possíveis erros na fase de testes é, por vezes, menor do que encontrar e corrigir esses erros durante os testes de integração (GANESAN et al., 2013).

Uma unidade é a menor parte testável de um aplicativo; tomando por exemplo o paradigma orientado a objetos, uma unidade pode ser uma classe (PASTERNAK et al., 2009). Portanto, é necessário testar uma classe do sistema de cada vez (RUNESON, 2006). De acordo com Pasternak e outros (2009), testar uma unidade isoladamente é um dos princípios mais importantes do Teste de Unidade.

De acordo com Ganesan e outros (2013), este tipo de Teste ocorre durante a fase de desenvolvimento sendo, por vezes, o primeiro nível de testes em software. A motivação para o Teste de Unidade é o fato de que o custo de encontrar e resolver um problema nesta fase é muito mais barato do que encontrar e corrigir problemas encontrados durante os testes de integração, testes de sistema ou quando o sistema está em produção (NEVES; VILAIN, 2014).

2.1.4 TESTES AUTOMATIZADOS

Enquanto nos Testes Manuais de Software um desenvolvedor, ou um profissional designando para tal função, tem a responsabilidade de produzir os testes necessários para aquele sistema, nos Testes Automatizados de software o desenvolvimento e a execução dos *scripts*, a verificação dos requisitos de testes e o uso de ferramentas de testes são todos automatizados (KARHU et al., 2009), (DUSTIN et al., 2008). Sua finalidade é auxiliar nos testes do produto de software (SULTANIA, 2015).

Os Testes Automatizados são um meio eficiente de minimizar os esforços de testes do sistema, pois as ferramentas de automação de teste são uma forma de alcançar a redução do esforço (NEVES; VILAIN, 2014). Com uma baixa complexidade, o teste automatizado se parece muito com um teste manual no qual o desenvolvedor pensa em um cenário, executa uma ação e por fim valida a saída (ANICHE, 2012).

Por vezes os testes podem não ser implementados computacionalmente e sim por testes de mesa ou busca por palavras chaves (HAUPTMANN et al., 2015). De acordo como a evolução do software, os testes também devem evoluir. As boas práticas de testes podem reduzir o tempo de desenvolvimento, melhorar a qualidade do produto, aumentar a satisfação do cliente e reduzir os custos de manutenção (NEVES; VILAIN, 2014).

2.1.5 DESENVOLVIMENTO ORIENTADO A TESTES

Assim como a Programação Pareada, o Desenvolvimento Orientado a Testes (*Test Driven Development - TDD*) também é uma prática ágil de desenvolvimento de software (MULLER; HOFER, 2007). Cockburn (2007) afirma que TDD teve início com o propósito de que um desenvolvedor escrevesse testes de software em todo o processo de codificação. Ao tomar este cuidado é possível perceber que o projeto se tornava mais simples, limpo e fácil de realizar alterações. Com isso TDD tornou-se uma das principais recomendações para o desenvolvimento de código (COCKBURN, 2007).

TDD é uma forma de desenvolver testes unitários automatizados em códigos de programas de maneira que uma parte do programa em desenvolvimento esteja trabalhando de acordo com o planejando (KAYONGO et al., 2016). Ele é escrito pelo programador que está desenvolvendo o código. Sua execução pode ser manual ou automatizada e usualmente ocorre após uma unidade ser codificada, podendo levar desde poucos minutos até alguns meses (JANZEN; SAIEDIAN, 2005). Com TDD o programador escreve um novo código quando um teste automatizado falhar, buscando eliminar duplicidades no código (BECK, 2002). Martin

(2011) apresenta três diretrizes para o desenvolvimento orientado a testes:

1 : Não codifique antes que tenha escrito uma unidade de teste que falhe;

2 : Não escreva mais do que uma unidade de teste;

3 : Não escreva mais código de produção do que o suficiente para passar no teste falhando no momento.

Heinonen e outros (2013) afirmam que antes de qualquer procedimento de programação, o desenvolvedor inicia uma tarefa de codificação escrevendo um teste que especifica uma pequena parte do comportamento pretendido de um software em desenvolvimento. Esta pequena parte pode ser um método ou uma função daquele software fazendo com que os testes escritos em TDD sejam testes unitários (BISSI et al., 2016)

Caso algum teste falhe, o código é corrigido antes de dar continuidade no desenvolvimento. Uma vez testado e aprovado, o código pode ser incluído na sua origem, e novos testes podem ser feitos. Com esse processo é possível identificar e corrigir rapidamente possíveis falhas antes da implementação efetiva (FRANCIS et al., 2009). Após a conclusão do teste, o programador passa a implementar o código que faz o teste passar na estrutura final de seu projeto (HEINONEN et al., 2013). Beck (2002) apresenta uma sugestão do ciclo de desenvolvimento empregado em TDD. Este ciclo deve seguir os três passos representados na Figura 1:

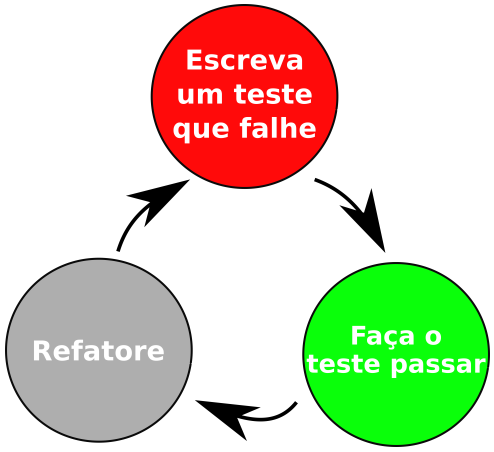

Figura 1: Ciclo de desenvolvimento com TDD. Fonte: Autoria própria.

- Vermelho - Escrever um pequeno teste que não funcione.
- Verde - Corrigir os erros apresentados no teste para que o código funcione.

- Refatorar - Eliminar as duplicidades do código.

A Figura 2 apresenta as etapas que ilustram este ciclo por meio de um diagrama de atividades. Uma vez que o desenvolvedor escreveu um teste seu próximo passo é escrever um código que passe por este teste. Depois que ele executa o teste verifica se houve erro ou não, caso haja erro significa que o código não passou no teste estando na condição vermelha.

O desenvolvedor faz as devidas correções para em seguida executar novamente o teste. Se não houver erro o código passou no teste assumindo a condição verde. Caso tenha sido concluída aquela funcionalidade, o desenvolvedor verifica a necessidade de refatorar o código. Havendo, ele faz a refatoração e executa o teste novamente até que o código assuma a condição verde. Não havendo a necessidade de refatorar, ele inicia uma nova funcionalidade para o software (BISSI et al., 2016).

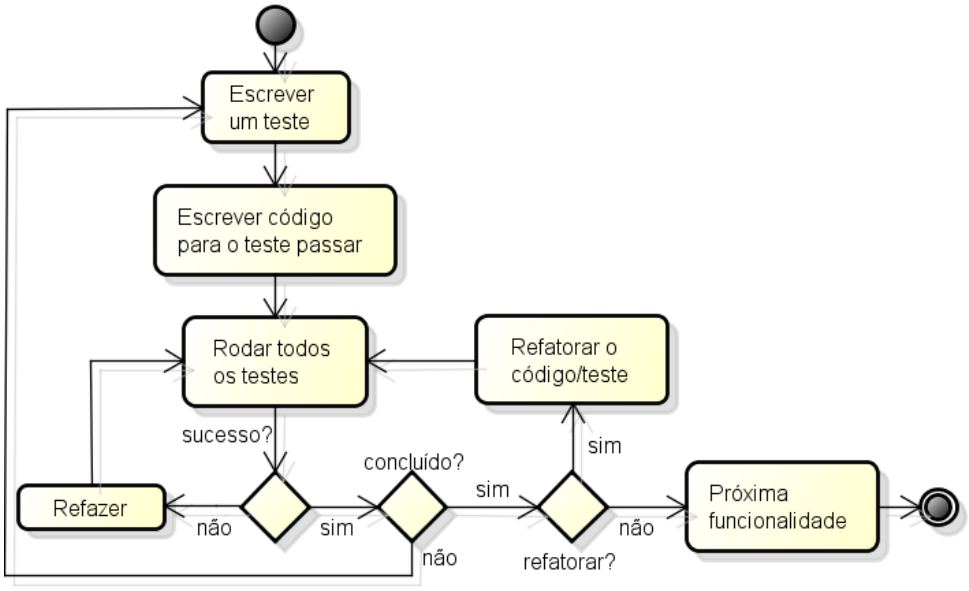

Figura 2: Etapas do ciclo de desenvolvimento com TDD. Fonte: Adaptado de (PANCUR; CIGLARIC, 2011).

2.1.6 REFATORAÇÃO

Refatoração é o processo de alterar o código de um software para melhorar sua estrutura interna sem no entanto, alterar o seu comportamento externo (FOWLER et al., 2012), (OLIVEIRA et al., 2019). Depois de concluir um ciclo de codificação de um software, a estrutura do código escrito poderá não estar otimizada, contendo trechos redundantes ou com falta de alguma abstração necessária. Este é o momento em que o desenvolvedor deve refatorar seu código para que se mantenha um bom nível de qualidade nesse código. Como a refatoração

deve ser feita sempre que necessário durante o desenvolvimento baseado em testes, o design do programa evolui à medida que o código toma forma (HEINONEN et al., 2013).

Liu e Liu (2016) afirmam que o objetivo da refatoração é melhorar a estrutura e qualidade de um código por meio de um conjunto de testes e a remoção de partes duplicadas desse código. Em (MENS; TOURWE, 2004) é apresentado um conjunto de atividades fundamentais para a refatoração:

- Identificar em que ponto o software deve ser reformulado.
- Determinar que tipo de refatoração deve ser aplicada nos pontos identificados.
- Garantir que a refatoração aplicada preserve o comportamento correto do software.
- Aplicar a refatoração.
- Avaliar o efeito da refatoração sobre as características do software.
- Manter a coerência entre o código do programa refatorado e outros artefatos de software.

Sempre que se faz a refatoração, a sugestão é que seja construído um sólido conjunto de testes. Os testes são importantes pelo fato de que mesmo seguindo um processo de refatoração estruturado, um desenvolvedor pode cometer erros imperceptíveis (FOWLER et al., 2012).

2.1.7 RETROSPECTIVA

Retrospectiva é uma reunião de programadores com uma curta duração de tempo. Nesta reunião a equipe analisa o que foi feito até o momento, procura tirar algum aprendizado com a experiência e planeja melhorias para os próximos passos do desenvolvimento do software. Esta prática ajuda aos desenvolvedores a refletirem sobre sua maneira de trabalhar e continuamente melhorar seus processos (GONÇALVES; LIDERS, 2015), (PRZYBYLEK; KOTECKA, 2017).

De acordo com Przybylek e Kotecka (2017), a Retrospectiva enfatiza um dos princípios do Manifesto Ágil que, em nossa compreensão, é o princípio que trata sobre os ”Indivíduos e interações mais que processos e ferramentas[2]”. Com esta ênfase é notório que um de seus principais objetivos é o de facilitar o desenvolvimento e o aperfeiçoamento da equipe (WEREWKA; SPEICHOWICS, 2017).

[2]https://http://agilemanifesto.org/

Dada a rápida velocidade em que o campo de desenvolvimento de software muda, é necessário revisar continuamente as práticas de trabalho. Retrospectivas asseguram que o processo do software se adapte aos avanços no campo. Uma retrospectiva pode ajudar a facilitar a melhoria dos processos e as mudanças que envolvem mais de uma equipe ajudando a construir uma cooperação mútua (KERTH, 2001).

Para que este tipo de reunião tenha sucesso, Werewka e Speichowics (2017) afirmam que é muito importante preparar, conduzir e encerrar uma retrospectiva de maneira adequada. Nestas reuniões, um time busca responder as seguintes perguntas (PRZYBYLEK; KOTECKA, 2017):

- O que funcionou bem que podemos esquecer de fazer se não discutirmos isso?
- O que não funcionou e como fazê-lo de forma diferente na próxima vez?
- O que aprendemos?

Uma retrospectiva bem organizada e gerenciada pode ajudar na compreensão das necessidades de melhoria motivando os envolvidos a mudar sua forma de trabalho. Ao identificar possíveis pontos de falha, um time saberá como se organizar e dar prioridade aos problemas a serem resolvidos fazendo com que esse time adquira experiência na resolução dos problemas identificados (KERTH, 2001).

2.2 METODOLOGIA DE ENSINO

Na metodologia de ensino tradicional o professor ainda é o centro de todo o processo. Ele expõe seu conteúdo de forma direta, e o aluno busca absorver o conhecimento por meio de anotações (SAVIANI, 1991), para em seguida de alguma forma por em prática.

Mesmo que esta maneira de ensino tenha o seu valor, em uma sala de aula existe uma diversidade de perfis distintos entre os alunos. Esta diversidade traz consigo múltiplas inteligências (GARDNER; THOMAS, 1989). Além dessas inteligências múltiplas entre os alunos, ainda existem diversos estilos de aprendizagem (ADAN-COELLO et al., 2008) no mesmo grupo. Por consequência, o aprendizado em sala de aula nem sempre é tão eficiente como deveria, se mostrando pouco eficaz na formação do estudante (ACOSTA et al., 2017).

2.2.1 TEORIAS DE APRENDIZAGEM

A diversidade de pensamentos que se propõem a tratar sobre o termo aprendizagem não permite uma definição única e precisa. Para muitos autores a aprendizagem é uma mudança de comportamento relativamente permanente devida a experiências vividas por um indivíduo. Outros a definem como sendo a simples aquisição de informação ou habilidades (MOREIRA, 1999).

Dentre as teorias de aprendizagem, duas correntes de pensamentos se destacam. As Teorias Conexionistas tratam a aprendizagem como sendo uma questão de conexões entre estímulos e respostas (também conhecidas como teorias Estímulo-Resposta ou Teorias Behavioristas). E as Teorias Cognitivas, que abordam como o indivíduo trata, processa e dá significado à informação que recebe. Nesta teoria, atitudes e crenças podem ser consideradas como variáveis intervenientes (MOREIRA, 1999).

De acordo com Moreira (1999), dois grandes pensadores das teorias cognitivas são Piaget e Vigotsky. Piaget entende que o princípio básico para explicar o desenvolvimento cognitivo no indivíduo é o equilíbrio, e seu desenvolvimento se dá por meio de estágios. Vigostky, por outro lado, afirma que este desenvolvimento não pode ser entendido sem considerar o contexto social e cultural em que o indivíduo está inserido.

Vigotsky se concentra nos mecanismos pelos quais se dá o desenvolvimento. Ele afirma que os processos mentais superiores (pensamento, linguagem e comportamento volitivo) de uma pessoa têm origem em processos sociais (MOREIRA, 1999). O indivíduo desenvolve seu aprendizado por meio da convivência social. Pelo processo de socialização, cada mudança cognitiva individual está associada a uma interação social correspondente (LOPES et al., 2006).

Uma das afirmações da Psicologia é que a interação social é o elemento que fornece a matéria prima para o desenvolvimento psicológico de uma pessoa. Nela o indivíduo adquire informações, habilidades, atitudes e valores (FONTANA, 2015). Por meio desta interação surge a colaboração. Nela o indivíduo pode se revelar mais forte e mais inteligente do que trabalhando sozinho (VIGOTSKY, 2001).

A argumentação pode ser vista como o principal mecanismo da colaboração. A argumentação em lugar do simplesmente escrever potencializa o aprendizado. Além de que a construção cooperativa é muito eficaz no desenvolvimento da aprendizagem (LOPES et al., 2006).

Um dos conceitos criados por Vigotsy (2001) com relação ao aprendizado é traduzido para o português como ”Zona de Desenvolvimento Proximal”. Ele diz respeito a um dos

estágios de aprendizado. Neste estágio o aluno pode desenvolver a capacidade tanto de fazer sozinho quanto com a colaboração de colegas mais adiantados o que faria com a ajuda de um professor.

Isso não dispensa o professor como mediador do ensino, mas destaca a capacidade de participação criativa do aluno podendo, inclusive, servir de medida para o desenvolvimento intelectual do estudante além de ser um critério de verificação do processo de ensino-aprendizagem (VIGOTSKY, 2001).

Além de incluir crianças e adultos com seus variados graus de conhecimento, a Zona de Desenvolvimento Proximal é um contexto de apoio mútuo no qual se pode agir. Neste contexto é possível envolver diversos artefatos tais como: livros, vídeos, equipamentos científicos e ambientes computacionais a fim de apoiar o processo de aprendizagem pois nesta Zona de Desenvolvimento Proximal se agrega o conceito de prontidão para aprender (SALOMON, 1996).

2.2.2 METODOLOGIA ATIVA DE APRENDIZAGEM

Diferentemente das metodologias tradicionais de ensino nas quais o professor ainda é o foco do ensino, as metodologias ativas de aprendizagem são centradas no aluno e buscam estimular os estudantes a participarem de atividades que os levem à reflexão e ao questionamento, permitindo que ele compreenda os conceitos e tenha capacidade de aplicá-los em um contexto real (ACOSTA et al., 2017).

Estas metodologias podem despertar a curiosidade dos alunos conforme buscam novos conceitos além do que foi abordado em aula. Ao se tornar protagonista de seu aprendizado, o senso de engajamento e pertencimento motiva o aluno a buscar mais do que simplesmente o que é ministrado pelo professor. Ao se tornar autônomo em suas ações acadêmicas, os alunos terão resultados positivos no que diz respeito à motivação, ao engajamento, ao desenvolvimento, à aprendizagem e outros (BERBEL, 2011).

2.2.3 MÉTODOS ÁGEIS E O APRENDIZADO EM DESENVOLVIMENTO DE SOFTWARE

O ensino de programação pareada e Desenvolvimento Orientado a Testes em um instituição de ensino superior ainda é um desafio (HEINONEN et al., 2013). Por outro lado, Boydens e outros (2015) afirmam que o emprego das práticas ágeis PP e TDD têm sido estudadas como métodos de ensino de programação de computadores, e a conclusão a que

chegaram é que ambas têm se mostrado eficientes para esse fim.

Rodrigues e outros (2017) afirmavam à época da publicação de seu trabalho que, na Universidade Federal do Pampa (UNIPAMPA), a taxa de falhas no aprendizado de algoritmos e programação em alunos de Engenharia de Software chegava a quase 25%. De acordo com os autores, existiam diversas razões que justificavam esta taxa, dentre elas a dificuldade em criar um modelo adequado para o ensino de programação.

A programação em pares incentiva o aluno a interagir com os colegas em suas aulas e laboratórios, propiciando um ambiente mais colaborativo. O envolvimento dos colegas nessa prática incentiva aos alunos a participarem ativamente na sala de aula (WILLIAMS et al., 2008). Outro fator importante é que o estudo por meio de programação pareada reduz a evasão dos cursos introdutórios de programação (ESTÁCIO et al., 2015a). Em estudo realizado por Kongcharoen e Hwang (2015) foi observado que, ao aplicar programação pareada em um grupo de estudantes, os que praticavam o desenvolvimento em pares conseguiam obter melhores resultados do que aqueles que trabalhavam sozinhos.

Em outra pesquisa realizada sobre o emprego de testes automatizados e desenvolvimento orientado a testes para alunos de ciência da computação observou-se que para alunos que já vem do ensino médio com alguma experiência de programação, TDD os motiva dando algo novo para aprender. Já para os alunos sem experiência, TDD permitiu que eles pudessem entender a lógica do código, haja vista esta ser mais simples (WELLINGTON et al., 2007). De acordo com Spacco e Pugh (2006), a construção de testes de software precisa ser parte do ensino de estudantes de graduação.

2.2.4 PRÁTICAS COLABORATIVAS NO ENSINO DE PROGRAMAÇÃO

O aprendizado colaborativo (ou aprendizado cooperativo), consiste em pelo menos duas pessoas se auxiliam na troca de informação (MCDOWELL et al., 2002). Esta técnica tem sido amplamente explorada em literaturas acadêmicas e o fato é que o desempenho acadêmico se aprimora quando os indivíduos aprendem uns com os outros produzindo melhores resultados (HORN et al., 1998).

As interações, tanto verbais quanto não verbais, permitem que os programadores troquem experiências dando novas perspectivas para a resolução de determinado problema, isso resulta em um potencial elevado para a elaboração de um plano mais assertivo de abordagem ao problema do que quando se trabalha sozinho (MCDOWELL et al., 2002), (FLOR, 1998).

Em estudos realizados tem se observado que o trabalho colaborativo por meio de

mecanismos adequados pode contribuir no aprendizado de programação de computadores. Trabalhando em grupos e com objetivos em comum, os envolvidos podem compartilhar suas ideias possibilitando novas ideias e novos conhecimentos (ADAN-COELLO et al., 2008).

No início dos anos 2000, Williams e Kessler (2001) realizaram um teste de Programação Pareada com um grupo de alunos. Aqueles que fizeram o teste em dupla apresentavam um rendimento no aprendizado de 15% maior do que aqueles alunos que que realizaram mesmo teste sozinhos.

Já no ano de 2017 Rodriguez e outros (2017) afirmavam que a programação em pares foi implementada com sucesso em muitos cursos de ciência da computação trazendo benefícios para os alunos que participavam desta atividade.

Coding Dojo, mesmo que com poucos estudos (ESTÁCIO et al., 2016), tem sido objeto de atenção de pesquisadores para compreender sua eficiência no ensino de práticas ágeis de programação como por exemplo, desenvolvimento orientado a testes (*Test-Driven Development* - TDD).

O principal objetivo de um *Coding Dojo* é o aprendizado pela prática. Com isso os participantes podem desenvolver novas habilidades de programação por meio do compartilhamento de experiências entre os diferentes níveis de programadores que se envolvem no *Coding Dojo* (SATO et al., 2008).

Um calouro pode se tornar um desenvolvedor competente. Ao promover o *Coding Dojo* em uma universidade, Sato e outros (2008) perceberam que um dos alunos, depois de ter participado de um *Coding Dojo*, passou a utilizar TDD na maioria de seus trabalhos. Além de ser capaz de escrever um código com clareza, seu código também possuía uma boa cobertura de testes.

Em pesquisa realizada por Luz e outros (2013) foi utilizada a modalidade *Randori* do *Coding Dojo* com o objetivo de ensinar práticas ágeis de programação. Os resultados desse trabalho sugerem que o *Coding Dojo* ajudou o grupo a compreender TDD, passos de bebê e programação pareada. A conclusão desse trabalho é que, como em qualquer atividade de aprendizado, é necessário que exista a prática, e *Coding Dojo* é uma atividade com foco nessa prática (LUZ et al., 2013).

Após dois anos de pesquisa com alunos voluntários da UNIPAMPA praticando *Coding Dojo*, Rodrigues e outros (2017) afirmam acreditar que tal prática colaborativa deve ser institucionalizada para que seus alunos possam usufruir de seus benefícios.

Durante o encontro de um *Coding Dojo*, os participantes têm a possibilidade de

aprendizado de várias formas: eles podem observar e analisar o desenvolvimento e solução de outros, fazer perguntas ou dar sugestões. E como piloto, pode ter *feedbacks* importantes de seu copiloto (HEINONEN et al., 2013).

Existem vantagens e desvantagens em realizar um *Coding Dojo* em um ambiente acadêmico. Dentre as vantagens pode-se observar que não há limitação no aprendizado, além de adquirir a habilidade no desenvolvimento de software, um aluno também pode desenvolver suas habilidades sociais por meio da troca de experiência e até mesmo por meio de sugestões na melhoria do código em desenvolvimento (HEINONEN et al., 2013).

No que diz respeito às desvantagens, a insegurança de um aluno sobre seu próprio desempenho pode provocar uma situação desconfortável. Outro problema apresentado é que os alunos acabam valorizando mais a velocidade na solução de um problema do que a qualidade da solução. Por fim, quanto mais participantes houverem, mais opiniões existem, o que torna ainda mais difícil formar um consenso sobre os passos seguintes (HEINONEN et al., 2013).

2.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Os conceitos aqui abordados tem por objetivo atender aos métodos e práticas estabelecidos no Manifesto Ágil possibilitando que um time de desenvolvedores interaja entre si e com seus clientes. Eles são utilizados, em maior ou menor grau, em práticas colaborativas de desenvolvimento de software tais como *Coding Dojo*, *Coderetreat* e *Mob Programming*. E cada uma dessas práticas tem propósitos que variam entre aprendizado pessoal, cumprir desafios ou desenvolvimento profissional de software.

3 TRABALHOS RELACIONADOS

Coding Dojo, *Coderetreat* e *Mob Programming* são práticas colaborativas de desenvolvimento de software que envolvem um grupo de pessoas que têm o mesmo propósito: desenvolver software com qualidade utilizando práticas ágeis de desenvolvimento.

Coding Dojo e *Mob Programming* fazem uso de programação pareada, passos de bebê, desenvolvimento orientado a testes, refatoração e retrospectiva. Entretanto, enquanto *Coding Dojo* tem o propósito de diversão e aprendizado, *Mob Programming* tem a finalidade de produção profissional de software. *Coderetreat* também faz uso de algumas práticas ágeis de desenvolvimento como programação pareada e desenvolvimento orientado a testes. Contudo, seu propósito é o aprendizado cumprindo desafios durante um período de tempo maior do que o *Coding Dojo*.

Cada uma destas práticas colaborativas têm o mesmo objetivo ao longo do processo de desenvolvimento de software: buscar a melhor solução para o código de forma que se possa alcançar os mesmos resultados com o melhor código possível por meio da troca de experiência, testes e refatoração daquele código.

3.1 PRÁTICAS COLABORATIVAS DE PROGRAMAÇÃO

Práticas colaborativas de programação buscam a ajuda mútua entre desenvolvedores de software. Cada um com sua experiência procura melhorar um código de maneira que possa chegar próximo ao ideal. Essa é uma tendência que surgiu com a Internet, onde um desenvolvedor disponibiliza seu código em um fórum ou rede social e outros programadores sugerem melhorias nesse código (STOREY et al., 2017).

3.1.1 *KATA* DE CÓDIGO

Um conceito importante e relacionado à prática colaborativa de desenvolvimento de software é o chamado *Kata* de Código (*Code Kata*). Ele tem papel fundamental, pelo menos, em

duas práticas, *Coding Dojo* e *Coderetreat*. Segundo Dave Thomas[1], o *Code Kata* é um exercício de programação que, ao ser resolvido repetidas vezes, fazendo pequenas melhorias no código, ajuda um desenvolvedor de software a aprimorar suas habilidades de programação. O Kata de Código é inspirado no caratê, arte marcial japonesa. A prática e a repetição de determinada sequência de movimentos (*Kata*) permite o melhor aprendizado de um golpe permitindo que o praticante possa aperfeiçoar aquele movimento (MARTIN, 2011).

De acordo com Thomas[1], os exercícios do *Kata* de Código podem durar entre trinta minutos a uma hora. Seu propósito é envolver o participante na programação propriamente dita, ou apenas levá-lo a uma reflexão sobre alguma questão relacionada a programação. Seu objetivo entretanto não é, necessariamente, chegar a uma resposta e sim o aprendizado ao longo do caminho. O propósito é a prática e não a solução (SATO et al., 2008).

Um exemplo de um *kata* de código é o *FizzBuzz*[2]. O problema consiste em escrever um programa que imprima os números de 1 a 100. Para os números múltiplos de três, deve ser impressa a palavra *Fizz* no lugar do número e para os múltiplos de cinco deve ser impressa a palavra *Buzz*. Por fim, para números múltiplos de três e cinco imprimir *FizzBuzz*. Um exemplo de código em *Python* é apresentado na Figura 3 e a saída para este exercício deve ser semelhante ao seguinte:

1, 2, fizz, 4, buzz, fizz, 7, 8, fizz, buzz, 11, fizz, 13, 14, fizzbuzz, ..., até 100 (buzz)

```
for i in range(1,101):
  if (i%3==0 and i%5==0):
    print ("FizzBuzz")
  elif (i%3==0):
    print ("Fizz")
  elif (i%5==0):
    print ("Buzz")
  else:
    print (i)
```

Figura 3: Código *FizzBuzz* em Phyton. Fonte: Autoria própria.

3.1.2 CODING DOJO

O Coding Dojo (*Dojo de Programação)*, inspirado no *Code Kata*, foi criado no ano de 2004 em Paris por Laurent Bossavit e Emmanuel Gaillot. A motivação para esta iniciativa foi o

[1]http://codekata.com/
[2]http://codingdojo.org/kata/FizzBuzz/

interesse em aprender uma nova linguagem de programação, além de entender melhor práticas como TDD e código simples. Os criadores reuniram-se com um grupo de pessoas e esboçaram um novo tipo de reunião para praticar programação. Esta reunião viria a ser o primeiro Dojo de Programação (ROOKSBY et al., 2014), (MARTIN, 2011), (BACHE, 2013).

Coding Dojo se trata de um encontro organizado, preferencialmente com poucos participantes. Bache (2013) relata ter obtido bons resultados com números entre 5 a 15 participantes. Neste encontro, programadores e interessados em aprender se reúnem com o objetivo de trocar experiência, se divertir e discutir temas como design, testes, refatoração, escolha do editor de código e ferramentas (BACHE, 2013), (SATO et al., 2008), (HEINONEN et al., 2013).

A Figura 4 apresenta um grupo de pessoas em um encontro de *Coding Dojo*. Nela é possível observar que o piloto e o copiloto estão no computador desenvolvendo suas ideias enquanto a tela é projetada para uma plateia que, em determinados momentos, pode interagir com o processo de codificação.

Figura 4: Seção de um *Coding Dojo*. Fonte: (LUZ et al., 2013).

Um dos propósitos de um Dojo de Programação é encontrar soluções simplificadas para problemas também simples, além de permitir que o público acompanhe a linha de raciocínio dos desenvolvedores podendo inclusive fazer sugestões quando permitido ou solicitado (SATO et al., 2008), (LUZ; NETO, 2012). A intenção é que, ao fim, se tenha um software com uma boa cobertura de testes (LUZ et al., 2013).

Outro propósito é criar um ambiente seguro que seja colaborativo, inclusivo e não competitivo onde as pessoas possam aprender continuamente (SATO et al., 2008). O

conhecimento de uma linguagem de programação pode acontecer pela observação passiva de uma pessoa ao que está sendo feito pelo piloto e copiloto. O participante pode aprender observando o que outros programadores fazem, estando em um ambiente favorável e aberto a novas ideias (ANICHE, 2012).

Não existe um consenso sobre a estrutura de um *Coding Dojo*. Porém, uma possibilidade é apresentada na Figura 5 e com algumas poucas variações esses elementos são comuns nos encontros (RODRIGUES et al., 2017).

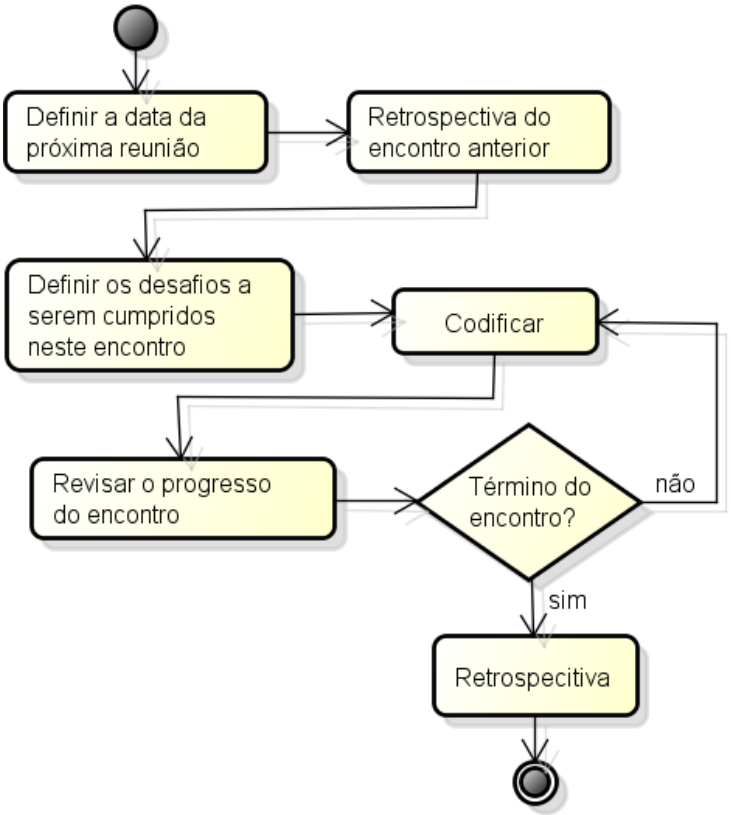

Figura 5: Estrutura de um *Coding Dojo*. Fonte: Autoria própria.

- Alguns minutos para a escolha da data do próximo encontro;
- Alguns minutos para uma retrospectiva do encontro anterior;
- Poucos minutos para decidir o desafio a ser cumprido no encontro;
- Codificação;
- Alguns minutos para revisar o progresso do encontro;
- Mais codificação;
- Alguns minutos para uma retrospectiva.

Mesmo sendo flexível e de fácil adaptação, Bache (2003) apresenta alguns elementos importantes, e são esses que realmente diferenciam um *Coding Dojo* de outras reuniões de programadores:

- Tanto a introdução, quanto a moderação devem ser feitas de forma que os participantes sintam-se à vontade para experimentar e aprender.
- A retrospectiva deve provocar a reflexão dos participantes sobre o aprendizado naquele encontro.
- Os testes são um mecanismo eficiente para demonstrar se o código realmente está de acordo com o definido e funcionando como deveria.
- Mostrando (em uma tela projetada para que todos vejam) como está sendo desenvolvido o código é possível que outros possam entender o processo, além de permitir opiniões sobre uma melhor maneira de desenvolvimento.
- O moderador (ou facilitador) é a pessoa que vai orientar e dirigir o encontro, além de conduzir todo o processo no *Coding Dojo*.

Ao término do encontro é importante saber dos participantes suas impressões e opiniões, coletadas por meio de algumas perguntas e as respostas devem ser apresentadas no mesmo instante, no que é chamado de Retrospectiva.

Retrospectiva. A retrospectiva, em um *Coding Dojo*, é um momento no encontro em que os participantes fazem uma reflexão sobre o aprendizado (BACHE, 2013). Isso pode acontecer ao final de uma reunião. Nela os desenvolvedores param o processo de codificação (mesmo que não tenham terminado) e passam a expor suas impressões sobre as experiências e o que foi aprendido no encontro (SATO et al., 2008). Rooksby e outros (2014) afirmam que, ao final de cada sessão do *Coding Dojo*, é realizado um momento de discussão reflexiva.

O tempo estipulado para a retrospectiva pode variar. Bache (2013), por exemplo, sugere um tempo entre 5 a 15 minutos e acrescenta que pode ser usada qualquer forma de retrospectiva ou outra atividade que julgue ser útil. Sato e outros (2008) propõem um período entre 5 a 20 minutos. Uma maneira de realizar a retrospectiva é permitir que os participantes exponham pontos positivos e negativos do encontro e que discutam sua experiência geral, a configuração do evento, e considerem possíveis melhorias futuras (ESTÁCIO et al., 2016), (HEINONEN et al., 2013).

Duas questões podem levar aos participantes a pensarem sobre o *Coding Dojo* (SATO et al., 2008):

- O que aprendemos?
- O que poderia melhorar?

O que é preciso para o encontro. Com poucas variações, estas são algumas condições para um *Coding Dojo* (LUZ et al., 2013), (SATO et al., 2008):

- Piloto, copiloto e plateia.
- Ambiente espaçoso.
- Projetor multimídia, tela e computador.
- Quadro para definir o problema, fazer anotações, desenhar diagramas e etc.

Pontos importantes. Sato (2008) e Heinonen e outros (2013) destacam alguns pontos importantes de um *Coding Dojo*:

- Não é uma competição.
- Os problemas não devem ser complexos.
- Não é necessário achar uma solução para o problema proposto.

3.1.3 ESTILOS DE *CODING DOJO*

Usualmente o *Coding Dojo* acontece em três estilos: *Randori*, *Prepared Kata* e *Kake* (RODRIGUES et al., 2017).

***Randori*.** Não exige muito preparo, pois não é necessário que o participante já tenha feito o *Kata*. Seu propósito é tomar decisões por meio da discussão, esclarecendo cada passo realizado durante o desenvolvimento (BACHE, 2013). A solução de um problema é realizada em tempo real, em um computador conectado a um projetor multimídia e dois programadores, o piloto e o copiloto que interagem entre si. Nesse estilo geralmente são empregados TDD e passos de bebê.

Enquanto o piloto assume o teclado digitando suas ideias, o copiloto por sua vez, sugere outras maneiras que possam ser aplicadas ao código. O tempo de duração de cada dupla

pode variar entre 5 a 10 minutos (em alguns casos o tempo pode durar 3 minutos para a troca de piloto). Terminado esse tempo o piloto deixa o teclado, o copiloto assume a posição de programador e um integrante da plateia assume a posição de copiloto (SATO et al., 2008).

Existe uma pequena variação do *Randori*. Com a tela projetada na parede, uma pessoa escreve o teste e em seguida se senta, para que outro participante possa vir ao computador e escrever outro teste. Isso pode ser feito em sequência ao redor de uma mesa (ROOKSBY et al., 2014).

***Prepared Kata*.** Neste estilo, diferente do estilo *Randori*, a solução de um problema já foi resolvida anteriormente pelo apresentador (ou por um grupo de pessoas). Os resultados da solução são apresentados aos participantes que são desafiados a começar do zero e alcançar o mesmo resultado sem conhecer o código original.

No *Prepared Kata*, o apresentador tem maior experiência em relação aos demais participantes (BACHE, 2013), ele tem maior vivência e mostra ao grupo a sua melhor solução. Nesse estilo a plateia pode interagir com os programadores fazendo sugestões. O objetivo é que todos sejam capazes de reproduzir os passos e resolver o problema (SATO et al., 2008).

***Kake*.** Várias duplas trabalham simultaneamente, cada uma em um computador, e todos devem ter conhecimento avançado de programação. As duplas podem tanto solucionar o mesmo problema com linguagens diferentes quanto resolver outros problemas. A cada período de sete minutos as duplas são trocadas, permitindo que todos os participantes se envolvam entre si.

Uma diferença em relação aos outros métodos de *Coding Dojo* é que a plateia não pode ver o que está sendo feito pelos programadores. Cada participante vai programar sem ter ideia do que aconteceu anteriormente. Segundo Cukier[3], é uma simulação da vida real, na qual um profissional vai trabalhar com código legado.

Os requisitos para a realização de um *Coding Dojo* estilo *Kake* são:

- Dois ou mais computadores;
- Turnos de 7 minutos;
- TDD, passos de bebê e refatorações;
- Divertir-se;
- Comida durante toda a sessão;

[3]http://www.agileandart.com/2010/08/16/dojo-kake/

- Ao final é feita a retrospectiva.

3.1.4 *CODERETREAT*

Coderetreat, junção das palavras inglesas *Code* (código) e *Retreat* (retiro) que em uma tradução livre significa algo como Retiro de Código, também tem como base o conceito de *Code Kata* e sua proposta é associar os métodos do *Code Kata* com práticas de desenvolvimento ágil de software, a programação pareada e desenvolvimento orientado a testes.

A ideia original surgiu de conversas entre Gary Bernhardt, Nayan Hajratwala e Patrick Welsh na conferência *Codemash*[4] (evento para desenvolvedores sobre práticas, metodologias e tendências tecnológicas atuais em uma variedade de plataformas) do ano de 2009 (HAINES, 2014).

O *Coderetreat* segue um processo de aprendizado, cujo objetivo não é se concentrar no problema em si, mas focar na atividade de desenvolvimento de maneira prazerosa escrevendo o melhor código possível refletindo e analisando os fundamentos de uma codificação simples e modular (PARSONS et al., 2014). Nele, um grupo de desenvolvedores se reúne para escrever código, sem a pressão de ter que concluir o que começou seguindo um conjunto de regras e desafios.

O principal objetivo das reuniões de *Coderetreat* é encorajar os quatro elementos que são empregados para um design simples (HAINES, 2014):

- *Passar por testes* - Os códigos em desenvolvimento devem passar por todos os testes. Essa regra trata também de correção e verificação do código desenvolvido.
- *Expressar sua intenção* - Os nomes de classes, métodos e variáveis devem ser expressivos e consistentes além de ter relação com seu propósito, ou seja, devem ser significativos.
- *Sem duplicação* - **Princípio Não Se Repita - NSR** (*Don't Repeat Yourself - DRY*). Segundo Hunt e Thomas (2010), cada bloco de informação deve ter uma representação oficial, exclusiva e sem ambiguidades dentro de um sistema.
- *Refatorar* - Voltar ao código e analisar se não existem trechos desnecessários, sem uso ou duplicados.

Além das 4 regras para que o encontro aconteça de maneira eficiente e atinja seu objetivo, são necessários alguns requisitos (HAINES, 2014), (PARSONS et al., 2014):

[4] http://www.codemash.org/

- O *Code Kata* a ser abordado é *Game of life* de John Conway
- Cinco a seis sessões de codificação, cada uma com duração de 45 minutos.
- Programação pareada e desenvolvimento orientado a testes (TDD) são usados em algumas sessões.
- A cada sessão, o código deve ser excluído e os pares devem ser trocados.
- Não há nenhuma expectativa de que uma solução completa para o problema vai ser escrita em qualquer sessão.
- Diferentes restrições podem ser aplicadas a diferentes sessões (tais como, os métodos não podem retornar valores, por exemplo) de modo que o espaço do problema é reinterpretado a partir de diferentes perspectivas.

Parsons e outros (2014) afirmam que mesmo que estas 4 regras sejam a estrutura normalmente utilizada, podem surgir variações, isso de acordo com o entendimento dos organizadores.

A organização *Coderetreat* é um esforço voluntário e globalmente coordenado que promove a melhoria das habilidades do desenvolvedor de software. Esta organização incentiva, em escala mundial, um dia no ano para celebrar o Dia Mundial de Retiro de Código (*Global Day of Coderetreat - GDCR*). Muitas organizações empresariais e instituições de ensino sediam o evento em suas dependências, inclusive fazendo contatos em tempo real com participantes de outras localidades[5].

De acordo com Haines (2014), no 3º encontro anual do GDCR em 2013 houve um recorde em números de fusos horários, países, cidades e desenvolvedores. No ano de 2017 foram registrado 136 pontos de encontro divididos em 41 países[6], entretanto não há uma estimativa de números de participantes total no evento.

3.1.5 *MOB PROGRAMMING*

Em tradução livre, Programação em Grupo (*Mob Programming*) abrange importantes conceitos de práticas ágeis de programação, como programação pareada e *Scrum*. Inspirando-se e estendendo o conceito de programação em pares, o *Mob Programming* propõe que todos os envolvidos possam atuar, tanto no desenvolvimento quanto nas sugestões a serem aplicadas ao

[5] http://coderetreat.org/
[6] http://coderetreat.org/hosts/

software (LILIENTHAL, 2017), com momentos de espaço aberto para conversas, retrospectivas e até um cafezinho (ZUILL, 2014).

Enquanto o navegador atua no teclado, o copiloto comenta a ideia que está sendo codificada e ambos discutem em voz alta para que todos da equipe possam ouvir e se envolver na codificação, tanto verbalmente quanto por meio de um quadro no qual estão informações sobre o desenvolvimento. Com isso, de acordo com Zuill (2014), cria-se uma espécie de ”inteligência coletiva”. Toda a equipe pode dar sugestões ao piloto. Desta forma, o trabalho em equipe permite uma melhor visualização de possíveis padrões, possibilitando uma ação mais eficiente. Além de atuar como piloto e copiloto, a equipe discute sobre a definição da história, design, testes e implantação.

Todos os envolvidos na criação do software são considerados membros da equipe, inclusive o *product owner*. Em outras palavras, todos trabalham na mesma coisa, ao mesmo tempo, no mesmo espaço e no mesmo computador (ZUILL, 2014). A Figura 6, adaptada do primeiro rascunho idealizado por Zuill, apresenta como deve ser o ambiente para um *Mob Programming* sendo este um de seus pilares (KATTAN et al., 2017). É possível perceber na Figura que o ambiente deve ser fisicamente confortável permitindo que os integrantes da equipe trabalhem próximos uns dos outros (KATTAN et al., 2017).

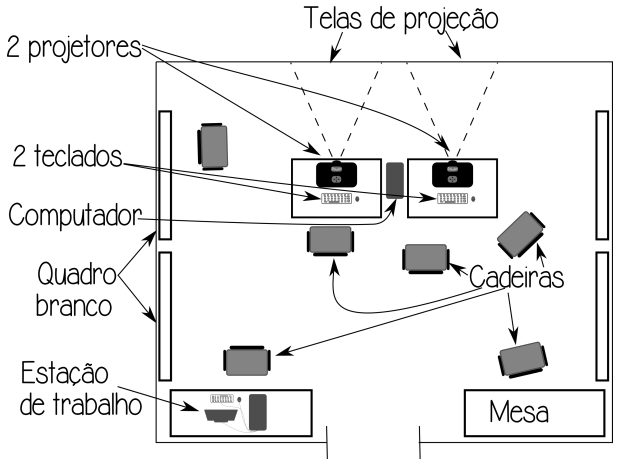

Figura 6: Esboço da área de trabalho do *Mob Programming*. Fonte: Adaptado de (ZUILL, 2014)

Neste local são utilizados dois projetores multimídia, dois teclados, um computador e uma estação de trabalho além de uma terceira mesa de trabalho. O time permanece junto neste ambiente e se comunica constantemente sobre suas tarefas (ZUILL, 2014). *Mob Programming* estende a ideia de programação pareada a toda a equipe (LILIENTHAL, 2017), pois todos os seus integrantes devem passar pela função de piloto por um período de 10 a 15 minutos (ZUILL, 2014). Um computador é usado por todos os integrantes da equipe para a codificação e apenas nele é realizado todo o processo de escrever o código. Contudo existem outros computadores

que são utilizados para pesquisas e outras tarefas paralelas (ZUILL, 2014).

O objetivo de uma equipe que trabalha utilizando *Mob Programming* é agregar o conhecimento e a experiência coletiva além de disseminar, entre seus integrantes soluções e decisões com relação ao trabalho em execução (LILIENTHAL, 2017). *Mob Programming* é, de acordo com seu idealizador, um evento onde todos na equipe trabalham simultaneamente no mesmo espaço, no mesmo projeto e no mesmo computador[7].

3.2 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Ao longo deste capítulo, é possível observar que as práticas colaborativas apresentadas têm as práticas ágeis de desenvolvimento como ponto em comum. A Figura 7 mostra que *Coding Dojo*, *Coderetreat*, *Mob Programming* e *Pair Programming* são práticas de desenvolvimento colaborativo. Nele é possível observar que tanto *Coding Dojo* quanto *Coderetreat* tem uma mesma raiz no *Code Kata*, contudo *Coding Dojo* busca se concentrar em TDD enquanto *Coderetreat* procura atingir o design simples no processo de desenvolvimento *Mob Programming*, por sua vez, mesmo sendo uma prática colaborativa e tendo como sua base programação pareada não utiliza o conceito de *Code Kata*.

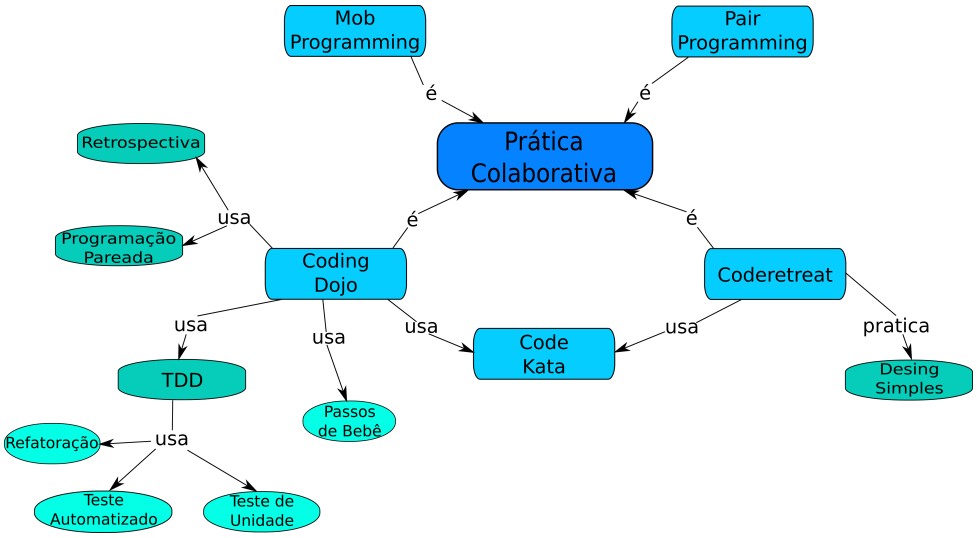

Figura 7: Relação entre as práticas colaborativas. Fonte: Autoria própria.

Com base no desenvolvimento deste capítulo, a Tabela 1 apresenta um comparativo entre os métodos de desenvolvimentos colaborativos *Coding Dojo*, *Coderetreat* e *Mob Programming*. No comparativo são apresentados a prática, o ambiente, o número de participantes e o propósito de cada atividade.

[7] http://mobprogramming.org/

Tabela 1: Comparativo entre os desenvolvimentos colaborativos. Fonte: Autoria própria.

Prática	Tipo de ambiente	Número de Participantes	Propósito
Coding Dojo	Descontraído	5 a 15 participantes	- Trocar experiência. - Praticar programação. - Diversão e técnica.
Coderetreat	Descontraído	Não definido	- Desafiador. - Praticar programação. - Diversão e desafio
Mob Programming	Profissional	3 a 12 desenvolvedores	- Produção de software. - Desenvolvimento profissional.

Enquanto em *Coding Dojo* e *Coderetreat* o ambiente é descontraído com o propósito de praticar programação, em *Mob Programming* seu ambiente é profissional e sua finalidade é a produção comercial de software. Mesmo existindo algumas semelhanças entre *Coding Dojo* e *Coderetreat*, sua principal diferença está em um dos propósitos. No primeiro a troca de experiência é evidente, enquanto no *Coderetreat* os desafios ao longo do encontro são o ponto forte. Outro destaque para *Coderetreat* é a definição do número de participantes. Diferente das outras práticas colaborativas comparadas aqui, *Coderetreat* é flexível com relação ao número de participantes.

Coderetreat, também tem uma certa flexibilidade quando se trata do desenvolvimento das atividades, entretanto, Para utilizar *Coderetreat* em sala de aula existem algumas limitações. Além de acontecer em um período que pode chegar a 8 horas de duração em um dia no ano, todas as atividades realizadas no encontro estão focadas no *Game of Life* de John Conway, Sendo importante a divisão do tempo em 5 ou 6 seções de 45 minutos cada uma. A cada seção o código deve ser excluído e começado novamente com novo desafio.

Mob Programming também apresenta algumas restrições para seu uso em sala de aula, seu ambiente tem característica formais, mesmo o ambiente sendo confortável, ele é um ambiente de produção, seu propósito é desenvolver software profissional. A programação é desenvolvida por toda a equipe e neste momento todos podem opinar sobre o desenvolvimento do código. Mesmo com a intenção de agregar conhecimento e experiência coletiva o perfil de quem se envolve com *Mob Programming* é de um desenvolvedor de software profissional.

4 METODOLOGIA DE PESQUISA

O processo de desenvolvimento desta pesquisa é representado na Figura 8. Este trabalho foi dividido em duas fases. Na primeira fase iniciamos com o referencial teórico, em seguida desenvolvemos a metodologia de nossa pesquisa, com a metodologia concluída e definidos os instrumentos de pesquisa trabalhamos com o levantamento de dados.

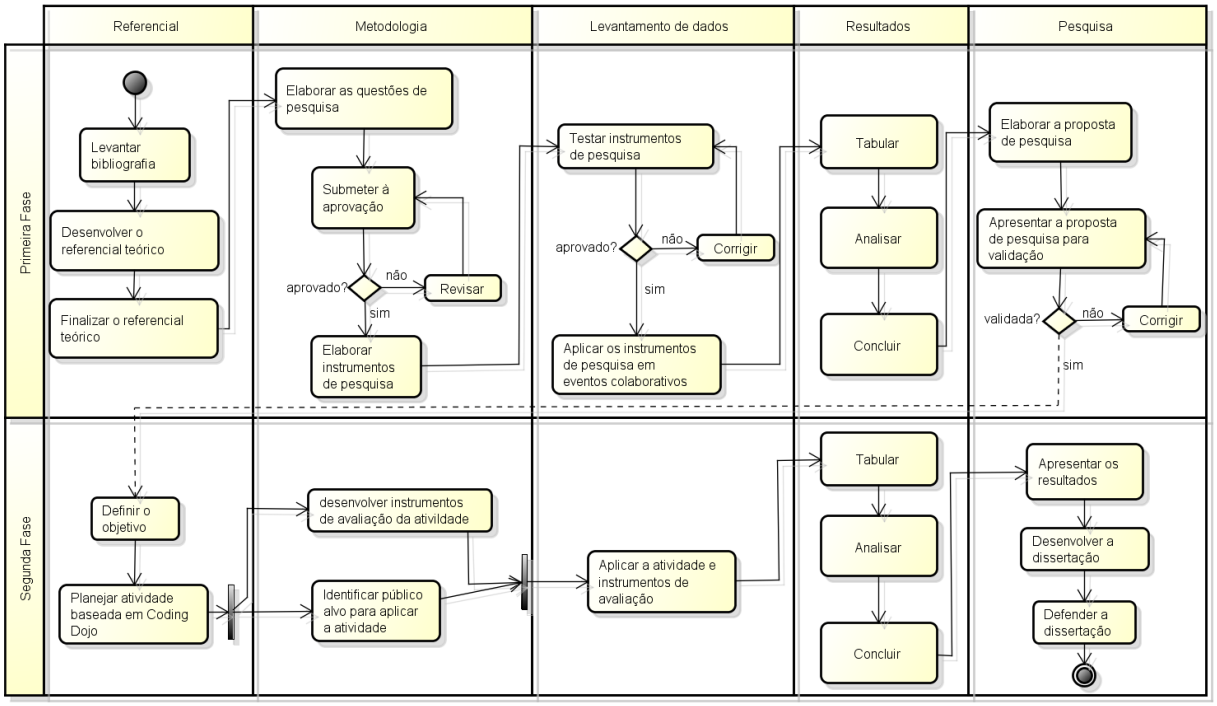

Figura 8: Desenvolvimento do trabalho de pesquisa. Fonte: Autoria própria.

Neste levantamento participamos de encontros de *Coding Dojo* e *Coderetreat* nos quais aplicamos os questionários aos participantes. Com os resultados em mãos fizemos uma análise cuidadosa dos dados para em seguida, elaborarmos nossa proposta de pesquisa.

Na segunda fase, com a proposta de pesquisa validada e o objetivo definido planejamos uma atividade baseada em *Coding Dojo* que pretendeu atender a alunos de cursos regulares de programação. Após identificar o público alvo e com os instrumentos de pesquisa em mãos aplicamos a atividade juntamento com o questionário aos alunos.

Como na primeira fase, com os dados da pesquisa em mãos fizemos cuidadosa análise para, por fim, apresentar os resultados na defesa desta dissertação.

4.1 LEVANTAMENTO DE DADOS

O levantamento de dados, também conhecido no meio científico como *survey*, ocorre por meio da aplicação de questionário a um grupo de pessoas cujo comportamento é desejado conhecer. Após a aplicação das perguntas, por meio de uma análise quantitativa, obtém-se as conclusões correspondentes aos dados coletados por meio das respostas ao questionário (PRODANOV; FREITAS, 2013).

Um questionário é um conjunto de perguntas pré-definidas e organizadas em uma ordem anteriormente determinada, frequentemente associado a uma pesquisa. Seus dados podem ser analisados e interpretados. Uma das vantagens dessa forma de captação de dados é a de ele poder ser autoadministrado. Neste caso o pesquisador não precisa, necessariamente, estar presente no ato de reposta do questionário (OATES, 2006).

Apesar de não existir uma normatização para a elaboração de um questionário, ainda assim é possível, embasado nas experiências de outros pesquisadores, estabelecer um conjunto de diretrizes práticas para desenvolver esse instrumento de pesquisa (GIL, 2002). Gil (2002) ainda apresenta um conjunto de diretrizes para que o questionário possa representar o propósito da pesquisa. Desse conjunto destacamos alguns:

- Iniciar com perguntas mais simples e finalizar com as mais complexas;
- Perguntas fechadas, mas amplas o suficiente para abrigar as respostas possíveis;
- Número limitado de perguntas, claras e precisas, com única interpretação e referindo-se a uma única ideia de cada vez;
- As perguntas devem levar em consideração o sistema, de referência e nível de informação do entrevistado;
- Evitar perguntas íntimas;
- Refletir uma única ideia de cada vez;
- Não sugerir respostas.

Uma das razões pelas quais um questionário pode ser interessante para a pesquisa é que este instrumento permite, de maneira eficiente, coletar dados de um número significativo

de pessoas com um esforço mínimo (OATES, 2006). As perguntas podem ser elaboradas de maneira que possam gerar dois tipos de dados (OATES, 2006), (MARCONI; LAKATOS, 2003):

- **Factuais**: aqueles que contém informações mais pessoais do respondentes;
- **De opinião**: nestes o respondente emite seu ponto de vista sobre determinado assunto. Este tipo de pergunta também pode ser dividida em perguntas **abertas**, quando o entrevistado é livre para escolher suas respostas, e perguntas **fechadas**, em que o entrevistado é obrigado a escolher a resposta de um conjunto pré-definido.

Marconi e Lakatos (2003) afirmam que, uma vez que o questionário seja enviado para seu público alvo, a taxa de devolução deles respondidos chega a 25%. Existem alguns fatores que influenciam esse retorno tais como, a forma, a extensão, o tipo de carta que acompanha o questionário solicitando a colaboração, as facilidades para o preenchimento, os motivos apresentados para a resposta e a classe de pessoas que recebem as perguntas.

Antes de ser publicado definitivamente, o questionário precisa ser testado aplicando-se alguns exemplares em uma pequena população definida. A análise dos dados evidenciará possíveis falhas, inconsistência ou complexidade das questões, além de oferecer uma estimativa sobre os futuros resultados (MARCONI; LAKATOS, 2003) e (PRODANOV; FREITAS, 2013).

A Internet possibilita o envio de questionários a pessoas em todos os lugares possíveis sem a preocupação com postagem, custos ou tempo de entrega, podendo ser encaminhado tanto por e-mail quanto elaborado em um formulário *web*. É recomendado que esta forma de pesquisa não seja muito extensa, pois um questionário muito longo pode cansar o respondente antes de chegar ao final das perguntas (OATES, 2006) e (PRODANOV; FREITAS, 2013).

4.2 DESENVOLVIMENTO DOS QUESTIONÁRIOS

Preparamos uma pesquisa por meio de um questionário impresso e entregue pessoalmente a participantes de encontros colaborativos de programação. Cada etapa executada desta fase do estudo é apresentada na Figura 9.

Definir o objetivo. O objetivo deste questionário é identificar o perfil dos participantes de encontros colaborativos e suas percepções de aprendizado sobre estes encontros. Logo após esta definição, iniciamos o processo de identificação dos participantes de encontros colaborativos de programação.

Identificar os participantes. O público alvo do questionários foram pessoas com

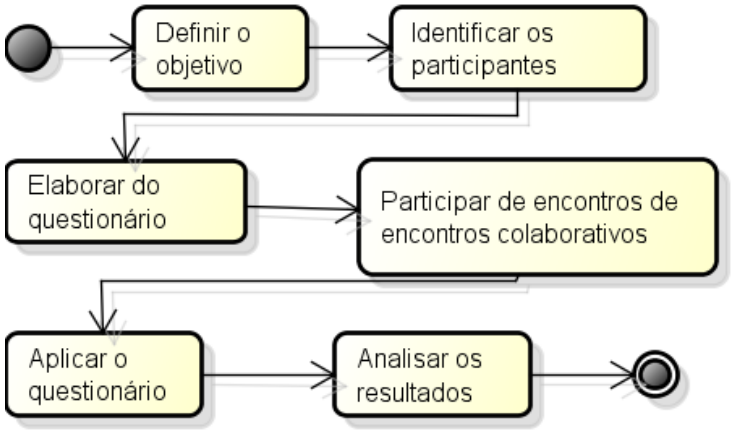

Figura 9: Etapas da metodologia aplicada aos encontros colaborativos. Fonte: Autoria própria

diversos níveis de conhecimento em programação e que participam de encontros colaborativos de programação.

Elaborar o questionário. Foram elaborados dois questionários, os respondentes foram aqueles que participam dos encontros colaborativos de desenvolvimento. O primeiro questionário era composto de perguntas factuais de múltipla escolha. Este questionário foi desenvolvido com base no trabalho realizado por (STOREY et al., 2017). Ali são apresentados diversos canais nos quais um desenvolvedor pode obter conhecimento e ajuda para seu código. O segundo questionário era composto por perguntas de opinião com o propósito de saber a opinião dos participantes sobre aquele encontro colaborativo que ele estava participando.

Participar de encontros colaborativos de programação. Cada um dos encontros aconteceu com grupos distintos de pessoas, estes grupos são descritos a seguir:

- Grupo interessado em conhecer o Motor de Jogos (*Game Engine*) *Unreal Engine*.
- Alunos de uma instituição de ensino superior interessados na linguagem de programação *Python*.
- Grupo organizado e com um perfil profissional de programação.
- Apresentar a linguagem de programação funcional Elixir e interessados em participar do *Global Day of Coderetreat*.

Aplicar o questionário. Em cada encontro colaborativo de programação que participamos entregamos os dois questionários aos respondentes. O primeiro questionário foi entregue no início do encontro, e o segundo questionário entregue ao final do mesmo encontro.

Analisar resultados. As respostas foram cuidadosamente analisadas e tabuladas a fim de obter os resultados precisos com base nas respostas informadas nos questionários. Estes resultados são apresentadas no Capítulo 6.

4.3 QUESTIONÁRIOS

O propósito destes questionários foi entender como as pessoas que participam de encontros colaborativos de programação de software, interagem com seus pares no processo de desenvolvimento e no aprendizado. Para isso este instrumento de pesquisa foi dividido em dois questionários, cada um com um grupo de perguntas:

- **Perguntas factuais**. O primeiro questionário possuía um conjunto de questões que procuravam saber a idade, sexo e ocupação do participante, além de algumas perguntas fechadas. Mesmo no conjunto de perguntas fechadas havia a possibilidade de o respondente marcar a opção *outros* e descrever brevemente a resposta caso as alternativas não se encaixassem no seu perfil.
- **Perguntas de opinião**. Neste segundo questionário, o propósito era entender a percepção dos encontros sobre o aprendizado em programação e sua compreensão relacionada às práticas ágeis utilizadas nestes encontros.

4.3.1 PRÉ-QUESTIONÁRIO

De acordo com as recomendações da Seção 4.1, foi desenvolvido um primeiro questionário (Pré-Questionário - apêndice A.1) dividido em duas partes. A primeira parte são questões factuais cuja intenção é traçar o perfil do respondente. Na segunda parte estão as questões de opinião, na qual o respondente emite sua percepção sobre o evento.

O pré-questionário foi encaminhado a um grupo denominado Grupo de Teste (GT). Após a análise do pré-questionário, consideradas todas as observações do GT e realizadas as devidas correções e ajustes necessários, chegou-se ao aprimoramento dos questionários apresentados nos apêndices A.2 e A.3, Questionário aplicado aos *Coding Dojos* e Questionário aplicado ao *Coderetreat* 2017 respectivamente.

O apêndice A.4 traz um quarto questionário, o Questionário DAPROG, entregue aos participantes de nossa proposta para uma nova maneira de usar o *Coding Dojo* em sala de aula. Esta proposta é discutida no capítulo 5 deste trabalho.

4.3.2 QUESTIONÁRIOS

Depois de validado o pré-questionário com o GT e atendidas as devidas observações do grupo, elaboramos dois outros questionários. O questionário Um era composto de perguntas factuais de múltipla escolha. Nestas questões procuramos saber a idade, sexo, ocupação, nível de conhecimento em programação, linguagens de programação conhecidas pelos participantes e se conheciam e utilizavam práticas ágeis de desenvolvimento de software.

O Questionário Dois era composto por perguntas de opinião e neste conjunto de questões a forma de obter as respostas foi utilizando um conjunto de afirmações baseado na escala de Likert conforme apresentado na Tabela 2.

Tabela 2: Valores da escala de Likert utilizados no Questionário Dois. Fonte: Autoria própria.

Escala	Definição	Descrição
-2	**Discordo totalmente**	O respondente é convicto ao não concordar com a afirmação.
-1	**Discordo**	O respondente apenas não concorda com a afirmação.
0	**Indiferente**	Para o respondente, a afirmação não causa nenhum impacto.
1	**Concordo**	O respondente apenas concorda com a afirmação.
2	**Concordo totalmente**	O respondente é convicto ao concordar com a afirmação.

Na Tabela 3 apresentamos como exemplo algumas das perguntas feitas aos participantes dos *Coding Dojos*. Neste questionário o respondente deveria escolher uma das colunas de acordo com sua opinião sobre aquela afirmação, que por sua vez, dizia respeito ao evento que ele havia acabado de participar.

Foi usado um conjunto de 17 perguntas que, além de captar a percepção dos participantes com relação ao evento que participou, também procurou saber seu possível aprendizado nas áreas de programação e práticas ágeis de desenvolvimento de software.

Tabela 3: Exemplo de perguntas do Questionário Dois. Fonte: Autoria própria.

Afirmação / Escala	-2	-1	0	1	2
Coding Dojo me ajudou a aprender mais sobre programação em geral e testes.					
Meu nível de conhecimento sobre a linguagem de programação melhorou significativamente.					
O Coding Dojo promoveu momentos de cooperação entre as pessoas que participaram.					
A experiência com Coding Dojo vai contribuir para meu desempenho em desenvolvimento de software.					
Coding Dojo está adequado à minha forma de aprender.					

4.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo foi apresentada a metodologia proposta para o desenvolvimento deste trabalho, nele apresentamos um conjunto de 4 questionários:

O primeiro, chamado de Pré-Questionário, foi aplicado a um grupo previamente selecionado. Por meio de suas observações este questionário foi aprimorado e serviu de base para o desenvolvimento dos outros 3 questionários que foram utilizados em encontros de *Coding Dojos*, no *Coderetreat* 2017 e no DAPROG. Estes 3 questionários foram aplicado a profissionais de desenvolvimento de software, interessados em aprender uma linguagem de programação e a alunos de cursos de graduação da Universidade Tecnológica Federal do Paraná, Câmpus Curitiba. Todos apresentados no capítulo 6 deste trabalho.

5 DOJO DE APRENDIZAGEM DE PROGRAMAÇÃO

Seguindo as orientações de Cohn (2006) para o bom funcionamento de uma equipe ágil, além de desenvolver sua capacidade de aprendizado em programação, o Dojo de Aprendizagem de Programação - DAPROG, motiva o desenvolvimento do trabalho em equipe, o trabalho em iterações curtas, a entrega em cada iteração, o foco nas prioridades do negócio e a auto-adaptação do aluno durante as aulas.

Esta atividade também se fundamenta nos conceitos de *Coding Dojo* de ”não competitividade”, ”problemas simples” e ”aprendizado pela prática” (SATO et al., 2008), (LUZ et al., 2013), (HEINONEN et al., 2013), associando estes conceitos às práticas ágeis de Programação Pareada, Passos de Bebê e Retrospectiva (KONGCHAROEN; HWANG, 2015), (RODRIGUES et al., 2017), (PRZYBYLEK; KOTECKA, 2017).

Outros dois conceitos também nos motivaram a pensar no DAPROG como uma atividade de ensino de programação de computadores foram a ”Interação Social”, estudada pela Psicologia e a ”Zona de Desenvolvimento Proximal” proposta por Vigostsky. Ambas afirmam que ao se relacionarem com outros colegas as capacidades de fazer sozinho, as habilidades e as atitudes se potencializam permitindo que o aluno possa ampliar seu desenvolvimento intelectual (VIGOTSKY, 2001), (FONTANA, 2015).

Diferente dos estilos de *Coding Dojo* atuais em que os participantes podem escolher que linguagem de progamação eles querem praticar momentos antes de iniciar o encontro, nesta proposta a linguagem de programação já está definida pela própria ementa da disciplina ou pelo professor responsável pela turma, não sendo necessário que os participantes definam que linguagem querem praticar. Também são dispensadas as demais discussões que são necessárias ao *Coding Dojo*.

Por se tratar de uma sala de aula, alguns elementos são importantes, por exemplo: as explicações sobre conceitos importantes, as orientações, as atividades práticas e avaliações do conhecimento sobre determinado conteúdo. Considerando estes itens esta proposta procura atender as demandas de uma sala de aula.

A escolha do *Coding Dojo* como base para o desenvolvimento de nossa proposta, está relacionada a maneira com que ele é realizado. Conforme descrito no Capítulo 3, (Trabalhos Relacionados), *Coding Dojo* é flexível em sua dinâmica permitindo que várias pessoas possam participar dentro de um ambiente seguro e colaborativo. Os exercícios realizado em um *Coding Dojo* não devem ser complexos e não é obrigatório chegar a uma solução, o aprendizado pode acontecer pela obervação do que está sendo feito e projetado em uma tela.

5.1 DESCRIÇÃO

Após as explicações sobre Programação Pareada, Passos de Bebê e Retrospectiva, inicia-se o Dojo de Aprendizagem em Programação. O DAPROG foi elaborado focando o ensino de programação em sala de aula. Nele buscou-se priorizar o aprendizado dos alunos por meio da junção e adaptação de alguns elementos existentes nos *Coding Dojos* estilos *Prepared Kata* e *Kake*.

Neste estilo a solução do problema já foi resolvida anteriormente pelo apresentador (ou por um grupo de pessoas). Os resultados da solução são apresentados aos participantes que são desafiados a começar do zero e alcançar o mesmo resultado sem conhecer o código original.

5.1.1 MINIKATAS

Os alunos sentarão em duplas, cada uma em um computador, e assim como acontece no *Coding Dojo Prepared Kata* apresentado na Seção 3.1.3, a solução do problema já foi resolvida anteriormente e será executada e apresentada pelo professor às duplas. Uma alternativa para a apresentação da solução é que o professor mostre, em um quadro, aos alunos a execução do programa por meio de um teste de mesa com entrada e saída.

Após conhecerem os resultados os alunos serão desafiados a reproduzi-los sem conhecer o código original. A qualquer momento poderão pedir auxílio para esclarecer suas dúvidas e a ajuda poderá vir do professor ou outra pessoa designada por ele para este fim. É fundamental que o exercício deve ser dividido em pequenos *katas* buscando a solução do problema em partes.

Tomando por exemplo o problema *FizzBuzz*, exercício matemático em que os múltiplos de 3 devem ser substituídos pela palavras *Fizz*, os múltiplos de 5 devem ser substituídos pela palavra *Buzz* e os múltiplos de 3 e 5 devem ser substituídos pela palavra *FizzBuzz*: No primeiro minikata os alunos devem fazer com que todos os números sejam apresentados na tela. Depois de resolvida esta parte eles devem fazer um novo *kata* para que sejam impressos na tela os

números e a palavra *Fizz* para os múltiplos de 3. Concluída esta etapa passam a desenvolver a solução *Buzz* para os múltiplos de 5. Por fim, os alunos devem escrever um programa que imprima na tela a palavra *FizzBuzz* para os múltiplos de 3 e 5.

A cada minikata finalizado poderá ocorrer o rodízio das duplas. A maneira que este rodízio deve ser feito será definido pelo professor. Porém, nossa sugestão é que neste dojo aconteçam os rodízios das duplas em intervalos de tempo de 7 a 10 minutos, tal qual acontece no *Coding Dojo* estilo *Kake*. Neste rodízio o aluno que era o piloto em uma dupla, passa a ser o copiloto em uma dupla diferente de sua anterior. A ordem dos rodízios também deverá ser definida pelo professor haja vista não existir um padrão de organização definido dos computadores em laboratórios de informática.

Em muitas circunstâncias, não é possível dividir uma turma em pares exatos. Para tanto a sugestão é que um grupo seja composto por três alunos. Neste caso o professor deve estar atento para que todos os integrantes do trio realizem todas as atividades. Outro ponto importante é que em todas as atividades realizadas no DAPROG, as duplas (ou o trio, quando for o caso) realizem o rodízio de forma que nunca uma mesma dupla passe para outra atividade. Acreditamos que com isso as trocas de experiências sejam enriquecidas.

A Figura 10 apresenta o diagrama com os processos do DAPROG e a Figura 11 apresenta como devem ser feitos os minikatas.

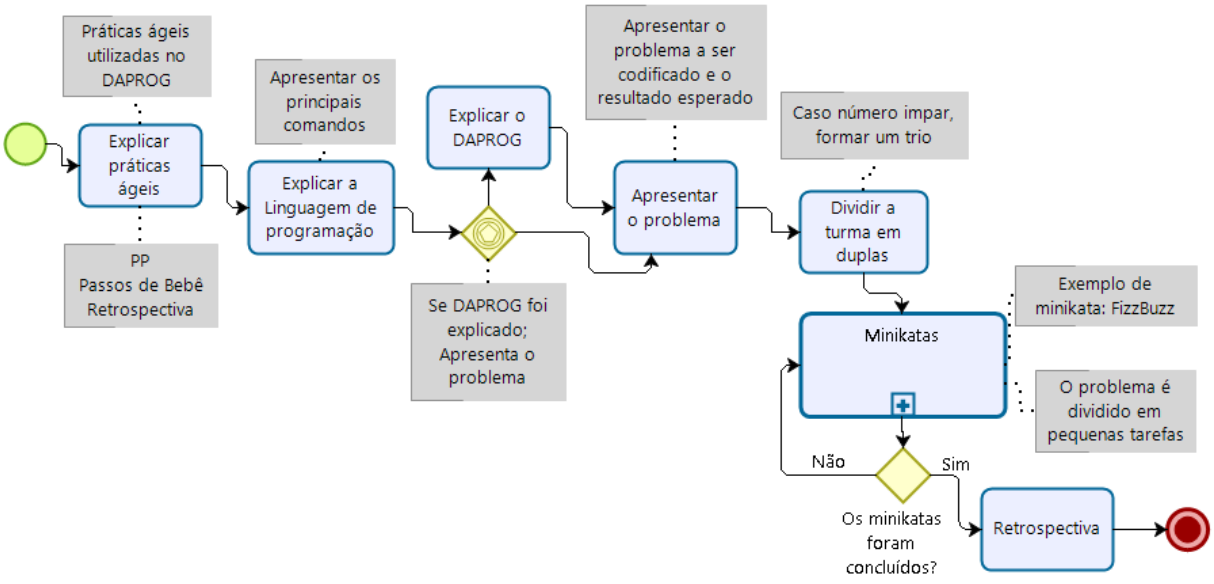

Figura 10: Diagrama de representação do DAPROG. Fonte: Autoria própria.

5.1.2 PASSO A PASSO.

1. Explicação de Programação Pareada, Retrospectiva e Passos de Bebê;

2. Explicação da linguagem de programação e suas principais funcionalidades;
3. Explicação do DAPROG;
4. Apresentação do problema a ser codificado;
5. Apresentação do resultado previamente desenvolvido pelo professor;
6. Escrever no quadro os principais comandos da linguagem de programação;
7. Separação da turma em duplas. Caso estejam em número ímpar, um grupo ficará com 3 alunos;
8. Iniciar o DAPROG com o primeiro minikata;
9. Ao término dos *katas*, fazer a retrospectiva;
10. Se o professor julgar que todos entenderam a linguagem poderá realizar o *Coding Dojo Randori*.

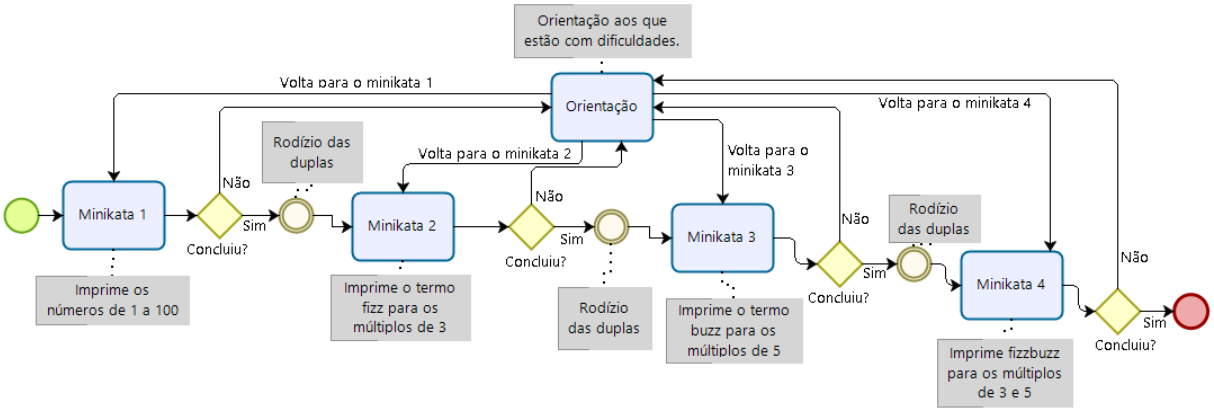

Figura 11: Diagrama de representação do subprocesso minikatas *FizzBuzz* do DAPROG. Fonte: Autoria própria.

Por se tratar de uma turma com foco no aprendizado de uma linguagem de programação, neste momento não é interessante que os alunos empreguem TDD e Refatoração no processo de codificação, pois eles devem estar concentrados em entender o funcionamento da linguagem. Em todo o momento do DAPROG é fundamental a orientação e o envolvimento do professor, intervindo sempre que necessário para que os alunos não se sintam inseguros e sem orientação ao desempenhar suas tarefas.

Itens necessários para o DAPROG.

- Sala de aula com computadores (laboratório de informática).

- Computadores em número suficiente para atender as duplas de alunos.
- Quadro.
- Giz ou canetas para quadro branco (conforme o quadro).
- Projetor multimídia.
- Tela para projeção.

Tarefas do professor e alunos.

O professor e os alunos devem se organizar para que o DAPROG tenha êxito em seu propósito. As responsabilidades de cada um são apresentadas na Figura 12. O professor deve organizar e testar os minikatas previamente escolhendo a tarefa que melhor se adapte à turma.

Depois de feitas todas as devidas apresentações sugeridas nos três primeiros processos da Figura 10 (Explicar práticas ágeis, explicar a linguagem de programação e explicar o DAPROG), ele pede para que os alunos se organizem em duplas. Ao final de cada minikata, ou no tempo estabelecido pelo professor, os alunos devem fazer o rodízio. O rodízio deverá ocorrer de acordo com as regras da Programação em Pares, o aluno no teclado e mouse é o piloto e o aluno ao seu lado é o copiloto.

Terminado o tempo definido de rodízio, o copiloto assume o teclado e o mouse. O aluno que era o piloto passa a ser copiloto em outra dupla. O professor é responsável por organizar as trocas de duplas de acordo com a disposição do laboratório de informática. Ele deve estar atento para que não se repitam as mesmas duplas a cada rodízio.

Ao longo dos minikatas o professor também é responsável por esclarecer possíveis dúvidas que os alunos tenham. E ao final dos minikatas os alunos devem responder às perguntas da retrospectiva apresentadas na tabela 4. Uma vez finalizada a etapa de escrita, o professor deverá dar o retorno das respostas aos alunos.

Tabela 4: Questões a serem feitas na retrospectiva DAPROG. Fonte: Autoria própria.

Questão
O que aprendemos?
O que pode melhorar?
Quais foram os pontos positivos?
Quais foram os pontos negativos?

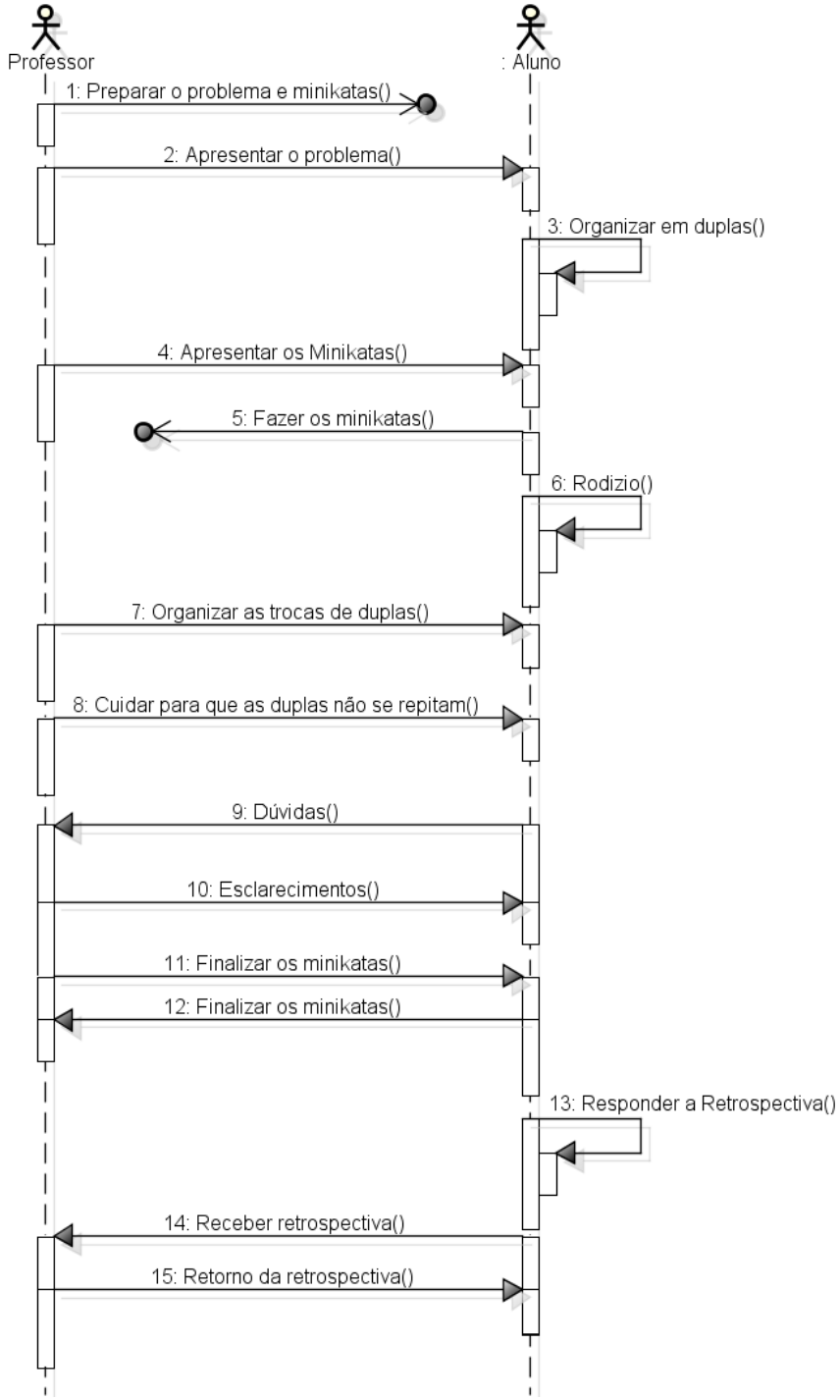

Figura 12: Tarefas do professor e alunos no DAPROG. Fonte: Autoria própria.

5.1.3 RETROSPECTIVA

Ao final de cada DAPROG, deverão ser realizadas as retrospectivas. A sugestão é que o tempo seja em torno de 20 a 30 minutos para que os alunos possam expor suas impressões sobre o que aprenderam, que dificuldades encontraram, quais foram os pontos positivos e negativos de cada dojo e sua experiência geral nas atividades. Uma sugestão para esta retrospectiva é

que seja por escrito e anônima. Em seguida, o professor seleciona aleatoriamente algumas para expor e discutir com a turma.

5.1.4 EXEMPLOS DE ATIVIDADE

As atividades apresentadas nesta Seção como exemplos que podem ser utilizados no DAPROG foram baseadas no *site Dojo Puzzle*[1].

Exemplo 1 - Anagrama.

Escreva um programa que gere todos os anagramas potenciais de uma *string*. Por exemplo, os anagramas potenciais de ”biro” são: biro bior brio broi boir bori ibro ibor irbo irob iobr iorb rbio rboi ribo riob roib robi obir obri oibr oirb orbi orib

Exemplo de como podem ser divididos os minikatas para esta atividade:

- Imprimir a *string*.
- inverter a *string*.
- comparar a *string*.
- Imprimir a *string* invertida

Exemplo 2 - Quiz Animal.

O desafio é criar um jogo de adivinhação sobre animais que deve funcionar do seguinte modo:

O jogo começa pedindo que você pense em um animal. Em seguida ele começa a fazer uma série de perguntas onde as respostas só podem ser sim ou não sobre este animal (exemplos de perguntas: Ele nada? Ele possui pelos? Ele voa?) Em um determinado momento, o jogo irá esgotar as suas perguntas e fará um palpite (É um rato?). Se o jogo fez o palpite correto, o jogo se encerra e pode ser reiniciado com outro animal.

Se o jogo errar o palpite, ele irá solicitar ao usuário o nome do animal que ele estava pensando e também uma pergunta que não tenha sido feita que possa identificar esse animal (a resposta deve ser apenas sim ou não). Essa pergunta irá compor o ”banco de dados” do jogo e poderá ser usada em outro animal no futuro (se apropriado).

Exemplo de como podem ser divididos os minikatas para esta atividade:

[1] http://www.dojopuzzles.com/problemas/todos/

- Imprimir a frase: pense em um animal.
- Imprimir o palpite.
- Imprime sim e não.
- Pede para marcar sim ou não.
- Alimenta o banco de dados.
- Interage como usuário.
- Imprime novo palpite.

5.2 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Mesmo tendo em vista que o professor, de acordo com seu entendimento, tem toda a liberdade de adaptar para as necessidades de sua turma a maneira de como esta proposta será aplicada. Acreditamos que alguns pontos são fundamentais para o êxito desta atividade: A organização em duplas, os rodízios e a troca de experiência entre os alunos.

6 ESTUDOS DE CASO

Para o desenvolvimento deste trabalho foram aplicados 7 questionários em circunstâncias distintas. Cada um destes instrumentos de pesquisa são apresentados no Apêndice A. O primeiro instrumento de pesquisa foi o pré-questionário (Apêndice A.1) e o seu objetivo foi aprimorar e ajustar as perguntas. É importante destacar que o questionário aplicado ao *Coderetreat* 2017 (Apêndice A.3), não foi tabulado neste trabalho por não entrar no escopo da proposta . Para o questionário do DAPROG (Apêndice A.4), a análise é apresentada na Seção 6.6. Os dados dos demais questionários (Apêndice A.2) foram agrupadas em uma única análise.

6.1 PRÉ-QUESTIONÁRIO

Um questionário de teste foi encaminhado ao GT (Grupo de Testes), composto por 18 pessoas. Desse grupo, 8 indivíduos não responderam ao questionário sem apresentar qualquer razão para isso, os outros 10 respondentes prestaram contribuição significativa para a melhoria deste instrumento de pesquisa. Todos os integrantes do GT são envolvidos com tecnologia da informação e a faixa etária está entre 19 e 63 anos. Na Tabela 5, é apresentado um perfil sucinto dos respondentes.

Tabela 5: Perfil dos respondentes. Fonte: Autoria própria.

Respondente	Sexo	Formação	Atividade
1	Feminino	Mestrando	Estudante
2	Feminino	Doutorando	Professor
3	Masculino	Graduando	Estudante
4	Masculino	Especialista	Professor
5	Masculino	Especialista	Professor
6	Masculino	Mestre	Professor
7	Masculino	Mestrando	Professor
8	Masculino	Mestrando	Estudante
9	Masculino	Mestrando	Professor
10	Masculino	Mestrando	Professor

O envio ao GT foi por meio da *web* no formato de um formulário eletrônico. Junto a esse questionário também foi encaminhada uma mensagem explicando o objetivo do questionário e da participação dessas pessoas. A fim de verificar a clareza de cada uma das questões, abaixo de cada pergunta, foi colocado um campo com o intuito de que, quando fosse o caso, o respondente fizesse qualquer observação crítica sobre a pergunta.

Mesmo sendo enviada uma carta-explicação instruindo que os respondentes comentassem qualquer ponto crítico no questionário, poucas perguntas tiveram alguma observação. Na Tabela 6 são apresentadas as perguntas e as respostas dadas por integrantes do GT que, de alguma forma não responderam as perguntas como o esperado. As respostas foram transcritas exatamente como o respondentes as escreveram.

Tabela 6: Questões que geraram dúvidas nas respostas dadas. Fonte: Autoria própria.

Questão	Pergunta	Resposta
Q6	Trabalha ou já trabalhou com desenvolvimento de software? Quanto tempo?	Sim, no desenvolvimento do TCC1
Q6	Trabalha ou já trabalhou com desenvolvimento de software? Quanto tempo?	Sim, 7 anos de forma intensa, mas trabalho até hoje.
Q8	Quais linguagens de programação você utiliza? Liste da mais importante para a menos importante	Eu sei algumas, mas não utilizo quase nenhuma

A Tabela 7 apresenta as perguntas e as observações diretas para aquelas perguntas feitas pelos participantes do GT para as questões que eles tiveram dúvidas. Na coluna observações foram transcritas exatamente da maneira como os participantes se expressaram.

6.2 QUESTIONÁRIOS DOS *CODING DOJOS*

Estes questionários foram aplicados pessoalmente nos seguintes encontros: *Coding Dojo* Unreal Engine, *Coding Dojo Phyton* FTSL e *Coding Dojo* Elixir. Estes realizados na UTFPR Câmpus Curitiba e o *Coding Dojo Phyton* GruPy-PR realizado pelo GruPy-PR no escritório de uma empresa de tecnologia na cidade de Curitiba.

Coding Dojo Unreal Engine

O *Coding Dojo Unreal Engine* para desenvolvimento de jogos foi realizado na Universidade Tecnológica Federal do Paraná, Câmpus Curitiba - UTFPR no dia 27/09/2017. Trinta e seis pessoas se inscreveram para participar do dojo dos quais 20 participantes compareceram. O primeiro questionário foi entregue no início do encontro. O outro, entregue ao final.

Tabela 7: Observações diretas sobre as perguntas. Fonte: Autoria própria.

Questão	Pergunta	Observação
Q1	Nome	É relevante este dado para sua pesquisa?
Q1	Nome	Nome completo?
Q5	Marque a característica que melhor te representa	Acho que não tem a ver muito as opções. Ex.: Posso ser tímida e também prefiro trabalhar em grupo.
Q8	Quais linguagens de programação você utiliza? Marque da mais importante para a menos importante.	Eu sei algumas, mas não utilizo quase nenhuma.
Q9	Você acredita que a troca de experiência com outros progamadores no desenvolvimento de sotware pode enriquecer seu código? Justifique sua resposta	Fica estranho, não sei como justificar.
Q12	Quais desses meios listados a seguir você utiliza para a troca de experiência em desenvolvimento?	Amigo e pessoalmente tem que ser só um, eu acho.
Q16	Em sua opinião, qual dessas é a melhor maneira de adquirir conhecimento?	Acho que pode ser várias opções.
Q16	Em sua opinião, qual dessas é a melhor maneira de adquirir conhecimento?	Aulas acadêmicas.

A Figura 13, apresenta parte dos participantes do *Coding Dojo* que aconteceu no evento ”Semana Technológica” promovido pela UTFPR. Neste encontro os participantes puderam praticar e conhecer melhor a ferramenta de desenvolvimento de jogos *Unreal Engine*[1].

Para muitos, este *Coding Dojo* foi o primeiro evento colaborativo de que participou. Quando questionados sobre o aprendizado por meio do *Coding Dojo*, apenas dois participantes afirmaram não terem tido bom aproveitamento. No geral, o percentual médio de respondentes que afirmou ter compreendido os conceitos de desenvolvimento de jogos por meio do *Coding Dojo Unreal Engine* e da programação por meio de *Blueprints Visual Scripting* (sistema de *scripts* para criar elementos de jogabilidade do *Unreal Editor*[2]) chegou a 80% dos respondentes.

Ao responderem a questão que trata sobre a adequação do *Coding Dojo* ao seu aprendizado 20% participantes afirmaram que esse sistema não está adequado à sua maneira de aprender. Os demais 80% disseram que este *Coding Dojo* foi uma maneira adequada de aprendizado para eles.

[1]https://www.unrealengine.com/en-US/what-is-unreal-engine-4
[2]https://docs.unrealengine.com/en-us/Engine/Blueprints

Figura 13: Grupo de participantes de um *Coding Dojo*. Fonte: Autoria própria.

Coding Dojo Python FTSL

O *Coding Dojo* da linguagem de programação *Python* foi realizado no Fórum de Tecnologia em Software Livre (FTSL) nas dependências da UTFPR, Câmpus Curitiba no dia 29/09/2017.

Neste evento 13 pessoas participaram e o perfil predominante do grupo era de profissionais da área de TI e estudantes do ensino superior. Neste grupo a grande maioria já tinha experiência em desenvolvimento de software adquirido na academia, empresa ou por hobby. A faixa etária dos participantes estava entre 21 e 42 anos de idade.

Do grupo, 53,85% se consideravam com nível intermediário de experiência em programação e outros 30,77% acreditavam estar no nível avançado de experiência em programação. Dois respondentes afirmaram ser iniciantes ou ter um nível básico em programação. Apenas 3 participantes afirmaram já ter participado de outros*Coding Dojos*.

Coding Dojo GruPy-PR

O GruPy-PR é um grupo organizado que se encontra regularmente na cidade de Curitiba-PR para praticar e trocar experiências de programação em *Python* utilizando *Coding Dojo*. O *Coding Dojo* da linguagem de programação *Python* aqui analisado foi realizado nas dependências de uma empresa de comércio eletrônico sediada na cidade de Curitiba

no dia 16/11/2017 das 19 às 22 horas. Neste encontro em específico, 12 profissionais de desenvolvimento de software participaram do encontro.

Este grupo tem características diferentes dos demais grupos de *Coding Dojo* entrevistados. Em sua maioria são profissionais de desenvolvimento de software que já participam de encontros regulares de Dojo de Programação e tem bons conhecimentos de programação. Mesmo com essas características, é possível perceber pelos questionários que muitos dos respondentes procuram no *Coding Dojo* para, além da diversão, ampliar seus conhecimentos em uma linguagem de programação.

Por meio da observação deste grupo foi possível constatar que os desafios lançados durante o encontro eram realmente para testar o conhecimento da dupla que estava no computador naquele momento, ao final do tempo determinado para o piloto as dicas de como melhorar o código ou o que fazer para cumprir a tarefa eram muito proveitosas.

Coding Dojo Elixir

O *Coding Dojo* da linguagem funcional Elixir[3] foi realizado na UTFPR Câmpus Curitiba no dia 08/03/2018. Trinta e quatro alunos da turma de primeiro período do curso de Bacharelado em Sistemas de Informação participaram deste *Coding Dojo*. O questionário de perfil foi entregue no início do encontro e o questionário de percepção foi entregue ao final do mesmo encontro. Por se tratar de um grupo específico de alunos de graduação, e considerando as respostas dadas por estes alunos no questionário, podemos notar que o *Coding Dojo* não atendeu às necessidades de aprendizado desta turma.

Nos encontros de *Coding Dojos* em que nos envolvemos, os participantes ou foram convidados ou participam por iniciativa própria. Nestes encontros havia o interesse pessoal no aprendizado de uma linguagem de programação portanto, o envolvimento e a aceitação foi mais significativa. No *Coding Dojo* Elixir, em específico, o perfil dos participantes era diferente. Eram alunos do curso de graduação da instituição de ensino superior e é possível perceber pelo gráfico da Figura 14 que a percepção de aprendizado do grupo foi baixo. Inclusive, ao analisar as retrospectivas do encontro, a aceitação do *Coding Dojo* pela turma foi mais baixa do que os questionários dos demais encontros analisados neste trabalho.

Fazendo um cruzamento das respostas apresentadas no questionário e os comentários escritos na retrospectiva do *Coding Dojo* Elixir (apêndice B.1), é possível perceber que, para esta turma com 34 alunos, é preciso mais dinamismo e envolvimento do maior número de alunos nas atividades.

[3]https://elixir-lang.org/

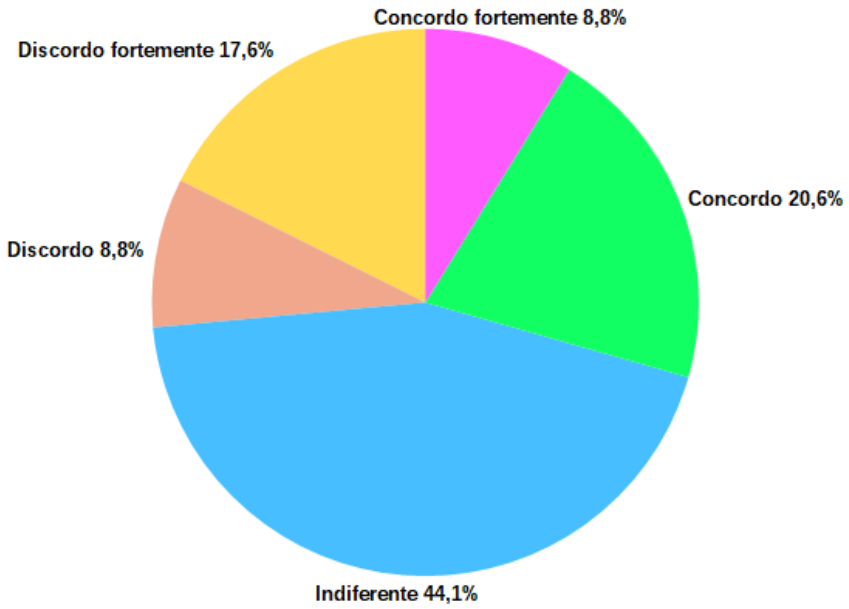

Figura 14: Percepção de aprendizado no *Coding Dojo*. Fonte: Autoria própria

Em nossas observações durante este *Coding Dojo*, ficou evidente a questão da insegurança discutida ao final da Seção 2.2.4. Ali foi possível observar que a insegurança de vários alunos os impediu de participarem ativamente do dojo e isso pode ter refletido nas observações feitas na retrospectiva e nas respostas do questionário.

6.3 RESULTADOS DOS QUESTIONÁRIOS APLICADOS NOS *CODING DOJOS*

Após as tabulações e análises das respostas foi possível obter o perfil do público participante dos *Coding Dojos* aqui pesquisados. Ao todo 79 pessoas responderam de forma voluntária e anônima aos questionários e a faixa etária dos respondentes estava entre 16 e 44 anos.

A área de atuação variou entre profissionais de desenvolvimento de software, estudantes de cursos técnicos e de graduação, gerentes de TI, empresários do ramo de tecnologia e desenvolvimento e interessados em aprender uma nova linguagem de programação. Dos participantes dos encontros aqui estudados, 41,77% dos respondentes não haviam tido qualquer contato com práticas ágeis de desenvolvimento de *software* até o *Coding Dojo*.

O nível de experiência em programação de computadores informado pelos participantes dos *Coding Dojos* é apresentado na Figura 15. Destes, 35,4% afirmaram ter um nível intermediário de conhecimento em programação, enquanto 16,5% afirmam ter nível avançado. A soma entre os valores de iniciantes, básicos e sem experiência em programação chegou ao percentual de 48,10% dos participantes, demonstrando um percentual relevante de

interessados em aprender sobre programação por meio do *Coding Dojo*.

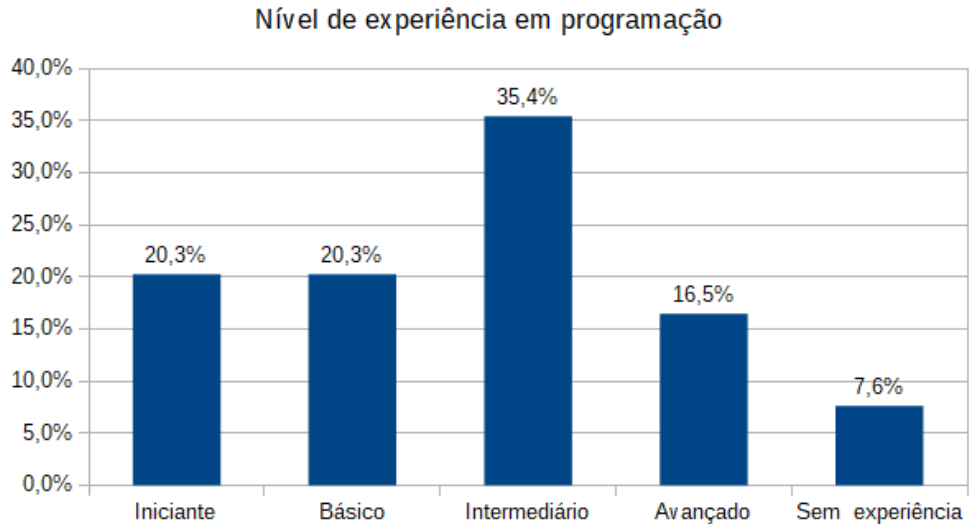

Figura 15: Nível de experiência em programação. Fonte: Autoria própria

As legendas para os gráficos de barra horizontal desta Seção são apresentadas na Figura 16 e os títulos de cada gráfico foram definidos de conforme com sua afirmação correspondente no questionário.

Consideramos que as afirmações marcadas como *indiferente* no segundo questionário denotam que o respondente ou não tem uma opinião formada sobre aquela afirmação, ou ela não provoca qualquer reação. Portanto, adotamos como critério de análise dos resultados não representar as respostas indiferentes nos gráficos a seguir. apenas as respostas que foram marcadas como *concordo totalmente*, *concordo*, *discordo totalmente* e *discordo* estão representadas nos gráficos.

Figura 16: Legendas dos gráficos de barra para o questionário Coding Dojo Elixir. Fonte: Autoria própria

Quando questionados sobre sua percepção de aprendido por meio de *Coding Dojo*, 78,5% dos participantes afirmaram que o encontro lhes ajudou a aprender mais sobre programação conforme apresentado na Figura 17.

Figura 17: Aprendizado de programação com *Coding Dojo*. Fonte: Autoria própria

A Figura 18 nos mostra que 81,0% dos respondentes afirmaram que tiveram uma melhor compreensão do funcionamento da linguagem de programação que estava sendo praticada nos dojos.

Figura 18: Compreensão da linguagem de programação com *Coding Dojo*. Fonte: Autoria própria

Quando questionados se *Coding Dojo* é adequado à forma de aprender dos participantes, a soma das respostas que, ou concordam ou concordam fortemente com a afirmação, foi de 60,08% dos respondentes. Por outro lado, 15,2% discordaram desta afirmação. (Figura 19).

Figura 19: *Coding Dojo* está adequado a forma de aprender do participante. Fonte: Autoria própria

Por meio dos gráficos apresentados nesta Seção podemos perceber que para aquele grupo de 79 entrevistados, *Coding Dojo* é uma boa maneira de aprender novos conceitos de programação de computadores. Entretanto, quando olhamos para o gráfico da Figura 15, 92,4% dos participantes daqueles *Coding Dojos* já tinham alguma experiência em programação. Restando 7,6% dos participantes sem qualquer experiência na área de programação e isto pode caracterizar um perfil de alunos em sala de aula.

6.4 ATIVIDADES DO DAPROG

Foram realizadas duas atividades DAPROG com turmas dos cursos de Bacharelado em Sistemas de Informação e Engenharia da Computação da Universidade Tecnológica Federal do Paraná totalizando em 54 participantes. A linguagem de programação escolhida pelo professor regente das turmas na disciplina de Introdução à Lógica para a Computação foi a linguagem funcional Elixir e as atividade foram realizadas nos dias 12/11/2018 e 22/11/2018 para as turmas do segundo período de Bacharelado em Sistemas de Informação (BSI) e primeiro período do curso de Engenharia da Computação respectivamente.

Por este trabalho tratar da análise da atividade de ensino DAPROG, e esta ser a mesma atividade aplicada às duas turma e o questionário aplicado aos alunos de ambas as turmas também ser o mesmo, os dados dos questionários foram agrupados em uma única análise. Como

critério de análise dos dados deste questionário consideramos que as afirmações marcadas como *indiferente* denotam que o respondente ou não tem uma opinião formada sobre aquela afirmação ou para ele não faz diferença o resultado daquela afirmação, portanto, representamos nos gráficos apenas as respostas que foram marcadas como *concordo totalmente*, *concordo*, *discordo totalmente* e *discordo*.

6.4.1 TURMA DE BSI

Com esta turma, o DAPROG foi realizado na disciplina de Introdução à Lógica para a Computação no dia 12/11/2018. A turma tem um total de 40 alunos registrados dos quais 30 estavam presentes no dia. O professor deixou livre para que os alunos escolhessem participar ou não da atividade.

A linguagem apresentada aos alunos para aquela atividade foi a linguagem de programação funcional Elixir e uma vez que todos concordaram em participar, o professor explicou os conceitos pertinentes à atividade proposta e sua sistemática conforme definido anteriormente. Para iniciar as atividades propostas o professor demonstrou por meio de exemplos repetidos pelos alunos em seus computadores o problema *FizzBuzz* que foi dividido em 4 pequenas tarefas.

Após a explicação os alunos foram divididos em duplas e foi apresentado o problema Livraria do Harry Potter do site dojopuzzles.com. Por conta do nível de dificuldade do problema (alguns alunos estavam com dificuldades na interpretação do problema) e do tempo já avançado não foi possível concluir todos os minikatas propostos para esta atividade. Ao final do DAPROG foram entregues aos alunos uma folha para que respondessem às questões da retrospectiva e um questionário com 19 perguntas sobre suas percepções do DAPROG.

6.4.2 TURMA DE ENGENHARIA DA COMPUTAÇÃO

O DAPROG para a turma de Engenharia da Computação também aconteceu na disciplina de Introdução à Lógica para a Computação. Porém, este foi realizado no dia 22/11/2018 com inicio às 9h20min e término às 11h39min.

A turma tem um total de 44 alunos registrados dos quais 24 estavam presentes no dia. O professor deixou livre para que os alunos escolhessem participar ou não da atividade. Dois alunos escolheram não participar da atividade. Da mesma forma que aconteceu com a turma de Sistemas de Informação, o professor apresentou a linguagem de programação funcional Elixir e passou a explicar aos alunos a sistemática do DAPROG.

Para iniciar as atividades propostas o professor demonstrou por meio de exemplos repetidos pelos alunos em seus computadores o exercício para gerar anagramas de palavras informadas ao computador. Observamos que este exemplo apresentou maior grau de dificuldade exigindo maior atenção dos alunos, então o professor passou a demonstrar como realizar média aritmética como exemplo que eram repetidos pelos alunos em seus computadores.

Após a explicação, foi escolhido o problema *FizzBuzz*[4]. O professor mostrou para os alunos como imprimir a palavra *Fizz*. Logo após eles deveriam imprimir as palavras *Buzz* e *FizzBuzz* respectivamente. Como aconteceu com a turma de Bacharelado em Sistemas de Informação, ao final do DAPROG foram entregues aos alunos uma folha para que respondessem às questões da retrospectiva e um questionário com 19 perguntas sobre suas percepções do DAPROG.

6.5 RETROSPECTIVAS

A Retrospectiva é uma parte fundamental das atividades colaborativas estudadas neste trabalho. No Apêndice B são transcritas as retrospectivas realizadas no *Coding Dojo* Elixir, fundamental para o desenvolvimento de nossa proposta, e a retrospectiva realizada no DAPROG, importante para entender o que os participantes perceberam da atividade.

As retrospectivas aconteceram logo após o encerramento das atividades dos *Coding Dojos* e DAPROG. Com estas respostas dadas de forma anônima, pudemos obter valiosas observações sobre o que os participantes entenderam, o que eles esperavam e como eles enxergavam cada uma das atividades realizadas. Dessa maneira pudemos cruzar as informações entre as respostas dos questionários e as obervações das retrospectivas para que, assim, fosse possível planejar uma proposta que atendesse, ao menos, parte das necessidades daquelas turmas com relação ao aprendizado de programação de computadores.

6.6 RESULTADOS DO DAPROG

Foram realizados dois estudos de caso para o DAPROG. O primeiro aconteceu no dia 12/11/2018 com a turma de segundo período do curso de Bacharelado em Sistemas de Informação. E o segundo estudo de caso aconteceu no dia 22/11/2018 com a turma de primeiro período do curso de Engenharia da Computação. Ambas as turmas da Universidade Tecnológica Federal do Paraná, Câmpus Curitiba.

[4] dojopuzzles.com

Nas duas turmas o professor regente deixou livre para que quem não quisesse participar do estudo de caso. Apenas dois alunos escolheram realizar uma atividade de programação diferente do DAPROG, esta atividade estava relacionada ao projeto final da disciplina naquele semestre. O total de participantes das duas turmas foi de 54 alunos e a linguagem de programação escolhida pelo professor dessas turmas foi a linguagem funcional Elixir.

Com o intuito de obter respostas dos alunos que participaram dos estudos de caso, foi desenvolvido e aplicado um questionário para as duas turmas. O propósito deste questionário para as turmas de Bacharelado em Sistemas de Informação e Engenharia da Computação era avaliar a atividade DAPROG e como o aluno viu esta atividade como forma de ensino. por se tratar dos mesmos questionários aplicados aos alunos de ambas as turmas, agrupamos os dados obtidos em uma única análise.

As respostas marcadas como indiferente nos dois questionários do DAPROG não estão representadas nos gráficos deste trabalho. Porém, para fins de visualização e compreensão dos dados tabulados, a tabela 8 apresenta as afirmações do questionário e os seus respectivos valores para cada resposta fornecida pelos alunos. Para que seja possível a leitura dos gráficos a seguir, as legendas são apresentadas na Figura 20.

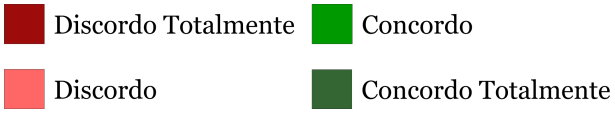

Figura 20: Legendas dos gráficos de barra para o questionário DAPROG. Fonte: Autoria própria

Mesmo que algumas respostas dadas nas retrospectivas afirmem que a explicação prévia sobre a linguagem não foi o satisfatória, os gráficos apresentados nas figuras 21 (47 respostas positivas) e 22 (52 respostas positivas) mostram que 87% dos alunos acreditam que a explicação prévia sobre a linguagem ajudou no desenvolvimento do código e 96,3% dos mesmos alunos afirmaram que a apresentação dos principais comandos sobre a linguagem ajudou no desenvolvimento do código.

Figura 21: A explicação prévia sobre a linguagem contribuiu para o desenvolvimento do código. Fonte: Autoria própria

Os gráficos das Figuras 23 e 24 ilustram que, mesmo alguns alunos tendo dificuldades na compreensão dos problemas e finalização dos mesmos, a maioria deles tiveram facilidades tanto em entender (39 respostas positivas) quanto em desenvolver (37 respostas positivas) os problemas apresentados.

Tabela 8: Total de respostas dadas por questão do DAPROG. Fonte: Autoria própria.

Afirmação	Figura	Respostas Positivas	Respostas Negativas	Indiferentes
A explicação prévia sobre a linguagem contribuiu para o desenvolvimento do código.	21	47	3	4
A apresentação dos principais comandos sobre a linguagem ajudou no desenvolvimento do código.	22	52	1	1
Tive facilidade em desenvolver o problema.	23	37	10	7
Tive facilidade em entender o problema. (1 resposta em branco)	24	39	9	5
O desenvolvimento em dupla contribuiu para entender o código.	25	45	0	9
O desenvolvimento em duplas me ajudou a compreender a linguagem de programação.(1 resposta em branco)	26	30	6	17
As discussões em duplas foram relevantes pois permitiram ver como outras pessoas pensam sobre o mesmo problema.	27	43	1	10
Acredito que a programação em pares torna o desenvolvimento mais eficiente.	28	50	2	2
Gostei da dinâmica de programação em pares.	29	46	1	7
A divisão da atividade em partes menores me ajudou a resolver o problema todo.	30	43	1	10
Esta atividade está de acordo com minha forma de aprender.	31	39	5	10
Gostei desta atividade de ensino.	32	40	4	10
Esta atividade me ajudou a aprender mais sobre a linguagem de programação. (1 resposta em branco)	33	40	2	11
Meu nível de conhecimento nesta linguagem de programação teve boa melhoria.	34	40	3	11

Figura 22: A apresentação dos principais comandos sobre a linguagem ajudou no desenvolvimento do código. Fonte: Autoria própria

Figura 23: Tive facilidade em desenvolver o problema. Fonte: Autoria própria

Figura 24: Tive facilidade em entender o problema. Fonte: Autoria própria

Os gráficos apresentados nas Figuras, 25, 26, 27, 28 e 29 trazem um retrato da percepção dos alunos sobre as atividades em duplas. É possível perceber pelas respostas positivas destes gráficos que parte dos alunos que realizaram a atividade do DAPROG entendeu que esta forma de troca de experiência é muito útil para seus aprendizados.

Figura 25: O desenvolvimento em dupla contribuiu para entender o código. Fonte: Autoria própria

Figura 26: O desenvolvimento em duplas me ajudou a compreender a linguagem de programação. Fonte: Autoria própria

Figura 27: As discussões em duplas foram relevantes pois permitiram ver como outras pessoas pensam sobre o mesmo problema. Fonte: Autoria própria

Figura 28: Acredito que a programação em pares torna o desenvolvimento mais eficiente. Fonte: Autoria própria

Figura 29: Gostei da dinâmica de programação em pares. Fonte: Autoria própria

Sobre a divisão das atividades em minikatas ajudar na resolução do problema, apenas 1 dos respondentes acredita que isso não ajuda na resolução do problema como um todo (Figura 30). Por outro lado, 79,6% dos participantes entendem que esta divisão, neste caso, é benéfica para o aprendizado em programação e os ajudou a resolver o problema como um todo.

Figura 30: A divisão da atividade em partes menores me ajudou a resolver o problema todo. Fonte: Autoria própria

Quanto à atividade proposta pelo DAPROG como um todo, as Figuras 31, 32 e 33 mostram que a maior parte dos alunos gostou da atividade, pois ela os ajudou a aprender mais sobre a linguagem. Além disso, mais de 70% dos participantes afirmaram que esta atividade de aula está de acordo com sua forma de aprendizado (Figura 31).

Figura 31: Esta atividade está de acordo com minha forma de aprender. Fonte: Autoria própria

Figura 32: Gostei desta atividade de ensino. Fonte: Autoria própria

Figura 33: Esta atividade me ajudou a aprender mais sobre a linguagem de programação. Fonte: Autoria própria

Por fim, o gráfico apresentado pela Figura 34 mostra que 74,1% dos respondentes acredita que seu nível de conhecimento na linguagem de programação funcional Elixir teve boa melhoria.

Figura 34: Meu nível de conhecimento nesta linguagem de programação teve boa melhoria. Fonte: Autoria própria

6.7 DISCUSSÃO

Múltiplas inteligências e estilos individuais de aprendizagem são características comuns em uma sala de aula (ADAN-COELLO et al., 2008), (GARDNER; THOMAS, 1989), (ACOSTA et al., 2017). Outrossim, a ”Interação Social”, discutida pela Psicologia e o comportamento definido como ”Zona de Desenvolvimento Proximal” proposto por Vigostsky também podem fazer parte da personalidade do indivíduo. Ao aluno se relacionar com outros colegas, sua capacidade de fazer sozinho, suas habilidades e as atitudes se potencializam permitindo que ele possa ampliar o próprio desenvolvimento intelectual (VIGOTSKY, 2001), (FONTANA, 2015). Associados a estes conceitos os fundamentos do *Coding Dojo* de ”não competitividade”, ”problemas simples”, ”aprendizado pela prática” (SATO et al., 2008), (LUZ et al., 2013), (HEINONEN et al., 2013), Programação Pareada, Passos de Bebê e Retrospectiva (KONGCHAROEN; HWANG, 2015), (RODRIGUES et al., 2017), (PRZYBYLEK; KOTECKA, 2017) foram essenciais para o desenvolvimento dos objetivos deste trabalho.

Os objetivos que estabeleceram o desenvolvimento deste trabalho foram, propor a atividade de ensino de programação de computadores nominada de DAPROG. E avaliar se esta atividade motiva o interesse pelo aprendizado de novas linguagens de programação, permitindo que a troca de experiências entre os alunos da graduação de cursos de computação os leve a perceber outras maneiras de alcançar um resultado. No planejamento da proposta consideramos as seguintes premissas:

- Não competitividade;
- Exercícios com problemas simples;
- aprendizado pela prática;
- Programação em Pares;
- Passos de Bebês;
- Retrospectiva;

- Interação entre os alunos.

Completando esta proposta, descrevemos os procedimentos para que o professor possa realizar o DAPROG em sala de aula definindo a formação de duplas entre os alunos para que pudessem trocar conhecimentos e experiência entre eles, o sistema de rodízio dos pares e exemplos de atividades, sugerindo como o professor pode dividi-las em pequenas tarefas, as quais chamamos de *minikatas*.

Para avaliar esta proposta e responder a pergunta de pesquisa RQ04 (DAPROG é útil para o ensino de programação para uma sala de aula com um número maior do que 20 alunos?), aplicamos o DAPROG em duas turmas, uma com 30 alunos e outra com 24 alunos totalizando 54 participantes desta atividade. Ao serem questionados se a divisão das atividades em partes menores os ajudou a resolver o problema (Figura 30), 43 participantes (79,6%) afirmaram positivamente, enquanto apenas 1 dos respondentes discordou da afirmação. Trinta e nove participantes afirmaram que a atividade estava de acordo com sua forma de aprender (72,2%) e 5 participantes (9,3%) discordaram da afirmação. Nestes mesmo grupo, 40 alunos (74,1%) disseram que gostaram da atividade de ensino e 4 (7,4%) discordaram da afirmação.

Com o intuito de alcançar os objetivos gerais deste trabalho definimos 3 objetivos específicos que são descritos a seguir. Descrevemos também as ações realizadas para cada objetivo específico e os seus resultados:

- Objetivo específico: Caracterizar a percepção dos participantes de encontros colaborativos de desenvolvimento de software sobre o uso da prática colaborativa no aprendizado de programação de computadores.

  Para que fosse possível alcançar este objetivo específico participamos de 4 encontros de *Coding Dojos*. Neles pudemos interagir e entrevistar por meio de questionários impressos 79 pessoas. O nível de experiência que se consideram os participantes destes encontros é apresentado na tabela 9. Para 60,8% deles o *Coding Dojo* é uma boa maneira de aprender mais sobre uma linguagem de programação.

  Outros 81% dos participantes afirmaram ter aprendido algo sobre a linguagem de programação por meio do *Coding Dojo*. Com os percentuais apresentados é possível concluir que, para estes grupos, o encontro colaborativo *Coding Dojo* é uma forma útil para aprender novas técnicas de programação e/ou até mesmo uma nova linguagem de programação.

  Porém, conforme discutido na Seção *Coding Dojo* Elixir (6.2), analisando os 34 participantes deste encontro, em conjunto com suas respostas na retrospectiva deste

Tabela 9: Nível de experiência em programação participantes dos *Coding Dojos* apresentados neste trabalho. Fonte: Autoria própria.

Experiência em programação	Núm. de respostas	% das respostas
Sem experiência.	6	7,6
Iniciante.	16	20,3
Básico.	16	20,3
Intermediário.	28	35,4
Avançado.	13	16,5

mesmo encontro, podemos observar que, para eles, *Coding Dojo* não representa necessariamente uma opção de aprendizado. Neste grupo 44,1% dos participantes ficaram indiferente com a afirmação sobre seu aprendizado por meio deste encontro colaborativo. Outros 26,4% entendem que o *Coding Dojo* não é uma maneira adequada de aprender programação.

- Objetivo específico: Testar se a participação na atividade Dojo de Aprendizagem de Programação resulta na auto percepção de aprendizagem dos alunos de graduação que cursam programação de computadores.

  Um vez caracterizado e entendido o perfil dos participantes de *Coding Dojo* e entendido que este encontro colaborativo é mais dinâmico para um grupo com menos participantes do que existente em uma sala de aula, elaboramos a proposta do DAPROG como uma atividade de ensino que pudesse atender a um grupo mais específico de aprendizes de programação de computadores, como por exemplo, os participantes do *Coding Dojo* Elixir que em sua totalidade eram alunos de curso de graduação.

  Os gráficos apresentados nas Figuras 31 e 34 informam que para 39 participantes do DAPROG que responderam o questionário, a atividade estava de acordo com sua forma de aprender chegando a um percentual de 72,2% de respostas positivas. Para este mesmo grupo, 40 pessoas (74,1%) afirmaram que o DAPROG lhes ajudou a aprender mais sobre a linguagem de programação. Com estes dados, é possível responder a pergunta de pesquisa RQ01, ”DAPROG facilita o aprendizado de novas linguagens de programação?”, pois todos os alunos que participaram da atividade a fizeram com uma linguagem de programação que ainda não conheciam, a linguagem de programação funcional Elixir.

- Objetivo específico: Avaliar se a participação na atividade Dojo de Aprendizagem de Programação motiva a interação dos alunos de graduação nas aulas de programação de computadores.

Em todas as perguntas sobre desenvolvimento em duplas (Figuras 25, 26, 27, 28 e 29), é possível observar que houve aceitação das turmas, quando questionados se o desenvolvimento em dupla contribuiu para o entendimento do código, dos 54 respondentes, 45 afirmaram positivamente (83,3%). Destes ainda, 50 participantes do DAPROG (92,6%) afirmaram acreditar que a programação em pares torna o desenvolvimento mais eficiente.

A Figura 25 apresenta o gráfico que trata da pergunta sobre a compreensão do código no desenvolvimento em dupla. O total de respostas positivas para esta pergunta chegou a 83,3% que corresponde a 43 pessoas. Nesta questão não houveram respostas negativas, os outros 16,7% (9 respostas) dos respondentes desta pergunta marcaram ela como indiferente. Neste objetivo específico as RQ02 (DAPROG possibilita a troca de experiências entre os alunos?) e RQ03 (DAPROG facilita a compreensão da programação em pares?) foram respondidas, pois, de acordo com os gráficos apresentados, a troca de experiência entre os alunos e a compreensão da programação em pares se mostrou satisfatória para este grupo.

Métodos colaborativos de aprendizagem em programação têm sido muito explorados na literatura científica. Williams e outros (2008), por exemplo, dizem que pesquisas sugerem que programação em pares traz muitos benefícios pedagógicos encorajando aos alunos a interagir com seus colegas criando um ambiente de aprendizado colaborativo além de trazer vantagens para os professores. Em outro estudo realizado foi observado que, ao aplicar programação em pares para um grupo de estudantes, os que praticavam o desenvolvimento em pares conseguiam obter melhores resultados do que aqueles que trabalhavam sozinhos (KONGCHAROEN; HWANG, 2015).

Os resultados apresentados neste Capítulo nos possibilitaram ter um vislumbre de que o DAPROG é uma atividade que possibilita servir de auxílio para o ensino de programação de computadores ao se utilizar do aprendizado colaborativo como método de ensino. Dos 54 participantes da atividade que responderem ao questionário, 74,1% (40 respostas) afirmaram que esta atividade ajudou na compreensão da nova linguagem de programação apresentada pelo professor (Figura 33).

O DAPROG permitiu que eles pudessem trocar experiências e a sua compreensão tanto do problema apresentado quanto da linguagem proposta os levou a ter a percepção de melhora no aprendizado, de acordo com as respostas dos questionários. Desse modo os participantes também puderam observar que existem outras formas de se chegar a uma mesma solução.

Estes resultados nos levam à discussão apresentada na Seção 2.2.3, Métodos ágeis e

o aprendizado em desenvolvimento de software, ao que afirmaram Boydens e outros (2015) que o emprego de práticas ágeis como PP e TDD têm se mostrado eficientes para o ensino de programação de computadores.

O ensino de programação pareada e Desenvolvimento Orientado a Testes em um instituição de ensino superior ainda é um desafio (HEINONEN et al., 2013). Por outro lado, Boydens e outros (2015) afirmam que o emprego das práticas ágeis PP e TDD têm sido estudadas como métodos de ensino de programação de computadores, e a conclusão a que chegaram é que ambas têm se mostrado eficientes para esse fim.

7 LIÇÕES APRENDIDAS COM AS RETROSPECTIVAS

Neste capítulo apresentamos as lições extraídas com as retrospectivas realizadas no encontro do Coding Dojo Elixir e nos encontros do DAPROG. Estas retrospectivas foram feitas de maneira que os alunos não precisaram se identificar, apenas responder as perguntas projetadas em uma tela ao final dos encontros. Entendemos que desta maneira as respostas refletiram melhor o pensamento dos participantes sobre o encontro.

7.1 RETROSPECTIVA DO CODING DOJO ELIXIR

Este *Coding Dojo* teve uma característica diferente dos demais que participamos, ele foi realizado com uma turma com 34 alunos da graduação do curso de Bacharelado em Sistemas de Informação da UTFPR. Com este grupo foi possível observar que para muitos alunos, o *Coding Dojo* não foi uma experiência proveitosa, isto ficou claro na retrospectiva que realizamos após o encontro. Nesta retrospectiva, por conta do tempo avançado do evento, optamos por pedir para que os participantes respondessem apenas sobre os pontos positivos e negativos do encontro.

Como ponto negativo do *Coding Dojo*, muitos alunos destacaram a falta de oportunidade para que todos pudessem participar e a falta de tempo para praticar o desenvolvimento. por ser um grupo grande muitos dos que estavam ali afirmaram que o encontro estava ”tedioso”, acreditamos que esse tédio se deu justamente pela falta de tempo de todos participarem e por apenas duas pessoas de cada vez se envolverem na programação, enquanto outros observavam sem, de acordo com as regras do *Coding Dojo*, poder opinar sobre o que estavam vendo.

De acordo com um dos participante o encontro estava ”chato e tedioso para quem tem mais conhecimento em programação”. Outros, seguindo a mesma linha de raciocínio sobre o dinamismo do encontro afirmaram que sentiram falta de ”por a mão na massa” durante o evento. Isto nos alerta para o tamanho ideal para o desenvolvimento do encontro, enquanto neste grupo haviam 34 participantes, de acordo com Bache (2013) o tamanho ideal para se

obter bons resultados com o Coding Dojo é entre 5 e 15 participantes por encontro. Podemos concluir que as observações feitas nesta retrospectiva refletem exatamente o número excessivo de participantes em um único encontro.

Com relação aos pontos positivos do encontro, muitos afirmaram que a cooperação dos pares os ajudou no entendimento da linguagem e da programação promovendo o pensamento coletivo. Além da cooperação, também foram destacadas positivamente a simplicidade e o compartilhamento de ideias inclusive foi mencionado que os diferentes níveis de conhecimento tornaram a atividade mais proveitosa. A atividade prática também foi mencionada por alguns participantes, de acordo com um deles ”não teve muita teoria abstrata” no encontro o que promoveu melhor entendimento da linguagem de programação e o trabalho em equipe. O aprendizado ficou evidente nos pontos destacado aqui e na observação de outro participante que afirmaram ter aprendido novos comandos por meio do *Coding Dojo*, além de poderem perceber os erros no momento do desenvolvimento. De acordo com um aluno estes encontro foi um ”passatempo cerebral”.

Estas respostas positivas ao *Coding Dojo* no levam a reafirmar que a programação em pares incentiva o aluno a interagir com os colegas e proporciona um ambiente colaborativo. de acordo com Williams et al. (2008) O envolvimento dos colegas nessa prática incentiva aos alunos a participarem ativamente na sala de aula e o aprendizado utilizando programação pareada pode reduzir a evasão dos cursos introdutórios de programação (ESTÁCIO et al., 2015a) pois em estudo realizado por Kongcharoen e Hwang (2015) alunos que praticavam programação pareada conseguiam obter melhores resultados em relação a outros que trabalhavam sozinhos.

7.2 RETROSPECTIVAS DAPROG

Estas retrospectivas aconteceram com alunos das turmas de Bacharelado em Sistemas de Informação e Engenharia da Computação da UTFPR. Foi entregue uma folha em branco para os alunos e em seguida foi projetada na tela do projetor multimídia as quatro perguntas da retrospectiva (*O que aprendemos? - O que pode melhorar? - Quais foram os pontos positivos? - Quais foram os pontos negativos?*) conforme apresentadas na Tabela 4, os alunos responderam de forma anônima a cada uma destas perguntas.

7.2.1 PRIMEIRO DAPROG

O primeiro DAPROG aconteceu com alunos da turma de 1º período do 2º semestre de 2018 de Bacharelado em Sistemas de Informação.

O que aprendemos: Nesta pergunta a percepção de aprendizado foi unânime, todos afirmaram que aprenderam sobre uma nova linguagem de programação funcional conhecida como Elixir. Outros ainda destacaram de maneira positiva o trabalho em grupos, a dinâmica e o melhor entendimento com o trabalho em duplas, a troca de ideias utilizando Elixir e o desenvolvimento da lógica de programação além de possíveis soluções para resolver um problema.

O que pode melhorar: Nas sugestões de melhoria do DAPROG muitos entenderam que faltou uma explicação mais detalhada da linguagem de programação e maior esclarecimento do problema a ser resolvido. Um dos participantes entendeu que a dinâmica das duplas deveria ser ”menos movimentada”, assim eles teriam mais tempo para o compartilhamento da informação. Outro participante afirmou que poderia haver maior interação entre as duplas para que pudessem comparar seus códigos.

Mesmo tendo acontecido o rodízio das duplas, nesta turma, quatro alunos se referiram diretamente aos rodízios das duplas, três deles afirmaram que não houve rodízio. O quarto aluno apenas informou a expressão ”Rodízio de duplas” não deixando claro se ele entendeu que não houve o rodízio, ou se ele está sugerindo que o rodízio deve ser melhorado. De qualquer forma, diante destas afirmações, entendemos que as explicações sobre a metodologia de funcionamento do DAPROG devem ser melhor esclarecida para que dúvidas como esta não surjam nas próximas atividades.

Uma outra lição que extraímos desta pergunta da retrospectiva é sobre a organização previa da atividade que será executada em sala de aula. Seis alunos, de uma forma ou de outra, apresentaram dificuldades com relação ao exercício proposto. Isto nos mostra que os exercícios devem ser melhor planejado haja vista a turma ser composta por alunos iniciantes em programação e a complexidade do exercício provocou certo desconforto a este grupo.

Pontos positivos: O conhecimento de uma nova linguagem de programação foi destacado como positivo para muitos alunos, outro ponto que teve destaque com esta turma foi o trabalho em duplas. A divisão da atividade em partes menores fez com que os alunos pudessem ter maior facilidade no desenvolvimento do exercício como um todo. Um dos alunos destacou que ”a dinâmica ajudou em uma melhora de pensamento lógico e rápido”. O dinamismo e o auxílio do professor durante a tarefa também teve destaque positivo pela turma.

Pontos negativos: Assim como na pergunta ”O que pode melhorar?”, 3 alunos afirmaram que não houve o rodízio das duplas, outro ponto negativo foi a falta de tempo, mostrando que o tempo do rodízio das duplas deve ser reavaliado. A complexidade do exercício proposto para aquela turma também foi destacado como negativa. Um dos participantes afirmou que o tempo gasto na discussão com seu par foi negativo. Alguns alunos destacaram que a explicação da linguagem ou não foi o suficiente ou foi confusa. Ao observar as demais respostas nesta pergunta podemos refletir que será necessário uma melhor explicação, inclusive com exemplos mais simples, da linguagem para que os alunos possam se sentir seguros ao trabalhar em equipe ao longo do DAPROG.

7.2.2 SEGUNDO DAPROG

O segundo DAPROG aconteceu com alunos da turma de 1º período do 2º semestre de 2018 de Engenharia da Computação.

O que aprendemos: Assim como aconteceu com a turma do primeiro DAPROG, nesta pergunta a percepção de aprendizado foi unânime, todos afirmaram que aprenderam sobre uma nova linguagem de programação funcional conhecida como Elixir. Além do trabalho em duplas para a resolução de problemas e a divisão da atividade em partes menores

O que pode melhorar: Nesta pergunta os alunos destacaram que pode ser melhorada a explicação sobre a linguagem seus comandos, sintaxes e conceitos básicos. Um dos participantes afirmou que poderia ter mais ajudantes em sala de aula para o desenvolvimento da atividade. Outro participante destacou o pouco tempo para o aprofundamento da atividade. Mais uma vez fica claro o melhor planejamento prévio da execução da atividade, não é possível desenvolver a atividade apenas seguindo o roteiro proposto neste trabalho, é necessário que o professor, antes de iniciar a aula, desenvolva sua proposta de atividade prevendo o exercício que será feito pelos alunos, a divisão do exercício em minikatas, quais comandos da linguagem deverão ser utilizados no exercício a explicação dos comandos e a exposição desses comandos no quadro negro ou qualquer outro meio de fácil acesso aos alunos.

Pontos positivos: Os participantes deste DAPROG destacaram como positivos o auxílio do professor ao longo da tarefa, o contado com uma nova linguagem, a dinâmica em equipe bem como a eficiência da programação em duplas e o desenvolvimento de novas habilidades de programar.

Pontos negativos: Nesta pergunta, 5 alunos, ou afirmaram que não tinham pontos negativos na atividade ou disseram que não havia nada a declarar sobre esta questão. Os demais

alunos viram como negativo a falta de tempo para a atividade, a execução da atividade e a falta de apresentação da sintaxe da linguagem. Um aluno justificou como negativo o pouco tempo para o aperfeiçoamento da linguagem, porém, com o tempo de aula regulamentar de uma sala de aula não é possível se aprofundar em qualquer conhecimento, pois, de acordo com Piazzi (2008), o aprendizado acontece de fato quando o aluno, em outro momento após a aula, revisa o conteúdo que lhe foi passado pelo professor.

8 CONCLUSÃO

Neste trabalho analisamos os participantes de encontros colaborativos de programação. Com os resultados obtidos nas entrevistas e observações feitas pessoalmente nestes encontros, propusemos e avaliamos uma atividade para ser utilizada em sala de aula; ela foi baseada na atividade *Coding Dojo* e a nominamos de DAPROG, sigla para Dojo de Aprendizagem de Programação.

Realizamos dois estudos de caso com o DAPROG, ambos para alunos de cursos de graduação. Após as atividades, os alunos foram questionados sobre a programação em pares e dos que que responderam ao questionário, 92,6% afirmou que a programação em pares realmente torna o desenvolvimento mais eficiente. Este mesmo grupo (85,2%) afirmou ter gostado da dinâmica de programação em pares.

Constatamos que a divisão das atividades de programação em partes menores (os minikatas) teve uma boa aceitação por parte dos alunos pois 79,6% dos participantes afirmaram que isso os ajudou a resolver o problema proposto mais facilmente. Como resultado de nossa pesquisa, 74,1% dos participantes do DAPROG afirmou que gostou desta atividade. E o mesmo percentual afirmou ainda que esta atividade lhes ajudou a aprender mais sobre programação estando ela de acordo com sua forma de aprender.

Os resultados apresentados neste trabalho indicam que houve uma boa aceitação da atividade proposta para os alunos daqueles estudos de caso. Assim podemos observar que, ao envolver os discentes tanto no processo de aprendizado, quanto no processo de compartilhar seu conhecimento, as aulas podem ficar mais atrativas e alcançar um maior número de alunos na sala de aula.

Em nossas observações durante os *Coding Dojos* de que participamos pudemos perceber em alguns dojos a falta de interesse de umas poucas pessoas, a timidez e a insegurança de outras. Isto nos leva ao que foi apresentado na Seção 2.2.4 do Capítulo 2 em que Heinonen et al. (2013) registram situações de timidez de alguns participantes de *Coding Dojo*.

Nas observações *in loco* que acontecerem durante o DAPROG pudemos perceber

algumas limitações da atividade com relação a uns poucos alunos, como por exemplo, a falta de engajamento, a valorização da rapidez em terminar a atividade e o não envolvimento no rodízio das duplas. Estas são situações que fogem ao controle do professor, porém, entendemos que são questões a serem tratadas em trabalhos futuros.

REFERÊNCIAS

ACOSTA, O. C.; REATEGUI, E. B.; BEHAR, P. A. Recomendações de conteúdo em um ambiente colaborativo de aprendizagem baseada em projetos. **Revista Brasileira de Informática na Educação - RBIE**, 2017.

ADAN-COELLO, J. M. et al. Conflito socio-cognitivo e estilos de aprendizagem na formação de grupos para o aprendizado colaborativo de programação de computadores. **Revista Brasileira de Informática na Educação**, v. 16, n. 3, p. 9–19, Setembro/Dezembro 2008.

ANICHE, M. **Test Driven Developement. Teste e Design no Mundo Real**. [S.l.]: Casa do Codigo, 2012.

BACHE, E. **The Coding Dojo Handbook a practical guide to creating a space where good programmers can become great programmers**. [S.l.]: Leanpub, 2013.

BECK, K. **Test Driven Development By Example**. [S.l.]: Addison Wesley, 2002.

BECK, K.; ANDRES, C. **Extreme Programming Explained: Embrace Change**. [S.l.]: Addison, Wesley, 2005.

BERBEL, N. A. N. As metodologias ativas e a promoção da autonomia de estudantes. **Seminário de Ciências Sociais e Humanas**, v. 32, n. 1, p. 25–40, jan 2011.

BISSI, W.; NETO, A. G. S. S.; EMER, M. C. F. P. The effects of test driven development on internal quality, external quality and productivity: A systematic review. **Information and Software Technology**, 2016.

BOSSAVIT, L.; GAILLOT, E. The coder's dojo – a different way to teach and learn programming. In: **Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering**. Berlin, Heidelberg: Springer-Verlag, 2005. (XP'05), p. 290–291. ISBN 3-540-26277-6, 978-3-540-26277-0.

BOYDENS, J.; CORDEMANS, P.; HALLEZ, H. On using test driven development to tutor novice engineering students using self assessment. **43rd Annual SEFI Conference June 29, July 2, 2015 Orleans, France**, 2015.

COCKBURN, A. **Agile Software Development Software Development as a Cooperative Game. 2nd edition**. [S.l.]: Pearson Education, Inc, 2007.

COCKBURN, A.; WILLIAMS, L. The costs and benefits of pair programming. **Extreme programming examined**, 2000.

COHN, M. **Agile Estimating and Planning**. [S.l.]: Pearson Education. Inc, 2006.

DIEBOLD, P.; ZEHLER, T. The right degree of agility in rich processes. In: ____. **Managing Software Process Evolution: Traditional, Agile and Beyond – How to Handle Process Change**. Cham: Springer International Publishing, 2016. p. 15–37. ISBN 978-3-319-31545-4.

DUSTIN, E.; RASHKA, J.; PAUL, J. **Automated Software Testing: Introduction, Management, and Performance**. 13. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2008. ISBN 0-201-43287-0.

ESTÁCIO, B. et al. Evaluating collaborative practices in acquiring programming skills: Findings of a controlled experiment. In: **Software Engineering (SBES), 2015 29th Brazilian Symposium on**. [S.l.: s.n.], 2015. p. 150–159.

ESTÁCIO, B. et al. Evaluating the use of pair programming and coding dojo in teaching mockups development: An empirical study. In: **System Sciences (HICSS), 2015 48th Hawaii International Conference**. [S.l.: s.n.], 2015. p. 5084–5093. ISSN 1530-1605.

ESTÁCIO, B. et al. On the randori training dynamics. In: **Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering**. New York, NY, USA: ACM, 2016. (CHASE '16), p. 44–47. ISBN 978-1-4503-4155-4.

FLOR, N. V. Side-by-side collaboration: a case study. **International Journal of Human-Computer Studies**, v. 49, n. 3, p. 201 – 222, sep 1998. ISSN 1071-5819.

FONTANA, L. A. M. Aprendizagem colaborativa e construção da inteligência coletiva no espaço cibernético. **Sobre educação e tecnologia: conceitos e aprendizagem**, 2015.

FOWLER, M. et al. **Refactoring Improving the Design of Existing Code 1 edition**. [S.l.]: Addison Wesley, 2012.

FRANCIS, J. P. et al. Chaste: A test driven approach to software development for biological modelling. **Computer Physics Communications**, v. 180, n. 12, p. 2452–2471, 2009. ISSN 0010-4655.

FRONZA, I. et al. Analysing the usage of tools in pair programming sessions. **Springer Verlag Berlin Heidelberg**, LNBIP 77, p. 1–11, 2011.

GANESAN, D. et al. An analysis of unit tests of a flight software product line. **Science of Computer Programming**, v. 78, n. 12, p. 2360 – 2380, 2013.

GARDNER, H.; THOMAS, H. Multiple intelligences go to school. educational implications of the theory of multiples intelligences. **American Educational Researcher Association**, 1989.

GIL, A. C. **Como Elaborar Projeto de Pesquisa**. [S.l.]: Atlas, 2002.

GONÇALVES, L.; LIDERS, B. **Getting Value out of Agile Retrospectives. A Toolbox of Retrospective Exercises**. [S.l.]: Leanpub, 2015.

HAINES, C. **Understanding the Four Rules of Simple Design. And other Lessons from watching Thousands of pair works on Conway's Game of Life**. [S.l.]: Leanpub, 2014.

HAUPTMANN, B. et al. Generating refactoring proposals to remove clones from automated system tests. In: **2015 IEEE 23rd International Conference on Program Comprehension**. [S.l.: s.n.], 2015. p. 115–124. ISSN 1092-8138.

HEINONEN, K. et al. Learning agile software engineering practices using coding dojo. In: **Proceedings of the 14th Annual ACM SIGITE Conference on Information Technology Education**. New York, NY, USA: ACM, 2013. (SIGITE '13), p. 97–102. ISBN 978-1-4503-2239-3.

HORN, E. M. et al. Individual differences in dyadic cooperative learning. **Journal of Educational Psychology**, v. 90, n. 1, p. 153–161, 1998.

HUNT, A.; THOMAS, D. **O programador Pragmático. De aprendiz a mestre**. [S.l.]: Artmed Editora S.A., 2010.

ISO; IEC; IEEE. International standard - systems and software engineering–vocabulary. **ISO/IEC/IEEE 24765:2017(E)**, p. 1–541, Aug 2017.

JANZEN, D.; SAIEDIAN, H. Test driven development: Concepts, taxonomy, and future direction. **IEEE Computer Society**, 2005.

KARHU, K. et al. Empirical observation on software testing automation. In: **Internationa Conference on Software Testing Verification and Validation**. [S.l.: s.n.], 2009.

KATTAN, H. M. et al. Mob programming in an academic setting. In: **WBMA**. [S.l.: s.n.], 2017.

KAYONGO, P.; CHIGONA, W.; MABHENA, Z. Why do software developers practice test-driven development? In: **2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)**. [S.l.: s.n.], 2016. p. 357–361.

KERTH, N. L. **Project Retrospectives. A handbook for team reviews**. [S.l.]: Dorset House Publishing Co., Inc, 2001.

KONGCHAROEN, C.; HWANG, W. Y. A study of pair-programming configuration for learning computer networks. In: **Ubi Media Computing (UMEDIA), 2015 8th International Conference**. [S.l.: s.n.], 2015. p. 369–375.

LILIENTHAL, C. From pair programming to mob programming to mob architecting. **Springer International Publishing**, 2017.

LIU, W.; LIU, H. Major motivations for extract method refactorings: analysis based on interviews and change histories. **Frontiers of Computer Science**, v. 10, n. 4, p. 644–656, 2016. ISSN 2095-2236.

LOPES, A. P. B. et al. Escrita cooperativa na promoção da aprendizagem. **Revista Brasileira de Informática na Educação**, v. 14, n. 3, p. 39–48, Setembro-Dezembro 2006.

LUZ, R. B.; NETO, A. Usando dojos de programação para o ensino de desenvolvimento dirigido por testes. **Anais do 23o Simpósio Brasileiro de Informática na Educação (SBIE 2012)**, Novembro 2012.

LUZ, R. B. da; NETO, A. G. S. S.; NORONHA, R. V. Teaching tdd, the coding dojo style. In: **2013 IEEE 13th International Conference on Advanced Learning Technologies**. [S.l.: s.n.], 2013. p. 371–375. ISSN 2161-3761.

MARCONI, M. de A.; LAKATOS, E. M. **Fundamentos de Metodologia científica. 5a Ed.** [S.l.]: Atlas, 2003.

MARTIN, R. C. **The Clean Coder: A Code of Conduct for Professional Programmers**. [S.l.]: Pearson Education, Inc, 2011.

MCDOWELL, C. et al. The effects of pair-programming on performance in an introductory programming course. **SIGCSE Bull.**, ACM, New York, NY, USA, v. 34, n. 1, p. 38–42, fev. 2002. ISSN 0097-8418.

MENS, T.; TOURWE, T. A survey of software refactoring. **IEEE Transactions on Software Engineering**, v. 30, n. 2, p. 126–139, Feb 2004. ISSN 0098-5589.

MOREIRA, M. A. **Teoria de Aprendizagem**. [S.l.]: EPU, 1999.

MULLER, M. M.; HOFER, A. The effect of experience on the test-driven development process. **Springer Science + Business Media**, 2007.

NAWAHDAH, M.; TAJI, D.; INOUE, T. Collaboration leads to success. **IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)**, 2015.

NEVES, G. S.; VILAIN, P. Test logic reuse through unit test patterns a test automation framework for software product lines. In: **Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference**. [S.l.: s.n.], 2014. p. 28–35.

NOSEK, J. T. **The case for collaborative programming**. [S.l.], mar 1998.

OATES, B. J. **Researching Information Systems and Computing**. [S.l.]: SAGE, 2006.

OLIVEIRA, J. et al. Revisiting the refactoring mechanics. **Information and Software Technology**, v. 110, p. 136 – 138, 2019. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584919300461>.

PANCUR, M.; CIGLARIC, M. Impact of test-driven development on productivity, code and tests: A controlled experiment. **Information and Software Technology**, v. 53, n. 6, p. 557 – 573, 2011. ISSN 0950-5849. Special Section: Best papers from the APSEC.

PARSONS, D. et al. Coderetreats: Reflective practice and the game of life. **IEEE Software**, v. 31, n. 4, p. 58–64, July 2014. ISSN 0740-7459.

PASTERNAK, B.; TYSZBEROWICZ, S.; YEHUDAI, A. Genutest: a unit test and mock aspect generation tool. **International Journal on Software Tools for Technology Transfer**, v. 11, n. 4, p. 273, Sep 2009. ISSN 1433-2787.

PIAZZI, P. **Aprendendo Inteligência. Manual de instruções do cérebro para alunos em geral.** [S.l.]: Editora Aleph Ltda, 2008.

PLONKA, L. et al. Collaboration in pair programming: Driving and switching. **Springer Verlag Berlin Heidelberg**, LNBIP 77, p. 43–59, 2011.

PRODANOV, C. C.; FREITAS, E. C. de. **Metodologia do trabalho científico: Métodos e técnicas da Pesquisa e do trabalho acadêmico. 2a Ed.** [S.l.]: Feevale, 2013.

PRZYBYLEK, A.; KOTECKA, D. Making agile retrospectives more awesome. In: **2017 Federated Conference on Computer Science and Information Systems (FedCSIS)**. [S.l.: s.n.], 2017. p. 1211–1216.

ROBERT, M. C.; MICAH, M. **Agile principles, patterns, and practices in C Sharpe**. [S.l.]: Prentice Hall, 2006.

RODRIGUES, P. L. R. et al. Coding dojo as a transforming practice in collaborative learning of programming: An experience report. In: **Proceedings of the 31st Brazilian Symposium on Software Engineering**. New York, NY, USA: ACM, 2017. (SBES'17), p. 348–357. ISBN 978-1-4503-5326-7.

ROOKSBY, J.; HUNT, J.; WANG, X. The theory and practice of randori coding dojos. In: ____. **Agile Processes in Software Engineering and Extreme Programming: 15th International Conference, XP 2014, Rome, Italy, May 26-30, 2014. Proceedings**. Cham: Springer International Publishing, 2014. p. 251–259.

RUNESON, P. A survey of unit testing practices. **IEEE Software**, v. 23, n. 4, p. 22–29, July 2006. ISSN 0740-7459.

SALOMON, G. **Distributed cognitions. Psychological and educational considerations**. [S.l.]: Cambridge University Press, 1996.

SATO, D. T.; CORBUCCI, H.; BRAVO, M. V. Coding dojo: An environment for learning and sharing agile practices. In: **Agile, 2008. AGILE 08. Conference**. [S.l.: s.n.], 2008. p. 459–464.

SAVIANI, D. **Escola e Democracia**. 4a. ed. [S.l.]: Cortez Editora, 1991.

SPACCO, J.; PUGH, W. Helping students appreciate test driven development (tdd). In: **Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications**. New York, NY, USA: ACM, 2006. (OOPSLA '06), p. 907–913. ISBN 1-59593-491-X.

STOREY, M. A. et al. How social and communication channels shape and challenge a participatory culture in software development. **IEEE TRANSACTIONS ON SOFTWARE ENGINEERING**, 2017.

SULTANIA, A. K. Developing software product and test automation software using agile methodology. In: **Computer, Communication, Control and Information Technology (C3IT), 2015 Third International Conference**. [S.l.: s.n.], 2015. p. 1–4.

VIGOTSKY, L. S. **A construção do Pensamento e da Linguagem**. [S.l.]: Livraria Martins Fontes Editora Ltda, 2001.

WELLINGTON, C. A.; BRIGGS, T. H.; GIRARD, C. D. Experiences using automated tests and test driven development in computer science i. In: **Agile Conference (AGILE), 2007**. [S.l.: s.n.], 2007. p. 106–112.

WEREWKA, J.; SPEICHOWICS, A. Enterprise architeture approach to scrum process, sprint retrospective example. In: **Federated Conference on Computer Science and Information Systems**. [S.l.: s.n.], 2017.

WILLIAMS, L. et al. Eleven guidelines for implementing pair programming in the classroom. In: **Agile, 2008. AGILE '08. Conference**. [S.l.: s.n.], 2008. p. 445–452.

WILLIAMS, L.; UPCHURCH, R. L. In suport of student pair-programming. **ACM**, 2001.

WILLIAMS, L. A.; KESSLER, R. R. Experiments with industry's pair-programming model in the computer science classroom. **Computer Science Education**, Routledge, v. 11, n. 1, p. 7–20, 2001. Disponível em: <https://www.tandfonline.com/doi/abs/10.1076/csed.11.1.7.3846>.

ZIERIS, F.; PRECHELT, L. On knowledge transfer skill in pair programming. In: **Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement**. New York, NY, USA: ACM, 2014. (ESEM '14), p. 11:1–11:10. ISBN 978-1-4503-2774-9.

ZUILL, W. **Mob Programming. A Whole Team Approach**. [S.l.], 2014.

APÊNDICE A – QUESTIONÁRIOS

A.1 PRÉ-QUESTIONÁRIO

Este foi o primeiro questionário desenvolvido para a pesquisa. Ele foi encaminhado para uma amostra de respondentes com o objetivo de captar desses suas opiniões e compreensão sobre cada pergunta. As respostas obtidas nele foram utilizadas para o seu aprimoramento.

Perguntas factuais.

Q1 - Qual seu nome?

Q2 - Qual sua idade?

Q3 - Qual é o seu sexo?

Q4 - Qual sua formação? (Cursos técnicos, graduação, especialização e/ou pós-graduação)

Q5 - Marque uma ou mais características que melhores te representam:

- Comunicativo.
- Tímido
- Prefere fazer coisas sozinho.
- Prefere fazer coisas em grupo.

Q6 - Trabalha ou já trabalhou com desenvolvimento de software? Quanto tempo?

Q7 - Caso tenha respondido afirmativamente à pergunta anterior, qual sua experiência em programação de software?

- Iniciante.
- Básico.
- Intermediário.
- Avançado.

Q8 - Quais linguagens de programação você utiliza? Marque da mais importante para a menos importante.

Q9 - Você acredita que a troca de experiências com outros programadores no desenvolvimento de software pode enriquecer seu código? Justifique sua resposta.

Q10 - Com que frequência você busca ajuda em redes sociais, fóruns ou sites de perguntas e respostas para o desenvolvimento de software?

- Nunca.
- Esporadicamente.
- Ocasionalmente.
- Sempre.

Q11 - Em que circunstância você se utiliza ou utilizou de redes sociais, fóruns ou sites de perguntas e respostas para trocar experiências?

- Quando precisa ter novas ideias para seu projeto.
- Quando não sabe por onde começar.
- Quando tem dúvidas sobre determinado trecho do código.
- Quando não consegue avançar no desenvolvimento.
- Outro.

Q12 - Quais desses meios listados abaixo você utiliza para a troca de experiência em desenvolvimento?

- Amigos - Pessoalmente - Telefone - Documentos - Livros - Meetup - Congressos - Campfire - NetBeans - Visual Studio - Eclipse - Skype - E-mail - Podcasts - Wikis - Blogs - Codewall - Yammer - Twitter - Facebook - Youtube - LinkedIn - Google Groups - Google Hangout - Slack - GitHub - Stack Overflow - SouceForge - Trello - Slashdot - Jazz - Masterbranch

Q13 - Você costuma ajudar a outros programadores com dúvidas nas redes sociais, fóruns ou sites de perguntas e respostas? Qual a frequência desse tipo de ajuda?

- Sim - Frequentemente.
- Sim - Ocasionalmente.
- Sim - Raramente.
- Não.

Q14 - Marque os problemas que você acha que o desenvolvedor pode encontrar ao utilizar redes sociais, fóruns ou sites de perguntas e respostas para a troca de experiência.

- Atraso no projeto.

- Distração.
- Perda de privacidade.
- Sobrecarga.
- Nenhum problema.

Q15 - Quais desses eventos sobre troca de experiência em programação você já participou?

- Coderetreats - *Coding Dojo* - Hackathons - Maratona de Programação - Olimpíada de Informática - Nenhum

Q16 - Em sua opinião, qual dessas é a melhor maneira de adquirir conhecimento?

- Coderetreat - *Coding Dojo* - Conferências para Desenvolvedores de Software - Encontros pessoais - Grupos de discussão - Hackathon - Livros - Redes sociais - Meetups - Fóruns

A.2 QUESTIONÁRIO APLICADO AOS *CODING DOJOS*

Com algumas poucas variações nas perguntas, por exemplo o perfil do público respondente e o *Coding Dojo* realizado, este questionário foi aplicado em quatro encontros distintos conforme descrito no Capítulo 6, Seção 6.2.

Perguntas factuais. Questionário de Perfil.

QPF01 - Qual sua idade?

QPF02 - sexo

QPF03 - Você é estudante? Qual curso e instituição que você estuda?

QPF04 - Caso não seja estudante, qual sua ocupação?

QPF05 - Já tem experiência com desenvolvimento de software? marque uma ou mais alternativa

( )Sim, como Hobby.

( )Sim, em Empresa.

( )Sim, na academia (Faculdade, ensino técnico, cursos).

( )Não.

QPF06 - Assinale o item que mais se aplicam ao seu grau de experiência em programação:

( )Iniciante.

( )Básico.

( )Intermediário.

( )Avançado.

( )Ainda não tenho experiência.

QPF07a - Já utilizou ou utiliza alguma ferramenta para desenvolvimento de jogos? Qual? Marque uma ou mais). Observação: Questão específica para o *Coding Dojo Unreal Engine*.

( )Unreal Engine 4.

( )Unity.

( )Construct.

( )3D Game Builder.

( )Blitz3D.

( )GameMaker.

( )RPG Maker.

( )CryENGINE.

( )Nenhum.

( )Outros.

QPF07b - Já utilizou ou utiliza alguma ferramenta para desenvolvimento de jogos? Qual? Marque uma ou mais) Observação: Questão para os demais Coding Dojos.

( )Python.

( )Java.

( )C, C++, C#.

( )PHP.

( )JavaScript.

( )Perl.

( )Ruby.

( )Assembly.

( )Nenhum.

( )Outros.

QPF07c - Já utilizou ou utiliza alguma ferramenta para desenvolvimento de software? Qual? Marque uma ou mais) Observação: Questão para o *Coding Dojo* Elixir.

( )Python.

( )Java.

( )C, C++ ou C#.

( )PHP.

( )JavaScript.

( )Ruby.

( )Erlang ou Elixir.

( )Nenhum.

( )Outros.

QPF08 - Antes deste *Coding Dojo*, já teve alguma experiência com: (Marque uma ou mais)

( )Programação em pares.

( )Desenvolvimento orientado a testes.

( )Métodos ágeis de programação.

( )Refatoração de código.

( )Nenhum.

QPF09 - Você já participou em um (ou mais) dos eventos listados abaixo?

( )Coderetreats.

( )Coding Dojo.

( )Hackathons.

( )Maratona de Programação.

( )Olimpíada de Informática.

( )Meetups sobre alguma linguagem de programação ou ferramentas de programação.

( )Nenhum.

( )Outros.

Perguntas de opinião. Questionário de Percepção dos Coding Dojos

QPC01 - Gostei de participar do Coding Dojo.

QPC02 - Coding Dojo me ajudou a aprender mais sobre programação em geral e testes.

QPC03 - Tive facilidade em entender a dinâmica deste Coding Dojo.

QPC04 - Compreendi o funcionamento da linguagem de programação utilizando Coding Dojo.

QPC05 - Aprendi sobre a linguagem de programação por meio do Coding Dojo.

QPC06 - Meu nível de conhecimento sobre linguagem de programação melhorou significativamente.

QPC07 - O Coding Dojo promoveu momentos de cooperação entre as pessoas que participaram.

QPC08 - Eu me diverti neste Coding Dojo.

QPC09 - Eu recomendaria Coding Dojo para meus conhecidos.

QPC10 - Gostaria de participar de um Coding Dojo novamente.

QPC11 - A experiência com este Coding Dojo vai contribuir para meu desempenho em desenvolvimento de software.

QPC12 - Vou considerar a utilização de práticas ágeis de programação em meus projetos.

QPC13 - Tive facilidade em entender como empregar a refatoração.

QPC14 - Tive facilidade em entender como funciona o desenvolvimento orientado a testes (TDD).

QPC15 - Compreendi como funciona a programação em pares.

QPC16 - Acredito que a programação em pares torna o desenvolvimento mais eficiente.

QPC17 - Coding Dojo está adequado à minha forma de aprender.

A.3 QUESTIONÁRIO APLICADO AO CODERETREAT 2017

Devido ao fato de que um *Coderetreat* ter um formato diferente de um *Coding Dojo*, este questionário teve alguns ajustes para adaptar-se ao evento, sem perder o propósito original.

Perguntas factuais.

Q1 - Qual sua idade?

Q2 - sexo?

Q3 - Qual sua atual ocupação?

Q4 - Já tem experiência com desenvolvimento de software? Marque uma ou mais alternativas.

( )Sim, como Hobby.

( )Sim, em Empresa.

( )Sim, na academia (Faculdade, ensino técnico, cursos).

( )Não.

Q5 - Assinale o item que mais se aplicam ao seu grau de experiência em programação:

( )Iniciante.

( )Básico.

( )Intermediário.

( )Avançado.

( )Ainda não tenho experiência.

Q6 - Já utilizou, ou utiliza, alguma linguagem para desenvolvimento? Qual? (Marque uma ou mais).

( )Python.

( )Java.

( )C, C++, C#.

( )PHP.

( )JavaScript.

( )Perl.

( )Ruby.

( )Assembly.

( )Outros:

( )Nenhum.

Q7 - Antes deste Coderetreat, já teve alguma experiência com: (Marque uma ou mais)?

( )Programação em pares.

( )Desenvolvimento orientado a testes.

( )Métodos ágeis de programação.

( )Refatoração de código.

( )Nenhum.

Q8 - Você já participou em um (ou mais) dos eventos listados abaixo?

( )Coderetreats.

( )Coding Dojo.

( )Hackathons.

( )Maratona de Programação.

( )Olimpíada de Informática.

( )Meetups sobre alguma linguagem de programação ou ferramentas de programação.

( )Outros:

( )Nenhum.

Q9 - Qual desses encontros você usa para aprender programação? Marque uma mais.

( )Coderetreats.

( )Coding Dojo.

( )Hackathons.

( )Maratona de Programação.

( )Olimpíada de Informática.

( )Meetups sobre alguma linguagem de programação ou ferramentas de programação.

( )Conferências para Desenvolvedores de Software.

( )Encontros pessoais.

( )Grupos de discussão.

( )Livros.

( )Redes sociais.

( )Fóruns.

( )Outros.

( )Nenhum.

Q10 - Qual dessas é a melhor maneira de adquirir conhecimento em programação? Marque no máximo 3 itens.

( )Coderetreats.

( )Coding Dojo.

( )Hackathons.

( )Maratona de Programação.

( )Olimpíada de Informática.

( )Meetups sobre alguma linguagem de programação ou ferramentas de programação.

( )Conferências para Desenvolvedores de Software.

( )Encontros pessoais.

( )Grupos de discussão.

( )Livros.

( )Redes sociais.

( )Fóruns.

( )Outros.

( )Nenhum.

Q11 - Com que frequência você busca ajuda em redes sociais, fóruns ou sites de perguntas e respostas para o desenvolvimento de software?

( )Nunca.

( )Esporadicamente.

( )Ocasionalmente.

( )Sempre.

Q12 - Em que circunstância você se utiliza ou utilizou de redes sociais, fóruns ou sites de perguntas e respostas para trocar experiências?

( )Quando está aprendendo uma nova linguagem.

( )Quando precisa ter novas ideias para seu projeto.

( )Quando não sabe por onde começar.

( )Quando tem dúvidas sobre determinado trecho do código.

( )Quando não consegue avançar no desenvolvimento.

( )Outros.

( )Nenhum.

Q13 - Quais desses meios listados abaixo você utiliza para a troca de experiência em desenvolvimento?

( )Pessoalmente.

( )Documentos (livros, etc).

( )Meetup.

( )Congressos.

( )Skype.

( )E-mail.

( )Wikis.

( )Blogs.

( )Twitter.

( )Redes sociais (Facebook, Youtube, LinkedIn, etc).

( )GitHub.

( )Stack Overflow.

Q14 - Você costuma ajudar a outros programadores com dúvidas nas redes sociais, fóruns ou sites de perguntas e respostas? Qual a frequência desse tipo de ajuda?

( )Sim - Frequentemente.

( )Sim - Ocasionalmente.

( )Sim - Raramente.

( )Não.

Q15 - O que você espera aprender neste Coderetreat?

Perguntas de opinião.

Q1 - Gostei de participar do Coderetreat.

Q2 - Coderetreat me ajudou a aprender mais sobre programação em geral.

Q3 - Coderetreat me ajudou a aprender testes.

Q4 - Meu nível de conhecimento sobre a linguagem de programação melhorou significativamente.

Q5 - Pude compreender melhor o funcionamento da linguagem de programação por meio deste Coderetreat.

Q6 - Tive facilidade em entender a dinâmica deste Coderetreat.

Q7 - Tive facilidade em entender como empregar a refatoração.

Q8 - Tive facilidade em usar o desenvolvimento orientado a testes (TDD).

Q9 - Tive facilidade em usar a linguagem de programação.

Q10 - Tive facilidade em compreender como funciona a programação em pares.

Q11 - Acredito que a programação em pares torna o desenvolvimento mais eficiente.

Q12 - Tive facilidades mesmo em face das diversas restrições impostas nas sessões (por exemplo, não falar).

Q13 - Tive facilidade na compreensão do jogo da vida.

Q14 - Tive facilidade em trabalhar com os outros.

Q15 - Eu me diverti neste Coderetreat.

Q16 - Eu recomendaria Coderetreat para meus conhecidos.

Q17 - Gostaria de participar de um Coderetreat novamente.

Q18 - A experiência com este Coderetreat vai contribuir para meu desempenho em desenvolvimento de software.

Q19 - Vou considerar a utilização de práticas ágeis de programação em meus projetos.

Q20 - O Coderetreat promoveu momentos de cooperação entre as pessoas que participaram.

Q21 - Acredito que a troca de experiências com outros programadores no desenvolvimento de software pode enriquecer meu código.

Q22 - Coderetreat está adequado à minha forma de aprender.

A.4 QUESTIONÁRIO DAPROG

Este questionário foi aplicado às duas turmas que participaram do estudo de caso. Com ele procuramos captar as percepções e opiniões dos alunos após o DAPROG.

Q01 - A explicação prévia sobre a linguagem contribuiu para o desenvolvimento do código.

Q02 - A apresentação dos principais comandos sobre a linguagem ajudou no desenvolvimento do código.

Q03 - Tive facilidade em desenvolver o problema.

Q04 - Tive facilidade em entender o problema.

Q05 - O desenvolvimento em dupla contribuiu para entender o código.

Q06 - A divisão da atividade em partes menores me ajudou a resolver o problema todo.

Q07 - O desenvolvimento em duplas me ajudou a compreender a linguagem de programação.

Q08 - O rodízio de duplas me permitiu ver outras formas de desenvolver o código.

Q09 - O rodízio de duplas me ajudou a entender a linguagem.

Q10 - O auxílio de outros colegas durante o rodízio me permitiu entender o funcionamento da linguagem.

Q11 - As discussões em duplas foram relevantes pois, permitiram ver como outras pessoas pensam sobre o mesmo problema.

Q12 - O auxílio do professor durante as atividades foi esclarecedor.

Q13 - Esta metodologia de aula está de acordo com minha forma de aprender.

Q14 - Gostei desta metodologia de ensino.

Q15 - Esta metodologia me ajudou a aprender mais sobre a linguagem de programação.

Q16 - Meu nível de conhecimento nesta linguagem de programação teve boa melhoria.

Q17 - Gostei da dinâmica de rodízios entre os alunos.

Q18 - Acredito que a programação em pares torna o desenvolvimento mais eficiente.

Q19 - Gostei da dinâmica de programação em pares.

APÊNDICE B – RETROSPECTIVAS

B.1 RETROSPECTIVA DO *CODING DOJO* ELIXIR

Pontos Negativos.

1.Tentar ensinar recursão para quem não tem experiência com programação

2.Faltou falar sobre o TDD (dar mais ênfase).

3.Professor podia tornar mais dinâmico, embora, em alguns momentos houve discussões interessantes.

4.Poderia incluir mais a plateia.

5.as funções da linguagem poderiam ir sendo escritas no quadro.

6.Muito lento as vezes.

7.linguagem desconhecida.

8.rápido demais as vezes.

9.difícil entender e participar sem ter visto a linguagem previamente.

10.acredito que forçar uma linguagem de programação nova em um período de 3 horas não leva a um aprendizado significativo, principalmente na primeira semana para alunos do primeiro período.

11.acredito que a linguagem usada não seria útil na formação.

12.muitos não conheciam nada de programação, se fosse feito o dojo em outro tempo o aproveitamento seria melhor.

13.poucos comandos ensinados.

14.atividades em duplas.

15.pouco tempo por aluno.

16.linguagem de programação desconhecida.

17.1 pc para 30 alunos.

18.diferença de experiência entre os participantes.

19.início lento.

20.falta de explicação dos pontos fortes de fracos da linguagem.

21.muito tempo sem por a mão na massa.

22.tempo mal otimizado.

23.desenvolvimento lento.

24.linguagem Elixir é pouco eficiente.

25.repetitivo.

26.desenvolvimento rápido.

27.pouco conhecimento em Elixir.

28.poucas pessoas participaram.

29.exigem muito tempo de aula.

30.cansativo.

31.algumas coisas acontecem sem explicação e fico sem entender.

32.gostaria de uma lista com os códigos para facilitar o acompanhamento sem precisar interromper.

33.confuso.

34.quando alguém não sabe programar tem dificuldade como motorista.

35.após se tomar um trajeto para resolução do problema é mais difícil alterá-lo.

36.faltou tempo para praticar.

37.nem todos participaram.

38.torna a implementação do algoritmo no Elixir de certa forma confusa, já que pode haver diferença na ideia de implementação de duas pessoas.

39.aprendi pouco sobre programação em geral.

40.foi tedioso.

41.como já tenho experiência em programação foi extremamente chato e tedioso.

42.acho também que não houve uma grande eficácia na organização da comunicação pois o público pouco participou.

43.não gostei da maneira que e ensinado porque as vezes não dá pra entender o professor que tem que ensinar devagar.

44.o motorista e o passageiro muitas vezes não explicavam o que iam fazer.

45.as vezes uma dupla ficava muito tempo.

46.faltou tempo para todos participarem.

47.nem todo mundo interage.

48.difícil de entender o que fazer por desconhecer a linguagem, faltou uma introdução melhor do funcionamento dela.

49.somente dois por vez participantes demora mais.

50.embora dinâmico, vai muito devagar.

51.nem todos participaram.

52.nem todos os participantes puderam realmente programar.

53.analisar o código de outra pessoa acredito não ser a melhor forma, o ideal seria analisar o próprio código.

54.devido a falta de conhecimento de muitos alunos em contraponto com a experiência de outros a integração pode ter sido pouco proveitosa para ambos.

55.A quebra da linha de raciocínio repentina do par ao final do tempo, precisando explanar a lógica ao próximo grupo.

Pontos Positivos.

1.a cooperação dos pares ajuda o entendimento e eficiência dos participantes.

2.o evento permite interação de pensamentos e ideias dos alunos.

3.de modo geral, o Coding Dojo foi interativo e contribuiu para a minha primeira experiência com Elixir.

4.ótimo para iniciantes, divertido.

5.ajudou a relembrar um pouco da programação.

6.aprender nova linguagem de programação.

7.compartilhamento de ideias.

8.desenvolvimento no aprendizado.

9.pude entender melhor o ritmo da programção (dinâmico).

10.entendi melhor as etapas da programação.

11.promoveu o espírito de cooperação.

12.programação em dupla melhora a eficiência.

13.Coding Dojo promove a interação entre os participantes.

14.relembrou a lógica de programação.

15.ensinou mais uma linguagem.

16.ensinou a programar em pares.

17.modelo piloto/copiloto é divertido.

18.contrato com uma linguagem diferente de programação.

19.possibilidade de interação entre os participantes para escrever o programa.

20.retirada das dúvidas e discussão deixa mais fluída a atividade.

21.diferentes níveis de conhecimento torna a atividade mais proveitosa.

22.maior interação permite entendimento mais fácil.

23.menor necessidade de saber programação.

24.uma maneira divertida/interativa de todos aprenderem.

25.promove o aprendizado como trabalhar em equipe.

26.acredito que programação em dupla pode deixar a criação do algoritmo mais rápida.

27.é uma iniciativa legal.

28.aprendi o básico de uma nova linguagem de programação, no caso Elixir.

29.dinâmico.

30.introdução a nova ferramenta.

31.gostei dessa linguagem porque parece fácil.

32.aprender na prática sem muita teoria abstrata.

33.aprendi um pouco sobre Elixir e sua sintaxe.

34.integração entre pessoas.

35.aprender uma nova linguagem de programação (Elixir).

36.conhecer a técnica de programação em dupla.

37.lógica.

38.participação dos alunos no exercício.

39.bastante comunicação.

40.ajuda a desenvolver pensamentos lógicos.

41.experiência com programação interativa.

42.aprendi novos comandos.

43.desenvolvi capacidade lógica de raciocínio.

44.aula dinâmica.

45.dinamicidade.

46.simplicidade.

47.aprendizado bom.

48.pensamento coletivo.

49.é didático.

50.visualização dos problemas (erros) na hora.

51.se empregado no tempo certo (todos com noção de programação) se torna útil na aprendizagem.

52.acredito ser ótimo para aprender o básico.

53.cooperação.

54.conhecimento.

55.é engraçado.

56.aprender detalhes e complexidades mais naturalmente.

57.passatempo cerebral.

58.aprendi Elixir.

59.ensina passo a passo.

60.forma dinâmica.

61.interação entre as pessoas.

62.cooperação.

63.explicação passo a passo.

64.desenvolvimento completo de um problema.

65.promoveu *pair programming*.

B.2 PRIMEIRA RETROSPECTIVA DAPROG

Esta retrospectiva é a transcrição das respostas dadas de forma anônima pelos alunos do segundo período de Bacharelado de Sistemas de Informação da UTFPR Campus Curitiba que participaram do DAPROG no dia 12/11/2018

O que aprendemos.

1. Aprendemos sobre Elixir uma linguagem que pode se utilizar para várias aplicações lógicas e matemáticas para por exemplo como foi passado em sala o desconto de produtos dependendo de certa condição que foi proposta
2. O básico de programação funcional em Elixir, variáveis, funções, condições, lógica.
3. Aprendemos um tipo de linguagem diferente da que estamos acostumados, e uma forma de resolver problemas com recursividade.
4. Aprendemos o funcionamento básico da linguagem de Elixir além disso aprendemos mais sobre programação funcional e suas particularidades.
5. Como a linguagem se diferencia das outras linguagens que conhecemos (C, C++, etc), por ser uma linguagem funcional, bem como a forma de programar diferenciada dela.
6. Como a linguagem Elixir funciona, e sua forma funcional que a difere das demais linguagens.
7. A trabalhar em grupos realizando a troca de ideias para achar as soluções básicas da linguagem de programação Elixir.
8. Aprendemos um pouco sobre a linguagem de programação Elixir.
9. O quão mais simples podem ser soluções que seus problemas.
10. Introdução à linguagem Elixir e como resolver problemas simples com a mesma.
11. Um novo tipo de linguagem de programação como também um modo diferente de desenvolvimento em dupla que permitiu uma melhor visão a como realizar a resolução de alguns problemas.
12. Simplificar a resolução do problema usando recursividade e a usar o básico da linguagem Elixir.
13. O desenvolvimento rápido da lógica e soluções para resolver um problema.

14.Uma nova linguagem e o quanto a dinâmica em dupla ajuda o entendimento.

15.Linguagem de programação Elixir. Programação em dupla

16.Aprendi sobre o básico da linguagem de programação Elixir. Aprendi onde o Elixir é interessante de utilizar e que a linguagem compila em cima de Erlang.

17.Um pouco de lógica, sintaxe e prompt a respeito do Elixir.

18.Os comandos básicos de Elixir

19.Comandos básicos de uma nova linguagem.

20.Aprendemos a introdução em Elixir, criar e programar uma função básica.

21.Aprendemos que o trabalho em dupla é importante para o bom entendimento do conteúdo. Além do auxílio providenciando pelo professor. Linguagem Elixir é muito diferente.

22.Uma nova linguagem Elixir.

23.Manuseio de valores tomando condições preestabelecidas como parâmetro, desenvolvendo de pequenas funções em Elixir.

24.Apenas conhecemos a existência de uma nova linguagem de programação.

25.Aprendemos comandos básicos de uma nova linguagem.

26.Sintaxe e funcionalidade básica em Elixir.

27.Aprendemos os conceitos básicos da linguagem Elixir.

28.Aprendemos operações básicas em Elixir.

29.Uma pequena ideia da programação em Elixir.

30.Explicar mais sobre a linguagem Elixir, para que serve e etc.

O que pode melhorar.

1.Acredito que a apresentação das funções que possui na linguagem e a explicação de como funciona a sintaxe que (...) várias vezes eu tive dificuldade para descobrir os erros do código.

2.Adicionar uma introdução sobre a linguagem, principalmente sobre as aplicações dela.

3.A explicação do problema.

4.Explicação dos exercícios.

5.O planejamento de como usar a dinâmica com a turma.

6.A dinâmica em pares podia ser menos movimentada, dando mais tempo para o compartilhamento da informação.

7.Algumas máquinas não tinham o programa indicado instalados e também não houve o rodízio de duplas.

8.Estudar mais a documentação e a sintaxe da linguagem, além de treinar mais.

9.A praticidade na explicação

10.As explicações sobre a linguagem poderiam ser mais completas.

11.Acho que a utilização de um ambiente visual para a realização dos problemas e tarefas mais complexas será interessante para aprender ainda mais e até auxiliar na lógica por trás da programação.

12.Poderia haver uma interação maior entre as duplas (quem fez o código mais rápido, mais eficiente).

13.A apresentação prévia da linguagem e metodologia aplicada.

14.Explicar melhor o objetivo do Elixir, assim como esclarecer o modo que ele funciona.

15.Exercícios seguindo uma escala de aprendizado. Mais exemplos.

16.Explicação mais profunda sobre a estrutura do programa ser algo interessante para melhorar.

17.Material de apresentação das estruturas da linguagem Elixir

18.Atuação do Elixir hoje foi pouco abordado.

19.Não utilizar exercícios tão complicados como o do Harry Potter para iniciantes e comentar melhor sobre alguns erros de sintaxe que ocorre devido ao espaço.

20.Trazer coisas interessantes sobre a linguagem.

21.Rodízio de duplas.

22. Conhecer melhor a estrutura da nova linguagem, ver outros exemplos de utilização de laços e estruturas dentro da nova linguagem antes de se propor a desenvolver um problema.

23. Sem texto.

24. Poderia ter mais tempo e de maneira real existir o rodízio de duplas. O tempo seria para entender melhor a linguagem e compreender melhor o problema.

25. Sem texto.

26. Não houve rodízios

27. Os exercícios propostos poderiam ser mais variados.

28. Descrever todas as introduções básicas antes de apresentar os problemas.

29. O conhecimento da linguagem.

30. Um pouco sobre Elixir.

Pontos Positivos.

1. A apresentação e introdução sobre a linguagem foi bom para conhecer a linguagem e ter uma base de como seria desenvolver uma aplicação nessa linguagem

2. Boa explicação, ensino teórico acompanhando a prática e problemas reais.

3. Conhecimento de uma nova linguagem de programação.

4. Gostei muito da programação pareada.

5. Trabalhar em dupla facilitou, as explicações foram boas.

6. Trabalhar em pares auxiliou muito na minha experiência.

7. Ver os problemas com dificuldade menor e depois os mais difíceis foi bem interessante para o aprendizado. Gostei de trabalhar em dupla também.

8. Linguagem de fácil compreensão e fácil de treinar, programação em dupla.

9. A intuitividade na prática dos novos métodos de programação.

10. Dinâmica em duplas ajudou na resolução dos problemas.

11.Os desafios foram um grande ponto positivo na minha opinião pois possibilitou um desenvolvimento de resolução interessante para a tarefa.

12.Tive um primeiro contato com Elixir e foi a primeira vez que programei em dupla.

13.A dinâmica ajudou em uma melhora de pensamento lógico e rápido.

14.Em dupla, facilidade de uso da linguagem, exercício proposto bem legal.

15.Trabalhar em grupo.

16.Aprender uma nova linguagem, a metodologia aplicada no 1 exercício.

17.Aprendizado e disseminação de uma nova linguagem.

18.Uma nova linguagem de programação com bons fatores positivos. Fácil de aprender. Interação com o terminal.

19.Começamos a aprender uma nova linguagem, junto com suas aplicações.

20.Exemplos de código.

21.Trabalhar em dupla, apresentação de uma nova linguagem, apresentação previa da linguagem, auxílio do professor.

22.Aula dinâmica e prática.

23.Aprender uma nova linguagem, pensamento lógico.

24.Discussão em duplas.

25.Uma nova abordagem do problema, por se tratar de uma atitude em dupla.

26.Trabalho em dupla.

27.O desenvolvimento em duplas e a divisão em partes do exercício possibilitou a atividade.

28.Auxílio do professor.

29.Saber sobre a linguagem.

30.Nova linguagem, conhecimento.

Pontos negativos.

1. Possui muitas dificuldades por não saber a sintaxe e também e porque não tive conhecimento ou tive dupla com outra pessoa creio que isso fez falta pra (...), pois só o meu conhecimento não foi o suficiente e tive muitas dificuldades em aprender
2. Sem texto.
3. Pouco tempo para desenvolver melhor as soluções.
4. Não foi apresentada a documentação da linguagem.
5. Não houve rodízio de duplas.
6. Não vejo nenhum que me chame a atenção.
7. O enunciado do puzzle do Harry Potter estava um pouco confuso.
8. Não teve o rodízio de duplas.
9. Sem texto
10. Professor poderia ter ensinado melhor o uso do terminal e das funções básicas da linguagem.
11. Por falta de conhecimento achei uma linguagem muito estranha e confusa, mas (...) esse não possui um ponto negativo que realmente chame a atenção.
12. A explicação do problema foi um pouco confusa.
13. Não entendi muito a linguagem devido a pouco explicação prévia
14. Falta explicação quanto ao Elixir e seu desenvolvimento.
15. Tempo.
16. Poderia ser um exercício mais simples, já que é o primeiro contato com a linguagem.
17. Nenhum.
18. Pouco perspectiva de uso e atuação.
19. Igual ao 2. (O que pode melhorar).
20. Poucos detalhes.
21. Pouco tempo, falta de rodízio entre duplas.

22.Conteúdo (nova linguagem) atropelada.

23.Explicação (ou falta dela) de alguns operadores.

24.Falta tempo.

25.Muitas vezes se gasta muito tempo discutindo com o par.

26.Falta de tempo.

27.Falta diversidade nos exercícios.

28.Breve explicação.

29.Não muito útil caso eu nunca vá usar a linguagem.

30.Processo da linguagem confuso.

B.3 SEGUNDA RETROSPECTIVA DAPROG

Esta retrospectiva é a transcrição das respostas dadas de forma anônima pelos alunos do Primeiro período de Engenharia da Computação da UTFPR Campus Curitiba que participaram do DAPROG no dia 22/11/2018.

O que aprendemos.

1. Sintaxe básica em Elixir.
2. Elixir, atom, *FizzBuzz*.
3. Comandos e sintaxes básicas da linguagem de programação Elixir.
4. Tivemos uma base de como funciona a linguagem Elixir.
5. Aprendemos a como desenvolver e um pouco da sintaxe da linguagem Elixir.
6. Sintaxe básica de Elixir, desenvolvimento em grupo, uso de Atom.
7. Uma nova forma de aprender programação com dojopuzzle e alguns conceitos básicos da linguagem Elixir.
8. Aprendi o básico da linguagem Elixir.
9. Um pouco mais sobre o CMD do Linux e como programar e executar códigos simples usando Elixir.
10. O que é Elixir e sua sintaxe básica.
11. Aprendemos o básico de Elixir e seus usos.
12. Aprendemos o básico da linguagem Elixir, sintaxe, funcionamento desenvolvimento em grupo.
13. Aprendemos o básico sobre a linguagem Elixir.
14. Aprendemos o básico de Elixir.
15. O básico de programação na linguagem Elixir.
16. Funções básicas da linguagem Elixir.
17. Aprendemos o básico da sintaxe e lógica da linguagem Elixir.

18. Aprendemos uma nova linguagem de programação assim como novos conceitos de programação.
19. Aprendemos a trabalhar em duplas, resolver problemas maiores dividindo-os em problemas menores.
20. Aprendi como trabalhar em equipe e desenvolver soluções em conjunto.
21. Conceitos básicos da linguagem Elixir. Resolução de pequenos problemas. E programação em pares. Dojo programming.
22. Um pouco sobre a linguagem Elixir. Como são os códigos basicos. Sintaxes e lógica.
23. Entregou em branco.

O que pode melhorar.

1. Problemas mais fundamentais e explicação própria dos comandos.
2. Didática usada.
3. Compreender melhor o raciocínio por trás das funções e atribuições da linguagem.
4. A explicação dos comandos usados, pois nem todos sabem usar Linux.
5. O rodízios de duplas e a dinâmica da atividade.
6. Planejamento prévio mais elaborado, dos problemas a serem resolvidos e da aula.
7. Explicar melhor os conceitos básicos da nova linguagem.
8. Uma melhor explicação da funcionalidade e principalmente da sintaxe da linguagem.
9. Maior compreensão do código e da lógica.
10. A explicação sobre os pontos básicos e sobre a utilidade da linguagem.
11. Nada a melhorar.
12. Planejamento prévio mais elaborado.
13. Ter mais ajudantes em sala de aula.
14. Deste jeito esta dinâmica é mais fácil de desenvolver o problema.
15. Planejar melhor a apresentação.

16.Uma introdução melhor e funções únicas de Elixir, como when.

17.Acho importante a sintaxe ser apresentada antes da resolução. Pois o conhecimento prévio da sintaxe aumenta o número de jeitos de se desenvolver o código.

18.As atividades propostas podem ser mais diretas.

19.Uma explicação prévia sobre a linguagem. Uma explicação prévia sobre o Linux.

20.Uma explicação prévia sobre a sintaxe da linguagem utilizada para ter melhor otimização do tempo usado.

21.Não consegui achar grandes problemas, pois não houve tempo o suficiente para se aprofundar na linguagem.

22.Eu achei interessante o jeito do aprendizado e não acho que pudesse ser melhor.

23.Entregou em branco.

Pontos Positivos.

1.Aprendi o que precisava.

2.Auxilio do professor ajuda bastante.

3.Contato com uma linguagem diferente.

4.Aprendi uma linguagem nova.

5.O aprendizado e a dinâmica em equipe

6.Legal aprender linguagens novas e em grupo.

7.Aprender uma nova forma de desenvolver habilidades de programar.

8.Foi bom conhecer uma nova linguagem de programação.

9.A dinâmica em duplas ajudou muito no desenvolvimento do código e da lógica.

10.A programação em duplas.

11.Apresentação de uma nova linguagem de programação.

12.Aprender sobre o uso de uma nova linguagem e ver como programar em dupla é mais eficiente.

13.Aprender uma linguagem nova que pode ser útil em algum momento.

14.Aprendemos uma linguagem nova.

15.Aprender uma nova linguagem.

16.A experiência de pair programming.

17.Introdução à linguagem é interessante.

18.A descoberta de novas técnicas de programação.

19.Foi legal, aprendi formas de trabalhar em dupla.

20.É uma ótima experiência de se ter a programação em pares e a troca de ideias.

21.A linguagem tem uma sintaxe simples que facilita sue entendimento. Isso também deixa o código mais simples de se ver (não é algo tão feio quanto C)

22.a ajuda dos colegas e a demonstração do professor, foi de grande importância, pois quando surgia uma dúvida alguém vinha ajudar.

23.Entregou em branco.

Pontos negativos.

1.Não houve rodízios.

2.Pouco conteúdo e aula podia render mais conhecimento.

3.Falta de tempo fazendo o aprendizado a ser de forma mecânica.

4.Não apresentar sintaxe da linguagem de forma previa.

5.Nenhum.

6.Elixir é uma linguagem muito estranha.

7.Pode parecer um pouco confuso no início.

8.Pouco tempo para aprender uma linguagem nova.

9.Não vejo pontos negativos para esta atividade.

10.A falta de conhecimento próprio por parte dos alunos dificultou um pouco a atividade.

11.Nada a declarar.

12.Acredito que a dinâmica.

13.Poderia aplicar o método de duplas com mais organização.

14.Pouco tempo para o aperfeiçoamento da língua.

15.Aula com dinâmica muito travada.

16.Não vi nenhum ponto negativo.

17.Como dito na pergunta 2, apresentação prévia de ferramentas da linguagem.

18.O pouco tempo.

19.Muito corrido, muita gente precisando de ajuda.

20.A execução foi um tanto quanto difusa.

21.A dinâmica podia ter mais tempo, assim poderiam ter uma experiência mais profunda sobre a programação em pares.

22.Não achei nenhum ponto negativo.

23.Entregou em branco.