CoDiVision: Uma Ferramenta para Apoio na Avaliação de Estudantes no Ensino de Programação.

Francisco Vanderson de Moura Alves[1], **Irvayne Matheus de Sousa Ibiapina**[1], **Werney Ayala Luz Lira**[1], **Pedro de Alcântara dos Santos Neto**[1]

[1]LOST - Laboratory Of Software Technology
DC - Departamento de Computação
CCN - Centro de Ciências da Natureza
UFPI - Universidade Federal do Piauí
Brasil

vanderson.moura.vm@gmail.com, irvaynematheus@yahoo.com

werney.zero@gmail.com, pasn@ufpi.edu.br

Abstract. *In this work is presented the CoDiVision tool, which was developed to infer the contribution level of developers in a software project. This tool can be used to support teachers in students evaluating in disciplines related to programming teaching. Through the use of this tool, instructors can evaluate the students individual contribution in works done in group, which are very common in the programming disciplines, since they allow to exercise several activities related to teamwork, but make the process of individual evaluation more complex. The CoDiVision was used in the evaluation of students in works developed in group, during a discipline designed to teach software development. Results show that the tool generates fundamental information to aid the students evaluation, especially with regard to notes attribution.*

Resumo. *Neste trabalho é apresentada a ferramenta CoDiVision, desenvolvida para inferir o nível de contribuição de desenvolvedores em um projeto de software. Tal ferramenta pode ser usada para apoiar professores a realizarem a avaliação de estudantes em disciplinas ligadas ao ensino de programação. Por meio da utilização dessa ferramenta, os instrutores podem avaliar a contribuição individual de estudantes em trabalhos feitos em grupo, que são bastante comuns em disciplinas de programação, pois permitem exercitar diversas atividades ligadas ao trabalho em equipe, porém, tornam o processo de avaliação individual mais complexo. A CoDiVision foi utilizada na avaliação de estudantes em trabalhos desenvolvidos em grupo, durante uma disciplina visando ensinar o desenvolvimento de software. Os resultados obtidos mostram que a ferramenta gera informações fundamentais para auxiliar na avaliação de estudantes, especialmente no que se refere a atribuição de notas.*

1. Introdução

A Programação é uma subárea da Ciência da Computação e pode ser entendida como o processo de escrita, teste e manutenção de um programa (software) de computador, criado com o intuito de resolver um problema [Van-Roy and Haridi 2004]. O

DOI: 10.5753/cbie.sbie.2017.1567

ensino de programação gera grandes benefícios para a sociedade, auxiliando diversas áreas da ciência, e proporciona também benefícios para alunos dessa área, como por exemplo, o desenvolvimento de habilidades para resolução de problemas gerais [Fessakis et al. 2013]. Em todo o mundo, muitos países já mobilizaram-se para implantar o ensino de programação à estudantes mais jovens, incluindo até mesmo crianças [Kalelioğlu 2015]. O ex-presidente americano Barack Obama, por exemplo, veio a manifestar interesse na inclusão da programação como uma disciplina básica nas escolas dos Estados Unidos.

Cursos de programação de computador são baseados em uma grande atividade prática, via execícios ou tarefas, que comumente abordam problemas maiores e mais representativos de situações reais, para serem resolvidos via trabalho em grupo. Essa abordagem é aplicada em disciplinas de diversas áreas e tem como objetivo repassar aos alunos princípios de trabalho cooperativo, em que os estudantes aprendem que existe a necessidade de ajuda mútua, para garantir que a tarefa seja realizada [Fehr and Schmidt 1999]. Além disso, com o trabalho em equipe, é possível a construção de programas mais complexos em um menor período de tempo. Isso é essencial para a futura vida profissional desses estudantes, uma vez que empresas de desenvolvimento de software costumam operar com grupos de programadores trabalhando em equipe.

Embora o uso de trabalhos em grupo seja fundamental para ampliar o processo de aprendizagem da programação, a avaliação de alunos é mais complicada nesses casos, podendo gerar uma avaliação injusta, uma vez que podem haver grandes discrepâncias nas contribuições de membros de uma equipe. Via de regra, um trabalho em grupo é feito por meio de alterações em um grande número de linhas de código do programa, diariamente. Isso pode ser feito por vários alunos ao mesmo tempo, resultando em uma alta taxa de variação dos programas e dificultando a identificação do que cada membro da equipe contribuiu com qual parte do código [Fritz et al. 2010]. Uma alternativa ao professor para tentar inferir o que foi desenvolvido por cada membro de um grupo é a realização de questionamentos orais, afim de identificar o quão cada aluno contribuiu para o projeto. Porém, tal alternativa é frágil, uma vez que existe muita imprecisão associada ao uso de questionamentos, além do fato dos próprios alunos poderem ter dificuldades em dizer exatamente o que fizeram em um projeto, especialmente em casos de trabalhos maiores.

Diante desse contexto, este trabalho apresenta a ferramenta *CoDiVision*, que auxilia professores no processo de avaliação de estudantes durante disciplinas de programação, especialmente na avaliação individual de alunos em trabalhos realizados em grupo. Ela oferece aos professores, uma maneira automática de identificar o grau de contribuição de alunos a um projeto, assim como partes do programa em que cada aluno mais realizou alterações. Os resultados são apresentados em forma de gráficos gerados automaticamente. A ferramenta utiliza técnicas de Mineração de Repositórios de Software (MRS) na extração de dados a partir de Sistemas de Controle de Versão (SCV), comumente utilizados no desenvolvimento de software, mesmo em ambientes educacionais. A *CoDiVision* foi avaliada em um contexto real, com a análise de projetos desenvolvidos por alunos em um curso de graduação em Ciência da Computação.

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta alguns conceitos sobre sistemas de versionamento e mineração de repositórios de software; a Seção 3 apresenta alguns estudos relacionados ao tema; na Seção 4 são detalhadas as principais

informações sobre a ferramenta *CoDiVision*; na Seção 5 é discutido o uso da ferramenta na avaliação do trabalho realizado por estudantes durante uma disciplina que tinha como objetivo o desenvolvimento de programas de computador. Por fim, a Seção 6 apresenta as conclusões a respeito deste trabalho.

2. Referencial Teórico

2.1. Sistemas de Controle de Versão

Um Sistema de Controle de Versão (SCV) consiste basicamente, em um local para armazenamento de arquivos gerados durante o desenvolvimento de software [Mason 2005]. Esse tipo de sistema é amplamente usado no ambiente industrial e também usado em cursos acadêmicos, uma vez que permitem o compartilhamento, cooperação e facilidade de acesso, além do histórico de versões do programa.

Sistemas de versionamento mantém um registro de todas as operações feitas no projeto, além da identificação da pessoa responsável por cada ação realizada. A cada nova modificação feita sobre um arquivo, ou conjunto de arquivos, é gerado uma nova versão do programa [Spinellis 2005].

2.2. Mineração de Repositórios de Software

A Mineração de Repositórios de Software (MRS) é um campo de pesquisa que analisa dados históricos disponíveis em repositórios de software, tais como sistemas de controle de versão e arquivos de listas de discussão (como *e-mail*, por exemplo), para descobrir informações úteis sobre projetos e programas de computador [Hassan 2008].

As etapas típicas de um processo de MSR são [Hemmati et al. 2013]: I) Extração, II) Modelagem de Dados, III) Síntese e IV) Análise. Primeiramente é realizada a Extração de dados "brutos" de vários tipos de repositórios. Na etapa de Modelagem é realizado uma preparação de dados para serem usados. Na etapa de Síntese é realizado um processamento dos dados extraídos. Finalmente, é necessária a etapa de Análise e interpretação dos resultados para formulação de conclusões.

3. Trabalhos Relacionados

Existem algumas ferramentas para o apoio no ensino de programação [Marcolino and Barbosa 2015]. Algumas dessas ferramentas têm o foco na avaliação de estudantes. Porém, a maioria delas, como as apresentadas em [Alves and Jaques 2014] e [Santos and Ribeiro 2011], focam em avaliar se os programas criados pelos alunos geram a saída correta. Poucas ferramentas auxiliam professores na avaliação de quanto o aluno contribuiu em termos de alterações de código e quais partes do programa foram alteradas por cada aluno, mesmo em trabalhos coletivos.

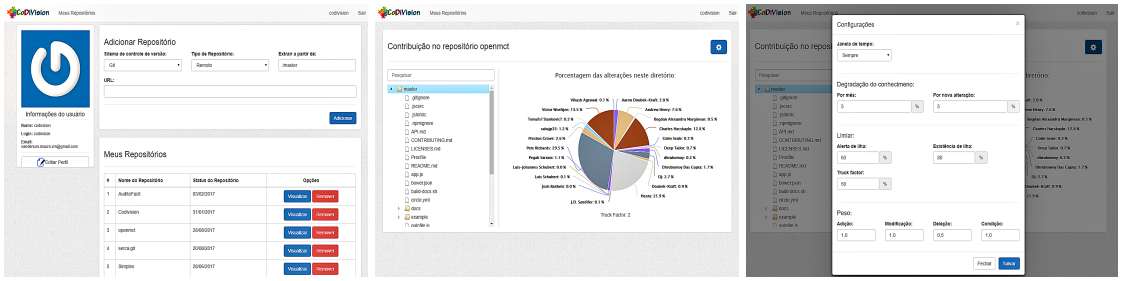
Em [Mittal and Sureka 2014], é proposto uma métrica para definir a colaboração de cada estudante no desenvolvimento de um projeto de software. Esta métrica consiste em analisar todos os componentes que foram adicionados, modificados ou excluídos por cada aluno ao longo do projeto. Os resultados obtidos em um estudo de caso realizado mostram que as métricas e visualizações propostas são úteis para o instrutor da disciplina, pois são capazes de detectar submissões de código redundantes ou desnecessários, e assim, ter uma visibilidade mais completa sobre os projetos em análise.

Em [Alexandre Barbosa 2015], é realizado um Mapeamento Sistemático da Literatura (MSL) para identificar trabalhos que realizam a análise de código em disciplinas de programação, e assim, proporcionar uma visão geral do estado da arte dessa linha de pesquisa. Foram definidas algumas questões de pesquisa para auxiliar na condução do MSL, como por exemplo, quais técnicas são empregadas na análise de código. Os resultados obtidos indicaram que a avaliação automática de código, utilizando teste de software, e a detecção de plágio, comparando similaridade de texto, são as principais técnicas presente na literatura sobre o contexto de analisadores de código para ambientes de ensino e aprendizado de programação.

O trabalho aqui proposto diferencia-se dos demais por utilizar uma ferramenta que realiza todo o processo de extração, análise e apresentação dos resultados de forma automatizada. As métricas utilizadas para definir o nível de colaboração dos alunos durante o desenvolvimento do programa analisam as alterações realizadas em linhas de código, como a adição, remoção e modificação. Com a ferramenta é possível obter informações sobre os participantes de trabalhos de programação, visualizando a contribuição que cada aluno obteve.

4. Ferramenta *CoDiVision*

A ferramenta *CoDiVision*[1] fornece uma visualização da contribuição de desenvolvedores em projeto de software. Neste trabalho a ferramenta foi aplicada em um contexto educacional para apoiar a correção de trabalhos de programação feitos em grupo. A Figura 1 apresenta as principais telas da ferramenta. É necessário realizar um cadastro prévio para ter acesso a todas as funcionalidades disponíveis pela *CoDiVision*. Após o cadastro, o usuário pode extrair dados de projetos a partir de um repositório de código.



(a) Projetos extraídos. (b) Gráficos. (c) Configurações.

Figura 1. Interface da ferramenta.

Os projetos aos quais já tiveram extraídas as informações, podem ser visualizados na tela apresentada na Figura 1(a). Caso o usuário opte por visualizar informações referentes a algum projeto específico, é exibida então a página apresentada na Figura 1(b), onde são apresentados gráficos que representam as contribuições realizadas pelos alunos, tanto em nível de arquivos quanto módulos. Há também a possibilidade de visualizar as submissões (ou *commits*) de código ao repositório, assim como a data em que foram realizadas.

A ferramenta permite também realizar configurações que irão afetar os resultados apresentados. Essas configurações definem o nível de importância que as operações de

[1] http://easii.ufpi.br/codivision/

adição, modificação ou deleção de código irão assumir durante a avaliação. Isso possibilita ao avaliador (professor) atribuir pesos diferentes para cada tipo de operação. Os pesos para cada uma das operações citadas podem ser ajustados na página apresentada na Figura 1(c).

Para fornecer as informações sobre contribuições realizadas por alunos, a ferramenta *CoDiVision* extrai dados de repositórios em SCV's. Esses dados referem-se a todo o histórico de alterações de um programa, incluindo: data da alteração, autor e arquivos alterados. Ao processar os dados extraídos, a ferramenta identifica as operações de adição, modificação e deleção de linhas de código, e assim é contabilizada a contribuição total de cada aluno de acordo com a importância (peso) dada a cada uma das operações realizadas.

5. Avaliação

Esta seção descreve a avaliação inicial realizada com o intuito de mensurar a adequação da *CoDiVision* na avaliação de trabalhos em grupo em disciplinas que possuem ênfase em programação.

5.1. Metodologia

A metodologia utilizada nesta avaliação baseia-se em algumas Questões de Pesquisa (QPs). Elas representam dúvidas que guiaram a avaliação inicial e que representam o ganho de informação que pode ser obtida por tutores/professores a respeito da contribuição feita por estudantes durante o desenvolvimento de programas de computador feitos por grupos de alunos. As questões de pesquisa são descritas a seguir:

- **QP1**: Os alunos que obtiveram maiores notas foram realmente os que mais contribuíram para o projeto?
- **QP2**: Houve uma maior dedicação dos alunos em datas próximas às entregas?
- **QP3**: Quais partes do programa foram alteradas por cada um dos alunos?

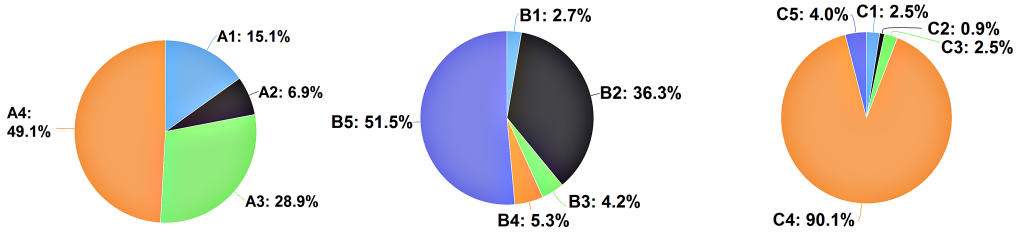
A questão de pesquisa **QP1** foi elaborada com o objetivo de avaliar a relação entre a contribuição feita pelos alunos e suas respectivas notas obtidas nos trabalhos analisados. A questão de pesquisa **QP2**, tem como objetivo verificar se as alterações realizadas pelos estudantes estão bem distribuídas ao longo do ciclo de desenvolvimento do trabalho, ou estão concentradas apenas nos períodos próximos aos prazos para entregas. Por fim, a questão de pesquisa **QP3** tem como objetivo apresentar a distribuição de conhecimento dos alunos acerca do trabalho, bem como a cooperação entre os alunos no que diz respeito as alterações feitas em partes que compõem um programa.

5.2. Resultados e Discussões

Para avaliar a ferramenta desenvolvida foram extraídos dados referentes a três trabalhos (projetos) desenvolvidos por estudantes durante uma disciplina de Engenharia de Software do curso de Ciência da Computação de uma Instituição de Ensino Superior (IES). A seleção da disciplina alvo foi feita porque os alunos usaram um repositório de código fonte, além do fato do professor da disciplina ter consentido com a avaliação e ter informado que auxiliaria na avaliação dos achados feito pela ferramenta.

Cada trabalho visava a construção de uma aplicação *Android* para controle imobiliário. Os projetos geraram, cada um, entre 50 e 70 arquivos. Com isso, realizar a

análise desses projetos, não é uma tarefa trivial para o professor. Isso porque uma única turma pode ser composta por vários grupos e além disso, o professor pode ministrar várias disciplinas com o tema envolvendo programação.



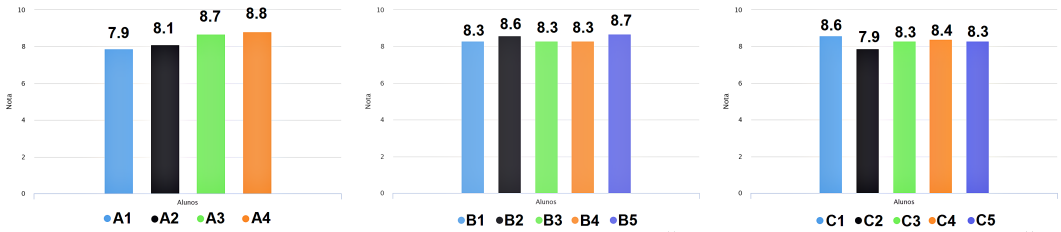
(a) Projeto A. (b) Projeto B. (c) Projeto C.

Figura 2. Porcentagem de alterações realizadas.

A contribuição realizada pelos alunos em cada um dos projetos é apresentada na Figura 2. Os gráficos exibidos foram gerados pela própria ferramenta *CoDiVision*, porém, a identificação dos alunos foi preservada.

- ***QP1*: Os alunos que obtiveram maiores notas foram realmente os que mais contribuíram para o projeto?**

Para responder a esta pergunta foram obtidas as notas finais dos alunos após concluírem a disciplina, com o consentimento do professor responsável. As notas finais de cada um dos alunos são apresentadas na Figura 3.



(a) Projeto A. (b) Projeto B. (c) Projeto C.

Figura 3. Notas finais dos alunos.

Analisando a contribuição dos alunos, apresentada na Figura 2, pode-se perceber que apenas nos projetos A e B os alunos que tiveram a maior nota foram os que mais contribuíram. No $Projeto\ C$, o aluno com a maior nota, $Aluno\ C1$, contribuiu com apenas 2.5% das alterações, todavia o aluno que mais contribuiu, $Aluno\ C4$, obteve a segunda maior nota.

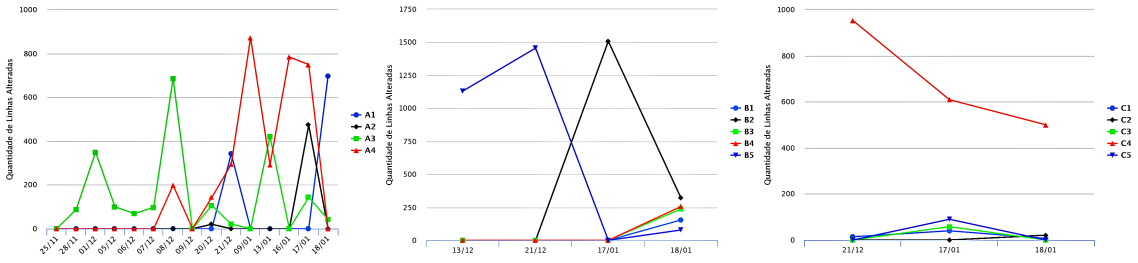
Uma informação bastante preocupante em relação aos trabalhos analisados é a grande concentração da contribuição por apenas um único aluno. No $Projeto\ C$, por exemplo, um único aluno obteve 90.1% das alterações realizadas no trabalho, demonstrando que no projeto houve uma ineficiente distribuição de tarefas, ou até mesmo omissão dos outros alunos do desenvolvimento do projeto.

Não foi identificada uma relação diretamente proporcional entre a contribuição dada pelos alunos e suas respectivas notas, evidenciando a existência de uma grande difi-

culdade em atribuir notas individuais nos trabalhos realizados em grupo, de forma justa, levando em consideração as alterações no projeto.

- ***QP2*: Houve uma maior dedicação dos alunos em datas próximas às entregas?**

Para responder a esta pergunta foram analisadas a quantidade de linhas alteradas e as datas dessas alterações. Essa informação pode ser vista na Figura 4. O desenvolvimento dos projetos ocorreu entre 18/11/2016 e 18/01/2017, com duas entregas. A primeira ocorreu no dia 21/12/2016, data em que foi realizada a apresentação demonstrativa do protótipo funcional da aplicação. A segunda aconteceu no dia 18/01/2016, data em que realizou-se a apresentação final do projeto.



(a) Projeto A. (b) Projeto B. (c) Projeto C.

Figura 4. Histórico de alterações.

Pode-se perceber que nos projetos B e C os alunos realizaram grande parte das alterações apenas em datas próximas às entregas, sendo que essas alterações limitavam-se a um grupo restrito de estudantes. No $Projeto\ A$ houve uma maior participação em todo o período disponibilizado para desenvolvimento do programa. Com essa informação é possível mensurar o comprometimento de cada aluno no projeto, bem como identificar situações que podem trazer riscos para a qualidade do processo de desenvolvimento e do próprio projeto, como por exemplo, o fato de adiar o início da realização do trabalho para uma data próxima à entrega, que é uma prática não recomendada e que pode comprometer parte do aprendizado.

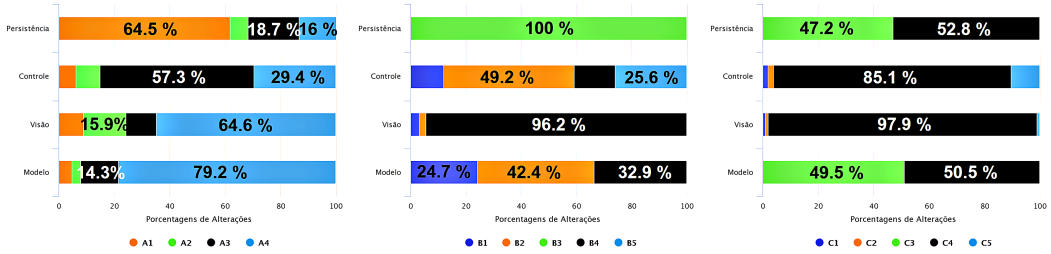
- ***QP3*: Quais partes do programa foram alteradas por cada um dos alunos?**

Os programas desenvolvidos durante os projetos analisados foram criados em linguagem de programação Java e seguem o padrão de arquitetura de software MVC (*Model-View-Controller*) [Sharan 2015]. Esse padrão arquitetural é divido em camadas (modelo, visão e controle) e ajuda a manter, principalmente, a flexibilidade do programa (sistema), tendo em vista que prima pelo desacoplamento. Além das camadas pertencentes ao padrão MVC, muitos projetos adotam ou levam em consideração mais uma camada de sistema: a camada de persistência.

Cada camada existente em um programa de computador pode ser considerada como uma parte específica que realiza uma determinada função. Em um programa que realiza a declaração de imposto de renda, por exemplo, a camada de modelo seria responsável por armazenar o estado atual de cada usuário que está realizando alguma operação no momento. A camada de visão seria a parte do programa que exibe a informação visual para o usuário, solicitando entrada de dados e exibindo resultados. A camada de controle processa as regras de negócio do sistema, como por exemplo o cálculo

do imposto de renda. E por fim, a camada de persistência é responsável por recuperar, atualizar e persistir informações em um banco de dados.

Os programas analisados neste trabalho utilizam o padrão arquitetural MVC e também adotam a utilização da camada de persistência. A Figura 5 apresenta as alterações realizadas pelos alunos em cada uma das camadas citadas anteriormente.



(a) Projeto A. (b) Projeto B. (c) Projeto C.

Figura 5. Alterações realizadas em cada módulo.

Pela análise da Figura 5(a), mesmo havendo um "domínio" na maioria das partes do programa por um único aluno, pode-se perceber que o conhecimento dos alunos está distribuído. No $Projeto\ A$, o $Aluno\ A4$, que possui a maior porcentagem de contribuição, como pode ser visto na Figura 2(a), foi o que mais contribuiu para o modelo e a visão, enquanto que na camada de controle, considerada como o "cérebro" do programa por ser responsável pela regras de negócio, o $Aluno\ A3$ foi o que mais contribuiu com alterações de código, totalizando 57.3%.

Analisando a Figura 5(b), pode-se perceber que que $Aluno\ B3$ foi responsável por 100% das alterações nos arquivos que são responsáveis pelas atividades de persistência, enquanto que o $Aluno\ B4$ realizou 96.2% das alterações na camada de visão. O $Aluno\ B2$ contribuiu aproximadamente com metade das alterações nas camadas de controle e modelo.

Como visto na Figura 2(c), o $Aluno\ C4$ contribuiu com aproximadamente 90% das alterações de código do $Projeto\ C$. Isso fica mais claro pela análise da Figura 5(c), em que pode-se perceber que as camadas de controle e visão tiveram alterações, em praticamente sua totalidade, feitas pelo $Aluno\ C4$, que foi responsável também por quase metade da contribuição realizada nas camadas de persistência e modelo.

Com isso, pode ser observado de maneira geral, que no $Projeto\ A$ o conhecimento dos alunos está distribuído no projeto como um todo, já que cada aluno realizou alterações em cada um das principais partes do sistema. Em termos de aprendizado, essa configuração de contribuições de código parece ser mais adequada, uma vez que os alunos trabalharam em todas as camadas do programa desenvolvido. Em contrapartida, nos projetos B e C houve uma maior especialização do trabalho, já que os alunos concentraram alterações de código em camadas específicas.

5.3. Considerações do Professor

Os resultados obtidos durante a avaliação foram apresentados ao professor responsável pela disciplina. Alguns pontos importantes foram levantados pelo docente. O caso de

alunos com mesma nota e com um percentual de alterações de código diferente foi justificado pelo fato de se tratar de uma disciplina de Engenharia de Software e, outros fatores além da programação, foram levados em consideração durante a avaliação, tais como: a geração de documentos referentes a especificação de requisitos, diagramas que mostram os componentes do programa, plano do projeto, dentre outros. Além disso, a participação dos alunos em fóruns da disciplina também influenciou em suas notas finais. No entanto, o professor afirmou que se caso tivesse acesso a esse tipo de informação no momento da atribuição de notas, certamente teria alterado algumas atribuições realizadas.

O professor ressaltou também que esperava o fato de que alguns alunos fizessem contribuições concentradas em módulos específicos do programa em vez de atuarem de forma generalista. Essa prática foi incentivada pelo professor em alguns projetos visando reduzir conflitos (gerados por alterações simultâneas) em partes do código e potencializar os resultados gerais do desenvolvimento.

Contudo, o professor responsável frisou que a ferramenta traz uma grande contribuição no sentido de monitorar de forma mais precisa a evolução dos projetos das equipes, a partir de um controle granular das alterações realizadas pelos alunos sobre o código do programa. Além disso, destacou que a ferramenta auxilia bastante na avaliação individual dos alunos em trabalhos realizados em grupo, proporcionando avaliar também a cooperação entres os estudantes, no que diz respeito a alterações realizadas sobre as mesmas partes do programa desenvolvido. O professor destacou ainda, que adotaria a ferramenta durante a disciplina em turmas futuras.

6. Conclusão

Neste trabalho foi apresentada a ferramenta *CoDiVision*, que visa auxiliar a contribuição de desenvolvedores em projetos de software. No entanto, essa informação é fundamental em contextos educacionais e podem auxiliar professores no processo de avaliação de estudantes durante disciplinas ligadas ao ensino de programação. Neste contexto, a *CoDiVision* apresentou-se como uma ferramenta para auxiliar professores na análise da contribuição realizada por alunos em trabalhos em grupo.

Foi realizada uma avaliação inicial utilizando três projetos de software desenvolvidos por equipes de um curso de graduação em computação. Foi possível obter uma visão geral dos resultados que podem ser gerados por meio da utilização da ferramenta, durante o processo de avaliação de alunos, mesmo em trabalhos realizados em grupo, em que o processo de avaliação individual se torna mais complexo. Os resultados demonstrados pela ferramenta foram validados pelo professor da disciplina. A *CoDiVision* mostrou-se bastante útil e possibilitou identificar de forma mais precisa as alterações realizadas, a cooperação entre alunos e a contribuição ao projeto como um todo. Essas informações servem de apoio aos instrutores na etapa de avaliação e atribuição de notas aos alunos. O professor responsável pôde ter uma visão das equipes bastante aprofundada e que seria muito útil para apoiar nas avaliações dos alunos e ajudar no decorrer de disciplinas de programação.

Como proposta futura, pretende-se expandir os experimentos em direção a um estudo de caso mais abrangente, utilizando diversas disciplinas ligadas ao ensino de programação, de modo a permitir a utilização da ferramenta CoDiVision, em tempo real, por professores no processo de acompanhamento de turmas. Assim, será possível avaliar

também o impacto da utilização da ferramenta no ponto de vista dos professores.

Referências

Alexandre Barbosa, Allan Correia, D. C. E. C. (2015). Um mapeamento sistemático sobre analisadores de código em disciplinas de programação. *Simpósio Brasileiro de Informática na Educação (SBIE 2015)*, pages 1235–1244.

Alves, F. P. and Jaques, P. (2014). Um ambiente virtual com feedback personalizado para apoio a disciplinas de programação. In *Anais do Simpósio Brasileiro de Informática na Educação*, volume 25, page 1078.

Fehr, E. and Schmidt, K. M. (1999). A theory of fairness, competition, and cooperation. *The quarterly journal of economics*, 114(3):817–868.

Fessakis, G., Gouli, E., and Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, 63:87–97.

Fritz, T., Ou, J., Murphy, G. C., and Murphy-Hill, E. (2010). A degree-of-knowledge model to capture source code familiarity. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 385–394. ACM.

Hassan, A. E. (2008). The road ahead for mining software repositories. In *Frontiers of Software Maintenance, 2008. FoSM 2008.*, pages 48–57. IEEE.

Hemmati, H., Nadi, S., Baysal, O., Kononenko, O., Wang, W., Holmes, R., and Godfrey, M. W. (2013). The msr cookbook: Mining a decade of research. In *10th IEEE Working Conference on Mining Software Repositories (MSR), 2013*, pages 343–352. IEEE.

Kalelioğlu, F. (2015). A new way of teaching programming skills to k-12 students: Code. org. *Computers in Human Behavior*, 52:200–210.

Marcolino, A. and Barbosa, E. F. (2015). Softwares educacionais para o ensino de programação: Um mapeamento sistemático. In *Anais do Simpósio Brasileiro de Informática na Educação*, volume 26, page 190.

Mason, M. (2005). *Pragmatic Version Control Using Subversion*. Pragmatic Bookshelf.

Mittal, M. and Sureka, A. (2014). Process mining software repositories from student projects in an undergraduate software engineering course. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 344–353. ACM.

Santos, J. C. and Ribeiro, A. R. (2011). Jonline: proposta preliminar de um juiz online didático para o ensino de programação. *Simpósio Brasileiro de Informática na Educação (SBIE 2011)*, 22:48.

Sharan, K. (2015). Model-view-controller pattern. In *Learn JavaFX 8*, pages 419–434. Springer.

Spinellis, D. (2005). Version control systems. *Software, IEEE*, 22(5):108–109.

Van-Roy, P. and Haridi, S. (2004). *Concepts, techniques, and models of computer programming*. MIT press.