Identificação de Autoria em Projetos Cooperativos de Programação Orientada a Objetos

David Eduardo Pereira[1], José Robson da Silva Araujo Jr.[1], Mariana Mendes e Silva[1], Matheus Gaudencio[1]

[1] Universidade Federal de Campina Grande (UFCG)
Caixa Postal 10.106 58429-900 -- Campina Grande, PB

{david.pereira, jose.robson.junior, mariana.silva}@ccc.ufcg.edu.br

matheusgr@computacao.ufcg.edu.br

Abstract. *Coding projects as a team is a common task of Object Oriented Programming (OOP) courses. When coding in a group it is important to know who did what. In this work, we propose using strategies to automatically detected ownership of each file on a shared project. We evaluate three ownership detection strategies in an OOP course. Students reported how much they own of each file and we compared those reports against our automatic strategies. Our results show that we were able to detect true ownership in at least 69.9% files of one project. One of our tested strategies were also able to detect different levels of ownership for each file.*

Resumo. *A criação de sistemas de forma cooperativa é uma atividade comum no ensino de programação orientada a objetos (POO) e é importante ter o conhecimento da contribuição de cada aluno ao projeto. Este trabalho propõe o uso de análises automáticas de detecção de autoria que identificam o quanto cada aluno pode ser considerado dono (autor) de um arquivo. Nós avaliamos três técnicas de detecção de autoria no contexto de projetos de POO e as comparamos com o que os alunos reportaram sobre sua produção de código. Nossos resultados apontam que os algoritmos mais básicos concordam em 69,9% a 88,7% das autorias indicadas pelos alunos. Uma das estratégias avaliadas foi capaz de, inclusive, detectar possíveis coautorias dos arquivos.*

1. Introdução

A programação orientada a objetos (POO) é um paradigma de programação importante no desenvolvimento de software e ganhou espaço no mercado devido à sua capacidade de promover alto reuso de código e fácil manutenção. No meio acadêmico, esse paradigma é parte da ementa dos cursos de computação em todo o mundo [ACM, 2013], pois seu uso promove abstrações que permitem pensar em responsabilidades de forma encapsulada.

No curso de Ciência da Computação da Universidade Federal de Campina Grande (UFCG) são feitas atividades de projetos, ainda que de pequeno porte, para facilitar o ensino de POO. Na disciplina de Laboratório de Programação II, essas atividades são desenvolvidas em Java com suporte ao paradigma de orientação a objetos. Tais projetos são desenvolvidos em grupos com intuito de simular o ambiente de trabalho e promover a troca de experiências entre os alunos.

DOI: 10.5753/cbie.sbie.2018.1663

Os projetos de computação têm uma natureza colaborativa e cooperativa. São colaborativos na medida que os alunos podem discutir a estrutura do código proposto, dialogar buscando esclarecer os requisitos a serem implementos, sugerir implementações e abordagens de código para a construção da solução. No entanto, a atividade de codificação, ou implementação, da solução é guiada por um indivíduo. Neste sentido,o projeto é também uma atividade cooperativa, na medida que a soma de cada um dos códigos produzidos serão incorporados à solução final. Enquanto o diálogo e construção da estrutura do código é relevante, é importante entender e identificar a contribuição individual principalmente para entender a capacidade do aluno em traduzir os conceitos de orientação à objetos em código.

Trabalhos realizados em grupo apresentam uma dificuldade em comum, identificar as contribuições de cada membro do time. Um contribuidor é uma pessoa que tem acesso a um projeto e que pode ser responsável por alterações significativas em um arquivo desse projeto, podendo ser também, mas não necessariamente, o criador desse arquivo. No domínio de ensino, é importante para o professor ter conhecimento do que foi estudado e desenvolvido por cada aluno com o intuito de se ter uma melhor visão do processo de aprendizagem do mesmo. Seria ideal que, por exemplo, cada aluno contribua nas diferentes partes que compõem o sistema em desenvolvimento de forma que um determinado conhecimento não fique restrito a um único aluno.

O trabalho em equipe requer também ferramentas próprias para essa atividade. Os sistemas de controle de versão (SCV) são ferramentas amplamente utilizadas no desenvolvimento de software e difundidas no mercado de trabalho para projetos em equipe. São sistemas responsáveis por manter um histórico de alterações dos artefatos, possibilitando o acompanhamento de dados como a identificação do contribuidor e o momento em que as modificações foram feitas. Por serem ferramentas indispensáveis no processo de desenvolvimento, especialmente para trabalhos em grupo, cursos de tecnologia estão incorporando o uso de SCV no ensino de programação [Cochez 2013].

Dentre os SCV disponíveis, o Git é a ferramenta adotada no contexto da disciplina que analisamos, e, dessa forma, foi feito uso de recursos provenientes desse SCV para acompanhar as contribuições dos alunos. O Git, assim como todo SCV, mantém um conjunto de dados relacionados às contribuições dos desenvolvedores de um projeto. Esses dados são importantes para manter um histórico individual das atividades de cada aluno. Entretanto, a visualização dessas informações não é direta o suficiente para que se possa inferir o nível de participação efetiva de cada aluno considerando o projeto final construído.

Com o intuito de melhorar a visão do professor sobre a atuação dos alunos em projetos em equipe, analisamos e avaliamos a autoria do aluno sobre cada artefato de código produzido no projeto. A autoria identifica o contribuidor com maior conhecimento sobre o arquivo, possivelmente considerando as diferentes alterações realizadas neste arquivo ao longo do tempo até o momento da análise. Para suportar nossa análise, nós desenvolvemos o *CodeOwnership* (http://github.com/JRobsonJr/CodeOwnership), uma ferramenta automática de análise de projetos do Git que busca identificar autoria de cada artefato (arquivo) de um projeto. Utilizamos três formas distintas para extrair a informação de autoria, cada uma explorando diferentes níveis de detalhamento sobre as contribuições.

A primeira técnica implementada na nossa ferramenta se baseia puramente na

criação do arquivo para identificar autoria. O aluno que cria a classe é identificado como autor dessa classe. A segunda análise dá a autoria de uma classe ao aluno com mais modificações mantidas até o momento atual do projeto nas linhas de código dessa classe. Por último, fizemos uma análise por porcentagem de linhas de código alteradas e mantidas, que introduz o conceito de coautoria, na qual o contribuidor pode ser dono de uma parte do código analisado dependendo da quantidade de alterações mantidas que foram realizadas por este autor na classe específica.

Para realizar a validação das análises propostas, nós realizamos um estudo de caso com três grupos de quatro alunos da disciplina de Laboratório de Programação II. Estes alunos desenvolveram projetos com 20, 24 e 39 classes cada. Nós executamos nossa ferramenta de detecção de autoria em cada projeto e, em seguida, os alunos foram consultados para identificar a contribuição individual de cada aluno a cada classe desenvolvida nos seus respectivos projetos. Nossas estratégias automáticas mais simples apontam uma concordância na autoria de, pelo menos, 69,9% dos arquivos de um dos projetos. De posse da informação de autoria, esperamos que o professor possa conhecer como os alunos têm desenvolvido seus projetos, identificando em que classes e, por consequência, aspectos do software, cada aluno deu mais destaque.

Este artigo está estruturado da seguinte forma, na seção a seguir iremos discutir os trabalhos relacionados. Na Seção 3, discutimos o conceito de Autoria e como é feita análise dessas métricas para, em seguida, apresentar a nossa validação e avaliação de resultados. Por fim, na Seção 5, discutiremos as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

A estratégia de detecção de autoria não é algo novo em computação, especialmente na engenharia de software. No trabalho de Hattori [Hattori, 2009] duas abordagens são citadas para a detecção de autoria. A primeira abordagem considera que o dono de um arquivo (a pessoa com a autoria) é aquele que tiver mais linhas de códigos escritas e mantidas na versão em que se é feita a avaliação de autoria. Essa técnica é uma das adotadas neste trabalho. A outra abordagem considera que o autor é aquele que fez um maior conjunto de operações de alterações (*commits*) que foram mantidas na versão em análise. Pela natureza simplificada do projeto explorado nas disciplinas de programação, nós não consideramos essa última abordagem como uma das técnicas a serem exploradas.

Trabalhos mais recentes indicam que a autoria baseada em contribuições de código é uma das possíveis estratégias para detecção de autoria, entretanto não é a única. Fritz [Fritz, 2014] sugere que além das contribuições de código, a maneira com que o usuário interage com o código pode indicar o seu grau de familiaridade, conhecimento e posse do mesmo. O nosso trabalho não faz considerações sobre o uso de interações com o código, mas estes são possíveis mecanismos que podem ser futuramente incorporados ao nosso ferramental.

Barros e Menezes [Barros, 2016] desenvolveram o EsCola, um ambiente para escrita colaborativa de texto e que apresenta o histórico de colaborações do estudante. Os conceitos apresentados aqui podem ser adaptados a outros sistemas colaborativos, como o EsCola, na medida de que é possível trabalhar na identificação de autoria de unidades do projeto colaborativo (no exemplo do EsCola, identificar autores de elementos textuais como seções ou parágrafos). Para tanto, é necessário que a

ferramenta forneça os dados indicados como entrada de cada uma das diferentes possíveis análises.

3. Autoria

Para determinar os níveis de autoria dos alunos sobre os códigos desenvolvidos, analisamos as informações de um sistema de controle de versão. Essas análises consistem em observar o histórico das alterações feitas por um aluno em cada linha de cada classe Java até o momento da análise e, a partir disso, inferir os níveis de autoria de cada arquivo do projeto.

Desenvolvemos um programa de código aberto (aguardando publicação), o *CodeOwnership*, com o objetivo de realizar tais análises. A ferramenta utiliza a biblioteca JGit que facilita a obtenção de dados de projetos Git. Ainda, a nossa ferramenta é responsável pelo tratamento de dados e extração das métricas para realizar a análise.

Quando o aluno submete qualquer alteração de artefatos ao Git, são gerados arquivos de registros que guardam informações detalhadas sobre estas alterações. Algumas dessas informações são: i) data de envio; ii) linhas alteradas, excluídas ou adicionadas; iii) dados do usuário (nome, email), e; iv) a mensagem que o usuário atribuiu à sua contribuição. Os registros são a base de dados que utilizamos como entrada para definir a autoria de cada classe do projeto.

Para realizar a detecção de autoria foram observadas as alterações feitas em cada linha de cada classe do projeto até o momento da análise (término do projeto). Essas alterações são tratadas pelo algoritmo que faz a definição do nível de autoria do aluno sobre uma classe. Foram usadas três formas de análises, e cada uma apresenta resultados diferentes e aborda estratégias distintas de detecção de autoria.

A descrição de funcionamento da nossa ferramenta está representada no Algoritmo 1. O programa inicia recebendo um repositório Git. Um repositório é a base de dados que armazena o projeto com seus artefatos e registros. A partir desse repositório, são identificados os contribuidores deste repositório, bem como todas as classes existentes no projeto. Para cada classe, e cada autor, o sistema detecta a autoria utilizando as informações das alterações realizadas na classe e baseando-se na estratégia adotada. O valor de autoria é a porcentagem de quanto um contribuidor detém de autoria sobre determinada classe.

Algoritmo 1. Estratégia de Análise de um Repositório

Entrada: repositório, estratégia
Saída: valor de autoria de cada classe e autor

```
contribuidores ← extrairContribuidores(repositorio)
classes ← extrairClasses(repositorio)
autorias ← [ ][ ]
para cada classe em classes faça
   alteracoes = extrairAlteracoes(classe)
   para cada contribuidor em contribuidores faça
      autorias[classe][contribuidor] ← analisaAutoria(estrategia, alteracoes, contribuidores,
                                                      contribuidor, classes, classe)
   fim
```

```
fim
retorna autorias
```

Nas subseções a seguir, iremos descrever cada uma das três diferentes estratégias de detecção de autoria abordadas neste trabalho. Cada uma dessas abordagens representa uma implementação da função *analisaAutoria* do Algoritmo 1.

3.1. Análise por Criação

A primeira estratégia adotada baseia-se na análise por criação que considera que cada classe tem autoria de apenas uma única pessoa. Nesta análise, exibida no Algoritmo 2, identificamos como autor de um arquivo aquele desenvolvedor que enviou pela primeira vez essa determinada classe ao repositório (i.e., criou o arquivo). Esta pode ser considerada a abordagem inteligente mais simples para a detecção de autoria pois o aluno que criou o arquivo passa a ser aquele que tem a autoria do arquivo.

Algoritmo 2. Estratégia de Análise por Criação

```
Entrada: alterações, contribuidor, classe
Saída: valor de autoria de um contribuidor sobre uma classe
para cada alteração em alterações faça
   se (alteração.contribuidor = contribuidor) e (alteração.tipo = "criação") então
      retorna 1
   fim
fim
retorna 0
```

Como exemplo, considere que a aluna Mariana criou os arquivos *Carro.java* e *Moto.java*, enquanto Raquel criou os arquivos *Onibus.java* e *Rodovia.java*. Mesmo que Raquel tenha feito alterações significativas nos arquivos *Carro.java* e *Moto.java*, nesta estratégia a autoria destes dois arquivos será 100% de Mariana. Por sua vez, os arquivos *Onibus.java* e *Rodovia.java* terão Raquel como autora.

Para realizar tal análise, não é preciso ter conhecimento sobre todos os contribuidores ou classes do sistema. Esta análise não é muito precisa pois, caso um aluno crie uma classe vazia, ele será considerado o dono mesmo não sendo um contribuidor significativo de seu conteúdo. Entretanto, esta análise pode fornecer informações importantes sobre qual aluno contribuiu inicialmente para cada parte do projeto.

3.2. Análise por Linhas de Código

A segunda forma de análise é feita avaliando a alteração mais recente de cada linha de código de uma classe. Para cada classe, identifica-se, em cada linha, qual o último contribuidor a ter feito uma modificação naquela linha. Isto pode ser uma simples edição, formatação de espaços ou até mesmo a simples criação da linha, o que tiver acontecido mais recentemente.

O autor de uma classe será aquele contribuidor que tiver mais contribuições em números de linhas recentemente alteradas nessa classe. Essa estratégia é apresentada no Algoritmo 3. O resultado desta análise é binário, o contribuidor com mais alterações é aquele que passa a ter autoria exclusiva da classe.

Algoritmo 3. Estratégia de Análise por Linhas de Código

Entrada: alterações, contribuidores, contribuidor, classe
Saída: valor de autoria de um contribuidor sobre uma classe

```
contribuições ← [ ]
para cada contribuidorTmp em contribuidores faça
   contribuições[contribuidorTmp] = 0
fim
para cada linha em classe faça
    alteração ← pegaAlteraçãoMaisRecenteNaLinha(alterações, linha)
    contribuições[alteração.contribuidor] ← contribuições[alteração.contribuidor] + 1
fim
se pegaMaiorContribuidor(contribuições) = contribuidor então
   retorna 1
senão
   retorna 0
fim
```

Considere a classe exibida na Tabela 1 abaixo, na qual cada linha da tabela representa uma linha de código e a primeira coluna representa o contribuidor que mais recentemente fez uma alteração nessa linha. Neste exemplo, Mariana criou a classe inicialmente apenas com o conteúdo da linha 1 e 5. Posteriormente, Raquel adicionou as linhas 2, 3 e 4. Neste exemplo, Raquel tem a autoria da classe pois foi a pessoa que têm mais linhas de código de sua autoria no código final da classe. Se, em seguida Mariana altera a mensagem de "Oi" para "Olá" (linha 3), Mariana passará a ter um maior número de linhas contribuídas na classe (linhas 1, 3 e 5) e passará a ser a autora exclusiva da classe.

Tabela 1. Exemplo de Código com Duas Contribuidoras

Autor	Linha	Código
Mariana	1	public class Exemplo {
Raquel	2	public static void main(String[] args) {
Raquel	3	System.out.println("Oi");
Raquel	4	}
Mariana	5	}

Essa abordagem é potencialmente mais precisa que a análise por criação, pois considera que quem mais recentemente alterou em maior quantidade as linhas do código é que deve ser considerado o don caso, o resultado não atribui aos dois a autoria, o que pode ser problemático.

3.3. Análise de Coautoria

A terceira e última forma de análise é semelhante à segunda, entretanto, ao invés de relacionar autoria apenas com um único aluno, é possível estabelecer o conceito de coautoria de uma classe. Nessa terceira modalidade de análise, os contribuidores podem ser autores de uma percentagem do arquivo de acordo com a quantidade de modificações mais recentemente realizadas.

A estratégia é exibida no Algoritmo 4. Novamente é percorrida cada linha do

arquivo e a cada contribuidor é atribuído o total de linhas alteradas ou adicionadas que foram mantidas na versão final do arquivo. Ao final é dividido o total de linhas de cada contribuidor pelo total de linhas da classe e assim é obtida a porcentagem de autoria de cada contribuidor sobre cada classe.

Algoritmo 4. Estratégia de Análise de Coautoria

Entrada: alterações, contribuidores, contribuidor, classe
Saída: valor de autoria de um contribuidor sobre uma classe

```
contribuições ← [ ]
para cada contribuidorTmp em contribuidores faça
   contribuições[contribuidorTmp] = 0
fim
para cada linha em classe faça
    alteração ← pegaAlteraçãoMaisRecenteNaLinha(alterações, linha)
    contribuições[alteração.contribuidor] ← contribuições[alteração.contribuidor] + 1
fim
retorna contribuições[contribuidor] / pegaQuantidadeDeLinhas(classe)
```

Considerando o exemplo mostrado na Tabela 1, podemos dizer que Mariana é responsável por 40% da autoria da classe de exemplo (responsável por duas das cinco linhas da classe) enquanto Raquel é responsável por 60% da autoria da classe.

4. Avaliação e Resultados

O estudo de caso realizado contou com a participação de três grupos de quatro alunos cada. Todos estavam cursando a disciplina de Laboratório de Programação 2 durante o período letivo de 2017.2. Ao final do desenvolvimento da atividade de projeto proposta na disciplina, os discentes em um estudo voluntário responderam a perguntas sobre as suas concepções acerca da autoria do código desenvolvido.

Os alunos responderam em uma escala entre 1 e 4 o seu nível de autoria de cada classe Java do seu projeto. A descrição da escala está exibida na Tabela 2 onde podemos ver que o aluno foi consultado quanto à "quantidade de código escrito". Nós não perguntamos diretamente sobre autoria, pois esse conceito é passível de diferentes interpretações.

Tabela 2. Escala de Avaliação de Autoria para o Estudo de Caso

Escala	Descrição
4	Escrevi todo ou praticamente todo o código da classe
3	Escrevi uma parte considerável do código da classe
2	Escrevi pouco código da classe
1	Não escrevi nenhum código da classe.

Durante a realização do questionário, os alunos puderam consultar livremente o código. As entrevistas aconteceram após a entrega do projeto e finalização da disciplina, para evitar qualquer eventual viés por receio quanto a avaliação acadêmica da própria disciplina. Ao mesmo tempo, os alunos não relataram terem esquecido o quanto foram responsáveis por cada classe.

4.1. Resultados da Análise por Criação e por Linhas de Código

Como as análises por criação e por linhas de código possuem resultados binários (o aluno é ou não dono da classe), criamos uma equivalência entre as respostas dos questionários e os resultados das análises automáticas. Quando um aluno responde que fez uma parte considerável ou praticamente todo o código, o mesmo pode ser considerado como tendo a autoria de uma classe (valor 1). Observe que isso pode gerar mais de um aluno com total autoria de uma classe, bem como classes sem autoria.

Uma vez feita essa equivalência, foi possível observar a concordância entre o que o aluno diz sobre sua autoria e o que algoritmo indica. Definimos concordância da seguinte forma: se a resposta do aluno em relação a uma classe for 3 ou 4, ele será considerado 100% responsável pelo código da mesma e, se o algoritmo indica que ele é de fato autor da classe (tem autoria de valor 1), então existe concordância; do contrário, não existe concordância entre as respostas. A situação é analogamente aplicada para as alternativas 1 ou 2, com o algoritmo indicando a não autoria para a classe (valor 0).

Em seguida calculamos, para cada análise, e para cada grupo, a proporção de resultados concordantes, ou seja, quando a análise mostrou um resultado igual à resposta do aluno. Os resultados das análises por Criação e por Linhas de código estão na Tabela 3. Podemos ver que os resultados totais de concordância das duas análises foram similares (aproximadamente 78% e 81%) e a forma de organização e divisão de tarefas no grupo parece ter influenciado para este resultado. No caso dos projetos avaliados, quando um determinado aluno cria uma classe é muito provável que esse mesmo aluno dê continuidade ao desenvolvimento da classe com mais participação que os outros integrantes.

Tabela 3. Concordância com os Grupos na Análise de Criação e por Linhas

Grupo	Criação	Por Linhas de Código
1	84,4%	82,3%
2	86,2%	88,7%
3	69,9%	77,6%

Nós investigamos o quanto as duas estratégias automáticas avaliadas concordam entre si para cada um dos grupos, ou seja, qual porcentagem da amostra em que as duas análises apontam a mesma autoria para uma determinada classe. Como resultado, obtivemos os valores de: 86,4%, 87,5% e 66,7% para os grupos 1, 2 e 3. Ao investigar o grupo 3, descobrimos que nossa ferramenta, em alguns casos, não é capaz de detectar o contribuidor original de um arquivo movido de pacote, impactando tanto nos valores apresentados na Tabela 3 e na concordância entre as duas análises.

4.2. Resultado da Análise de Coautoria

Utilizando as análises apresentadas anteriormente, percebemos que os alunos que contribuíram consideravelmente na implementação de uma classe podem não ser considerados autores da mesma. Para cobrir essa possível deficiência das análises binárias, introduzimos o conceito de coautoria, no qual os alunos são considerados autores de uma porcentagem do código de uma mesma classe.

Considere uma classe com quatro contribuidores em que cada contribuidor ajuda

com 26%, 25%, 25%, 24% das linhas de código, respectivamente. Nas estratégias anteriores, o primeiro contribuidor é identificado como dono da autoria da classe mesmo sem ter contribuído com a maior parte da mesma (26% versus 74% de contribuição dos demais membros). Enquanto as estratégias anteriores foram úteis para dar autoria ao criador da classe e identificar o contribuidor com a maior participação, consideramos que é necessário também observar os diferentes níveis possíveis de contribuição.

Executando essa estratégia foi possível identificar, para cada classe, um valor de autoria entre 0 e 1 para cada aluno. Para avaliar o resultado dos algoritmos nós buscamos o valor de correlação, para cada aluno, entre a resposta deste aluno (valor entre 1 e 4) e o valor gerado pela estratégia automática de coautoria. Os doze valores de correlação encontrados estão exibidos na Figura 1. Podemos observar no gráfico uma baixa correlação para um aluno (valor abaixo de 0,25) e a maioria dos alunos com valores acima de 0,75 de correlação.

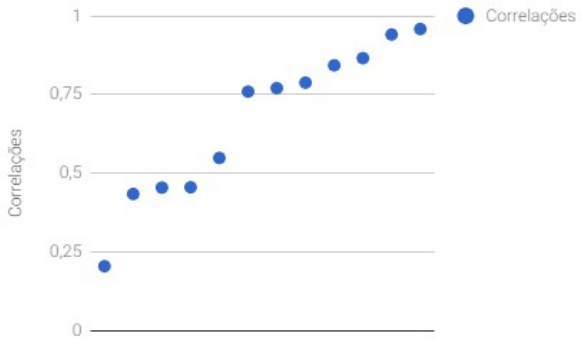

Figura 1. Dispersão das Correlações Encontradas

Ao investigar o aluno com baixo resultado de correlação, identificamos que o algoritmo parece não funcionar de maneira satisfatória quando o próprio aluno reporta baixa participação (“escrevendo pouco” ou “escrevendo nenhum” código) na quase totalidade de classes do sistema. Essa estratégia parece ser mais adequada para membros que tem um nível de participação significativo em pelo menos algumas classes do projeto.

4.3. Limitações da Avaliação

Ainda que os resultados obtidos tenham sido promissores, algumas limitações de nossa avaliação puderam ser observadas. Tais limitações estão listadas abaixo.

Conjunto limitado de dados: o trabalho atual representa um estudo de caso e uma investigação inicial das técnicas automáticas de autoria. Neste trabalho exploramos três técnicas automáticas realizando a avaliação de três projetos de pequeno porte (com quatro alunos cada). As estratégias podem ter comportamentos diferentes para grupos de outros tamanhos ou mesmo outros tipos de linguagem e contexto.

Pair programming: essa é uma prática de programação na qual dois programadores trabalham em conjunto em um único computador. Enquanto um escreve o código, o outro dá suporte. Essa prática é muito comum no contexto da disciplina — nos momentos de aula, pudemos observar os alunos fazendo o uso dessa técnica. Isso implica que a submissão ao Git está atrelada a uma única pessoa, mesmo não sendo essa

a única que participou da concepção do código. A natureza dos SCV exige um único responsável por contribuição. Outras ferramentas de cooperação podem ser planejadas para incorporar a informação de múltiplos responsáveis por uma mesma contribuição.

Reconfiguração de ambiente: Ao efetuar mudanças de classes entre diretórios, o Git pode compreender isso como a criação de novas classes, levando assim a pessoa que fez apenas a mudança de diretório a ter autoria total sobre uma classe. Boa parte dessas alterações são automaticamente detectadas e tratadas, mas ainda é preciso melhorias na ferramenta desenvolvida.

5. Conclusões e Trabalhos Futuros

A detecção de autoria é um mecanismo viável para detecção automática do quanto cada aluno tem de autoria de cada classe. Duas estratégias simples foram testadas neste trabalho e apresentaram bons resultados: a análise baseando-se na criação do arquivo e a que se baseia no total de linhas contribuídas para a versão final de cada arquivo. Uma terceira estratégia foi avaliada também de forma positiva e permite diagnosticar múltiplos autores para um mesmo arquivo.

Como trabalhos futuros, pretendemos usar novas estratégias de detecção de autoria além de realizar o estudo de caso com mais alunos e no contexto de outras disciplinas mais avançadas. Dentre as possíveis estratégias possíveis a serem abordadas temos: i) o uso de conjuntos de contribuições para detectar autoria; ii) o uso de interações com o código (visualização e exclusão) para complementar o mecanismo de detecção de autoria; iii) o uso de outros artefatos (diálogo e diagramas) para a identificação de contribuições, especialmente para contemplar as atividades de programação em pares.

Referências

Association for Computing Machinery (ACM) e IEEE Computer Society (2013), Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, ACM, New York, NY, USA.

Barros, E. e Menezes, C. (2016) EsCola Ambiente Educacional para Escrita Colaborativa. Em: Anais do XXVII Simpósio Brasileiro de Informática na Educação (SBIE 2016), 2016, Uberlândia.

Cochez, M., Isomottonen, V., Tirronen, V. e Itkonen, J. (2013) The Use of Distributed Version Control Systems in Advanced Programming Courses. Em: ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer ICTERI 2013, p.221-235.

Fritz, T., Murphy, G. C., Murphy-Hill, E., Ou, J. e Hill, E. (2014). Degree-of-knowledge: Modeling a developer's knowledge of code. Em: ACM Trans. Softw. Eng. Methodol. 23, 2, Article 14 (April 2014), 42 pages. DOI: http://dx.doi.org/10.1145/2512207

Hattori, L. e Lanza, M. (2009) Mining the history of synchronous changes to refine code ownership. Em: Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR '09). IEEE Computer Society, Washington, DC, USA, 141-150. DOI=http://dx.doi.org/10.1109/MSR.2009.5069492