When Students Become Contributors: Leveraging OSS Contributions in Software Engineering Courses

Clarice Ferreira
Universidade Federal do Pará (UFPA)
Belém, Pará
clarice.ferreira@icen.ufpa.br

Cleice Souza
Instituto Federal do Pará (IFPA)
Belém, Pará
cleice.talitha@gmail.com

Gustavo Pinto
Universidade Federal do Pará (UFPA)
Belém, Pará
gpinto@ufpa.br

Igor Steinmacher
Universidade Tecnológica Federal do Paraná (UTFPR)
Campo Mourão, Paraná
igorfs@uftpr.edu.br

Paulo Meirelles
Universidade Federal de São Paulo (UNIFESP)
São Paulo, São Paulo
paulo.meirelles@unifesp.br

ABSTRACT

Traditional Software Engineering courses commonly prioritize the teaching of methodologies and concepts in small and controlled environments. This decision is partly justified by the difficulty of bringing real software projects to the classroom. The ubiquity of Open Source Software (OSS) projects contributes to mitigating this problem. Several instructors already make use of contribution to OSS as part of the teaching and evaluation process in their courses. However, little is known about how students perceive the approach of contributing to OSS projects in the context of a Software Engineering course. This paper aims to uncover challenges and benefits from the students' perspective. To achieve this, we conducted14 semi-structured interviews with students who attended to this kind of courses in five different Brazilian universities, resulting in findings not so well known. For example, we noticed that, although instructors point to the projects that students are required to contribute to, students (and the project community) are involved in the process of choosing projects and tasks (issues). We also identified that students' contributions vary in terms of number of lines added and removed in commits, as well as the use of different programming languages.

CCS CONCEPTS

• **Software and its engineering → Open source model**;

KEYWORDS

Software Livre, Engenharia de Software, Educação, Comunidades.

ACM Reference Format:
Clarice Ferreira, Cleice Souza, Gustavo Pinto, Igor Steinmacher, and Paulo Meirelles. 2018. When Students Become Contributors: Leveraging OSS Contributions in Software Engineering Courses. In *Proceedings of XXXII Brazilian Symposium on Software Engineering – Education Track, Brazil, 2018 (SBES),* 10 pages.

SBES, 2018, Brazil

ACM ISBN . . . $00.00
https://doi.org/

https://doi.org/

1 INTRODUÇÃO

A disciplina de Engenharia de Software (ES) é uma das mais desafiadores para ensinar e aprender [6, 20]. Existem os cursos tradicionais de engenharia de software, que enfatizam as metodologias e conceitos teóricos, mas não focam em ensinar os alunos a lidar com sistemas de software existentes e complexos [3, 11]. Além disso, a área de desenvolvimento de software requer habilidades além do conhecimento técnico, visto que a construção de um software é resultado de um esforço colaborativo, em que os aspectos sociais também são protagonistas. Ainda, a indústria de software depende de desenvolvedores de software que possam lidar com o código legado, mas tem pouco tempo disponível para treinar mão-de-obra.

Para alinhar-se com o atual panorama da área de desenvolvimento de software, o ensino de ES deve incluir habilidades e atitudes que vão além de conceitos, métodos e técnicas [1, 15]. Isso requer ir além das fronteiras do ensino tradicional, dando mais atenção à complexidade das interações e de como se dá a construção de software colaborativamente, em um ambiente real [18].

Entretanto, não é fácil trazer projetos desenvolvidos na indústria para dentro da sala de aula, ou interagir com empresas no contexto educacional. Uma estratégia comum é, então, conduzir projetos colaborativos, em grupos formados com alunos da própria disciplina – em alguns casos grupos formados por alunos em universidades distintas. Contudo, esses projetos são, em grande parte, *toy projects* desenvolvidos desde o início, e o software produzido, em geral, não é algo que se torne um produto para um usuário final.

A fim de preencher essa lacuna, uma abordagem que tem se tornado cada vez mais comum é promover a participação dos alunos em projetos de software livre [15, 18, 19]. A participação nesse tipo de projeto possibilita que os alunos interajam com sistemas reais, problemas reais e com uma equipe de desenvolvimento interessada em construir software de qualidade. Assim, os estudantes têm a oportunidade de aprender habilidades, atitudes e os desafios inerentes ao desenvolvimento de software no mundo real, o que pode aumentar a confiança dos estudantes quando forem iniciar suas atividades profissionais [4].

Em um trabalho anterior, investigamos a perspectiva dos professores com relação à utilização de projetos de software livre em

cursos de Engenharia de Software[18]. Percebemos que, segundo os professores, os benefícios superam os desafios. Adotar esse tipo de prática melhora não apenas as habilidades técnicas, mas também desenvolve as habilidades sociais dos estudantes. Além disso, os professores ressaltaram que a possibilidade de ter contribuições em projetos de software livre é benéfico por compor um portfólio a ser utilizado em futuras oportunidades de emprego. Com relação aos desafios, os professores relatam que a restrição de tempo da disciplina pode dificultar o engajamento total dos alunos com os projeto, e torna o acompanhamento individual muito raso.

Como uma evolução do nosso trabalhado anterior, buscamos investigar a percepção dos estudantes com relação a esse tipo de prática. Para isso, conduzimos 14 entrevistas com estudantes que tiveram a oportunidade de cursar disciplinas que se beneficiaram da participação em projetos de software livre durante a graduação ou pós-graduação. Neste trabalho, o objetivo é entender a importância, os benefícios e desafios percebidos pelos estudantes deste tipo de abordagem.

Para alcançar o objetivo, as respostas às entrevistas com estudantes foram analisadas qualitativamente. Os indícios obtidos dessa análise resultam em importantes lições que podem servir de inspiração e recomendações para professores que desejem se beneficiar da participação dos alunos em projetos de software livre. Os resultados podem ainda ser de interesse das comunidades de software livre que queiram se beneficiar das contribuições dos estudantes. Os principais achados deste artigo incluem:

1. A escolha do projeto e da tarefa a ser realizada na disciplina pode ser melhor escolhida se feita em colaboração com professor, alunos e as comunidades de software livre;

2. As contribuições feitas pelos alunos variam tanto em termos de complexidade (aferido pela quantidade de adições, deleções e arquivos editados) bem como diversidade (aferido pelo domínio dos projetos, bem como as linguagens de programação utilizadas);

3. A participação do professor da disciplina em atividades extra-classe relacionadas às comunidades de software livre foram percebidas como de grande importância para a condução das disciplinas.

2 MÉTODO

Nesta seção apresentamos as questões de pesquisa e as etapas do processo de realização das entrevistas.

2.1 Questões de Pesquisa (QP)

Definimos cinco questões de pesquisa que guiaram a condução deste estudo:

- **QP1.** Como é feita a escolha de projetos de software livre para os alunos trabalharem?
- **QP2.** Como é feita a escolha da tarefa nos projetos para os alunos trabalharem?
- **QP3.** Qual a natureza das tarefas realizadas pelos alunos?
- **QP4.** Quais são as dificuldades em contribuir para um projeto de software livre no contexto de uma disciplina?
- **QP5.** Quais foram os benefícios em fazer uma disciplina baseada em contribuições para projetos de software livre?

Cada questão de pesquisa objetiva levantar respostas de alunos que cursaram disciplinas com enfoque em projetos de software livre. As perguntas de pesquisa foram, primariamente, respondidas com base nas entrevistas realizadas. Em específico, a resposta à QP3 também baseou-se nas contribuições realizadas pelos alunos nas plataformas de desenvolvimento, além das respostas às entrevistas.

2.2 Entrevistas

A seleção dos participantes da pesquisa foi realizada com base em nosso estudo anterior [18], que analisou a perspectiva de professores que ministram disciplinas utilizando software livre como forma de ensino de Engenharia de Software. A partir dos contatos estabelecidos com os professores naquela pesquisa, solicitamos os contatos dos alunos que participaram dessas disciplinas. Contatamos potenciais participantes via e-mail, de forma que o número de participantes de cada universidade não destoasse. Durante 5 semanas, enviamos 52 convites (aproximadamente 10 convites por semana), dos quais 20 alunos manifestaram interesse na pesquisa; e, ao final, conseguimos 16 entrevistados. As duas primeiras entrevistas foram realizadas em formato piloto, para avaliarmos o roteiro da entrevista e a adequação das perguntas. Após as duas entrevistas-piloto, o roteiro foi revisado e algumas perguntas foram removidas e outras foram adicionadas. Os dois participantes das entrevistas piloto foram removidos e não compõem as entrevistas analisadas. Os dados demográficos dos 14 participantes entrevistados são apresentados na Tabela 1.

Conduzimos entrevistas semi-estruturadas, via teleconferência online, gravadas com a permissão dos participantes, As entrevistas foram conduzidas entre outubro a dezembro de 2017. Para minimizar viés de análises qualitativas, após a transcrição, os áudios foram analisados por três dos autores deste artigo. As entrevistas duraram, em média, 49 minutos (mínimo: 24; máximo: 77). As entrevistas foram baseadas nas questões de pesquisa **QP1–QP5** sendo compostas de cinco partes.

1. Investigamos o perfil dos participantes, particularmente a instituição que estudou e a(s) disciplina(s) que utilizava(m) contribuições em projetos software livre;

2. Buscamos entender a disciplina, incluindo o objetivo e o período em que foi ofertada. Ademais, investigamos a participação do aluno na disciplina, e as formas de avaliação;

3. Colocamos em discussão a escolha do projeto de software livre, incluindo o nível de complexidade, a linguagem de programação utilizada, o conhecimentos dos alunos, como era feita as escolhas de tarefas, os tipos de atividades realizadas no projeto, a interação com a comunidade de software livre, bem como as dificuldades em interagir com a comunidade;

4. Investigamos ainda o processo de contribuição e colaboração com as comunidades. Em particular, questionamos os entrevistados se eles já tinham contribuído com um projeto de software livre antes da disciplina, se conseguiram contribuir durante o curso da disciplina, o número de vezes que o participante contribuiu nos projetos selecionados, se tiveram auxílio para executar as tarefas, as dificuldades em contribuir com um projeto de software livre e, por fim, explicar o impacto em sua motivação ao ter uma contribuição e/ou colaboração aceita;

5. Solicitamos aos entrevistados um relato sobre os benefícios e desafios em fazer uma disciplina que utiliza contribuições em projetos software livre como parte da avaliação, bem como explicar a

importância do professor que participa de comunidades de software livre (fora da sala de aula). Por fim, provocamos os entrevistados a sugerirem melhorias nas disciplinas cursadas.

2.3 Análise das Entrevistas

Analisamos as entrevistas com base nas seguintes etapas:

Codificação inicial dos dados. Nesta etapa atribuímos códigos, ou seja, rótulos que dessem significado aos trechos das entrevistas, que representassem ações e percepções relevantes dos entrevistados. Os códigos iniciais foram considerados provisórios e, em alguns casos, necessitaram de refinamento. Os códigos foram identificados e refinados durante todo o processo.

Categorização dos códigos. Agrupamos os códigos inicias de acordo com características similares em categorias mais genéricas. Esse processo foi realizado durante toda a etapa de análise das entrevistas.

Revisão e definição das categorias. As etapas da análise foram realizadas por dois autores deste artigo, e discutidas em detalhes em reuniões com os demais autores, sendo dois deles também professores dos alunos entrevistados. Durante esse processo, revisamos a categorização, bem como os nomes dados foram refinados para melhor representar os códigos agrupados. As categorias finais relatadas são resultados do consenso obtido após essas reuniões.

Mapeamento das categorias versus quotes. Três autores deste artigo mapearam as categorias identificadas na etapa de "Revisão e definição das categorias" com os trechos das entrevistas (*quotes*), em que partimos das categorias refinadas para os dados mais amplos. Com base nesse mapeamento, identificamos as respostas para a discussão das questões de pesquisa.

2.4 Sumário dos participantes

A Tabela 1 apresenta um sumário demográfico dos alunos que participaram das entrevistas deste estudo. Os 14 entrevistados estão identificados de P1 a P14, com suas idades e seu gênero apresentados nas colunas seguintes. A média de idade era de 27 anos, na data das entrevistas. Dos 14 entrevistados, 4 eram mulheres. As universidades onde eles cursaram as disciplinas estão identificadas de U1 até U5. A coluna nível da disciplina informa se o curso era de graduação ou pós-graduação. Note que o participante P9 cursou duas disciplinas (uma na graduação e outra na pós-graduação), em duas instituições diferentes. Nesse caso, a percepção desse participante foi dividida de acordo com a instituição que fez a disciplina. Em relação à ocupação, 12 dos participantes haviam concluído o curso em questão quando entrevistados, com a maioria deles atuando como Engenheiros de Software profissionalmente. Por fim, cinco participantes tinham experiência com projetos de software livre antes de participarem da disciplina em questão.

3 RESULTADOS

Nesta seção apresentamos os resultados agrupados pelas questões de pesquisa.

3.1 QP1: Como é feita a escolha de projetos de software livre para os alunos trabalharem?

Dentre as estratégias utilizadas para escolher os projetos de software livre mencionadas pelos estudantes, destacamos (i) a liberdade para escolha (6 ocorrências), (ii) indicação do professor (4 ocorrências) e (iii) conhecimento prévio sobre a comunidade de software livre (4 ocorrências). A seguir, apresentamos essas estratégias.

Liberdade para escolha dos projetos. De acordo com 6 alunos, seus professores deram total liberdade na escolha dos projetos. O aluno P1, por exemplo, mencionou que: "*[...] [o professor] deixou livre em relação a qual o projeto em si a gente queria contribuir*". P3 também mencionou que "*o professor deixou livre a escolha*". Ainda, P11 afirmou que "*[...] eles apresentavam os conceitos e deixaram livres [...] pessoas escolhessem o projeto*". Adicionalmente, P8 descreveu que "*a gente poderia escolher qualquer projeto de software livre [...] nós escolhemos um dos projetos que inclusive havia saído de dentro da universidade, mais por questões de proximidade*". Identificamos outros casos em que o professor da disciplina oferecia liberdade parcial para escolha do projeto. Nesses casos, o professor fornecia algumas diretrizes que os alunos deveriam adotar ao procurarem seus projetos. Por exemplo, P5 mencionou que "*o critério para a escolha do projeto seria a simplicidade do projeto, o conhecimento sobre a linguagem e também a afinidade com o projeto*". Em outra instituição e com outro professor, P9 cita essa liberdade na disciplina de pós-graduação que cursou: "*foi mais livre, basicamente a única exigência é que o projeto deveria estar ativo, inclusive podia ser correlacionado ao nosso mestrado*".

Indicação do Professor da disciplina. Em disciplinas de diferentes instituições houve casos em que os projetos foram escolhidos majoritariamente por uma indicação do professor. Quatro alunos relataram que essa indicação foi objetiva, como mencionado por P2: "*a primeira vez, o professor foi lá e escolheu para gente*". No entanto, houve casos em que o professor indicava possíveis projetos para que os alunos fizessem uma escolha final. P5 relatou que "*tinha que escolher um projeto do GNOME ou KDE*". A participação do professor em projetos de software livre também aparece como fator relevante para escolha de um projeto. P9 mencionou que "*o professor faz uma triagem com vários mantenedores de alguns projetos já consolidados, depois ele vê quem tem disponibilidade ou não desses mantenedores de dar um certa atenção aos alunos*". Finalmente, P6 indicou uma forma colaborativa para escolha dos projetos: "*o professor pega a lista de matriculados no início do semestre, e vai distribuindo o pessoal com base na experiência dos alunos com a linguagem de programação do projeto. Caso o aluno não tenha a experiência é colocado como desafio. Ele também pede ajuda para os alunos do laboratório. Todos os alunos dão opinião sobre os projetos, se teve disciplina com o colega ou não, etc. Por fim, o professor verifica o background do aluno. Se, por exemplo, o aluno tem background em C, então ele trabalharia com esse projeto aqui[...]* ". Nesse caso, a escolha do projeto aconteceu utilizando critérios estabelecidos pelo professor ao avaliar o perfil do aluno, que preenche um questionário no início do semestre. Nesse questionário, o aluno indica três opções de projetos, dentre aquelas previamente apresentadas pelo professor em sala de aula. O objetivo era ter equipes equilibradas em termos de conhecimento prévio dos alunos, de forma que o pareamento em sala de aula possa fazer o conhecimento circular entre os integrantes. Os alunos

Tabela 1: Dados demográficos dos participantes.

Participante	Idade	Gênero	Universidade	Nível da Disciplina	Ocupação	Experiência Prévia?
P1	27	Feminino	U2	Graduação	Engenheiro de Software	Não
P2	23	Feminino	U5	Graduação	Engenheiro de Software	Não
P3	25	Feminino	U5	Graduação	Estudante	Não
P4	24	Masculino	U3	Graduação	Professor Substituto	Sim
P5	25	Masculino	U5	Graduação	Estagiário	Sim
P6	26	Masculino	U3	Graduação	Engenheiro de Software	Sim
P7	26	Masculino	U2	Graduação	Engenheiro de Software	Não
P8	24	Masculino	U2	Graduação	Engenheiro de Software	Não
P9	27	Masculino	U3/ U4	Graduação / Pós-Graduação	Estudante	Não
P10	25	Feminino	U3	Graduação	Consultor de desenvolvimento	Não
P11	30	Masculino	U3	Graduação	Estudante	Não
P12	34	Masculino	U1	Pós-Graduação	Analista de Sistemas	Não
P13	33	Masculino	U1	Pós-Graduação	Empresário	Sim
P14	31	Masculino	U2	Graduação	Engenheiro de Software	Sim

com mais conhecimentos e experiência prévia eram indicados como "*coach*" (papel definido no método ágil XP) das equipes.

Conhecimento prévio sobre a comunidade do projeto de software. Identificamos também alguns casos em que os projetos eram escolhidos com base no conhecimento prévio da comunidade responsável pelo projeto em questão. Esse conhecimento prévio pode ser tanto por parte do professor quanto por parte dos alunos. Por exemplo, um caso em que o aluno conhecia a comunidade foi relatado por P14, que mencionou: '*escolhi o projeto no qual já era familiarizado*". Além dele, o participante P1 mencionou que "*já conhecia os desenvolvedores que desenvolveram o jogo. Dessa forma, ficou mais fácil a escolha, pois a gente podia contatá-los diretamente*". Em dois casos, o professor tinha contato com a comunidade, um foi apresentado por P4: "*o professor escolhe quais projetos que a gente tem contato com os desenvolvedores [...] de várias linguagens e tecnologias.*." Outro caso foi relatado por P10: "*o próprio professor seleciona alguns projetos que ele acha interessante, ou que ele tem contato com alguma pessoa*".

Sumário da QP1: Dentre as estratégias adotadas para definição dos projetos, alguns professores deram total ou parcial liberdade para os alunos escolherem o projeto, outros professores faziam indicações objetivas, enquanto que outros professores escolhiam colaborativamente com base em um conhecimento prévio sobre o projeto e a comunidade de software livre em questão.

3.2 QP2: Como é feita a escolha da tarefa nos projetos para os alunos trabalharem?

Após escolher o projeto, o próximo passo dos alunos é a seleção das tarefas a serem executadas nos projetos. P5 chamou atenção para o fato de que, antes de procurar uma tarefa, é preciso "*estudar o projeto*". Em seguida, P5 mencionou que é importante "*pegar uma tarefa periférica, e não tentar pegar uma tarefa da parte principal do projeto*". Ainda, P5 indicou que uma possível tarefa para se familiarizar é "*escolher um bug e tentar reproduzir o bug*".

Como vários dos projetos envolvidos nas disciplinas estão hospedados em plataformas de desenvolvimento colaborativo como Github e Gitlab, P6 mencionou o fato de que existe "*issue tracker que tem uma lista de tarefas e problemas que o software tem*". Com base nas *issues*, P6 comentou que um projeto específico criou um *label* (uma maneira de etiquetar as *issues*) para identificar "*as issues que os alunos deveriam trabalhar*". P10 também indicou uso de *label* em *issues* para auxiliar na escolha da tarefa: "*o projeto tem issues que tinham* labels *que determinavam se era fácil ou tinha alguma tarefa. Os mantenedores desse projeto também auxiliavam na configuração do projeto e orientavam os alunos sobre quais issues eram fáceis. Os alunos determinavam quando iam pegar uma issue maior*".

A busca por tarefas via as *issues* cadastradas no repositório do projeto foi também recorrentemente realizada pelos alunos em diferentes projetos. Por exemplo, P7 ressaltou que "*normalmente, os alunos iam atrás de tarefas no issue tracker*". Como forma de avaliar a viabilidade da implementação da tarefa ao decorrer da disciplina, P7 indicou que "*os alunos tentavam se comunicar com o time de desenvolvimento do projeto para checar se fazia sentido o trabalho previsto, ou se seria algo viável*". De forma similar, P8 relatou que "*procurava por issues abertas [...] pegava alguma issue, que achava que conseguiria resolver e que seria interessante para contribuir para o projeto*".

Por fim, identificamos uma forma colaborativa para a escolha de tarefas, como relatado por P11: "*era uma junção do professor, com o mantenedor do projeto e os alunos. O mantenedor sugeria algumas coisas, os alunos escolhiam o que queriam fazer, e o professor falava se era viável para o escopo da disciplina.*". Adicionalmente, o participante P9 corroborou com a estratégia colaborativa, ao indicar que "*a definição das tarefas era bem democrática, bem métodos ágeis. Os alunos sentavam um dia da semana junto com o professor, abriam as issues e discutiam o que poderia ser feito*". A abordagem colaborativa variava entre os projetos e também entre qual ciclo/etapa a equipe estava. Nos ciclos iniciais, geralmente, o professor participava das reuniões de planejamento (e de revisão ao final do ciclo de 15 dias). O objetivo era assegurar um conjunto de tarefas factíveis de serem realizadas naquele ciclo inicial para que os alunos, assim que possível, tivessem a experiência de terem sua contribuição revisada (para aceite ou rejeição). Portanto, o aluno logo no primeiro mês da disciplina teria a oportunidade de passar por um ciclo completo de interação com a comunidade em questão, o que os fazia

melhor entender a dinâmica da disciplina e se motivarem para as contribuições ao projeto ao longo do semestre.

Sumário da QP2: Umas das formas de iniciar a colaboração em um projeto de software livre é começar com uma tarefa mais fácil, que não faça parte do núcleo do software, e se possível indicada também pela própria comunidade. Procurar *issue* ou *bug* com indicações (geralmente, através de *label* nos repositórios) de tarefas que possam ser resolvidas por novatos é uma estratégia recomendada. A proximidade com a comunidade, participando das suas listas e canais de comunicação, inclusive via os comentários na *issues*, pode facilitar a decisão sobre qual tarefa trabalhar. Durante a disciplina, a intermediação do professor com a comunidade e a participação dele no planejamento, principalmente no início da disciplina, é importante para ajudar na escolha de tarefas que possam ser concluídas no curto prazo, o que motiva os alunos ao longo do semestre.

3.3 QP3: Qual a natureza das tarefas realizadas pelos alunos?

A Tabela 2 apresenta os projetos de software livre que receberam contribuições dos alunos (P#), bem como os dados quantitativos sobre esses projetos: número de contribuições (#C), ou seja, *commits*; número de contribuidores diferentes (#CT); número de linhas de código (LOC), e linguagem de programação (LP) mais utilizada no projeto. O número de linhas de código foram calculadas através do ferramenta `cloc`, que considera linhas em branco e comentários.

Tabela 2: Projetos de software livre que receberam contribuições dos alunos.

P#	Projeto	#C	# CT	# LOC	LP
P1	Catch-The-Pigeon	197	11	7K	Java
P2	JabRef	11K	143	138K	Java
P2	Gnome Music	2K	198	31K	Python
P3	L.Office Impress	1K<	39	18k	Objective-C
P4	Noosfero	15k	126	631K	JavaScript
P6, P9	Mezuro/Prezento	1.7K	35	13K	Ruby
P7	Diaspora	19K	338	151K	Ruby
P8	Amadeus	3K	13	114K	Python
P9	Mezuro/Kalibro	1.2K	12	7K	Ruby
P9	Yosys	3.8k	54	121K	C++
P10	Radar Parlamentar	2k	56	35K	Python
P10	GestorPSI	2k	14	103K	Python
P4, P11	Analizo	1.1k	18	7K	Perl
P13	CakePHP	35k	505	184K	PHP
P14	Liferay-Portal	264k	478	5,262K	Java

Neste trabalho, consideramos uma contribuição um *commit* aceito no repositório oficial do projeto. Como nem todos os alunos tiveram suas contribuições aceitas no repositório oficial, apesar de todos terem as enviado para as comunidades e terem sido avaliadas pelo professor, os participantes P5 e P12 não estão listados na Tabela 2. Por outro lado, há projetos que receberam contribuições de mais de um dos nossos entrevistados, como é o caso do Analizo. Também, há participantes que colaboraram com mais de um projeto, na mesma disciplina, como é o caso de P9, que colaborou com Prezento e o Kalibro , quando estava na graduação; e colaborou com o projeto Yosys enquanto pós-graduando.

Como é possível perceber na Tabela 2, os projetos são diferentes em termos de tamanho e linguagem de programação. O domínio dos projetos variou entre jogos para plataforma móveis, redes sociais, analisadores de código, gerenciador de clínicas, arcabouço de desenvolvimento web, dentre outros. Alguns dos projetos foram criados e são mantidos por contribuidores da própria universidade (e.g., o *Catch-The-Pigeon*); outros são populares e conhecidos pela ampla comunidade de desenvolvimento de software (e.g., o *CakePHP*).

Para definição do tipo de atividade realizada pelos alunos, utilizamos a definição de Hattori e Lanza [9], que categorizam contribuições em código em quatro grandes grupos. **forward engineering** (adicionar novas funcionalidades); **reengineering** (atividades de refatoração); **corrective** (corrigir bugs), e **management** (atualizar documentação).

Para cada participante, analisamos o trecho das entrevistas em que eles explicam suas contribuições. O agrupamento dos tipos de atividades foi apoiado em uma prévia lista de palavras-chave também introduzidos por Hattori e Lanza [9]. Observamos que a maioria das contribuições foram do tipo **forward engineering**; 7 participantes a adição de novas funcionalidades (P1, P3, P4, P8, P9, P10 e P11). Por exemplo, P3 mencionou que o seu grupo fez "*uma contribuição no LibreOffice Impress, pra passar os slides com o celular. Não tinha essa funcionalidade, e a gente colocou*".

Atividades do tipo **corrective** receberam contribuições de seis participantes diferentes (P2, P7, P9, P13 e P14). Por exemplo, P2 observou um bug no "*processo de importação pra banco de dados. Funcionava apenas para o MySQL, mas não funcionava para o Postgres.*". O participante confirmou o bug e implementou a solução. Ademais, apenas três participantes mencionaram contribuições do tipo **reengineering** (P6, P8 e P9). Dentre os exemplos, destaca-se a contribuição do participante P8, que fez uma *"refatoração de alguns layouts que não estavam muito adequados ao padrão Android*".

Por fim, apenas P10 mencionou realizar contribuições do tipo **management**. Este participante mencionou que relatava bugs ou adicionava *labels* em *issues* como parte das contribuições. Importante ressaltar que cinco participantes (P4, P6, P9, P10, P11) utilizaram metodologias ágeis no processo de colaboração para os projetos durante das disciplinas, como destacado por P4: "*A gente fazia todo o processo do Scrum, tinha as sprints, estórias, pontuava as estórias, etc.*". O pareamento era uma estratégia utilizada pelo professor para que o conhecimento circulasse entre os alunos e a escrita de testes automatizados era fundamental, pois, em geral, os projetos de software livre não aceitam contribuições sem seus respectivos testes automatizados.

Investigamos também os *commits* dos entrevistados que apontaram para o repositório *fork* usado na disciplina ou que tiveram suas contribuições aceitas no repositório principal do projeto. A amostra analisada neste estudo foi de 39 contribuições realizadas em 11 projetos de software livre usados nas disciplinas.

Ao analisar a quantidade de adições e deleções dessas contribuições, observamos a presença de vários *outliers*. Por exemplo, uma única contribuição introduziu 2.711 adições e 2.589 deleções em um único commit[1]. Esta contribuição modificou o layout de um jogo, logo foi necessário realizar modificações em 39 arquivos

[1] https://github.com/rodolfoasantos/catch-the-pigeon/commit/51607083

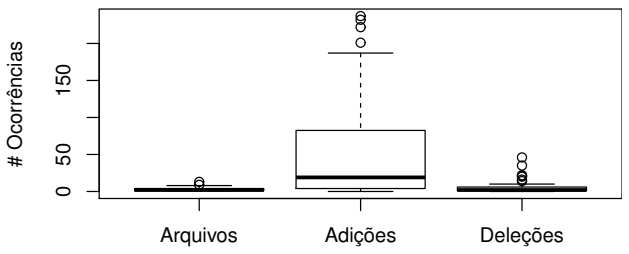

Figura 1: Distribuição das contribuições feitas pelos alunos nos projetos de software livre.

diferentes, possivelmente por se tratar de uma interface de um aplicativo Android modificando também os arquivos XML gerado. Sem considerar os *outliers*, em média, as contribuições modificaram 12,5 arquivos, introduzindo 56 linhas e removendo 7 linhas de código.

A Figura 1 apresenta a distribuição do número de arquivos modificados, adições e remoções realizadas. Por um lado, contribuições pequenas foram também frequentes, ficando de acordo com o apresentado em estudos recentes que sugerem que em projetos de software livre as contribuições tendem a ser pequenas [7, 8, 17]. Na Figura 1, as contribuições do primeiro quartil de contribuições inseriram 4 linhas de código e removeram 1 linha de código. Por exemplo, a contribuição feita no projeto `Mezuro/Prezento`[2] foi de uma única adição: trocou uma imagem em uma página web. Por outro lado, em geral, nossa amostra indica uma média alta de arquivos modificados e de linhas de códigos alteradas por contribuição (*commit*). A recomendação, e sendo algo que evolui conforme a experiência do desenvolvedor, é que se tenha *commits* pequenos e autocontidos (por exemplo, com seus testes automatizados), o que facilita a revisão das novas contribuições por parte dos membros das comunidades dos projetos. Dessa forma, contribuições com um certo volume de modificações sinalizam a possível falta de experiência dos estudantes no processo colaborativo de desenvolvimento de software livre, o que é o esperado nas disciplinas, uma vez que maioria não tinha experiência prévia de colaboração com projetos de software livre. Portanto, este fato corrobora a importância de o aluno passar por este tipo de experiência prática em uma disciplina durante sua formação.

Sumário da QP3: Sete participantes relataram tarefas relacionadas a adição de novas funcionalidades nos projetos de software livre. Já as tarefas que visavam a correção de *bugs* foram citadas por seis dos participantes. Em média, na amostra analisada, cada contribuição introduziu 56 linhas de código em 12 arquivos diferentes, indicando um volume de modificações por contribuição acima do relatado nos trabalhos relacionados, confirmando a falta de experiência dos alunos em como organizar suas contribuições para aumentar as chances de aceite pelos revisores das comunidades de software livre. Ainda, alguns participantes mencionaram a utilização de práticas de metodologias ágeis no processo de desenvolvimento de software durante a disciplina para uma melhor disseminação do conhecimento e colaboração entre os próprios alunos nas equipes formadas nas disciplinas.

[2]https://github.com/mezuro/prezento/pull/114/files

3.4 QP4: Quais são as dificuldades em contribuir para um projeto de software livre no contexto de uma disciplina?

Selecionamos os principais obstáculos para contribuir para os projetos de software livre durante as disciplinas: (i) a estrutura do código; (ii) a interação com a comunidade; (iii) entender e configurar o ambiente de desenvolvimento do projeto, e (iv) a duração da disciplina ser insuficiente.

O estudante P2 relatou que "*[...] a minha dificuldade foi com a estrutura do código [...] Achar um bug em si é difícil[...]. Achar o problema era muito difícil também [...]*". P3 disse " *[...] a gente tinha que analisar o projeto inteiro [...]até você adaptar com esse projeto também é difícil*". Ainda, P4 mencionou que "*[...] sempre tem a dificuldade de entender a arquitetura, o código do projeto*". O entrevistado P5 adicionou que "*a dificuldade que eu tive foi com o código, [...] parecia um monte de cartas embaralhadas, que a gente não conseguia colocar em ordem*". Por fim, P9 cita uma questão de curva de aprendizado por conta da "*[...] complexidade do projeto, por ter que lidar com hardware*".

Entender e configurar o ambiente do projeto também foi um dificuldade relatada pelos entrevistados. De acordo com P2, o obstáculo encontrado foi "*[...]configurar o ambiente*". P1 disse que "*a minha dupla já tinha um conhecimento maior com o ambiente Linux, e eu não*". De forma complementar, P5 relatou que "*[...] o sistema de versionamento utilizado era desconhecido*". Já P7 afirmou que " *[...] configurar o projeto, às vezes pode dar um pouquinho de trabalho*". Concluindo, o participante P9 disse que foi "*[...] a curva de aprendizado da linguagem, aprender o Git, e entender como a comunidade funciona [...] tudo ao mesmo tempo.*"

Essas dificuldades, do ponto de vista técnico, corroboram o quão real foram as experiências dos alunos em lidar com projetos de software livre. Para mitigar este tipo de problema quanto à curva de aprendizado do projeto e código, um dos professores acompanhava mais de perto o projeto e organizava as duplas de forma que quem tinha mais dificuldade tivesse a ajuda de alguém mais confortável com o código do projeto [18]. O professor entrava em contato com mantenedores do projeto para agendar videoconferências para sanar as dúvidas (isso era facilitado, em especial, quando se tratava de projetos nacionais ou com brasileiros envolvidos, mas também pelo contato prévio antes de selecionar o projeto para a disciplina). Também, algumas atividades de estudo de código em grupo, como *coding dojo*, eram organizadas em sala de aula.

De acordo com os relatos, durante o desenvolvimento das atividades, os alunos muitas vezes interagiam diretamente com as comunidades. É algo necessário e positivo para a experiência de colaboração, mas essa comunicação nem sempre é fácil. O estudante P3 relata que "*interagir nas listas de discussões é difícil, por que você não sabe quem é que vai te responder [...]alguém do nosso grupo mandou mensagem na lista de discussão, e um cara foi bem ríspido*". P4, por exemplo, cita a dificuldade em entender "*[...] como é a organização da comunidade em si [...]*". P9 também relatou que a dificuldade estava em "*[...] entender como a comunidade funciona[...]*". Já P11 cita como obstáculo "*[...] conseguir o primeiro contato com alguém da comunidade[...]*", o que impacta em algo enfatizado por P13 com principal dificuldade: "*[...] foi escolher o que fazer*".

Por outro lado, por ser a primeira experiência de muitos em uma projeto real, lidando com desenvolvedores acostumados a colocar software em produção, parte das dificuldades tinham origem no receio em como interagir com a comunidade do projeto. P10 cita que teve "*[...] medo de contribuir [...] dificuldade inicial de entender como funciona, de ter coragem de submeter um pull-request, porque um committer irá avaliar isso, medo de fazer uma besteira e quebrar tudo"*. Já P9 fala em "*[...] não saber o que fazer para o meu nível*". P11 completa: "*[...] acho que começar é mais difícil, perder o medo de mexer no código, mas depois que você quebra esse medo as coisas começam a andar um pouco melhor*". Por isso, o recomendado por professores que passaram por estas experiências em sala de aula [18] inclui encorajar os alunos, passar confiança e, quando necessário, especialmente no início da disciplina, fazer diretamente a comunicação com a comunidade, também estando presente nas listas de discussão e nos canais de comunicação.

Além disso, o atraso na avaliação das contribuições por parte das comunidades impacta nessa interação com os estudantes. P12 cita como dificuldade '*[...] a gente tinha algumas dúvidas sobre como fazer algumas coisas, a gente teve uma certa dificuldade em alguns momentos conseguir obter o retorno num prazo satisfatório*". Ainda, P9 mencionou que fez quatro contribuições para um projeto, no entanto, *"até agora, dois dos meus quatro pull-requests estão abertos [...] eles até hoje não responderam*". P1 também percebeu comportamento similar em trabalhos de colegas: "*alguns colegas tiveram feedback rápido e conseguiram contribuir efetivamente [...] Tiveram outros colegas que submeteram, mas não tiveram a contribuição avaliada durante a disciplina. Até hoje não sei se foi aceita ou não*". Comunidades de software livre tendem a demorar mais para processar contribuições quando o autor não é um integrante ativo da comunidade [17]. Isso também observado pelos participantes deste estudo, e os professores devem ficar atentos a essa demora não desmotivar os alunos. O contato do professor com os membros das comunidades antes do início da disciplina explicando a metodologia em sala de aula e as necessidades dos alunos, é importante para mitigar a falta de retorno nas avaliações das contribuições enviadas [18].

Quando questionados sobre sugestões para as disciplinas que adotam software livre como alvo de contribuições, dado a dificuldade de se ambientar com o projeto, alguns participantes mencionaram que a duração da disciplina foi insuficiente. O participante P4 mencionou que "*[...] essa disciplina poderia, inclusive, ser divida em duas: a primeira, com foco em como se ambientar na comunidade e a segunda, o qual os alunos de fato contribuiriam com o projeto de software livre*". De forma similar, o participante P1 mencionou que a disciplina poderia ser "*[...]dividida em dois semestres*". O participante P11 foi mais extremo e mencionou que "*[...]toda disciplina de graduação poderia utilizar software livre. Sempre que um professor precisar mostrar código aos alunos, o professor poderia mostrar a implementação em um projeto software livre*". Esta dificuldade gerada pela duração da disciplina, uma vez que a contribuição dos alunos para projetos de software livre é uma iniciativa individual de alguns professores em seus cursos, foi citada no estudo que tratou a perspectiva dos professores [18], e é corroborada aqui na visão dos estudantes.

Sumário da QP4: Entender a estrutura do código, interagir com a comunidade, entender e configurar o ambiente do projeto, bem como o tempo insuficiente da disciplina, foram as principais dificuldades que os estudantes ressaltaram nas entrevistas. Em conformidade com os relatos, para atenuar os problemas da interação com a comunidade, o professor indicava os projetos que tinha contato com mantenedores. Para facilitar o entendimento da estrutura do código, segundo relatos de entrevistados, uma estratégia é a escolha de um projeto que o aluno já tenha conhecimento sobre a linguagem de programação usada. De toda forma, as dificuldades relatadas pelos entrevistados serviram para os alunos obterem maturidade na área de desenvolvimento de software. Sob outra perspectiva, se este tipo de iniciativa com projetos de software livre fosse adotada em mais disciplinas, a questão do tempo insuficiente da disciplina para uma experiência ainda mais completa seria mitigada.

3.5 QP5: Quais foram os benefícios em fazer uma disciplina baseada em contribuições para projetos de software livre?

Questionamos os estudantes sobre os benefícios de cursar uma disciplina com ênfase em contribuições para projetos de software livre. Baseado nos relatos, identificamos três categorias de benefícios: (i) participar de um projeto real, (ii) aperfeiçoar o portfólio e (iii) continuar contribuindo após a disciplina.

Os alunos se sentiram motivados e se mostraram satisfeitos por terem a chance de participar de um projeto real. Tal benefício foi mencionado por sete alunos. P2, por exemplo, cita que " *[...] te dá um pouco mais de confiança, que você colabora, e aceita, um negócio real, que tá no ar, qualquer pessoa pode utilizar [...] é o mais próximo da realidade que a gente vai ter, a experiência mais próxima do mundo real*". P4 adicionou que "*[...] é uma grande oportunidade para o aluno. Primeiro, para ter essa parte de contribuir com um software grande que já existe [...] e também dá a oportunidade para ele conhecer que existem projetos de software livre e que não é um bicho-de-sete-cabeças contribuir com eles*". Já P14 citou que o "*[...] conhecimento filosófico, da importância de você contribuir e colaborar com um repositório de software livre*".

Do ponto de vista pessoal em relação a trabalhar com um projeto real, P7 compartilhou que "*[...] você fica bem mais contente de saber que você aprendeu, e além de ter aprendido, você contribuiu um pouquinho com um projeto que está sendo usado*". P12 também relatou que "*[...] você poder ver como é que o ecossistema do projeto sobrevive, como é que se consegue trabalhar com uma rede formada basicamente de desenvolvedores*". Nesse contexto, P9 completa: "*[...] a visão que software não é feito por uma pessoa, mas é feito colaborativamente, por várias pessoas, seja livre ou não*".

Aperfeiçoamento de portfólio foi observado como um benefício por 11 estudantes. De acordo com P8, a experiência na disciplina "*[...] foi o ponto de partida pra ingressar nesse mundo de software livre*". P6 relatou que "*[...] hoje na empresa onde eu trabalho, eu sou referência no versionamento (controle de versão e repositório de código), exatamente por conta dessa experiência[...] um reconhecimento e destaque pessoal, que é chegar em uma empresa e falar que tinha tido contribuições em projetos de software livre na entrevista. Isso é um benefício imensurável*". P10 adicionou que foi uma oportunidade para "*[...] trazer um portfólio, que são contribuições que estão no nosso GitHub, e a gente consegue levar em uma entrevista de emprego, e isso é bem visto[...] então muitas pessoas que gostaram e que contribuíram*

fortemente dentro das disciplinas conseguiram mestrados, empregos em grandes empresas no exterior, e muito por causa do portfólio que a gente vai fazendo enquanto estuda".

Com relação à continuidade nas contribuições para projetos de software livre após a disciplina, oito participantes mencionaram que continuaram a contribuir. O relato mais rico foi feita por P11: "*depois da disciplina eu passei bastante tempo procurando me envolver em algum projeto. Eu comecei a mandar pull request em projetos do GitHub que achava interessante. Até encontrar um projeto específico e contribuo pro projeto já tem um tempo*". P11 completou seu relato destacando: "*eu contribuo com software livre, viajo para conferências, trabalho com software livre. Então, fez bastante diferença na minha vida. A gente resolveu criar uma ferramenta, o Kiskadee [...] a gente colocou essa ferramenta no Google Summer of Code deste ano, debaixo do projeto Fedora (de grande reconhecimento internacional, liderado pela RedHat). Então, eu fui mentor de um aluno [...] E depois que o programa acabou, a gente colocou a ferramenta na disciplina, a mesma que eu fiz no passado [...]*". Além disso, P9 (que não tinha experiência prévia antes da disciplina) mencionou que se tornou contribuidor de projetos de software, incluindo distribuições Linux: "*atualmente, eu estou bem focado no Linux [...] Eu também contribuo com o Debian, só que não é implementação, é com empacotamento; eu mantenho dois pacotes lá, e estou preparando outros três*".

Sumário da QP5: Colaborar e contribuir com o desenvolvimento de software livre geram benefícios imediatos durante a disciplina, e consequências positivas depois dela. Através da experiência com software livre na disciplina, os entrevistados relatam oportunidades profissionais de alto nível, devido ao portfólio de colaboração em projetos de software livre, muitas vezes iniciado na disciplina. Outro benefício, bastante relevante colhido das entrevistas foi o relato da continuação nas contribuições após a disciplina (e ainda depois da conclusão do curso de graduação, por exemplo), uma vez que alguns dos entrevistados tornaram-se desenvolvedores oficiais de projetos de software profissionalmente, com reconhecimento internacional. Com exemplo, um dos entrevistados colabora com *Kernel* do Linux e outro participou como mentor do *Google Summer of Code* por conta de uma ferramenta livre que ele mesmo desenvolveu.

4 IMPLICAÇÕES

Com base em nossos resultados, apresentamos as potenciais implicações para professores, alunos e comunidades de software livre. Discutimos tais implicações a seguir.

Professores de Engenharia de Software: dada a dinâmica entre professor, alunos e comunidades, os participantes mencionaram que o envolvimento do professor em atividades relacionadas aos projetos de software livre é importante para a boa condução da disciplina. Em particular, P3 indicou que esse envolvimento "*era essencial, uma vez que o professor tinha mais experiência. Então, muitas das vezes em que os aluno desanimavam ou não sabiam muito bem o que fazer, o professor conseguia muito bem direcioná-los*". Ademais, P5 indicou que, através de pesquisas acadêmicas conduzidas pelo professor, os alunos já sabiam que tipos de dificuldades enfrentariam:"*o professor estuda formas pra fazer com que as pessoas desistam menos de contribuir para projetos de software livre, e a gente ali teve que enfrentar essas dificuldades*". Além disso, P10 ressaltou o fato de que "*se o professor não está preparado, não tem como essa dinâmica ocorrer. Tem que ser uma pessoa que entenda e seja apaixonado pelo software livre.*". Isso significa que mais professores podem se motivar e fomentar a utilização de projetos de software livre em suas disciplinas, ao conhecerem melhor as dinâmicas de colaboração com as comunidades de software livre, se envolvendo com as comunidades. Tal envolvimento pode facilitar a comunicação entre professores e alunos com as comunidades. Existem várias formas de contornar essa dificuldade, e se envolver em comunidades de projetos de código aberto, como já reportado na literatura [13]. Professores podem começar um envolvimento utilizando como base as dificuldades que os alunos passam em sala de aula (por exemplo, relatando um bug que os alunos encontraram). Além disso, durante nossas entrevistas, alunos relataram a participação de professores assistentes e convidados que possuíam experiência com contribuições em projeto de software livre, auxiliando os alunos e ministrando algumas aulas e palestras relacionadas. Essas abordagens podem ajudar o professor com pouco envolvimento.

Alunos de Engenharia de Software: a experiência prévia com software livre não é requisito obrigatório. Nove dos 14 participantes entrevistados não tinham experiência prévia com projetos de software livre. Ainda assim, de maneira geral, os participantes relataram um bom desempenho nas disciplinas. É importante o aluno ter em mente que em projeto reais, como os de software livre, o uso de diferentes linguagens de programação, inclusive no mesmo projeto, é comum. Ademais, além de linguagens como C, C++, Java, Python e Ruby, observamos a utilização de linguagens de programação não comumente encontradas em currículos de curso de graduação em computação, como Perl, Objective-C e JavaScript. Esses pontos implicam na possibilidade de expandir o aprendizado para além de uma disciplina convencional de Engenharia de Software do ponto de vista da experiência com diferentes linguagens de programação.

Mantenedores de projetos de Software Livre. Uma vez que cada vez mais professores introduzem esse tipo de abordagem em sala de aula, é esperado que outros alunos participem e se envolvam ainda mais com as comunidades. Mantenedores de projetos de software livre poderiam se inspirar em alguns dos achados deste trabalho e, por exemplo, propor *issues* que sejam propícias para serem resolvidas por alunos num curto período de tempo. Mantenedores poderiam também realçar na documentação e nos repositórios dos projetos quais os canais de comunicação (listas, IRC, entre outros) estão disponíveis para sanar dúvidas de alunos e facilitar o seu envolvimento no projeto, estando cientes que, enquanto estudantes, eles estão inseguros e com receios, não sabendo algumas vezes fazer a abordagem correta para interagir com os membros das comunidades. Nesse contexto, alguns entrevistados mencionaram algumas práticas que, se adotadas por comunidades de software livre, poderiam facilitar o processo de inserção de desenvolvedores externos às comunidades, em particular, alunos. Por exemplo, o participante P5 indicou que "*[...] tem a documentação do que o projeto faz, mas não tem uma documentação voltada para o desenvolvedor*". P13 também corroborou com a importância de boa documentação nos projetos: "*a gente estudou por que um projeto faz mais sucesso que outro, e geralmente isso se atribui a uma documentação muito bem feita*". Um outro ponto, mencionado por P5, é "*label nas issues pra identificar se é uma tarefa para iniciante, intermediário, etc.*". De maneira geral,

esse participante foi além e mencionou que o projeto poderia, inclusive, ser divido de forma que pessoas inexperientes possam começar contribuindo com partes para iniciantes. Por fim, quando o projeto for inserido em uma disciplina, os mantenedores devem ajudar a mobilizar a comunidade para ficar em contato com o professor e os alunos, pois, na prática, muitas vezes uma equipe de desenvolvimento formada por estudantes talentosos, por um período contínuo (4 a 5 meses), estará colaborando para o projeto. E, em alguns casos, gerando contribuidores para além da disciplina.

5 TRABALHOS RELACIONADOS

Diversos esforços recentes envolveram alunos em projetos de software livre dentro da sala de aula [2, 5, 14, 19, 21]. Smith e colegas [21] se concentraram em selecionar os projetos de software livre mais apropriados para os alunos. Os autores argumentam que, devido à curta duração de uma disciplina típica de Engenharia de Software (4-5 meses), os professores não deveriam selecionar projetos de software livre grandes ou complexos, porque os alunos teriam dificuldade em contribuir significativamente durante o curso.

Da mesma forma, outros trabalhos sugerem que os projetos de software livre não devem ser muito pequenos ou muito simples, uma vez que os alunos não colocariam em prática muitos princípios de ES. Morgan e Jensen [14] detalharam a experiência de ministrar disciplinas de ES com o apoio de software livre e observaram que o tamanho do projeto (nesse caso, o Ubuntu) acabou sendo um obstáculo significativo para os alunos contribuírem. Da mesma forma, Buchta e seus colegas [2] relataram experiências de ensinar evolução de software em disciplinas de ES. Os autores relataram a necessidade de 60 horas antes da disciplina para selecionar os projetos e configurar o sistema de controle. Neste estudo, no entanto, os autores não discutiram benefícios, implicações ou desafios da utilização desta abordagem em disciplinas de ES.

Além disso, Holmes e colegas [12] relataram que em seu programa de Trabalhos de Conclusão de Curso em Projetos de Software Livre (UCOSP – do inglês *Undergraduate Capstone Open Source Projects*), os projetos devem manter um *issue tracker* ativo, usar um sistema de controle de versão e ter processo de revisão de código definido. Mais tarde, Holmes et al. [11] também analisaram a percepção dos alunos com relação à adoção de contribuição para projetos de software livre como Trabalho de Conclusão de Curso. Eles observaram que os alunos valorizam muito a oportunidade de aplicar seu conhecimento a tarefas reais, em projetos reais com uma comunidade de usuários, enquanto recebem orientação real de membros da equipe de desenvolvimento. Além disso, relatam que contribuir para projetos reais fornece maior compreensão da engenharia de software do que poderia ser obtido através de meios mais tradicionais. Os resultados de Holmes e colegas se alinham aos resultados obtidos por nós na presente pesquisa.

No estudo de Petrenko *et al.* [16], os autores descreveram seu sistema de avaliação: os alunos são classificados com base em seu esforço individual, que inclui implementar, depurar e testar mudanças, mas também justificar suas decisões arquitetônicas. No trabalho de van Deursen *et al.* [22], a avaliação é baseada em equipe (por exemplo, apresentações, análise de código, etc.) e entregas individuais (revisão do código, participação em aulas, etc.).

Embora software livre já tenha uma presença formal na ensino de ES durante a última década [10, 19], a maioria dos trabalhos publicados nessa área foram relatórios de experiência (*eg,* [2, 12, 14, 21]). Esses relatórios não capturam todo o espectro de dinâmicas que podem surgir nessas disciplinas. Além disso, como esses estudos fazem diferentes perguntas de pesquisa, os resultados não são unificados e, dessa forma, são dificilmente comparáveis. Nosso trabalho conduzido com professores anteriormente [18] alinha-se com este presente estudo, pois mostra que as percepções dos professores são alinhadas às dos alunos ao que se refere aos benefícios e desafios. O presente trabalho complementa aquele [18] por buscar entender a perspectiva dos alunos com relação à importância dessas atividades na formação acadêmica e como os alunos tem interagido/contribuído com as comunidades. Os trabalhos de Holmes *et al.* [11, 12] são os que mais se aproximam deste trabalho. Pode-se dizer que os resultados são complementares, visto que foram analisadas diferentes abordagens de uso de Software Livre (sala de aula versus trabalho de conclusão de curso). Além disso, as questões de pesquisa aqui apresentadas visam cobrir diferentes aspectos, incluindo o entendimento da natureza das contribuições, bem como as especificidades da escolha de tarefas e dificuldades enfrentadas.

6 CONCLUSÕES

Com o advento das plataformas de desenvolvimento colaborativo e codificação social, como o Github e o GitLab, vários projetos de software livre se tornaram mais acessíveis, ajudando na disseminação do processo colaborativo de software já usado pelas comunidades de software livre. Recentemente, uma população de professores de Engenharia de Software tem mostrado interesse em utilizar projetos de software livre com parte das atividades e da avaliação das suas disciplinas. Trabalhos recentes discutiram algumas das percepções desses professores, e forneceram lições sobre a dinâmica da disciplina, bem como alguns possíveis benefícios e desafios encontrados ao colaborarem com projetos de software livre em suas disciplinas.

No entanto, pouco se sabe da percepção dos alunos que participaram de disciplinas com esse contexto. Neste trabalho, conduzimos 14 entrevistas semi-estruturadas com alunos que participaram de disciplinas que utilizavam contribuições em projetos de software livre como parte do processo de ensino-aprendizado. Apesar de dificuldades recorrentes para identificar um projeto de software livre adequado e uma tarefa que pudesse ser implementada durante a disciplina, e que ainda entregasse valor ao projeto, vários alunos relataram uma satisfação em ter cursado as disciplinas, bem como avaliaram positivamente suas próprias participações nas disciplinas. Ademais, identificamos que a participação do professor em atividades das comunidades de software livre (extra-classe) são notoriamente importantes para um melhor dinamismo entre professor, alunos e comunidades. Observamos também que a maior parte das contribuições dos alunos adicionaram novas funcionalidades no projeto, embora contribuições que realizavam correções no software também foram comuns. Em suma, respondemos e discutimos em detalhes as nossas cinco questões de pesquisas, o que gerou também implicações para os três diferentes atores dessa dinâmica de ensino-aprendizado e colaboração: professores, alunos e comunidades de software livre.

6.1 Limitações

Cientes das limitações e ameaças à validade deste trabalho, em primeiro lugar, reconhecemos que 14 participantes de entrevistas não nos possibilita generalizar os resultados e que os achados desse trabalho são limitados às entrevistas realizadas com os alunos das cinco instituições em que os participantes cursaram as disciplinas investigadas. De forma relacionada, este trabalho se limita a estudar disciplinas de Engenharia de Software que utilizam contribuições em projetos de software livre como parte do processo de ensino-aprendizado. Entretanto, dada a natureza qualitativa deste estudo, o número de participantes não é inexpressível e nos permitiu confirmar e estender resultados prévios, adicionando ao estado-da-arte.

A utilização de contribuições em projetos de software livre em outras disciplinas — que pode levar a diferentes achados — está fora do escopo deste trabalho. Por fim, a condução do estudo foi primariamente baseado nas entrevistas (embora enriquecido por uma análise dos dados das contribuições feitas nos projetos de software livre), dessa forma, está limitado ao entendimento dos autores que conduziram e analisaram as entrevistas. Para minimizar as ameaças relacionadas ao processo manual de codificação das entrevistas, as entrevistas foram primeiramente transcritas e a sua análise foi feita por pares, seguido por reuniões para resolução de conflitos. Disponibilizamos o roteiro das entrevistas bem como o resultado da análise, possibilitando que o estudo seja replicado e os resultados comparados [3].

6.2 Trabalhos Futuros

Como trabalhos futuros, além de ampliar o escopo do trabalho realizando mais entrevistas com outros alunos, planejamos realizar um estudo de larga escala em repositórios de software livre de forma a identificar e categorizar contribuições feitas por alunos em contextos de disciplinas. Por fim, conduziremos entrevistas com mantenedores e líderes de projetos de software livre de forma a triangular os achados deste estudo, bem como com resultados de estudos com análise da perspectiva do professor, de forma a confirmar ou refutar resultados conhecidos na literatura.

AGRADECIMENTOS

Agradecemos os participantes das entrevistas pela colaboração com o trabalho. Também agradecemos o apoio financeiro do CNPq (proc. 430642/2016-4) e FAPESP MCT/CGI.br (proc. 2015/24527-3).

REFERÊNCIAS

[1] Moara Brito, Fernanda Gomes, Debora Nascimento, Christina Chavez, and Roberto Bittencourt. [n. d.]. FLOSS in Software Engineering Education: An Update of a Systematic Mapping Study. In *Brazilian Symposium on Software Engineering - Education Track (SBES 2018)*. ACM.

[2] Joseph Buchta, Maksym Petrenko, Denys Poshyvanyk, and Vaclav Rajlich. 2006. Teaching Evolution of Open-Source Projects in Software Engineering Courses. In *22nd IEEE International Conference on Software Maintenance*. 136–144.

[3] Oisin Cawley, Stephan Weibelzahl, Ita Richardson, and Yvonne Delaney. 2014. *Incorporating a self-directed learning pedagogy in the Computing Classroom: Problem-Based Learning as a means to improving Software Engineering learning outcomes*. IGI Global, 348–371.

[4] Christina Chavez, Antonio Terceiro, Paulo Meirelles, Carlos Santos, and Fabio Kon. 2011. Free/Libre/Open Source Software Development in Software Engineering Education: Opportunities and Experiences. In *4th Software Engineering Education Forum, FEES@SBES@CBSoft, São Paulo, Brazil*.

[5] David Coppit and Jennifer M. Haddox-Schatz. 2005. Large Team Projects in Software Engineering Courses. In *36th SIGCSE Technical Symposium on Computer Science Education*. 137–141.

[6] Fabian Fagerholm and Max Pagels. 2014. *Examining the Structure of Lean and Agile Values among Software Developers*. Springer International Publishing, Cham, 218–233.

[7] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*. 345–355.

[8] Georgios Gousios, Andy Zaidman, Margaret-Anne D. Storey, and Arie van Deursen. 2015. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. 358–368.

[9] Lile P Hattori and Michele Lanza. 2008. On the nature of commits. In *23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, III–63.

[10] Gregory W. Hislop, Heidi J.C. Ellis, Allen B. Tucker, and Scott Dexter. 2009. Using Open Source Software to Engage Students in Computer Science Education. In *40th ACM Technical Symposium on Computer Science Education*. 134–135.

[11] Reid Holmes, Meghan Allen, and Michelle Craig. 2018. Dimensions of Experientialism for Software Engineering Education. (2018).

[12] Reid Holmes, Michelle Craig, Karen Reid, and Eleni Stroulia. 2014. Lessons Learned Managing Distributed Software Engineering Courses. In *36th International Conference on Software Engineering*. 321–324.

[13] Clif Kussmaul, Heidi J.C. Ellis, and Gregory W. Hislop. 2012. 50 Ways to Be a FOSSer: Simple Ways to Involve Students and Faculty. In *43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*. ACM, New York, NY, USA, 671–671. https://doi.org/10.1145/2157136.2157393

[14] Becka Morgan and Carlos Jensen. 2014. Lessons Learned from Teaching Open Source Software Development. In *10th International Conference on Open Source Systems, OSS 2014, San José, Costa Rica, May 6-9, 2014*. Berlin, Heidelberg, 133–142.

[15] D. M. Nascimento, K. Cox, T. Almeida, W. Sampaio, R. A. Bittencourt, R. Souza, and C. Chavez. 2013. Using Open Source Projects in software engineering education: A systematic mapping study. In *2013 IEEE Frontiers in Education Conference (FIE)*. 1837–1843.

[16] Maksym Petrenko, Denys Poshyvanyk, Václav Rajlich, and Joseph Buchta. 2007. Teaching Software Evolution in Open Source. *Computer* 40, 11 (Nov. 2007), 25–31.

[17] Gustavo Pinto, Luiz Felipe Dias, and Igor Steinmacher. 2018. Who Gets a Patch Accepted First? Comparing the Contributions of Employees and Volunteers. In *11th IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE@ICSE 2018, Gothenburg, Sweden, May, 2018*.

[18] Gustavo Pinto, Fernando Figueira Filho, Igor Steinmacher, and Marco Gerosa. 2017. Training Software Engineers Using Open-Source Software: The Professors' Perspective. In *30th Conference on Software Engineering Education and Training (CSEET)*. 117–121.

[19] Anita Sarma, Marco Aurélio Gerosa, Igor Steinmacher, and Rafael Leano. 2016. Training the Future Workforce Through Task Curation in an OSS Ecosystem. In *2016 24th ACM FSE*. ACM, 932–935.

[20] Yvonne Sedelmaier and Dieter Landes. 2013. A Research Agenda for Identifying and Developing Required Competencies in Software Engineering. *International Journal of Engineering Pedagogy* 3, 2 (2013).

[21] Therese Mary Smith, Robert McCartney, Swapna S. Gokhale, and Lisa C. Kaczmarczyk. 2014. Selecting Open Source Software Projects to Teach Software Engineering. In *45th ACM Technical Symposium on Computer Science Education*. 397–402.

[22] Arie Van Deursen, Maurício Aniche, Joop Aué, Rogier Slag, Michael De Jong, Alex Nederlof, and Eric Bouwers. 2017. A Collaborative Approach to Teaching Software Architecture. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 591–596.

[3] https://goo.gl/Yc4JJq