UNIVERSIDADE FEDERAL DO PAMPA

PATRICIA PADULA LOPES

PROPOSTA DE UM SISTEMA PARA O MONITORAMENTO DAS ATIVIDADES DE PROGRAMAÇÃO PARA ALUNOS INICIANTES – CFACIL+

Bagé
2018

PATRICIA PADULA LOPES

PROPOSTA DE UM SISTEMA PARA O MONITORAMENTO DAS ATIVIDADES DE PROGRAMAÇÃO PARA ALUNOS INICIANTES – CFACIL+

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Orientador: Érico Marcelo Hoff do Amaral

Co-orientadora: Marina Silva Gomes

Bagé
2018

PATRICIA PADULA LOPES

PROPOSTA DE UM SISTEMA PARA O MONITORAMENTO DAS ATIVIDADES DE PROGRAMAÇÃO PARA ALUNOS INICIANTES – CFACIL+

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 10 de Julho de 2018.

Banca examinadora:

_____

Prof. Dr. Érico Marcelo Hoff do Amaral
Orientador
UNIPAMPA

_____

Prof. Dra. Sandra Dutra Piovesan
UNIPAMPA

_____

Prof. Dr. Sandro da Silva Camargo
UNIPAMPA

Dedico este trabalho especialmente, aos meus pais, minha irmã e namorado por estarem sempre ao meu lado.

AGRADECIMENTO

Talvez não existam palavras suficientes e significativas que me permitam agradecer, às pessoas que me ajudaram, com o devido merecimento. A ajuda e o apoio de cada um de vocês foram para mim de valor inestimável. Apenas posso me expressar através da limitação de meras palavras o sentimento de gratidão que tenho por cada um, prestando esta humilde, mas sincera homenagem.

Agradeço ao meu namorado, Thiago, por sempre estar comigo e me incentivar a concluir este trabalho, especialmente por apresentar sempre um sorriso, quando sacrificava os dias, as noites, os fins-de-semana e os feriados em prol da realização deste estudo.

Aos meus amigos queridos.

À Mégui, por me ajudar quando mais preciso.

À Lili, por saber como espantar a tristeza e toda negatividade dos obstáculos que precisei enfrentar durante esta jornada.

À Mariana e Fernanda, por me fazerem sorrir até mesmo quando eu achava que nada poderia.

À Camila, por fazer a saudade não ser tão dolorosa quando estamos distantes.

À Maíra, por todas as dificuldades superadas em conjunto, por todo apoio que de você nunca me faltou.

À Vanessa, por todas as incontáveis vezes que me ajudou a superar todos os momentos difíceis pelos quais passei.

À Elda, pela força que me deu, obrigada por ter sempre uma palavra amiga.

Ao Dionatan, por todo empenho, dedicação e paciência para me ajudar.

Ao André, por sempre estar presente para me escutar e me apoiar.

Ao meu orientador, Prof. Dr. Érico Marcelo Hoff do Amaral, pela sua orientação prestada, pelo seu incentivo, disponibilidade e pelo apoio que sempre demonstrou.

RESUMO

Este trabalho apresenta uma proposta de auxílio no atendimento às dificuldades de aprendizagem dos alunos iniciantes das disciplinas de Algoritmos e Programação na Universidade Federal do Pampa. Implementou-se um sistema de monitoramento, o CFacil+, a partir da integração da ferramenta CFacil em um ambiente de rede cliente e servidor. Para tanto, neste estudo, foram realizadas pesquisas bibliográficas, bem como o aprimoramento da aplicação CFacil. Posteriormente, efetuaram-se testes de funcionamento dessa ferramenta de monitoramento e, também, pesquisa de satisfação de docentes da disciplina de Algoritmos e Programação. Os resultados da utilização da ferramenta demonstram que a mesma torna as aulas mais produtivas, uma vez que o sistema proporciona uma visualização imediata das atividades dos alunos. Além disso, os docentes participantes da pesquisa, as quais se utilizaram da aplicação, afirmaram haver realizado o acompanhamento dos alunos de forma efetiva por meio de uma interface simples e intuitiva. Por fim, as avaliações dos professores apontaram para o CFacil+ como uma alternativa de auxílio ao ensino de algoritmos e programação.

Palavras-Chave: Algoritmos e Programação, Cliente, Servidor.

ABSTRACT

This work presents a proposal of support for attending the beginner students' learning difficulties in the subject of Algorithms and Programing at the Federal University of Pampa (Universidade Federal do Pampa). A monitoring system, CFacil+, was developed after an integration of the tool CFacil in a network (client and server). In order to accomplish this goal, both bibliographic researches and an improvement of the application CFacil were carried out. Later on, functioning tests were executed in the monitoring tool (CFacil+), as well as a satisfaction survey conducted among the professors of Algorithms and Programing. The results reached after the use of CFacil+ by the professors show that it helps classes become more productive. This is due to the fact that the system eases an immediate visualization of the learners' activities. Moreover, the participants representing the faculty members, after making use of the application, claimed to have accomplished the monitoring of the students in an effective way through a simple and intuitive interface. Finally, the survey portrayed CFacil+ as an alternate type of collaboration to the teaching of Algorithm and Programming.

Keywords: Algorithms, Programming, Client, Server.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACM - *Association for Computing Machinery*

AIIP - Ambiente Inteligente para Iniciantes em Programação

ALICE - *Algorithm Learning Internet-based Computer Environment*

GCC - *Gnu Compiler Collection*

IEEE - *Institute of Electrical and Electronic Engineers*

IP - *Internet Protocol address*

RBIE - Revista Brasileira de Informática na Educação

RENOTE - Revista Novas Tecnologias na Educação

SCIELO - *Scientific Electronic Library Online*

TCP - *Transmission Control Protocol*

UNIPAMPA - Universidade Federal do Pampa

SUMÁRIO

1 INTRODUÇÃO

As disciplinas de Algoritmos e Programação constituem a base para o ensino de programação e abordam os princípios da lógica de programação, com o objetivo de desenvolver a capacidade de análise e resolução de problemas dos alunos através da descrição das soluções dos problemas na forma de algoritmos. Além disso, esta disciplina faz parte do plano curricular de vários cursos na área de tecnologia, com a finalidade de introduzir os conceitos de programação, exigindo dos alunos determinadas habilidades e desenvolvendo competências. Entre essas, pode-se mencionar a capacidade de pensar logicamente, a fim de organizar as ideias em forma de código e resolver problemas, tendo como propósito a construção de soluções algorítmicas.

Os estudantes, na maioria das vezes, não estão habituados com as novas formas de pensar requeridas pela disciplina, bem como não possuem certas aptidões necessárias. Dessa forma, encontram dificuldades que, por vezes, culminam na reprovação e evasão (BARBOSA, 2011). Tal problemática tem sido motivo de preocupação dos alunos e dos próprios docentes (VIEIRA *et al.* 2015). Dentre os problemas vivenciados em sala de aula, Ochoa *et al.* (2011) apontam que professores têm demonstrado dificuldades na elaboração de estratégias no atendimento de alunos de forma individualizada, podendo também ser uma das causas para a falta de motivação dos alunos na aprendizagem de algoritmos e programação (RAPKIEWICZ *et al.* 2006).

Reconhecendo que existe uma lacuna no processo de aprendizagem para compreensão da lógica de programação, causada pela distância entre professor e aluno durante as atividades didáticas, e agravada pela dificuldade dos estudantes em entender as linguagens de programação, observa-se cada vez mais a adoção de recursos computacionais como ferramenta de apoio que visam possibilitar uma melhor interação entre docentes e discentes.

Trabalhos como o de Gomes e Amaral (2016) buscam reduzir esta lacuna com a proposta de uma ferramenta para o apoio ao processo de ensino e aprendizagem de programação. Voltado para alunos iniciantes, através de um sistema que torna a identificação e correção de erros de compilação em linguagem C mais simples e compreensível, esse recurso foi denominado CFacil. Com esta solução, o estudante, em seus primeiros contatos com a linguagem de programação, pode ter uma maior

facilidade na abstração da complexidade envolvida na correção de erros durante o processo de programação. Estimula-se, desta forma, a construção de conhecimento sobre a lógica envolvida na implementação de soluções computacionais para problemas simples.

A disciplina de Algoritmos e Programação exige uma forte demanda de interação a fim de atender, acompanhar, mediar e avaliar os alunos em atividades propostas em sala de aula. Em muitas situações, esta demanda de interação é inviável de ser atendida por intermédio de ferramentas como o CFacil, devido à quantidade de alunos e a diversidade de dificuldades apresentadas por estes. Desta forma, muitas vezes as demandas com relação à aprendizagem não são detectadas e atendidas em tempo hábil, ocasionando problemas como desmotivação, reprovação ou evasão.

Com base neste contexto, este trabalho se caracteriza como uma proposta de uma solução que permita ao professor acompanhar as atividades de programação desenvolvidas pelos alunos em ambientes de laboratório. Reconhecendo a finalidade do CFacil, este estudo vislumbra implementar uma aplicação que disponibilize uma infraestrutura para que o docente acompanhe o desempenho dos alunos nas tarefas propostas aos estudantes em sala de aula. Dessa forma, o professor conseguirá reconhecer as dificuldades e os erros mais comuns dos discentes na aprendizagem inicial da programação, acompanhando os passos do estudante dentro do ambiente e registrando seu progresso por meio das atividades a serem desenvolvidas. A proposição e ajuste das estratégias didáticas então adotadas visam um aprimoramento da experiência de ensino e aprendizagem na disciplina de Algoritmos e Programação.

1.1 Problema de pesquisa

As dificuldades encontradas pelos professores em acompanhar de forma individual o desempenho dos alunos em atividades de programação em laboratório são muitas, talvez pela falta de um método de ensino adequado ou um material didático especial. Para tentar minimizar essa dificuldade, busca-se, então, resposta à pergunta:

É possível implementar uma solução para o monitoramento das atividades de programação, desenvolvidas por iniciantes em laboratórios de informática, que auxilie ao docente no processo de ensino e aprendizagem de Algoritmos e Programação?

1.2 Objetivos

A partir da definição da questão norteadora para o desenvolvimento deste trabalho, passam-se às descrições dos objetivos que constituem as principais ações a serem desenvolvidas.

1.3 Objetivo geral

Este trabalho tem como objetivo o desenvolvimento de um recurso didático que visa aprimorar a forma de interação entre aluno e professor em ambientes de laboratório através do acompanhamento individualizado dos alunos, permitindo assim uma maior eficiência na identificação e atendimento das dificuldades de aprendizagem de Algoritmos e Programação.

1.4 Objetivos específicos

Para que o objetivo geral seja alcançado, propõem-se os seguintes passos a serem executados no decorrer deste trabalho:

- Estudo do referencial teórico e estado da arte sobre o tema elencado;
- Estudo sobre o funcionamento da ferramenta CFacil;
- Proposição de um modelo que integre a ferramenta CFacil a uma aplicação cliente/servidor;
- Análise e projeto da solução cliente/servidor para avaliação das atividades de cada aluno;
- Implementação do software;
- Definição dos métodos de validação e teste do software junto aos alunos das disciplinas de Algoritmos e Programação;
- Avaliação dos resultados.

1.5 Estrutura do trabalho

Este trabalho está estruturado em 5 capítulos. No capítulo 2 é realizado o levantamento teórico sobre conceitos relevantes para o desenvolvimento do trabalho. O capítulo 3 apresenta a caracterização da pesquisa e os procedimentos metodológicos adotados. O capítulo 4 apresenta o desenvolvimento de um sistema para monitoramento das atividades dos alunos iniciantes em programação. Na sequência, descrevem-se a modelagem, as técnicas adotadas no processo da implementação, bem como os resultados analisados da utilização dessa ferramenta. resultados atingidos com os testes de validação realizados. Por fim, o capítulo 5 apresenta as considerações finais.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentadas as bases teóricas utilizadas para realização deste trabalho. A Seção 2.1 apontará as dificuldades apresentadas pelos alunos e professores no ensino de algoritmos e programação. Na Seção 2.2 será abordada a relação professor-aluno no processo de ensino-aprendizagem de Algoritmos e Programação. Na seção seguinte, 2.3, serão tratados os métodos de ensino de Algoritmos e Programação como práticas de ensino. Por último, na Seção 2.4, serão apresentados alguns trabalhos correlatos.

2.1 Ensino de algoritmos: Dificuldades encontradas

O ensino de Algoritmos e Programação, em cursos de Computação ou afins tem por objetivo estruturar o pensamento do aluno de maneira que ele seja capaz de conseguir utilizar a lógica de programação como ferramenta para a resolução de diversos problemas computacionais. Esse é um fator indispensável para disciplinas mais avançadas (SCOLARI *et al.* 2007). A importância do aprendizado desses conceitos já nas disciplinas do primeiro semestre é percebida tão logo o aluno inicie o aprendizado de linguagens de programação e precise estruturar os passos para resolução de um problema, repassando-os ao computador, sob a forma de comandos ou instruções de linguagens de programação (SANTOS *et al.* 2015).

No entanto, o processo de aprendizagem de programação é uma experiência nova, sendo permeada por diversos desafios relacionados principalmente à compreensão de abordagens abstratas e ao desenvolvimento do raciocínio lógico-dedutivo. Contudo, tais habilidades nem sempre são desenvolvidas no ensino básico, levando o aprendiz a ter grandes dificuldades e, consequentemente, um mau desempenho durante o curso (GIRAFFA *et al.* 2016).

Em sua pesquisa, Campos (2009) identificou juntamente com os alunos dois motivos que causam maior preocupação em sua totalidade: a inabilidade em conseguir ler/entender algoritmos, sejam próprios ou propostos por terceiros, para a solução de um problema com ou sem a sua definição, e a dificuldade em desenvolver a lógica para a solução de problemas de baixa para média complexidade.

Outra problemática, levantada a partir dos estudos de Barbosa (2011), vivenciada pelos alunos, compõem um conjunto de subproblemas, são eles: a falta de

experiência e de habilidades exigidas não adquiridas no ensino médio, a metodologia de ensino utilizada pelos professores em sala de aula, muitas vezes pouco estimulante e pouco preocupada com o conhecimento prévio do aluno; e, por fim, a falta de compreensão de como resolver problemas reais com algoritmos. Estas dificuldades, segundo Da Silva e Montenegro (2016), residem na organização em compreender e empregar certos conceitos abstratos na programação de algoritmos, e, em particular, na aplicação de noções básicas, acerca da elaboração de estratégias para resolução de problemas.

Aliado a isto, a utilização de aulas expositivas com baixos níveis de interação professor-aluno aumenta o nível de dificuldade para que se tenha êxito no ensino de algoritmos. Todos estes problemas, dada a problemática em torno do ensino de programação, acabam, segundo Schultz (2003), sendo agravados pela dificuldade dos próprios professores em identificar e reconhecer nos alunos tais habilidades prévias, de forma a aproveitá-las melhor para o desenvolvimento das competências de construção de algoritmos e programação, conjugada, muitas vezes, com turmas grandes. A forma de ensino dentro da sala de aula é a mesma para todos os alunos. É difícil para um professor levar em consideração o perfil, as metas, as necessidades, as expectativas, as preferências e o nível de conhecimento de cada aluno, de modo a proporcionar um ensino adaptado (FALCKEMBACH; DE ARAÚJO, 2013).

No entanto, para produzir melhores resultados no processo de aprendizagem, torna-se necessário que o docente consiga compreender como as experiências relacionadas à programação de algoritmos estão relacionadas aos conceitos estudados e como podem ser organizadas para alcançar os objetivos esperados. Além disso, também é muito comum que as atividades sejam realizadas sem a necessária análise da solução. Segundo Stamouli e Huggard (2006), no ensino de programação, embora a compreensão da correção lógica do programa pelo estudante seja importante, a análise da solução pelo docente também é fundamental, porque afeta diretamente na abordagem didática adotada pelo professor e otimiza a relação de ensino-aprendizagem, proporcionando uma melhor percepção das dificuldades encontradas pelos discentes durante as aulas de algoritmos e programação.

2.2 A relação professor-aluno no processo de ensino-aprendizagem de algoritmos e programação

A relação professor-aluno tem sido um dos principais motivos de preocupações com relação ao ensino e aprendizagem de Algoritmos e Programação, destacando-se em vários trabalhos e relatos de professores e alunos. Segundo Barcelos *et al.* (2009), alguns fatores como as características do professor e do aluno, a didática utilizada, o método de ensino e as relações interpessoais estabelecidas são de extrema importância para que sejam identificadas as causas relacionadas às dificuldades de interação entre os estudantes e professores em sala de aula.

Alguns motivos podem ser apontados como causadores desta problemática; um destes, é citado por Pereira *et al.* (2007), e está relacionado ao número de alunos em sala de aula nas disciplinas de Algoritmos e Programação, que, segundo o autor, tem dificultado o acompanhamento individualizado dos estudantes.

Por ser o primeiro componente curricular que trabalha com os conceitos básicos de resolução de problemas via computador, as disciplinas de programação são ministradas no início dos cursos relacionados às tecnologias da informação e comunicação. É comum que essas turmas alcancem um número consideravelmente elevado de alunos, visto que há um alto nível de reprovação e muito dos alunos são repetentes.

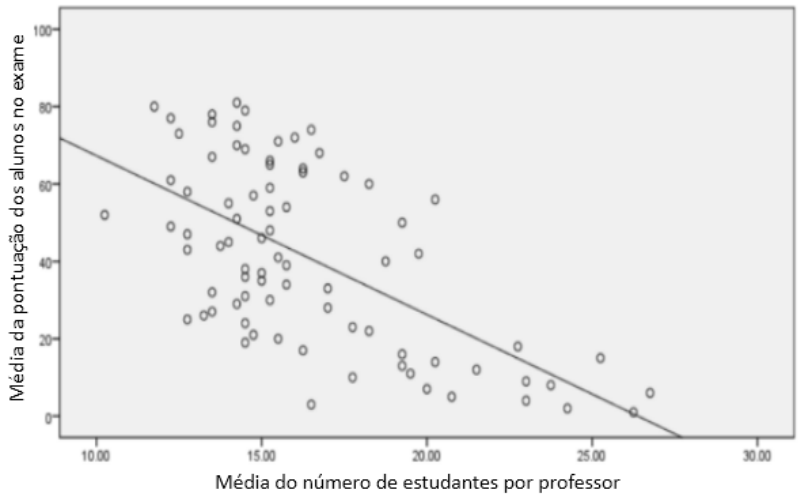
Todos estes fatores dificultam uma avaliação individualizada do professor com relação às atividades propostas em laboratório, além de limitar a quantidade de avaliações que se pode realizar em um semestre: é muito difícil para o educador avaliar individualmente todas as soluções de um determinado algoritmo, e pontuá-lo de acordo com a quantidade de acertos ou erros dos estudantes (MOREIRA *et al.* 2009).

Um estudo apresentado por Blatchford *et al.* (2011), examinando o efeito do tamanho da turma no envolvimento na sala de aula e a interação aluno e professor, demonstrou que em classes maiores os alunos tendem a desviar-se mais facilmente das tarefas. Em sua pesquisa, foram realizadas observações sistemáticas da interação professor-aluno em 49 escolas primárias e secundárias na Inglaterra e no País de Gales, onde foram estudados os comportamentos dos alunos e professores tendo por base algumas categorias comportamentais. Através dessas observações, pôde-se perceber que um aumento de 5 alunos em uma turma diminuía 25% o nível de atenção que o professor conseguia dar aos estudantes individualmente. O

aprendizado se tornava ainda mais defasado devido às interrupções decorrentes da indisciplina.

Na mesma linha de pesquisa, Koc *et al*. (2015) realizaram uma análise de correlação entre o número médio de alunos por professores em sala de aula e o desempenho destes estudantes no exame de admissão universitária, em 81 cidades da Turquia. Através desta análise, percebeu-se uma correlação negativa com relação à transição dos alunos para o ensino superior. A correlação negativa, que pode ser observada através do gráfico da Figura 1, indica que, quanto maior o número médio de estudantes por professor em sala de aula, menor era o aproveitamento acadêmico dos alunos no exame.

Figura 1 - Correlação entre a média de alunos por professor e a média da pontuação dos alunos no exame



Fonte: Koc *et al.* (2015)

Ao identificar o contexto apresentado, percebe-se que a redução do número de alunos em sala é um dos fatores para melhoria da qualidade do ensino e da aprendizagem dos estudantes. Salas com um número elevado de alunos impedem que o professor consiga minimamente reconhecer as dificuldades de cada aluno, de maneira a estabelecer estratégias adequadas para potencializar a aprendizagem. De acordo com Tardif (2002), os docentes consideram difícil gerenciar um grupo grande de alunos quando realizam atividades práticas, pois este tipo de atividade requer ao professor acompanhá-los de forma individual ou em pequenos grupos no processo de aprendizagem.

Com base nestas características, percebe-se que a disponibilização de um recurso computacional, que permita ao docente acompanhar e identificar os alunos com maiores dificuldades em relação as atividades propostas em sala de aula, irá possibilitar um maior contato entre alunos e professores, proporcionando o acompanhamento constante do processo de ensino e aprendizagem.

2.3 Práticas e soluções para o ensino de Programação

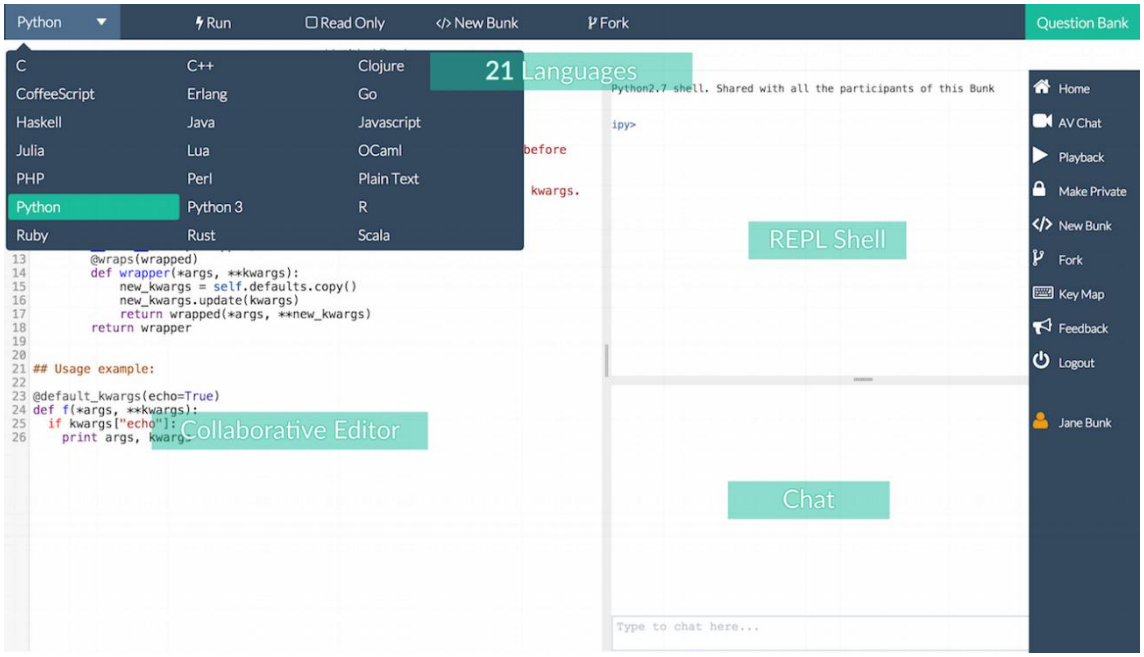
O ensino de Algoritmos e Programação tem como propósito permitir que os alunos desenvolvam um conjunto de competências necessárias para conceber programas e sistemas computacionais capazes de resolver problemas reais. O professor deve ter a capacidade de instruir os alunos a fim de que eles consigam aprender a desenvolver programas, de forma a garantir uma metodologia de análise de problemas através da construção de algoritmos, capacitando-os a traduzir estes algoritmos em programas, a partir de uma linguagem em alto nível. Porém, muitas das práticas de ensino adotadas não conseguem suprir a necessidade dos docentes em acompanhar todos os estudantes durante as aulas de programação realizadas em laboratórios, devido ao quantitativo elevado de alunos.

Neste sentido, devido ao seu caráter técnico-formal, a disciplina carece de modelos pedagógicos que facilitem o acompanhamento efetivo das atividades dos estudantes (AMARAL, 2015). Diante dessa problemática, novas ferramentas computacionais vêm, ao longo dos anos, sendo incorporadas para auxiliar os professores no monitoramento das turmas em aulas práticas de algoritmos e programação. Uma das ferramentas que pode ser utilizada com tal finalidade é o *Codebunk*, um editor de código colaborativo que permite desenvolver e testar programas de forma online. O sistema possibilita monitorar em tempo real códigos desenvolvidos em até 21 linguagens diferentes, por intermédio de seções, conhecidas como "*Bunks*". Dentro destes *Bunks* é possível visualizar tudo o que é digitado, e os resultados que aparecem no compilador, se utilizado em sala de aula, poderia auxiliar o professor no entendimento das soluções desenvolvidas pelos estudantes e tornar possível uma ajuda imediata em sala de aula. Além disso, o software disponibiliza um recurso de bate-papo onde seria possível a comunicação do professor com o aluno através de mensagens de texto. Pesquisas envolvendo a utilização desse tipo de

ferramenta foram apontadas por Guoli *et al.* (2010), como alternativa para ampliar o processo de ensino e aprendizagem na área educacional. Os pesquisadores descrevem que essas aplicações podem fornecer maneiras mais fáceis e eficientes para os professores monitorarem o progresso dos alunos de forma individual. Entretanto, seu uso em um ambiente acadêmico pode ser considerado limitado, por ter somente versões pagas.

A Figura 2 apresenta a interface do *Codebunk*, dentro da aplicação, o usuário encontrará um menu no topo, onde é possível selecionar a linguagem e executar um código. Há mais três janelas, duas delas em que é possível escrever as linhas de comando, mais uma de bate-papo. A ação pode ser visualizada pelo proprietário da seção, e o resultado das compilações pode ser monitorado assim que o botão *Run* for acionado.

Figura 2 - Interface do Codebunk

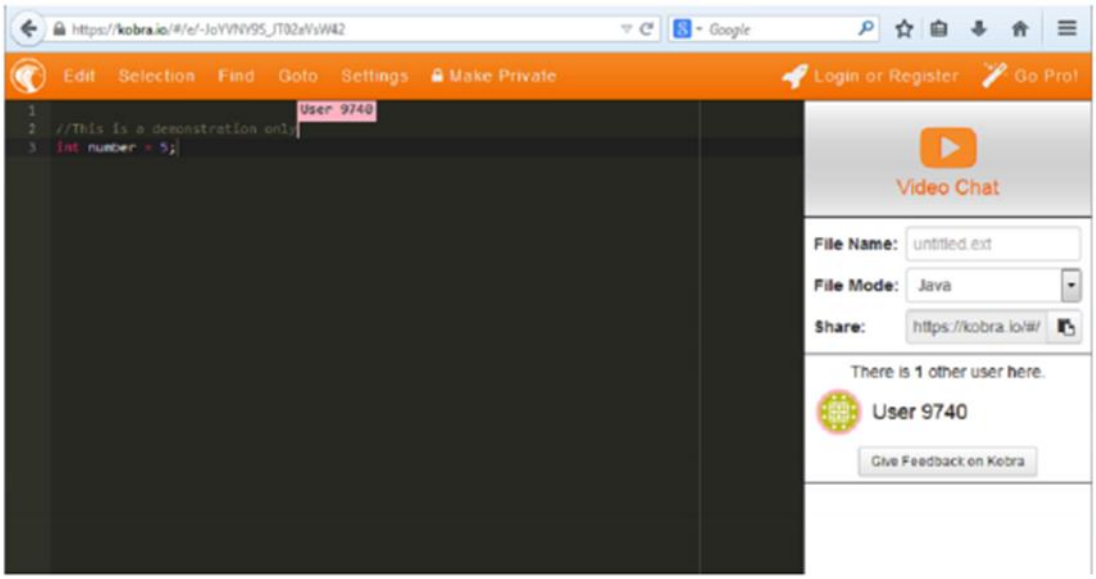


Fonte: Codebunk (2018)

Outro ambiente que poderia ser adotado como instrumento de apoio ao professor em aulas de Algoritmos e Programação é o Kobra, um editor de código-fonte colaborativo, que permite o acompanhamento individualizado de códigos diretamente na *Web*, tornando-os acessíveis a partir de qualquer lugar (KOBRA, 2018). O acesso do editor é concedido com um endereço da *Web* e qualquer pessoa que recebe um

link para o arquivo pode participar para editá-lo. Ao editar os documentos, os desenvolvedores podem ver imediatamente as modificações feitas por todos os outros participantes, através de um editor com realce do cursor dos convidados. Na Figura 3, é possível observar um usuário convidado, com o nome de “Usuário 9740”, que foi requisitado por meio de um endereço da *Web* compartilhado. A localização do cursor é destacada com uma etiqueta de identificação colorida durante a edição do código.

Figura 3 - Interface gráfica do Kobra

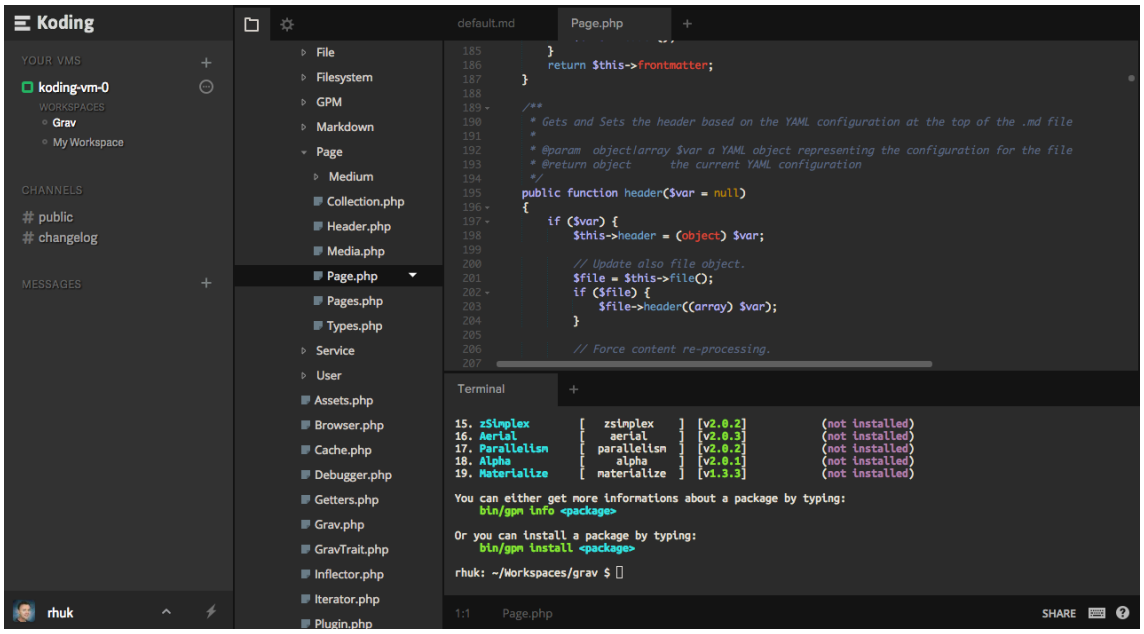


Fonte: Rantala (2015)

Diante das funcionalidades que a ferramenta apresenta, a introdução do Kobra como apoio ao ensino de uma disciplina introdutória de programação poderia viabilizar a correção de códigos-fonte ao professor, bem como reduzir o tempo necessário para correção e apresentação dos resultados das atividades desenvolvidas pelos alunos, possibilitando ao docente, dessa forma, acompanhar os estudantes. Essas características puderam ser associadas com trabalhos educativos, como o feito por Jung *et al.* (2016) na qual foi constatada a relevância deste tipo de ferramenta para a aprendizagem colaborativa no ensino.

Dentre outros serviços similares envolvendo o uso destas tecnologias no processo de ensino e aprendizagem, Amber *et al.* (2015) apresenta em sua pesquisa a aplicabilidade do Koding.com em um ambiente educacional. A Figura 4 apresenta a interface gráfica desta ferramenta.

Figura 4 - Interface Web do Koding.com



Fonte: Amber *et al.* (2015)

De acordo com os pesquisadores, este tipo de sistema permite a utilização de recursos para prover um *feedback* automático ao estudante através do seu terminal, disponibilizando desta maneira um método para identificação imediata de erros e a construção de um raciocínio correto para problemas propostos. Além disso, possibilita o compartilhamento de códigos em tempo real com os demais colegas, ao passo que o documento está sendo editado. Cada aluno poderá observar a correção do outro e inserir comentários, caso encontre algo que não esteja correto segundo sua opinião, de modo que o outro aluno possa ver, e ambos debatam utilizando a ferramenta de comentários ou mesmo o próprio *chat* da ferramenta. As correções e os próprios códigos podem ser compartilhados com os professores da disciplina, os quais podem analisar as soluções propostas pelos estudantes, tornando a correção mais dinâmica e interativa, permitindo que as avaliações estejam sob a luz de vários olhares.

Outra prática pedagógica alinhada à concepção de um sistema para acompanhamento dos alunos e que vêm sendo explorada há alguns anos, é o BOCA (*On-line Contest Administrator*), desenvolvido por De Campos e Ferreira (2004) para ser utilizado em maratonas de programação da Sociedade Brasileira de Computação. O sistema pode ser também aplicado no apoio a disciplinas em que se faça uso de submissão e correção de trabalhos durante as aulas; quando utilizado para este propósito, o programa é capaz de compilar e executar códigos escritos em diversas

linguagens de programação, e os códigos submetidos são então avaliados quanto a erros de compilação e execução em um processo automático e em tempo real (NAZARIO; DE SOUZA, 2010).

Por intermédio do BOCA, é possível que o professor acompanhe os alunos por meio de uma tela de visualização dos resultados, que pode ser observada através da Figura 5. Na tabela apresenta, é possível visualizar os usuários com os respectivos nomes de cada um dos alunos no sistema após a submissão das soluções das atividades propostas em sala de aula. As células das colunas com as respectivas letras: A, B, C, D, E, F e G correspondem a todos os problemas previamente cadastrados, que foram identificados pelo professor por uma letra (A, B, C, D, E, F e G) e por uma cor (vermelho, verde, azul, preto, branco e amarelo). No sistema, o aluno receberá o balão correspondente à cor do exercício somente se o programa for aceito.

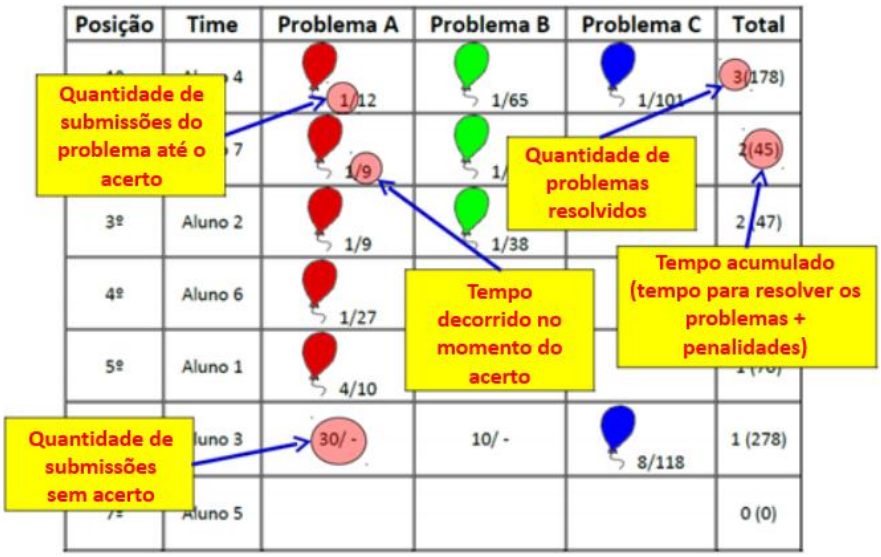
Figura 5 - Tela de visualização dos resultados

#	User	Name	A	B	C	D	E	F	G	Total
1	estudante01/1	Alessandro dos Santos Ferreira	2/28	1/33	1/43	1/33	2/76	10/-	2/133	6 (432)
2	estudante62/1	Leonardo Mauro Pereira Moraes	1/33	1/37	1/7	2/54	3/148		4/184	6 (583)
3	estudante61/1	Lucas Abreu da Silva	1/14	1/24	3/25	2/48	5/129	2/-	3/207	6 (628)
4	estudante49/1	Shih Ting Ju	1/36	1/74	1/61	1/102	1/155	1/-	1/213	6 (644)
5	estudante43/1	Reinaldo Felipe Soares Araujo	1/38	1/65	3/28	2/90	2/154		2/187	6 (693)
6	estudante39/1	Matheus Vicente Pinheiro de Oliveira	2/149	1/20	2/153	1/72	1/214		1/137	6 (785)
7	estudante15/1	Emerson Jair Reis Oliveira da Silva	7/88	1/39	2/19	2/53	4/-	2/-	1/181	5 (477)
8	estudante14/1	Edison Gabriel Goncalves Borghezan	1/73	3/83	2/19	3/118		1/-		4 (367)
9	estudante25/1	Jean Patrick Tremeschin Torres	1/43	1/80	4/102	3/87	1/-		2/-	4 (392)
10	estudante12/1	Doglas Wendll Sorgatto	2/171	1/48	2/72	3/112		1/-		4 (481)

Fonte: Facom (2018)

Para exemplificar em maiores detalhes os dados que estão disponíveis ao docente, no acompanhamento dos alunos na tela de visualização de resultados, é apresentada a Figura 6.

Figura 6 – Acompanhamento dos alunos no BOCA



Fonte: Facom (2018)

Nesta figura, é possível observar que o docente tem acesso as seguintes informações sobre o desempenho dos alunos: quantidade de submissões do problema até o acerto, tempo decorrido no momento do acerto, quantidade de submissões sem acerto, quantidade total de problemas resolvidos e o tempo que é acumulado pelos alunos para resolver os problemas.

No entanto, a utilização destas ferramentas para apoiar os professores apresenta algumas limitações que podem gerar problemas para alunos iniciantes em programação, tais como o excesso de configurações avançadas, visto que o Koding.com, o Kobra e o Codebunk são IDEs profissionais e possuem uma lógica voltada à construção de projetos de maior porte do que os abordados na programação introdutória (PEARS *et al.* 2007). Além disso, apresentam todos os textos da interface gráfica (menus, ajuda, mensagens para o usuário, etc) no idioma inglês, o que cria um obstáculo, em especial nos primeiros contatos com a ferramenta, aos estudantes que não possuem domínio deste idioma (ANIDO, 2015). Com relação ao software BOCA, observou-se que os alunos podem ter dificuldades em compreender as mensagens de erros que são apresentadas. Estas mensagens são mostradas em inglês, o que nos leva ao problema do idioma já mencionado anteriormente. Além disso, a ferramenta não apresenta um compilador próprio, sendo necessária a utilização de uma IDE externa que permita desenvolver e compilar o código. Após a verificação dos erros, o aluno ainda precisa salvar o programa em um código fonte e submeter ao

BOCA. Todas estas etapas dificultam a utilização do sistema por usuários que não possuem suficiente conhecimento sobre linguagem de programação ou mesmo do funcionamento de um compilador, que é o caso da maioria dos alunos que estão iniciando nos cursos da área que envolve programação de computadores (DA SILVA *et al.* 2017).

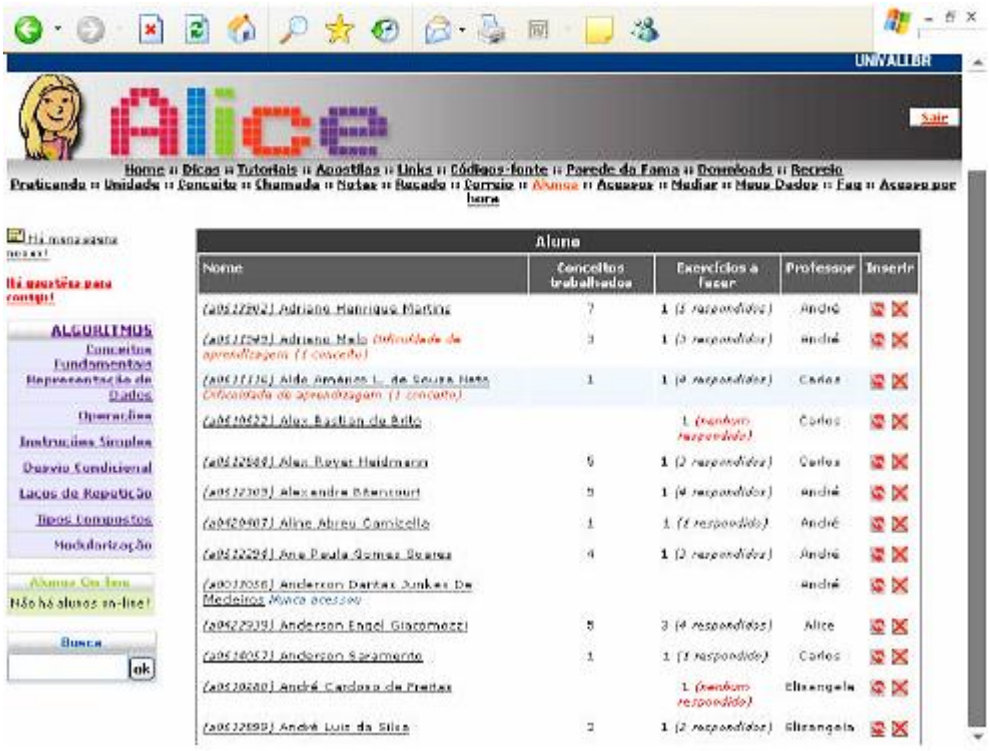
Embora estes trabalhos citados tragam importantes contribuições e sirvam como embasamento teórico para o desenvolvimento deste estudo, eles se diferenciam da proposta do CFacil+, pois apesar do foco desta aplicação ser o professor, também preocupa-se com o aluno no processo de ensino e aprendizagem, permitindo a prática, fixação do conteúdo trabalhado em aula, e a correção automática dos algoritmos através de sua integração com o CFacil, que utiliza termos da língua portuguesa em seu vocabulário, sendo de simples compreensão, principalmente para os iniciantes em programação.

2.4 Trabalhos Correlatos

Nesta seção serão apresentados algumas ferramentas que podem ser utilizadas com a finalidade de auxiliar o professor em sua tarefa de dar acompanhamento adequado ao aluno em disciplinas de algoritmos e programação. Serão citadas neste trabalho: “ALICE”, “AIIP” e “Codeboard”. Existem várias outras aplicações que permitem monitorar o desempenho dos alunos em sala de aula, porém, estes projetos apresentados a seguir possuem semelhanças com o CFacil+.

No ambiente ALICE, proposto por Raabe *et al.* (2005), o propósito é fornecer subsídios ao professor para o monitoramento do desenvolvimento do aluno, sendo possível o acompanhamento individualizado sobre as dificuldades de aprendizagem dos estudantes. A aplicação assemelha-se com a proposta deste trabalho, pois oferece recursos que permitem ao docente acompanhar o desempenho acadêmico, não em tempo real, mas ao longo do semestre, de acordo com a realização dos exercícios no próprio ambiente, avaliados pelos professores. Na Figura 7, é possível visualizar uma tela do sistema em que um professor pode identificar cada aluno pelo nome, acompanhar quais são as suas dificuldades, verificar os conceitos que foram trabalhados na resolução das atividades e observar quantos exercícios foram respondidos e quantos ainda estão pendentes.

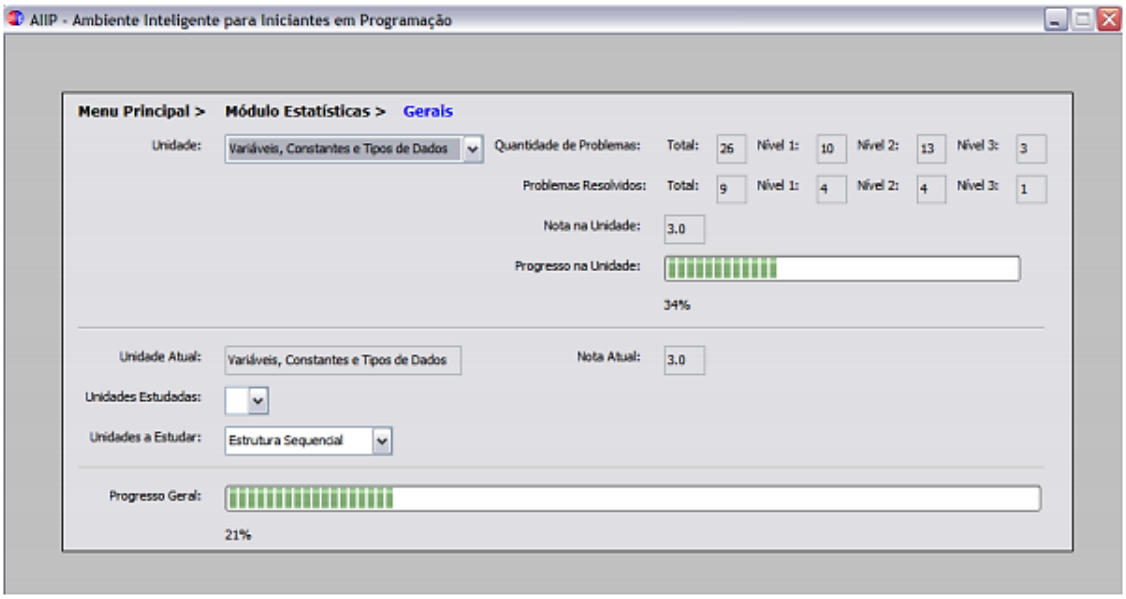
Figura 7 - Identificando as dificuldades de aprendizagem dos alunos no ambiente ALICE



Fonte: Raabe *et al.* (2005)

Diferente do ambiente ALICE, a ferramenta AIIP implementada por Gomes *et al.* (2011), permite o acompanhamento das atividades dos alunos em tempo real. O professor pode ter acesso à quantidade de problemas resolvidos por um aluno em uma determinada unidade. Cada unidade é estruturada de acordo com o nível de dificuldade encontrado nos conteúdos de algoritmos e programação e fornece dicas e *feedbacks* apropriados durante o processo de resolução de problemas. Na medida que um aluno avança de unidade, novos problemas são liberados para que ele tente resolvê-los. A Figura 8 apresenta a tela do AIIP, onde o docente poderá verificar a quantidade de problemas que foram resolvidos e a quantidade de dicas disponíveis, de acordo com o nível de conhecimento do aluno e o progresso na unidade.

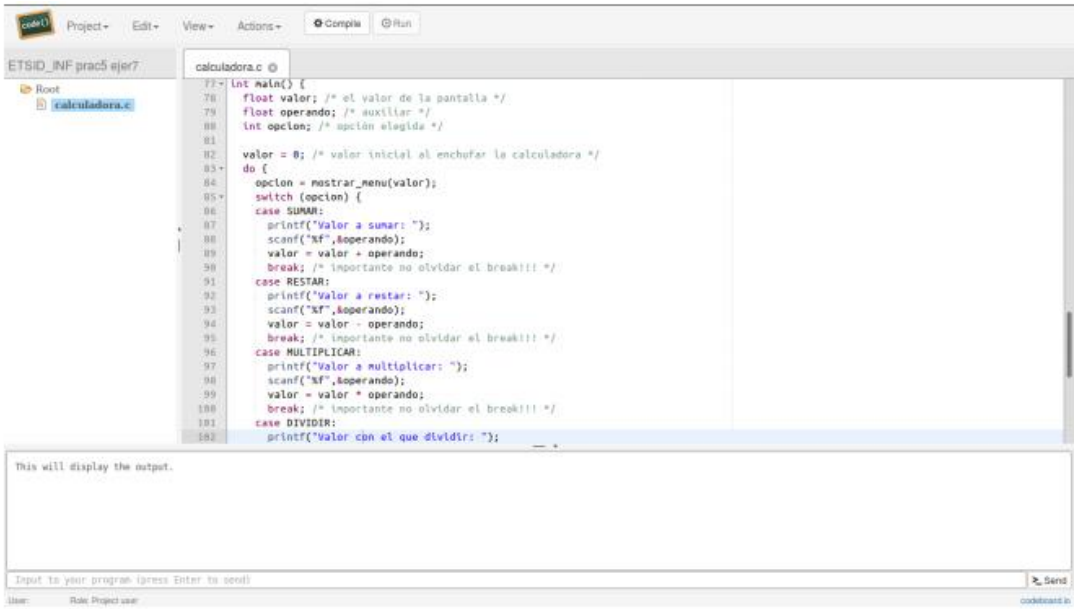
Figura 8 - Acompanhamento dos alunos no AIIP



Fonte: Gomes *et al.* (2011)

Ao realizar uma comparação entre as características presentes no Codeboard com as funcionalidades disponíveis no CFacil+, verificou-se que esta ferramenta também possui a capacidade de coletar estatísticas sobre o número de compilações de acordo com a quantidade de exercícios resolvidos pelos alunos. Contudo, o fluxo de trabalho usual desta aplicação quando utilizada em sala de aula é um pouco diferente da proposta do sistema de monitoramento das atividades de programação aqui apresentada. No Codeboard, o professor cria alguns projetos, cada um contendo um exercício que normalmente é um modelo a ser preenchido pelo aluno. Nesse ponto, o criador do projeto pode decidir se este projeto é público ou privado, a linguagem de programação usada no exercício e se o envio de código está ativado ou não. O aluno, após efetuar *login*, abre o projeto correspondente ao exercício e edita-o, podendo compilar, executar ou salvar, como em qualquer outra IDE de programação, conforme ilustrado na Figura 9.

Figura 9 - Captura de tela do IDE Codeboard



Fonte: Codeboard (2018)

O professor ou, em geral, o criador do projeto, pode inspecionar a lista de usuários que acessaram seus projetos e também as soluções dos alunos, onde é possível analisar o progresso dos estudantes, com base no número de acessos globais por dia em um determinado projeto, número de compilações e execuções por projeto e por dia, número de compilações e execuções do projeto correspondente por usuário, incluindo também a última data. Esses dados parecem úteis para dois propósitos diferentes: monitorar o processo de aprendizagem, fornecendo informações sobre o esforço dedicado a cada parte do programa (número de compilações / execuções, número de exercícios resolvidos, etc.), e auto avaliar a qualidade do material, o nível de dificuldade dos exercícios propostos, etc. Isso pode ser estimado com o esforço dedicado a cada exercício a partir dos dados obtidos com o Codeboard. Além disso, essas informações podem auxiliar o professor a identificar o aluno que está com maior dificuldade.

Na Tabela 1 é feita uma comparação entre as características observadas e definidas como pertinentes nos trabalhos correlatos com o CFácil+.

Tabela 1 – Trabalhos Correlatos

Características	ALICE	AIIP	Codeboard	CFacil +
Permite a identificação dos alunos	✓	✓	✓	✓
Possui informações sobre o nº de compilações em cada exercício	✗	✗	✓	✓
Inclui informações sobre a quantidade de erros em cada exercício	✓	✓	✓	✓
Contém informações sobre os exercícios resolvidos	✓	✓	✓	✓
Permite identificar alunos com maiores dificuldades	✓	✓	✓	✓
Possibilita a visualização imediata sobre o monitoramento dos alunos	✗	✓	✓	✓

Fonte: Próprio autor (2018)

Apesar dos trabalhos analisados neste capítulo serem relacionados ao proposto, observam-se alguns problemas associados à utilização destas ferramentas. No AIIP é utilizado um algoritmo-resposta para realizar a análise das soluções dos exercícios realizados pelos alunos. O sistema compara o algoritmo submetido pelo aluno com o algoritmo-reposta cadastrado na base de dados do AIIP pelo professor, considerado ideal para resolução do problema em questão, porém, por pensarem diferente, cada aluno pode ter a sua própria solução. Como consequência, o docente pode ter dificuldades em compreender a lógica individual de cada aluno, podendo levar à construção de soluções equivocadas que não condizem com a lógica de programação adotada pelos alunos no desenvolvimento de seus algoritmos.

A identificação deste mesmo problema pode ser observado também no ambiente ALICE. Ao cadastrar uma questão no sistema, o professor é quem informa quais conceitos envolvidos e quais elementos léxicos (*tokens*) a resposta deve obrigatoriamente apresentar. Assim, cada resposta é submetida a uma análise léxica onde um conjunto de *tokens* é identificado e, caso um dos solicitados pela questão não constar neste conjunto, o professor recebe uma notificação no momento da correção, de que existe uma possível ocorrência de erro na resposta. Devido os raciocínios diferenciados, na grande maioria das vezes não vai existir um gabarito de correção que possa ser seguido.

No Codeboard, embora os dados sobre o monitoramento dos alunos estejam disponíveis em sua interface da web (conforme mostrado na Figura 10), atualmente a ferramenta não permite salvar esses dados para serem usados em análises estatísticas.

Figura 10 - Captura de tela da interface Web do Codeboard mostrando um gráfico de compilações e execuções de um exercício

Fonte: España-Boquera *et al.* (2017)

Além disso, não é possível obter um relatório detalhado da quantidade de erros dos alunos, pois nem sempre o aluno que realiza um maior número de compilações é o que possui mais dificuldades. Portanto, a interpretação dessas variáveis (acessos / compilações / execuções) permanece incerta. Por exemplo, um grande número de compilações pode ser explicado por diferentes razões: alguns alunos compilam após cada pequena modificação do código, enquanto outros podem ser mais reflexivos e não tentam compilar até o final (ESPAÑA-BOQUERA *et al.* 2017).

Após esta análise, independente de qual categoria um ambiente ou ferramenta se enquadre, é importante ressaltar que cada uma tem sua contribuição individual. Encontrar um ambiente ou ferramenta voltado para o ensino de programação que englobe todas as características e funcionalidades desejadas ainda é algo difícil. Contudo, a proposta de ambiente apresentada neste trabalho tenta englobar algumas funcionalidades e características que não foram encontradas nas ferramentas

apresentadas, como por exemplo um recurso de depuração de erros que permita ao aluno desenvolver a sua própria lógica de programação; isso foi possível através da integração com o CFacil.

Além disso, o professor poderá acompanhar os alunos continuamente, por intermédio da visualização dos resultados das atividades. Dessa maneira, permite-se monitorar o número de compilações, a quantidade total de erros e acertos cometidos pelos discentes em cada exercício e um *status* de desempenho que permite verificar quais alunos estão obtendo sucesso ou necessidade de uma intervenção, agilizando o *feedback* necessário para o estudante reavaliar a sua solução. É também disponibilizado via *e-mail* um relatório geral, que contém um registro de todos os erros e acertos e ocorrências de cada erro apresentados por cada um dos alunos durante as atividades em laboratório, permitindo realizar análises estatísticas sobre o nível de aprendizado dos estudantes durante as aulas. Enfim, pretende-se que a ferramenta proposta no presente trabalho possua um diferencial das já existentes, podendo agregar um fator motivacional aos alunos, e apresentar-se como uma alternativa inovadora para auxiliar o professor no ensino dos algoritmos e programação.

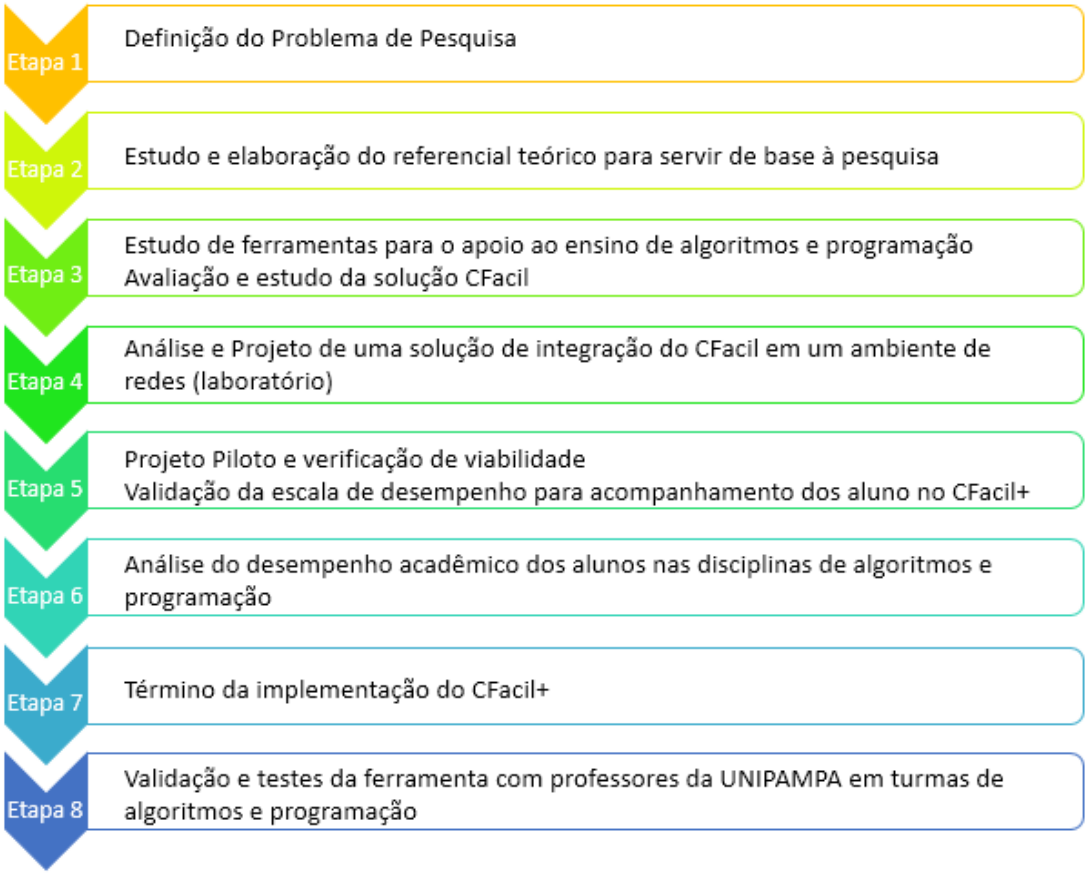
3 METODOLOGIA

Para que a pesquisa atinja seus objetivos, será preciso adotar um método científico que permita o desenvolvimento deste estudo de maneira a sistematizar o processo de pesquisa em etapas e mostrar os procedimentos a serem adotados em cada uma delas. Desta forma, ao analisar os métodos descritos na literatura, a pesquisa realizada neste trabalho constitui-se como aplicada e exploratória. Objetivou-se, com ela, propor uma solução para auxiliar o docente a acompanhar as atividades de programação desenvolvidas em laboratório, permitindo observar/registrar de forma individual as dificuldades apresentadas pelos alunos em sala de aula.

De acordo com a forma de abordagem, este estudo, segundo Wainer *et al.* (2007), constitui-se numa análise qualitativa com o objetivo de avaliar a aplicabilidade do sistema no monitoramento do desempenho dos estudantes durante as práticas de programação, validando as propostas apresentas neste trabalho. Quanto aos procedimentos técnicos, a pesquisa classifica-se como experimental, pois baseia-se em um estudo de caso aplicado a alunos de turmas de Algoritmos e Programação na UNIPAMPA, que permitiu a elaboração de uma escala para classificação do desempenho dos estudantes utilizada no sistema de monitoramento que é proposto neste trabalho.

Segundo Fonseca (2002) a pesquisa pode também ser considerada como bibliográfica, já que implica no estudo de artigos, teses, livros e outras publicações usualmente disponibilizadas por editoras e indexadas. Tendo em vista a metodologia adotada para o desenvolvimento deste trabalho, propôs-se um conjunto de ações, organizadas formalmente em etapas, por meio de atividades específicas, com o intuito de apontar um método efetivo para o acompanhamento das atividades de programação pelos professores. Um resumo de cada uma destas etapas é apresentado a seguir, na Figura 11.

Figura 11 - Etapas da Metodologia



Fonte: Próprio autor (2018)

A primeira etapa consiste na definição do problema de pesquisa com base nas dificuldades reportadas pelos professores da Universidade Federal do Pampa – UNIPAMPA, no ensino das disciplinas introdutórias de programação (lógica, algoritmos e programação). Face aos aspectos levantados, buscou-se, então, resposta à pergunta já descrita em sessões anteriores, a qual conduziu este trabalho.

Na etapa seguinte, foi realizado um estudo do estado da arte com base em alguns repositórios e periódicos, dentre eles, destacam-se: IEEE, ACM, SCIELO, RENOTE, RBIE, entre outros. Com base neste estudo foi elaborado um referencial teórico em relação ao ensino de programação e suas dificuldades.

Em relação à terceira etapa, foi feito um levantamento das ferramentas voltadas ao ensino de algoritmos, elucidando as soluções atualmente utilizadas e tendências futuras que podem ser empregadas pelos professores para o monitoramento dos alunos em sala de aula. Paralelo a isso, uma análise da solução CFacil foi realizada, estando descrita na Seção 4.1 do próximo capítulo.

A partir deste estudo, na quarta etapa, foi elaborada uma proposta de solução de integração da ferramenta CFacil em um ambiente de laboratório. Também foram avaliadas e estudadas as formas de integração desta solução em um ambiente de rede (cliente servidor).

O passo seguinte foi a implementação de um projeto piloto, que consistiu na criação de um modelo inicial com intuito de avaliar a viabilidade da proposta em um ambiente de rede. Com o desenvolvimento do projeto piloto pretendeu-se experimentar as técnicas aplicadas na construção do CFacil+ de forma a validar e adequar possíveis falhas que surgiram. Para o desenvolvimento da ferramenta, foi utilizada uma linguagem de programação orientada a objetos – Java –, por ser a base de praticamente todos os tipos de aplicações em rede (JAVA, 2017). Nesta etapa, também verificou-se que, para fornecer dados de referência em uma escala de indicação de desempenho dos alunos na ferramenta, seria necessário realizar uma validação desta escala a partir de um estudo de caso. Logo, este estudo foi aplicado para três turmas da disciplina de algoritmos e programação, onde foi distribuído um formulário para que os alunos pontuassem a quantidade de compilações e erros ao solucionar quatro exercícios, variando do nível mais fácil ao nível mais avançado. Uma descrição mais detalhada do desenvolvimento desta escala é apresentada na Seção 4.4.3.

Posteriormente, após finalizado o desenvolvimento da ferramenta, esta foi colocada em teste junto aos professores de algoritmos e programação da UNIPAMPA. A aplicação foi utilizada por aproximadamente 3 aulas de 2 períodos por aula. Neste processo, buscou-se analisar os resultados obtidos pelos docentes através do monitoramento do desempenho dos alunos durante as atividades práticas de programação por meio do CFacil+.

4 DESENVOLVIMENTO

O objetivo deste tópico é demonstrar a concepção da proposta de aplicação para o acompanhamento das atividades dos estudantes, com base no CFacil. Na Seção 4.1 é apresentado uma análise da ferramenta CFacil e um melhor detalhamento sobre a integração ao sistema de monitoramento apresentado neste estudo. A seguir, na Seção 4.2 será exposto o processo de modelagem do CFacil+ e os casos de uso da ferramenta. Na Seção 4.3 são apresentadas as principais técnicas adotadas para o desenvolvimento do CFacil+ e a especificação de suas funcionalidades quando comparado a ferramentas para identificação e correção de erros de compilação em C que já foram utilizadas em disciplinas de algoritmos e programação na UNIPAMPA. Na Seção 4.4 serão abordados os detalhes da implementação da ferramenta proposta. Por fim, na Seção 4.5 são descritos os testes e os resultados finais da aplicação do CFacil+ em aulas de Algoritimos e Programação.

4.1 Análise da ferramenta CFacil e a proposta da solução de monitoramento

A ferramenta desenvolvida por Gomes et al. (2016) é um analisador de códigos em linguagem C, e tem como objetivo tornar a depuração dos mesmos mais atraente ao aluno, através da apresentação de mensagens de erros no idioma português. Com esta solução, o estudante, em seus primeiros contatos com a linguagem de programação, pode ter maior facilidade na abstração da complexidade envolvida na correção de erros durante o processo de programação. Estimula-se, desta forma, a construção de conhecimento sobre a lógica envolvida na implementação de soluções computacionais para problemas simples. Foi possível perceber que o CFacil está focado nas dificuldades encontradas pelo aluno na disciplina de algoritmos e programação, não dando atenção aos problemas enfrentados pelo professor no ensino individualizado necessário nesta disciplina.

O CFacil realiza uma análise recursiva descendente sobre o código a ser analisado. Um analisador recursivo descendente é uma coleção de funções recursivas que processam uma expressão. Ao final desta análise a ferramenta de software mostra a saída com os erros na tela para o aluno, ao mesmo tempo gera um arquivo de *logs*, capturando a saída que foi enviada para o terminal. O arquivo de logs gerado apresenta padrões em sua construção, como armazenamento de data e hora da compilação, a frase “Erro na linha x”, e um contador de erros que é apresentado ao

final de cada compilação. Ao total 32 mensagens de erros podem ser apresentadas. Porém, por ser uma ferramenta para alunos iniciantes, não contempla análises para ponteiros e estruturas, conteúdos que não são apresentados nas disciplinas de algoritmos e programação.

Com os arquivos de logs que são gerados durante as compilações com o CFacil, a organização e a disposição interna das informações nestes arquivos tornam possível gerar estatísticas sobre as dificuldades que os alunos estão encontrando na prática de programação. A seguir, esses dados são apresentados ao professor, de maneira que ele consiga acompanhar as atividades individuais realizadas por cada estudante e, também de forma centralizada, de toda a turma. Desta forma, o professor pode analisar o desempenho destes, sendo possível intervir em atividades ou conteúdos que um aluno, ou mais de um, estejam com dificuldades.

A proposta de solução para o monitoramento das atividades dos alunos tem como propósito coletar os logs de erros. Estes são emitidos através da ferramenta CFacil e são analisados para realizar uma correlação geral da quantidade de compilações, erros e acertos, bem como a descrição de cada erro cometido em cada exercício que será compilado pelos alunos durante as tarefas de programação. Possibilita-se, com esses procedimentos, que o professor consiga acompanhar as dificuldades dos estudantes através do monitoramento destas informações que pode ser realizado através da configuração de uma rede cliente e servidor. Essa conexão disponibiliza uma infraestrutura para que o docente consiga acompanhar as tarefas propostas aos estudantes e identificar cada um deles em sala de aula. Dessa forma, o professor conseguirá reconhecer as dificuldades e os erros mais comuns dos discentes na aprendizagem inicial da programação, acompanhando os passos do estudante dentro do ambiente e registrando seu progresso por meio das atividades a serem desenvolvidas. Garante-se, desta forma, a proposição e ajuste das estratégias didáticas adotadas, que visam um aprimoramento da experiência de ensino e aprendizagem na disciplina de Algoritmos e Programação.

4.2 Modelagem do sistema

Primeiramente, foi realizado um levantamento dos requisitos do sistema (Apêndice I), com objetivo de delimitar o escopo do conjunto de funcionalidades que

o sistema iria prover aos usuários. Em seguida, foram elencadas as principais técnicas para modelagem do sistema baseado nos princípios da engenharia de *software*.

A descrição dos requisitos essenciais para o desenvolvimento da ferramenta apresentada neste trabalho serão descritos a seguir e foram divididos em requisitos funcionais e não funcionais.

4.2.1 Requisitos funcionais

O objetivo principal do sistema é possibilitar ao professor um controle do desempenho dos alunos durante a realização das atividades práticas de programação em sala de aula, através da integração do CFacil a um ambiente de rede cliente e servidor. No lado cliente da aplicação, a ferramenta deve fornecer funcionalidades como:

- Instrumento de coleta de dados – Permitir a compilação de códigos em linguagem C, através do analisador CFacil, aos alunos e gerar um relatório de erros a partir das compilações dos programas desenvolvidos durante as práticas de programação em sala de aula.
- Monitoramento dos logs de erros – O sistema deverá ser capaz de realizar o monitoramento das máquinas dos alunos a partir da especificação do número das suas máquinas e do diretório raiz da aplicação Cliente, onde será armazenado o *log* de erros gerado através das compilações. Assim, as alterações deverão ser detectadas.
- Análise dos logs de erros – Identificar o aluno, pelo nome e pela numeração da máquina a qual ocupa em sala de aula, contabilizar a quantidade total de compilações realizadas, a quantidade total de erros e acertos, a descrição e a ocorrência dos erros cometidos em cada exercício, gerando um outro relatório contendo estas informações para ser enviado ao servidor. Simultaneamente, este relatório deverá ser monitorado, pois a cada nova compilação este arquivo deve ser atualizado com novas informações referentes a execução das tarefas pelos estudantes.

  Já no lado servidor da aplicação, o principal requisito visa proporcionar ao professor o monitoramento contínuo do ambiente computacional utilizado pelos estudantes em sala de aula através de uma:

- Interface para acompanhamento dos alunos – Possibilitar o monitoramento das atividades que estão sendo desenvolvidas pelos alunos durante as práticas de programação, através de uma tela que exibe ao professor os seguintes dados: o nome dos alunos, a identificação das máquinas, o nº total de compilações, a quantidade total de acertos e erros e uma escala de indicação de desempenho.

4.2.2 Requisitos não funcionais

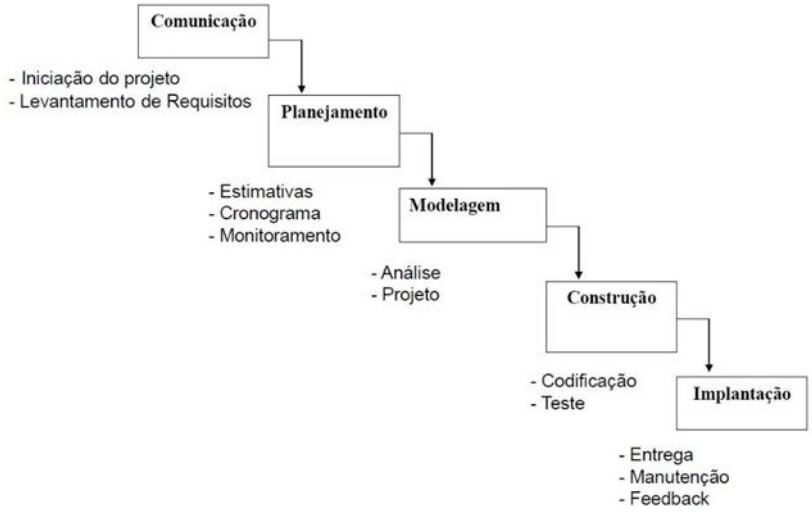
Conforme Sommerville (2011), os requisitos não funcionais se refere às restrições ou funções do sistema. Sendo assim os requisitos não funcionais do sistema devem ser:

- Usabilidade – As interfaces entre os processos das aplicações cliente e servidor devem ser intuitivas, possibilitando o fácil entendimento e uso da ferramenta.
- Desempenho – O servidor deve ser capaz de receber várias requisições dos clientes ao mesmo tempo para recebimento e apresentação de relatórios, permitindo o acompanhamento dos estudantes em sala de aula.
- Integridade – Todos os relatórios gerados através das compilações realizadas pelos alunos e que são monitorados pelo sistema devem ter sua integridade garantida a fim de eliminar qualquer inconsistência existente para o efetivo monitoramento dos discentes.

4.2.3 Modelagem do CFacil+

Para desenvolver a ferramenta proposta neste trabalho, foi escolhida a metodologia de desenvolvimento em cascata, visto que era preciso realizar adaptações ou aperfeiçoamentos em um sistema já existente. Em outras palavras, necessitava-se de adaptar a ferramenta CFacil a um ambiente de rede cliente e servidor onde fosse possível ao docente acompanhar os alunos na execução de suas tarefas em sala de aula (PRESSMAN, 2011).

Figura 12 - Modelo Cascata



Fonte: Pressman (2011)

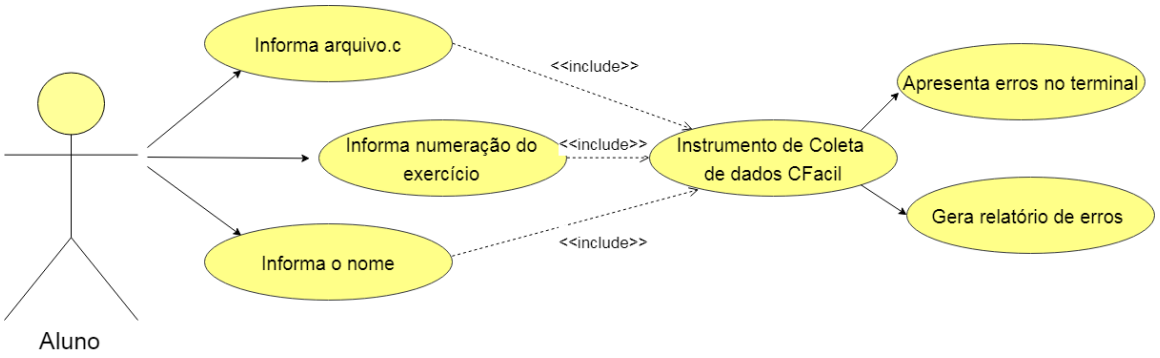
Conforme é apresentado na Figura 12, na fase de comunicação, foram levantadas todas as funcionalidades e restrições do sistema. Em geral, nesta etapa foi elaborado o documento de requisitos (Apêndice I), alguns dos quais, desenvolvidos na ferramenta CFacil, foram considerados. A documentação dessa ferramenta e até mesmo o seu código puderam ser reutilizados para desenvolver o CFacil+, que, além de adaptar-se às necessidades de aprendizado do estudante, está focado em auxiliar o professor em sua tarefa de dar acompanhamento adequado ao aluno, o que não era possível por intermédio do CFacil.

Na etapa de planejamento foram definidos os passos que o sistema deveria seguir, como a verificação das etapas para o seu desenvolvimento. Na etapa de modelagem os requisitos foram analisados, visando facilitar a criação do código fonte e sua respectiva implementação. Uma vez definidos os requisitos, faz-se necessário realizar a previsão e o estudo das tecnologias que a nova plataforma iria necessitar.

A etapa de construção configurou-se na implementação do sistema, ou seja, foi a fase em que ocorreu a codificação da ferramenta CFacil+ utilizando a linguagem de programação Java. Após esta etapa de codificação, o sistema foi testado utilizando um plano de testes, conforme o Guia para Elaboração de Documentos de Teste de Software (Apêndice II) baseado na Norma IEEE 829 e foi aplicado em uma turma da disciplina de algoritmos e programação da UNIPAMPA. Nesta fase também ocorreu o treinamento dos usuários em sala de aula. Por fim, foram realizadas correções de erros que foram descobertos com a utilização do programa pelos usuários finais.

Apresentados os processos para modelagem, a seguir será exposto o comportamento do sistema, frente às funções que o usuário pode utilizar através da especificação dos casos de uso, diagramas de sequência e os diagramas de classe da ferramenta. Essas representações foram baseadas em notação UML e geradas a partir da ferramenta online Draw.io[1]. O diagrama de Casos de Uso do CFacil+, conforme as Figuras 13 e 14, contemplam as principais funcionalidades da aplicação, demonstrando o comportamento do sistema, quanto à utilização da aplicação pelos alunos.

Figura 13 - Diagrama de Casos de Uso do aluno



Fonte: Próprio autor (2018)

Inicialmente, no lado Cliente CFacil+, onde estão os alunos, ocorre uma solicitação, ao estudante, do preenchimento com as seguintes informações: o nome do arquivo que deseja compilar, a numeração do exercício ao qual está sendo solucionado e o seu nome. Utiliza-se, para isto, o instrumento de coleta de dados proposto por Gomes *et al.* (2016) que foi readequado as necessidades do CFacil+.

O instrumento de coleta de dados faz a análise do código fonte enviado pelo aluno; após, é avaliado se não foi encontrado nenhum erro nesta análise. Em seguida, a aplicação Cliente CFacil+ realiza a coleta dos logs de erros gerados a partir deste relatório; se forem encontrados erros é gerado um arquivo com um relatório de erros e apresentado na tela os erros ao usuário.

Estes relatórios são monitorados a cada 10 segundos de acordo com as compilações que são efetuadas por intermédio das atividades desenvolvidas pelos alunos. Simultaneamente, o sistema analisa os *logs* de erros que são coletados e

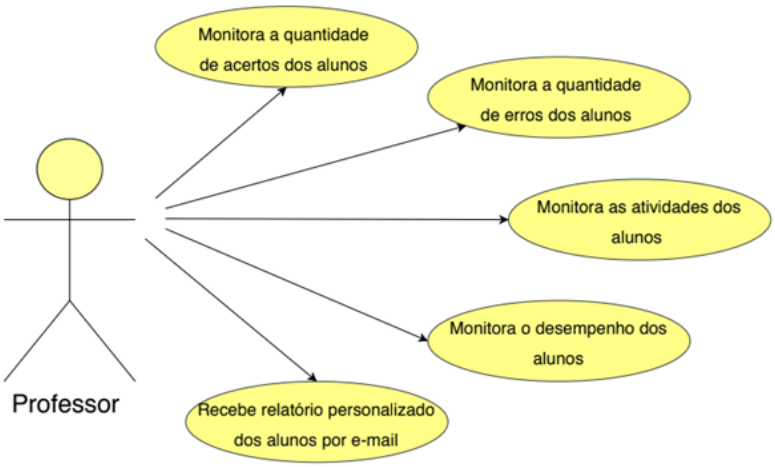
[1] Disponível em: https://www.draw.io

começa a apurar a quantidade de compilações realizadas, de erros e acertos cometidos, bem como a descrição de cada erro na prática de cada exercício.

Para o envio destas informações, uma conexão com o Servidor CFacil+ é estabelecida, permitindo o envio de relatórios que contém a análise dos *logs* de erros coletados e analisados pelo CFacil+. Com isso, como pode ser observado nos casos de uso da Figura 14, o professor poderá observar quais são os erros mais comuns cometidos pelos alunos. Então, estes relatórios são analisados de duas formas oferecidas pela ferramenta que foi implementada: a partir do monitoramento destes arquivos armazenados em um diretório ou através de uma interface, que contém uma tabela. Nesse quadro, estão dispostas as informações, tais como o nome do aluno, a identificação da máquina ao qual ocupa em sala de aula, a quantidade de erros e acertos e uma escala de desempenho que irá determinar a situação de cada um dos alunos com relação aos exercícios compilados.

Os alunos que apresentarem um menor desempenho na realização de suas atividades, ou seja, aqueles com uma maior quantidade de erros acumulada na realização dos exercícios ficaram no topo da tabela. Isso permite o acompanhamento contínuo de cada aprendiz e o tratamento personalizado a cada estudante para reverter o quadro de dificuldades constatadas. Além disso, nos relatórios que são disponibilizados ao professor é especificado a numeração das máquinas as quais os alunos ocupam, sendo possível identificar os estudantes em sala de aula com maior facilidade, a fim de estimulá-los a verificarem seus erros e corrigi-los. Quando todos os alunos estiverem finalizado suas atividades, um relatório personalizado contendo a quantidade de erros, acertos, compilações e descrição de cada erro em cada exercício pelos alunos poderá ser gerado e enviado via e-mail para o professor.

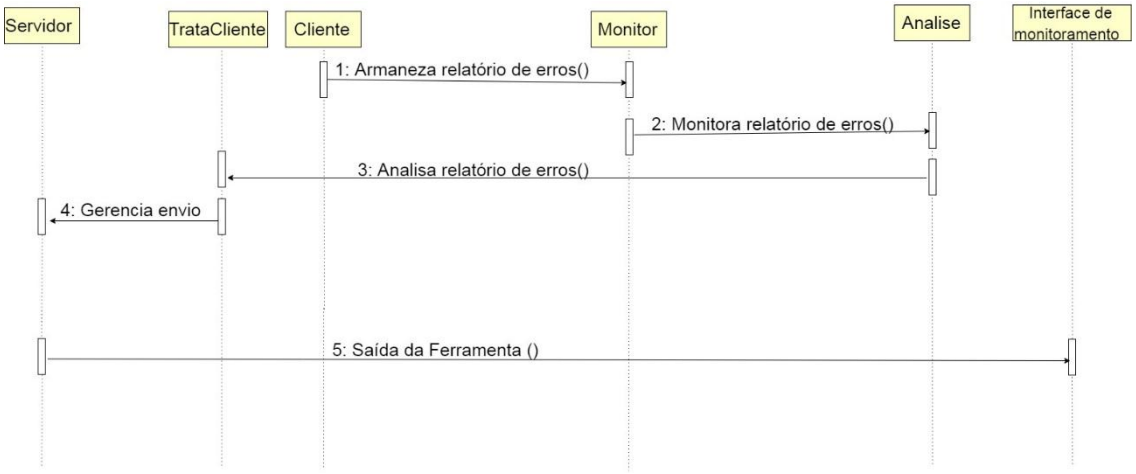
Figura 14 - Casos de Uso do Professor



Fonte: Próprio autor (2018)

Baseado nos casos de uso que foram especificados, foi possível construir um diagrama de sequência para determinar a ordem em que cada um dos eventos ocorre no sistema, bem como definir as mensagens que são enviadas e os métodos que são chamados. A Figura 15 apresenta o diagrama de sequência do CFacil+.

Figura 15 - Diagrama de Sequência do CFacil+



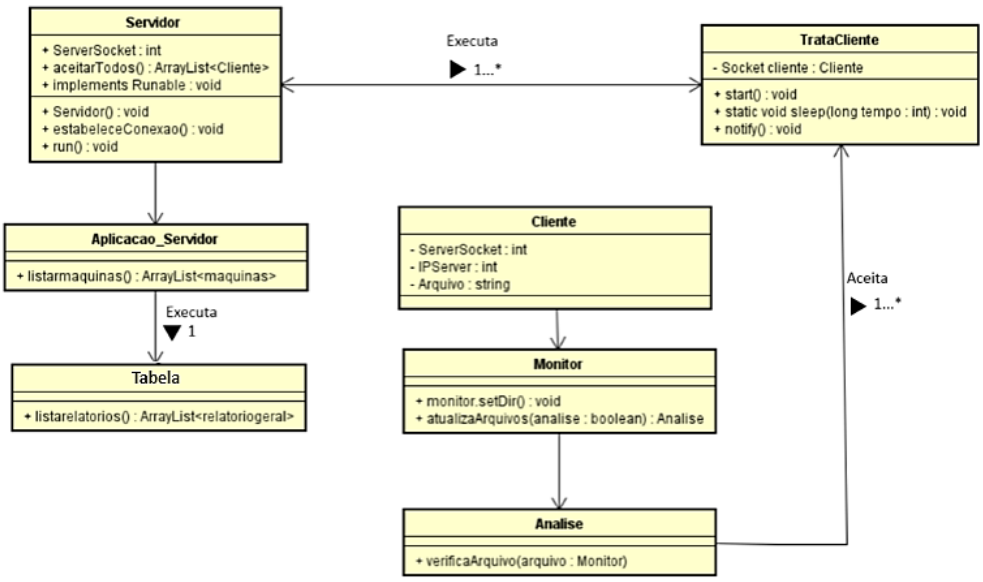
Fonte: Próprio autor (2018)

A Figura 15 mostra que cada vez que o “Cliente” armazena os *logs* coletados em um relatório, o sistema detecta a alteração a partir de um módulo da aplicação, intitulado “Monitor”. Esse envia aquelas informações à Análise para computação dos dados que serão encapsuladas em um relatório geral a ser enviado para a

“TrataCliente”, que é responsável por gerenciar o envio destes arquivos ao “Servidor”, atendendo mais de uma conexão simultaneamente por meio da implementação de *threads*. O sistema, então, irá confirmar o envio de mais um arquivo e, após estas etapas, na “Interface de Monitoramento” será apresentado em uma tabela a listagem de todos os alunos que estão conectados. A referida tabela conterá o nome de cada aluno, identificação da máquina, quantidade de erros, acertos, compilações, e uma escala de desempenho para acompanhamento do professor. Além disso, os arquivos transmitidos via cliente para o servidor são armazenados em um diretório.

Seguindo a descrição da modelagem do projeto piloto implementado, desenvolveu-se um diagrama de classes de implementação apresentado na Figura 16. Pretendeu-se, dessa forma, apresentar como as classes implementadas interagem entre si a fim de especificar a responsabilidade de cada uma delas na realização das operações solicitadas pelos atores.

Figura 16 - Diagrama de Classes



Fonte: Próprio autor (2018)

Na classe servidor, através de um ServerSocket, é possível, atender os clientes via rede e em determinada porta. A classe executa uma ou mais vezes a TrataCliente, responsável por realizar o tratamento de cada uma das solicitações para o envio dos relatórios gerais através de *threads*, deixando assim o servidor livre para esperar por novas conexões. A classe Cliente, por sua vez, informa o endereço IP para que cada

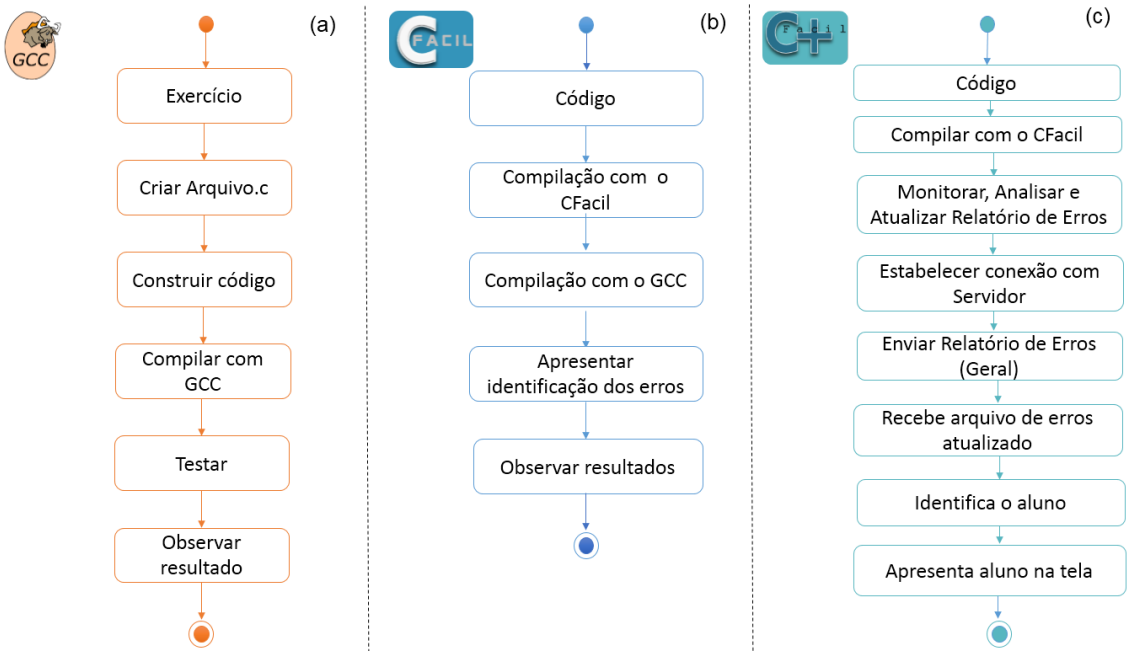
máquina seja identificada na rede; também é especificado a porta que irá se comunicar com o servidor. Além disso, a classe contém a especificação do diretório que irá armazenar os relatórios de erros que vão sendo gerados a partir da compilação dos códigos em linguagens C pelos alunos.

Os arquivos são monitorados, a cada dez segundos, a partir da atualização das informações contidas nos relatórios com a classe Monitor, que são enviadas para Análise. Nela, acontece a verificação dos arquivos e a identificação dos alunos no sistema. Logo, a classe contabiliza o número total de compilações, erros, acertos e informa a descrição dos erros que foram cometidos em cada compilação pelo número do exercício informado pelos alunos. Todas estas informações são computadas em um relatório geral, que é enviado para a classe Trata Cliente, onde todo processo é realizado novamente. No momento em que se pretende verificar os relatórios que são gerados pelo CFacil+, a classe Aplicação_Servidor lista cada um dos alunos a partir do seu respectivo nome e número da máquina em uma Jtable, que apresentará as informações contidas nos relatórios de cada um dos estudantes e um *status.*

4.3 Projeto CFacil+

O principal objetivo deste tópico é demonstrar as principais técnicas para o desenvolvimento do CFacil+ e elucidar o funcionamento da aplicação de forma geral. Tem-se aqui, como ponto de partida, a especificação do propósito das seguintes ferramentas: GNU *Compiler Collection* (chamado usualmente por GCC), CFacil e CFacil+, que está sendo proposta neste trabalho, apresentadas na Figura 17.

Figura 17 - Ferramentas para identificação e correção de erros de compilação em linguagem C



Fonte: Próprio autor (2018)

A estrutura de utilização do GCC, Figura 17 (a), mostra os passos da aplicação para a compilação de um código em linguagem C. Primeiramente, é preciso escrever o código fonte em um editor de texto qualquer e salvar com a extensão.c. Logo após, deve-se compilar o código com o compilador GCC. Se o programa consistir de um único arquivo, é preciso executar o comando $ gcc nome_do_arquivo.c -o nome_do_arquivo no terminal. Onde nome_do_arquivo.c é o nome do arquivo que contém o código. Os outros dois parâmetros, -o nome_do_arquivo, indicam respectivamente, o arquivo de saída do compilador e -o arquivo executável que conterá o programa. Se o programa não apresentar erros, nenhuma mensagem será apresentada pelo compilador e será possível executá-lo. Basta digitar ./ antes do nome do programa (por exemplo, ./nome_do_arquivo). Caso contrário, o compilador irá exibir através de mensagens onde está o erro do programa. Será necessário corrigir o erro no código fonte e compilá-lo novamente.

Com relação a estrutura de utilização do CFacil, proposta por Gomes *et al*. (2016), na Figura 17 (b), ao ser executada, assim como no compilador GCC, é solicitado ao usuário um nome para o arquivo de saída e o nome do arquivo que se deseja compilar. Após isto, o arquivo enviado pelo usuário é compilado duas vezes, uma com a ferramenta CFacil e outra com o compilador GCC, caso não sejam

encontrados erros na primeira compilação com o CFacil. Desta forma, se forem encontrados erros nesta compilação, são exibidas mensagens no terminal de compilação para análise do usuário e um arquivo de erros. Se não forem encontrados erros, o script cria o arquivo executável e o usuário pode executar o programa.

No entanto, a ferramenta CFacil+, proposta neste trabalho, quando aplicada nas aulas práticas de algoritmos e programação, além de permitir a correção de erros de compilação em linguagem C para os alunos, também oferece ao professor uma visão geral e atualizada do desempenho de todos os estudantes em uma sala de aula. A partir do *feedback* fornecido, é possível ao docente identificar os focos de maior dificuldade, conseguindo, assim, atender aos alunos que demandam maior ajuda e oferecendo um aporte a ferramenta CFacil através de sua integração a um ambiente de rede (cliente e servidor).

No CFacil+, Figura 17 (c), a partir da análise dos relatórios, emitidos ao sistema através da análise dos resultados pelo CFacil, o professor, no lado servidor da aplicação, poderá verificar as dificuldades que os estudantes estão apresentando durante a programação de algoritmos em linguagem C em cada uma das tarefas propostas em sala de aula. Isto é possível, através da apresentação de um relatório individualizado de cada estudante presente na turma. Os alunos, no lado cliente da aplicação, são identificados no sistema através do seu nome e identificação da máquina a qual ocupam em sala de aula, a cada nova compilação, é especificada a quantidade total de erros, acertos e compilações cometidas pelos discentes. A ferramenta também oferece um indicador de desempenho dos estudantes que varia de acordo com a quantidade de erros acumulados durante a execução de suas atividades.

O processo de registro do aluno no lado cliente da aplicação, implica nele informar ao sistema o seu nome e, para a compilação dos algoritmos implementados, é preciso que o estudante especifique o nome do arquivo “.c”, a numeração do exercício que está sendo solucionado de forma algorítmica em linguagem C e a identificação da máquina. A cada nova compilação, o aluno deve seguir informando ao sistema em qual exercício ele está, se houve troca ou não de atividade.

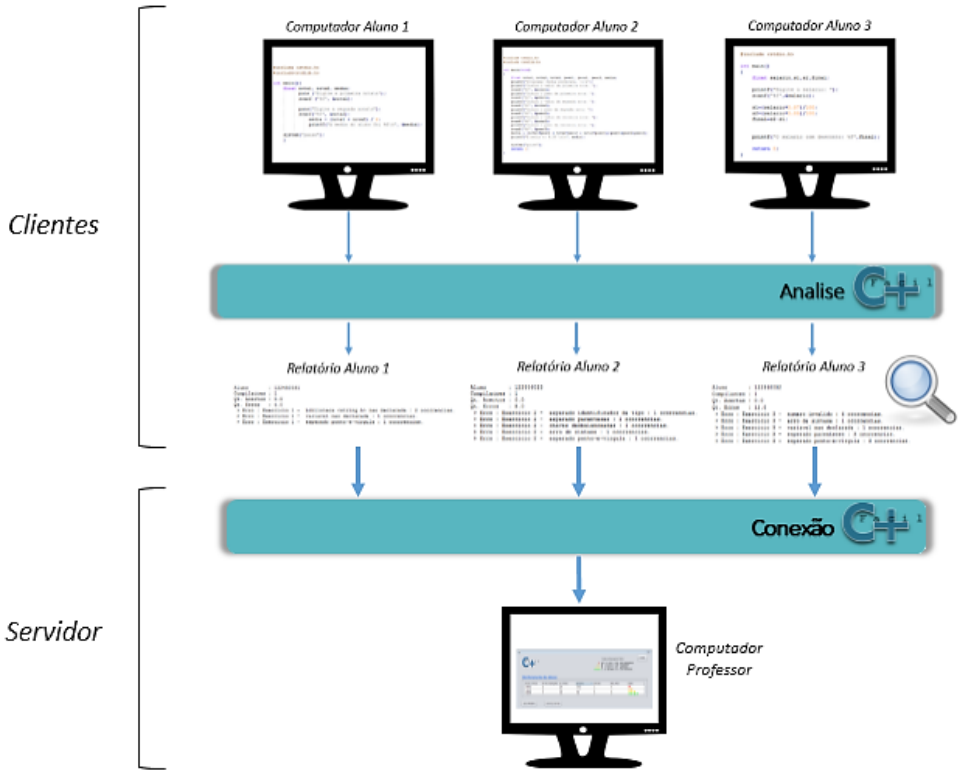
Dentre as funcionalidades previstas para o software, estão a capacidade de classificar as informações de logs coletadas, gerar relatórios a partir da análise das

informações recebidas pelo sistema e realizar o monitoramento em das atividades desenvolvidas pelos alunos.

Para que estes recursos possam ser oferecidos por intermédio do CFácil+, após ser estabelecida uma conexão com o servidor, o sistema envia ao professor as informações do cliente (aluno) onde contabiliza o total de erros, acertos, número de compilações e o tipo de erro do aluno a partir de cada um dos relatórios emitidos pelos estudantes. Um novo arquivo é gerado para ser enviado ao docente. Neste arquivo, é mostrada uma correlação final de todos os erros e acertos e ocorrências de cada erro apresentados em cada exercício pelos alunos conectados na rede.

A Figura 18 apresenta a arquitetura (cliente e servidor) do CFacil+. Na aplicação, o sistema monitora as compilações a partir da classe Monitor implementada no CFacil+, e gera estatísticas de erros através da classe Analise, que permitem o professor acompanhar a evolução dos alunos. A classe Analise identifica o aluno através da identificação do seu nome, computa a quantidade total de compilações realizadas, a quantidade total de erros e acertos, informa a descrição e a ocorrência dos erros cometidos em cada exercício e gera um relatório de erros do aluno com estas informações. Este arquivo, é atualizado a cada nova compilação e monitorado a cada 10 segundos no diretório da máquina do aluno que arquiva o relatório de erros.

Figura 18 - Arquitetura (cliente servidor) CFacil+



Fonte: Adaptado de Lopes *et al.* (2017)

Para o envio dos relatórios emitidos através da classe Analise para o servidor é preciso estabelecer uma conexão entre o computador do aluno e o computador do professor. Isto foi possível a partir da implementação de sockets, que constituem canais de comunicação entre os dois processos cliente e servidor da aplicação. A partir dessa conexão, é preciso especificar o endereço IP do processo servidor em cada um dos clientes. Posteriormente, os dados são enviados via rede cliente e servidor e são apresentados em uma tela ao professor. A fim de permitir a conexão de todos os alunos ao sistema de monitoramento (Cliente/Servidor), adotou-se a tecnologia de *threads*, garantindo dessa forma um padrão eficiente e confiável de comunicação entre as aplicações.

4.4 Implementação do CFacil+

Nesta seção serão apresentados detalhes da implementação da ferramenta proposta. Para uma melhor compreensão será abordado cada elemento que compõe a estrutura do CFacil+, sendo estas as aplicações cfacilplus_cliente e

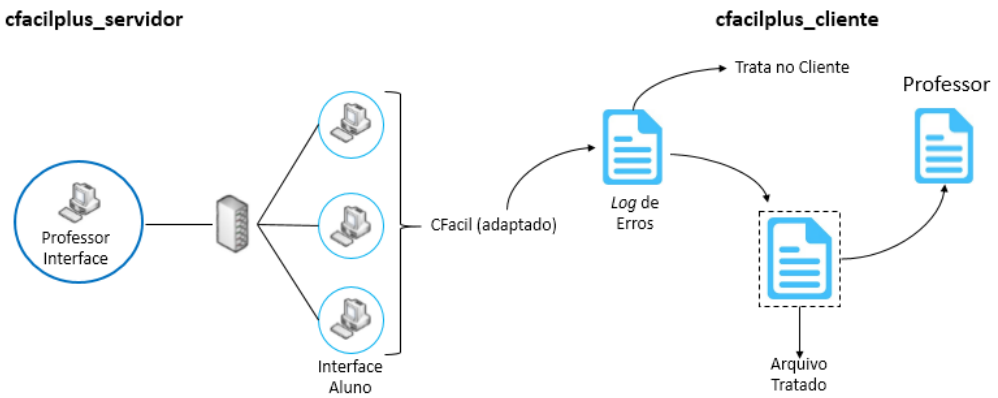
cfacilplus_servidor, implementadas na plataforma Netbeans IDE 8.1 e desenvolvidas em linguagem Java.

As aplicações que foram desenvolvidas, além de utilizarem o mecanismo *socket*, utilizam também os componentes GUI (*graphical user interfaces* – interfaces gráficas para os usuários), que fornece os recursos necessários para criação de interface. Deitel (2003) diz que interfaces são muito importantes na criação de qualquer software, já que é através delas que o usuário pode aprendê-lo mais rapidamente e utilizá-lo melhor.

Para Correia (2006), o mecanismo de comunicação *socket*, é o mais utilizado entre aplicações e permite seu uso através do Modo Orientado à Conexão, que funciona com o uso do protocolo TCP/IP, fornecendo serviços confiáveis, sem perda de dados na rede e ordem dos pacotes e ainda possibilita o uso de *DataStreams*.

A Figura 19 apresenta uma visão geral do CFacil+, onde no lado cliente ficam as aplicações cfacilplus_cliente responsável por tratar e enviar ao servidor os *logs* de erros gerados através da ferramenta CFacil, implementada por Gomes *et al.* (2016) e que foi integrada ao CFacil+ para possibilitar o acompanhamento dos alunos ao professor no lado servidor da aplicação através do cfacilplus_servidor.

Figura 19 – Aplicação CFacil+



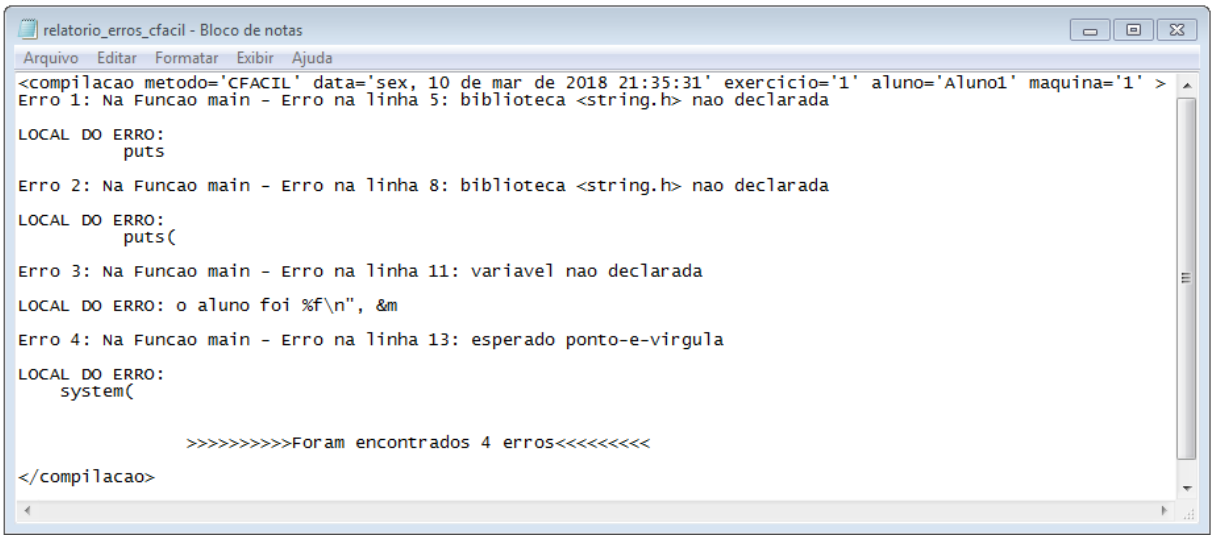
Fonte: Próprio autor (2018)

Uma vez evidenciado os objetivos e as principais particularidades para a construção da ferramenta, será apresentado detalhadamente o processo de implementação das aplicações cfacilplus_servidor e cfacilplus_cliente, referentes à proposta. A finalidade na descrição desta escrita é mostrar o desenvolvimento dos códigos mais significativos para a implementação do CFacil+.

4.4.1 Implementação da aplicação cfacilplus_cliente

A aplicação cfacilplus_cliente é iniciada através de uma *thread* principal, que é executada quando o programa começa na implementação da classe Monitor.java. Paralelo a isso, com as compilações dos arquivos.c feitas pelos alunos, um *log* de erro em formato de arquivo do tipo “txt” é gerado (Figura 20) e armazenado no diretório raiz da aplicação Cliente.

Figura 20 - *Log* de Erro gerado após uma compilação



Fonte: Próprio autor (2018)

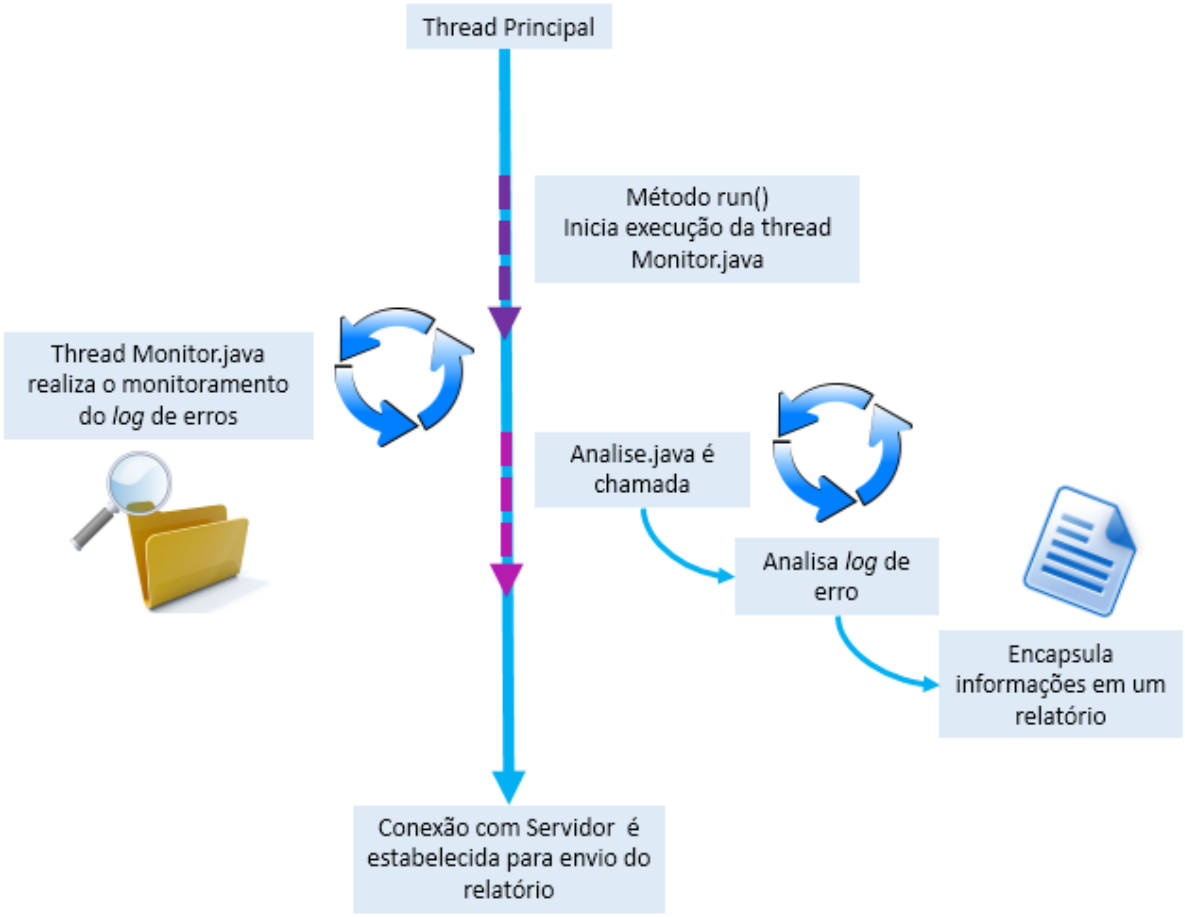
Tal arquivo é obtido por meio da utilização de uma variável do tipo FileFilter, uma classe que cria um filtro abstrato e seleciona, dentre arquivos dos mais diversos formatos, exclusivamente aqueles de extensão "txt". Em seguida, o método accept() é chamado, retornando um *boolean true* caso atenda à condição do filtro. O método listFiles() listará todos os arquivos presentes no diretório e retorna um vetor de *File*. Dado um intervalo de tempo, é realizada uma nova varredura na pasta para determinar se houve alguma modificação.

Conforme é apresentado na Figura 21, para que todos os procedimentos realizados sejam executados de maneira concorrente dentro da aplicação, cada um dos processos deve corresponder diretamente a uma *thread*[2]. A funcionalidade (programação) das *threads* é feita na implementação do método run(). Uma *thread*

[2]Definição: um thread é um fluxo único de controle sequencial dentro de um programa. Disponível em: http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/threads/threads1.html

fica monitorando o diretório onde está o *log* de erro, enquanto que a *thread* principal segue sua execução. Neste momento, a classe Analise.java é chamada e encapsula em um outro arquivo um relatório contendo as informações obtidas após a análise do *log* de erro.

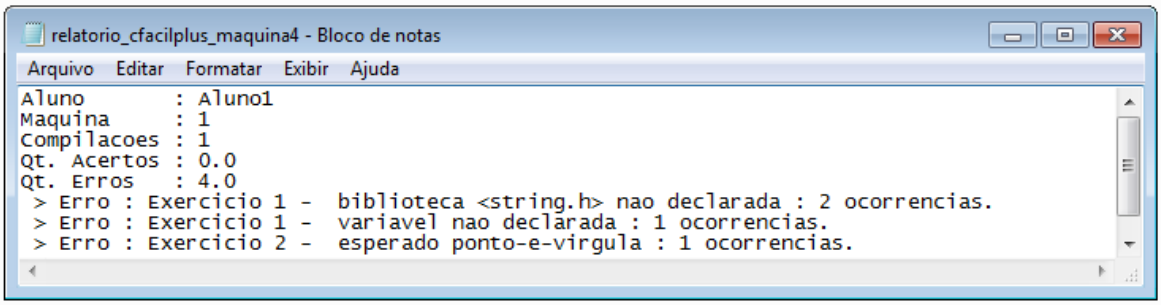
Figura 21 – Thread Monitor.java e Classe Analise.java



Fonte: Próprio autor (2018)

Neste relatório, apresentado na Figura 22, são passadas informações sobre a quantidade de compilações, erros e acertos cometidos, bem como a descrição de cada erro na prática de um determinado exercício pelo aluno. Isso ocorre toda vez que houver uma nova compilação. Em seguida, a partir da definição do endereço IP na aplicação, uma conexão com o servidor é estabelecida e este arquivo é enviado ao servidor.

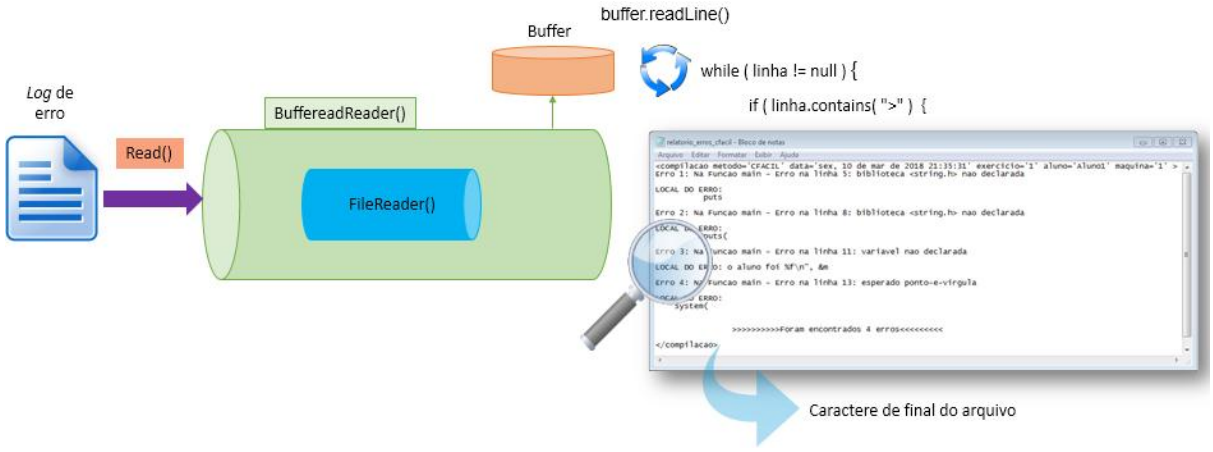
Figura 22 - Relatório de Erros enviado ao Professor



Fonte: Próprio autor (2018)

O funcionamento da classe Analise.java, como mostra a Figura 23, inicia a partir da leitura do *log* de erro com o método FileReader, que é associado a um fluxo de entrada de dados baseado em caracteres, através de um BufferedReader. Este método, por sua vez, armazena a entrada do *log* especificado em um *buffer* até encontrar o caráter do final do arquivo com a chamada do método buffer.readLine(), que é executado dentro de um laço de repetição *while* para ler todas as linhas do *log*. Dentro deste processo é declarado uma condição que utiliza o método contains(). O método contains() não tem como retorno uma nova *string*, mas sim um *boolean*. Ele avalia se a sequência de caracteres que está sendo buscada está contida na *string* informada como parâmetro para o método.

Figura 23 - Funcionamento da Classe Analise.java

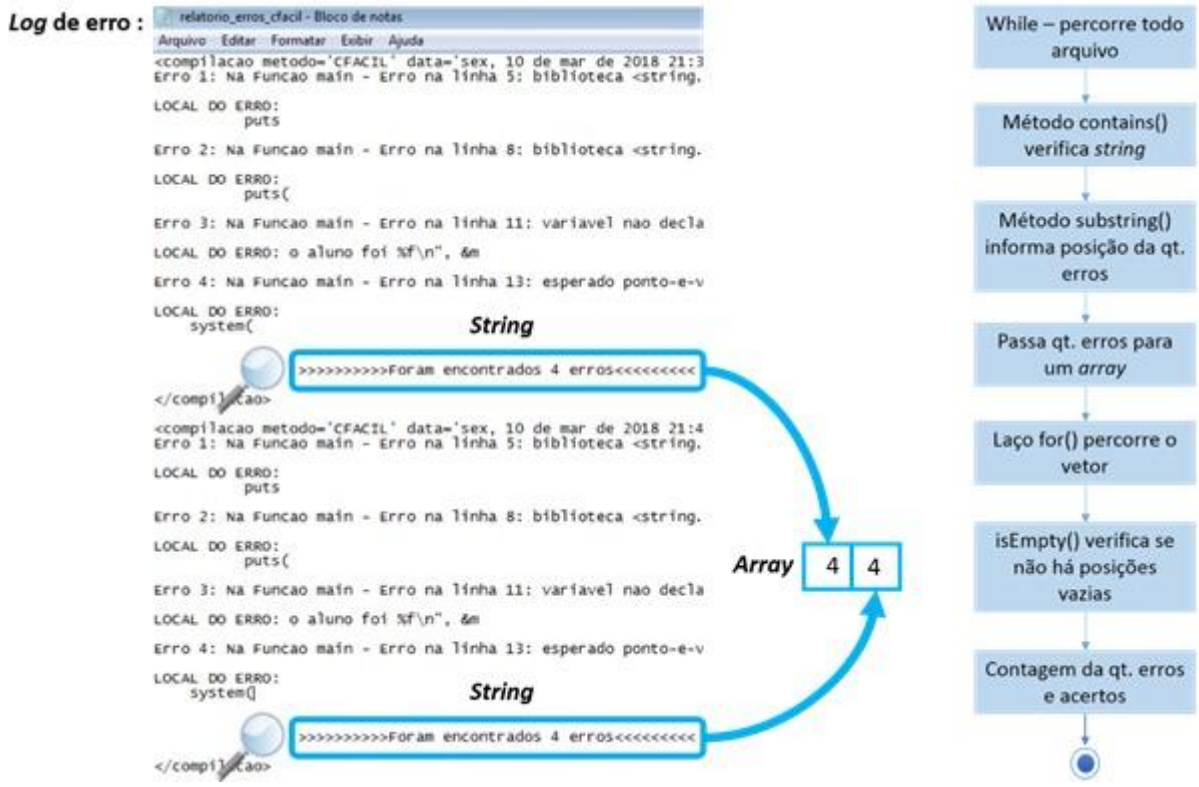


Fonte: Próprio autor (2018)

Para a contagem da quantidade de erros, como mostra a Figura 24, o método contains() verifica se a *string* inserida como parâmetro contém o conjunto de

caracteres formado por (>); então, é percorrido todos os possíveis erros referentes a cada uma das compilações efetuadas pelo aluno. Através do método split() é possível separar cada um dos erros que forem encontrados em um *array*. Assim, o método substring() informa a posição dos caracteres os quais correspondem a cada um dos erros, para que possam ser armazenados no *array*, e define a posição dos demais caracteres do *log* de erro, os quais não serão incluídos. Utilizando um laço de repetição *for*, todas as posições do *array* são percorridas, quando chega na última posição, todo o conjunto de possíveis erros são encontrados, logo é realizado o cálculo do somatório dos erros que foram armazenados.

Figura 24 - Contagem da Quantidade de Erros e Acertos

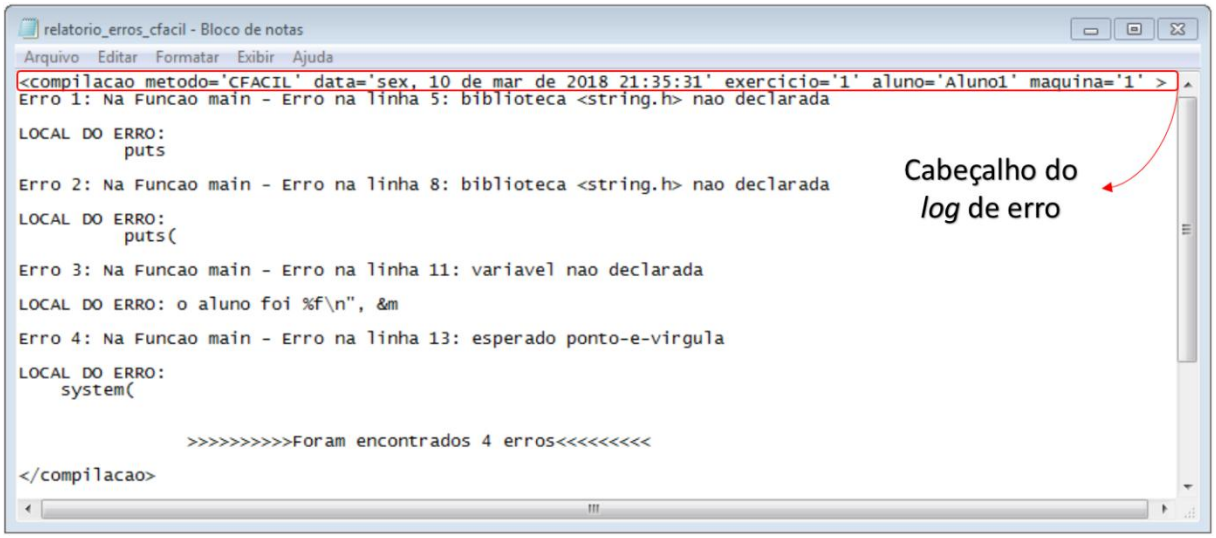


Fonte: Próprio autor (2018)

No entanto, se alguma das posições do *array* está com vazio, significa que dentre as compilações que foram realizadas, houveram acertos, ou seja a ocorrência de nenhum erro. Para isto, o método isEmpty() é utilizado, verificando se o caractere especificado como erro tem um valor vazio, então, somam-se as ocorrências de cada um dos caracteres vazios contidos no *array*. O resultado deste somatório corresponderá a quantidade total de acertos cometidos pelo aluno.

Após, realiza-se a análise do cabeçalho do *log* de erro que é mostrado na Figura 25, para contabilizar o número de compilações realizadas, utilizando para isto, novamente o método contains() onde é especificado a *string* “<compilação”. Então por meio de uma variável auxiliar, conta-se a ocorrência desta *string* durante a leitura do arquivo.

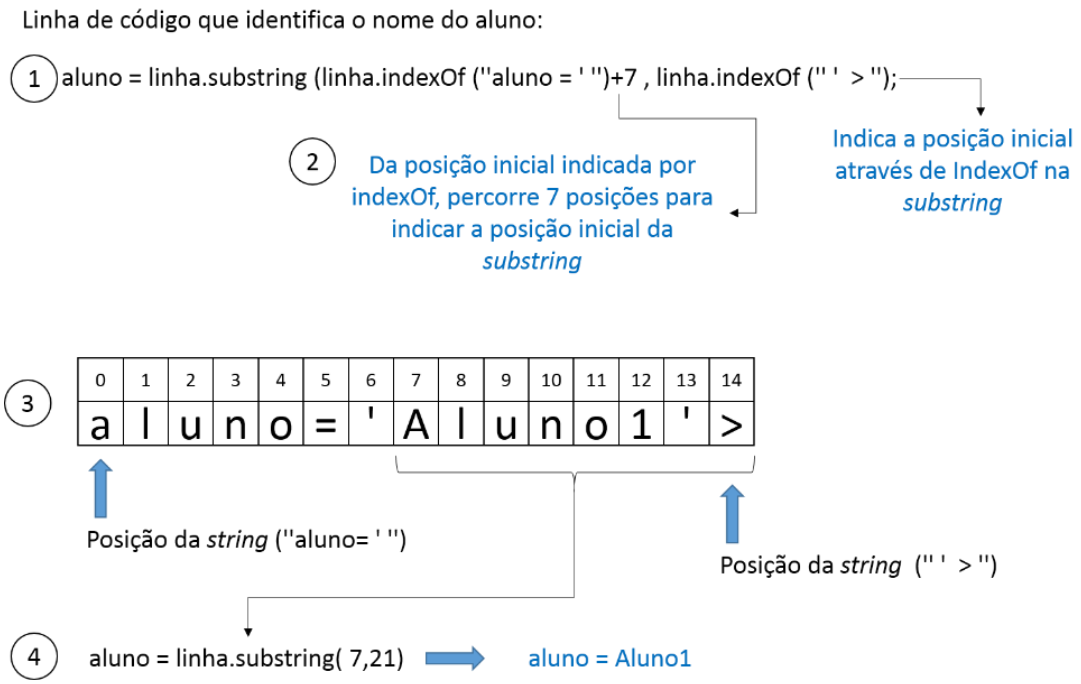
Figura 25 - Cabeçalho do *log* de erro



Fonte: Próprio autor (2018)

Conforme a Figura 26 apresentada a seguir, o nome do aluno e a numeração do exercício compilado são identificados através do método substring() referenciando-se as sequências de caracteres “aluno= ’ ” e “exercicio= ‘ ” no método indexOf() para buscar a posição inicial destas *strings* nas linhas do *log* de erro. O primeiro parâmetro, obrigatório do método substring(), é a posição que a *substring* deve iniciar (lembrando-se que *strings* sempre começam da posição 0) e o segundo parâmetro, é a posição final da *substring*. O caractere na posição inicial fará parte da *substring*, porém o caractere na posição final não faz parte da *substring*.

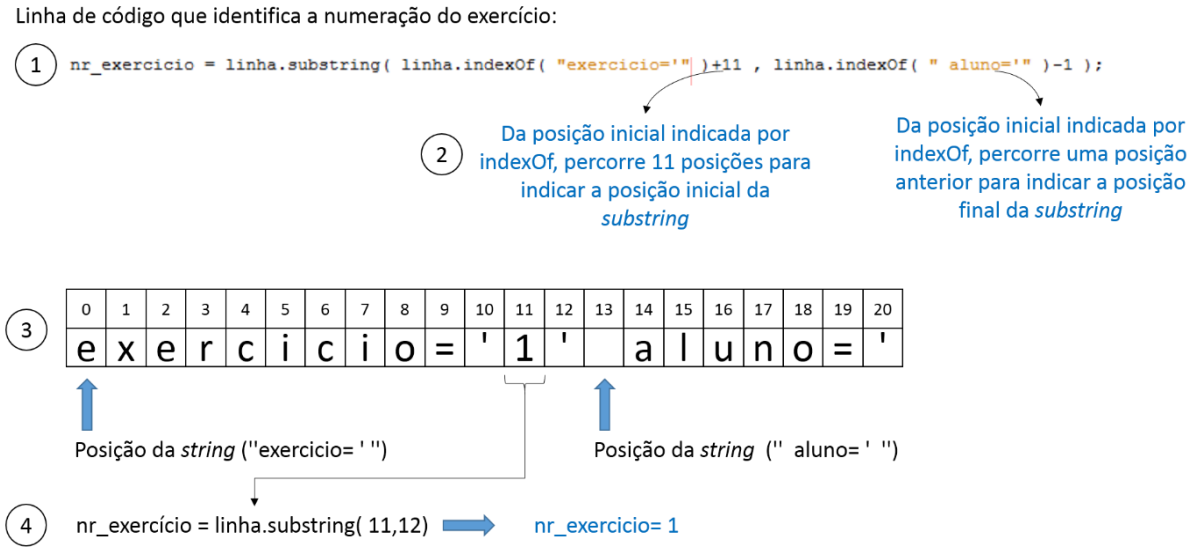
Figura 26 - Identificação do nome do aluno no log de erro



Fonte: Próprio autor (2018)

Para identificação da numeração do exercício, também são utilizados os métodos mencionados anteriormente. A Figura 27, demonstra como isto acontece.

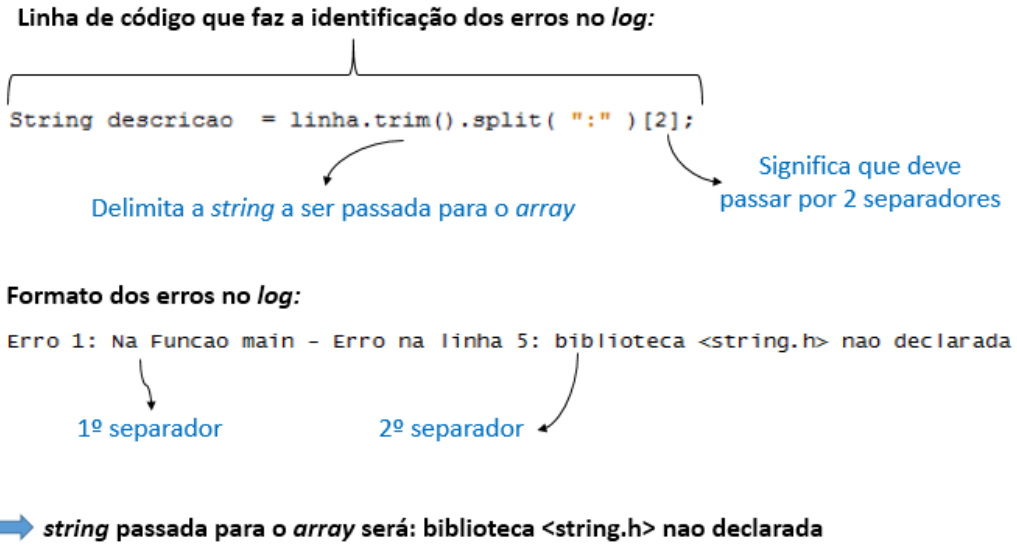
Figura 27 - Identificação da numeração do exercício no log de erro



Fonte: Próprio autor (2018)

As *strings* correspondentes a cada uma das descrições dos erros em um determinado exercício são detectadas por meio dos métodos contains() e split(). O *split* funciona da seguinte maneira: são passadas para ele uma *string* e um separador, e ele retorna um array, com a *string* desse separador dividido em cada posição, como é mostrado na Figura 28.

Figura 28- Identificação das descrições de cada erro por exercício



Fonte: Próprio autor (2018)

Posteriormente, para quantificar o número de ocorrências de cada erro cometido pelo aluno durante as suas compilações, estas *strings* podem ser comparadas através do método compareTo(), a comparação é feita letra a letra enquanto cada letra comparada resultar em 0 (equivalente) passa-se para a próxima letra.

Por fim, após serem efetuadas cada uma das operações descritas anteriormente, as informações referentes ao nome do aluno, quantidade total de compilações, quantidade de acertos e erros e a descrição da ocorrência de cada um dos erros, são organizadas em um arquivo e é armazenado em um diretório. Este arquivo, é atualizado a cada 10 segundos por meio da classe Monitor.java e pode ser enviado para a aplicação cfacilplus_servidor.

Para a utilização da ferramenta, no lado cliente da aplicação, como mostra a Figura 29, é indispensável que o aluno especifique previamente o arquivo.c a ser compilado e a numeração do exercício ao qual está resolvendo, pois é a partir disto

que o sistema identificar os erros e as compilações que foram cometidas na realização de cada uma das atividades durante as práticas de programação.

Figura 29 - Interação dos alunos com a ferramenta CFacil (adaptada)

Arquivo.c a ser compilado:

*exercicio1.c

```
1  #include <stdio.h>
2  #include<stdlib.h>
3
4  int main(){
5      float nota1, nota2, media;
6            puts ("Digite a primeira nota\n");
7            scanf ("%f", &nota1);
8
9            puts("Digite a segunda nota\n");
10           scanf("%f", &nota2);
11               media = (nota1 + nota2) / 2;
12               printf("A media do aluno foi %f\n", &media);
13
14     system("pause");
15     }
```

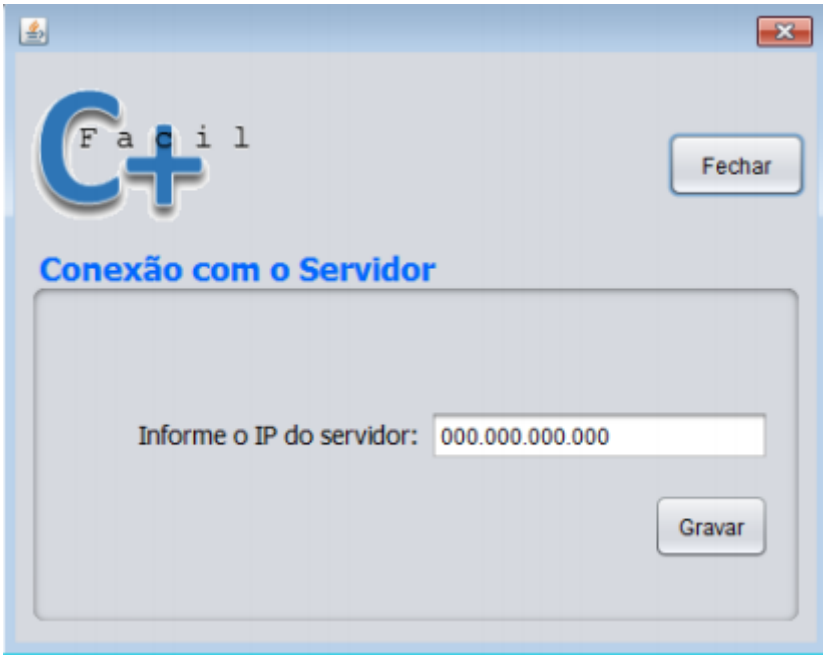
Interação com a ferramenta CFacil (adaptada):

```
Informe o arquivo de saida, o arquivo.c a ser compilado e o nr do exercicio:
exercicio1 exercicio1.c 1
Com o CFACIL

Erro 1: Na Funcao main - Erro na linha 5: biblioteca <string.h> nao declarada

LOCAL DO ERRO:
          puts

Erro 2: Na Funcao main - Erro na linha 8: biblioteca <string.h> nao declarada

LOCAL DO ERRO:
          puts(

Erro 3: Na Funcao main - Erro na linha 11: variavel nao declarada

LOCAL DO ERRO: o aluno foi %f\n", &m

Erro 4: Na Funcao main - Erro na linha 13: esperado ponto-e-virgula

LOCAL DO ERRO:
   system(

          >>>>>>>>>>Foram encontrados 4 erros<<<<<<<<<
```

Fonte: Próprio autor (2018)

Após a primeira compilação, o *log* de erros é gerado e para que seja tratado e enviado ao servidor, é preciso especificar o endereço IP na interface apresentada na Figura 30 e clicar no botão “gravar”, em seguida a aplicação cfacilplus_cliente será iniciada e a conexão com o servidor será estabelecida para que o professor consiga monitorar os alunos.

Figura 30 - Interação do Aluno com o CFacil+



Fonte: Próprio Autor.

Os logs de erros são então monitorados pela classe Monitor.java, analisados pela classe Analise.java e vão sendo direcionados ao servidor a partir do *socket* cliente.

4.4.2 Implementação da aplicação cfacilplus_servidor

Como primeiros passos na criação do projeto cfacilplus_servidor, na classe Servidor.java, foi necessário importar o pacote java.net e em seguida instanciar um objeto do tipo ServerSocket. Após receber uma conexão, um objeto do tipo Socket é criado para manter a comunicação entre o cliente e o servidor. O servidor possui um endereço IP único de maneira que possa ser identificado pelos clientes na rede. A classe InetAdress permitiu obter estas informações. Os principais métodos implementados desta classe foram os seguintes: getHostAddress(), que retorna uma *string* contendo o endereço IP no formato “000.000.000.000”, e o método getLocalHost(), que retorna informações sobre a máquina local onde está hospedado o servidor.

Para cada solicitação de cliente é criada uma nova *thread* para gerenciar a troca dos arquivos que são enviados via rede. Na classe Servidor.java, um novo objeto Thread é criado, sendo passado um objeto Runnable no seu construtor. Quando a *thread* é criada com o método start(), a nova *thread* começa a execução no método run() do objeto Runnable dentro de uma estrutura de repetição *while*. A execução do servidor fica bloqueada ao se chamar o método accept da classe ServerSocket. Quando algum cliente se conectar ao servidor, o método desbloqueia e retorna com um objeto da classe Socket, que é a porta da comunicação. Logo, cria-se uma nova *thread* a partir da classe TrataCliente para aceitar uma nova conexão e volta a repetição do laço *while*, esperando mais alguém se conectar. Quando vier uma conexão, a referência do cliente será armazenada na variável “cliente”, do tipo *Socket*. Um objeto chamado “dis” da classe DataInputStream é criado. Esta classe é a responsável por receber os *bytes* dos arquivos que vem através da conexão do servidor com o cliente modelando um canal (*stream*) através do qual os *bytes* poderão ser lidos.

É através deste canal que os métodos readUTF e readInt são utilizados para ler os dados dos arquivos transmitidos do cliente para o servidor. O método readUTF lê os bytes do canal e decodifica eles em caracteres Unicode, que irá retornar, em uma *string*, a especificação do nome do arquivo.

Para salvar os arquivos que são recebidos em um diretório na máquina onde é hospedado o servidor, utiliza-se o método FileOutputStream. Na sequência, defini-se o arquivo com *File* e repassam-se os *bytes* lidos para cliente.getOutputStream(), que transforma os *bytes* em um objeto denominado “fos” via BufferedOutputStream. No outro lado da conexão, captura-se os *bytes* com o cliente.getInputStream(), via BuffereInputStream. Com isto, os dados são gravados em um buffer de saída para que sejam enviados da aplicação cfacilplus_cliente e lidos através de um buffer de entrada na aplicação cfacilplus_servidor.

Por fim, foi preciso implementar um *loop* para ler os *bytes,* um por um, dos arquivos de origem para seus respectivos arquivos de destino, enquanto houvesse conexão entre os *sockets.* Definiu-se, também, o tamanho máximo dos arquivos, através de um *array,* utilizando-se para isto o método readInt() e retornando a uma variável chamada “tam” o tamanho máximo de cada um dos arquivos que trafegassem via rede.

Os arquivos que são armazenados no diretório, são exibidos através de uma Jtable, um componente de interface gráfica que permitirá ao professor, no lado servidor da aplicação, monitorar os alunos por meio de uma tabela. Para sua implementação, primeiramente construiu-se uma estrutura para instanciar um objeto do tipo *File*, passando como parâmetro o diretório e retornando um *array* de *strings* que logo após é populado, por meio da invocação do método list(). Uma vez populado o *array*, é utilizado um laço de repetição “for” aprimorado para percorrê-lo. Então, a classe BufferedReader irá realizar a leitura do fluxo de entrada do *array*.

Para obtermos a interface dessa Jtable, foi preciso implementar a classe DefaultTableModel utilizando o método getModel(). Este, então, retorna uma instância da classe que implementa o TableModel, fornecendo-nos todo o controle dos dados da JTable para a inserção e manipulação das informações. Tais informações estão contidas em cada um dos arquivos que controlam as atividades de compilação dos alunos. Antes de adicionar os dados na tabela, indica-se o número de linhas relativas

à quantidade de arquivos presentes no diretório, e isso é feito através do método setNumRows().

O comando switch() é então utilizado para fazer a seleção das *strings* que devem ser apresentadas na Jtable. Então, detalhadamente, o switch() recebe o índice de cada uma das colunas da tabela e abre um bloco de dados ({ }). Dentro desse bloco de dados há os chamados *cases*. Cada *case* recebe o índice de cada uma das colunas da tabela e uma condição que utiliza o método split() para selecionar nas linhas de cada um dos arquivos as que contém os seguintes dados: o nome do aluno, a quantidade total de compilações, a quantidade de acertos e a quantidade de erros, para que estas informações possam ser exibidas na interface para o monitoramento dos alunos.

Para que o professor tenha acesso a um relatório personalizado dos alunos, cada um dos arquivos é aberto para operações de saída através da variável arquivoDestino que é instanciada e criada a partir da classe File. Em seguida, o objeto de gravação denominado “escritor” é associado a um fluxo de saída de dados através da classe PrinterWriter. Posteriormente, foi implementado um processo de repetição (for) para gravar o resultado da leitura dos arquivos resultantes do monitoramento dos alunos em um arquivo de texto externo nomeado como Relatorio_Geral. Assim, o processo é finalizado através do método leitor.close().

O Relatorio_Geral pode ser enviado para o *e-mail* do docente, através da utilização da API Commons Mail. Esta API requer o uso de seis bibliotecas com a extensão .jar que foram inseridas no *classpath* da aplicação cfacilplus_servidor. Logo após, configura-se o endereço do servidor de *e-mails* que será utilizado para enviar o relatório e chamamos o método setSSLOnConnect passando o parâmetro true para o arquivo ser enviado por uma conexão SSL. Então, define-se a porta do servidor de *e-mails* e, como esta aplicação está utilizando SSL, implementa-se o método setSslSmtpPort que recebe o número da porta como *string*.

Um objeto DefaultAuthenticator foi criado para definir o usuário e a senha para autenticação do servidor de *e-mail*. Então, esse objeto criado é adicionado à classe MandarRelatorio. Uma configuração também foi efetuada no endereço de envio chamando este método com dois parâmetros que devem ser informados pelo usuário, sendo o primeiro o endereço de *e-mail* e o segundo o nome pessoal. Então, adiciona-

se o assunto, o corpo do texto e o destinatário da mensagem. Por fim, é chamado o método send() que efetivamente enviará o *e-mail*.

Como resultado será apresentada a seguir na Figura 31, a interface que permite a interação dos usuários, neste caso, do professor na aplicação cfacilplus_servidor.

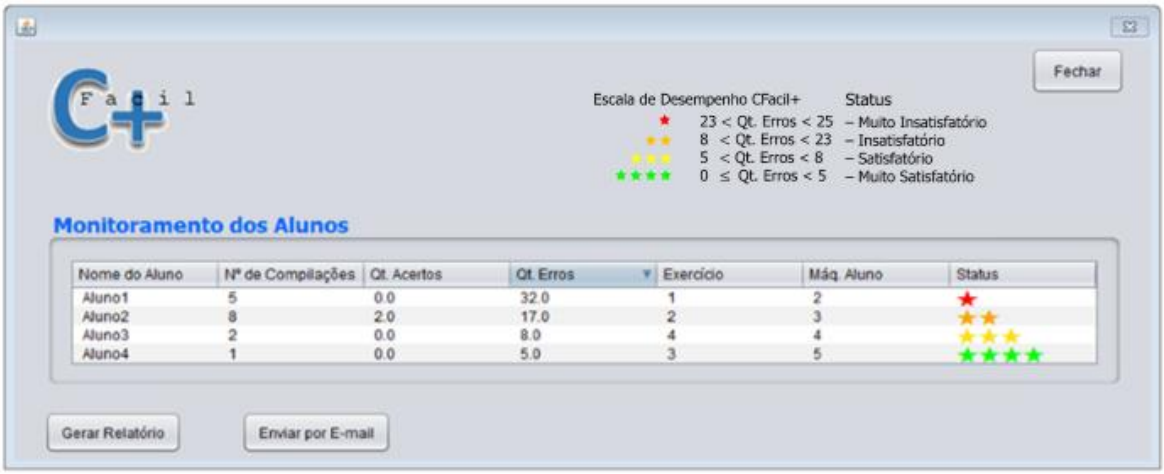
Figura 31 - Interface Inicial da Aplicação Servidor

Fonte: Próprio autor (2018)

Primeiramente, para utilização da ferramenta, é preciso clicar no botão “Iniciar Servidor” por meio desta ação, a classe Servidor.Java será iniciada e o servidor ficará à espera da conexão dos clientes na rede. Nesta primeira etapa, o servidor deve ter criado um *socket* que aceita o contato do cliente. O *socket* servidor é definido por um endereço IP, associado a um número identificador de porta. Este endereço é especificado ao clicar no botão “Obter Endereço IP”, na segunda etapa, que chama a classe InnetAddress. Este endereço deve ser informado pelo professor aos alunos, para que na aplicação cfacilplus_cliente possa ser estabelecida a conexão com o servidor. Isto foi planejado para que o docente possa utilizar a ferramenta em qualquer laboratório, já que se pode disponibilizar a ferramenta para mais de uma instituição de ensino e consequentemente atender um número maior de alunos.

Por último, na terceira etapa, ao clicar no botão “Monitorar Alunos” serão apresentados os arquivos que são recebidos da aplicação cfacilplus_cliente por meio da Jtable.

Figura 32 - Interface para Monitoramento dos Alunos no CFacil+

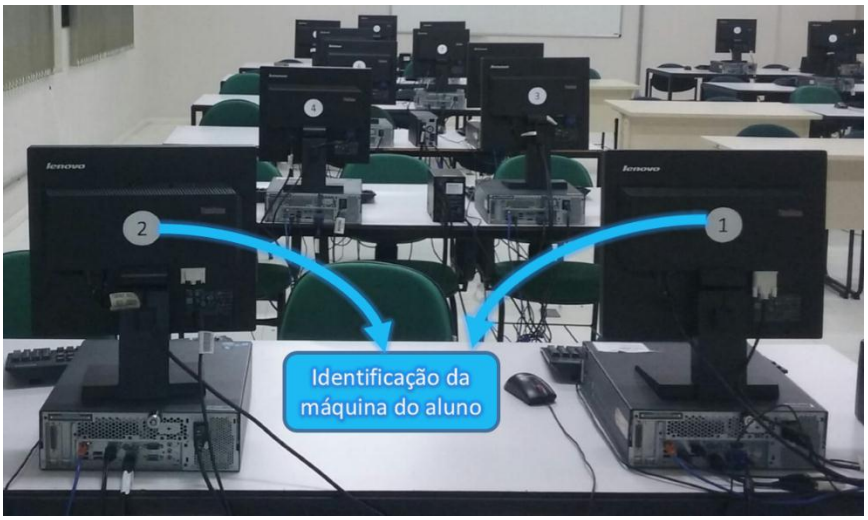


Fonte: Próprio autor (2018)

Através desta interface, apresentada na Figura 32, o professor, no lado servidor da aplicação, poderá verificar quais são os estudantes que estão apresentando maiores dificuldades durante a programação dos algoritmos em cada uma das tarefas executadas em sala de aula. Para que isso fosse possível, durante a implementação da JTable, implementaram-se o método TableRowSorter e uma SortKey que especifica o índice da coluna pela qual a tabela é classificada, mostrando o aluno com a maior quantidade de erros no topo da tabela.

Como mostra a Figura 33, sugere-se que os alunos sejam dispostos em máquinas previamente numeradas para que o professor consiga identificá-los na sala de aula por intermédio da interface de monitoramento. Dessa forma, o professor terá condições de perceber como está o desenvolvimento dos seus alunos e auxiliar aqueles que apresentam problemas com base na identificação dos discentes em cada máquina.

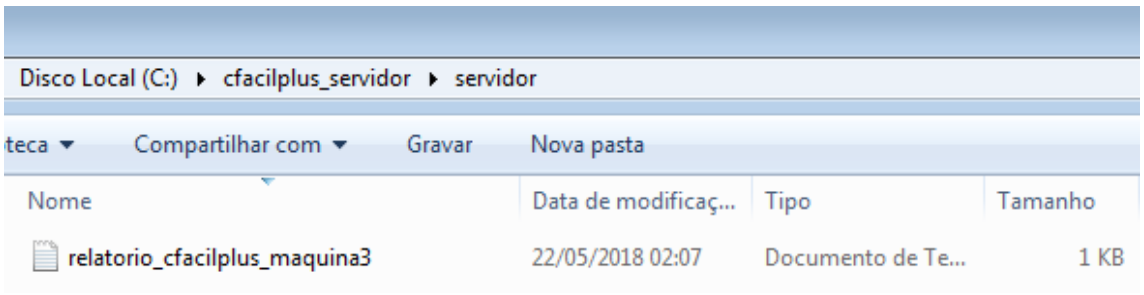
Figura 33 - Identificação da máquina do aluno em sala de aula



Fonte: Próprio autor (2018)

Para identificar quais são os erros cometidos pelos estudantes, é possível ter acesso aos relatórios de cada um dos alunos em sua sala de aula adentrando o diretório que hospeda o servidor. Em uma situação hipotética, suponha que o “Aluno1” esteja ocupando a máquina 3 e ele é quem possui maior dificuldade, então o professor deve acessar nesta pasta o seguinte relatório apresentado na Figura 34. Para que isto seja possível, no momento da instalação da ferramenta é preciso configurar um arquivo.cfg e indicar a numeração das máquinas dos alunos.

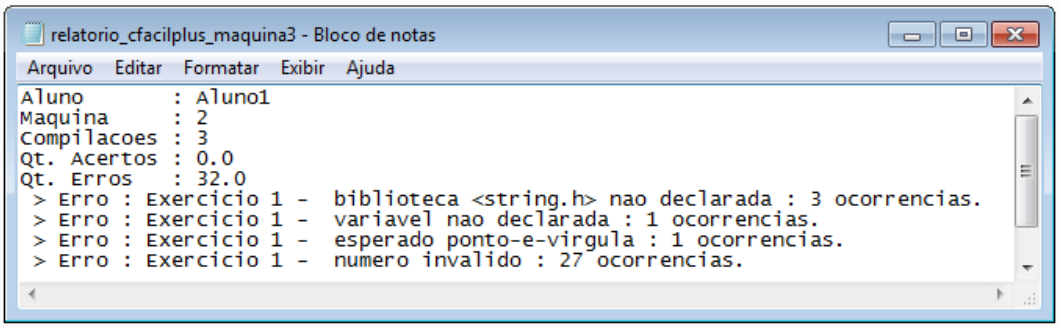
Figura 34 - Acesso ao Relatório de Erros dos Alunos



Fonte: Próprio autor (2018)

Então, conforme mostra a Figura 35, obter-se-á em detalhes a descrição de cada um dos erros que este aluno está cometendo na resolução de suas atividades. Assim sendo, o professor poderá auxiliar esse estudante.

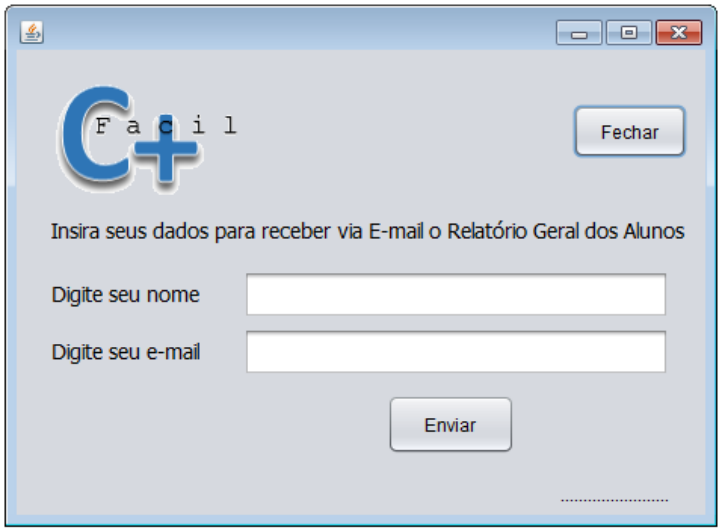
Figura 35 - Detalhes do Monitoramento das Atividades dos Alunos Apresentados no Relatório de Erros



Fonte: Próprio autor (2018)

No botão “Gerar Relatório”, o Relatorio_Geral é gerado e armazenado no diretório raíz da aplicação cfacilplus_servidor. Para enviar este relatório para o e-mail do professor basta clicar no botão “Enviar por E-mail”, e na tela aparecerá a janela apresentada na Figura 36.

Figura 36 - Acesso ao Relatorio_Geral via e-mail



Fonte: Próprio autor (2018)

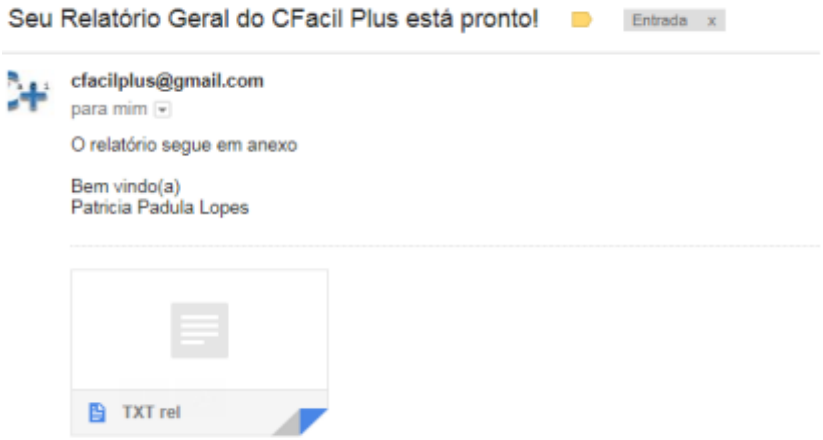
Por fim, após seguir as instruções que são apresentadas na Figura 38, basta clicar no botão “Enviar”. Uma mensagem confirmando o envio do e-mail aparecerá nesta janela. O professor, ao acessar o seu e-mail, poderá verificar o recebimento do arquivo (Figura 37).

Figura 37 - Verificação do recebimento do Relatorio_Geral

Fonte: Próprio autor (2018)

Ao abrir o e-mail, como mostra a Figura 38, o professor terá acesso ao relatório geral do monitoramento das atividades dos alunos realizado durante as suas aulas.

Figura 38 - Acesso ao Relatório Geral do Monitoramento dos Alunos

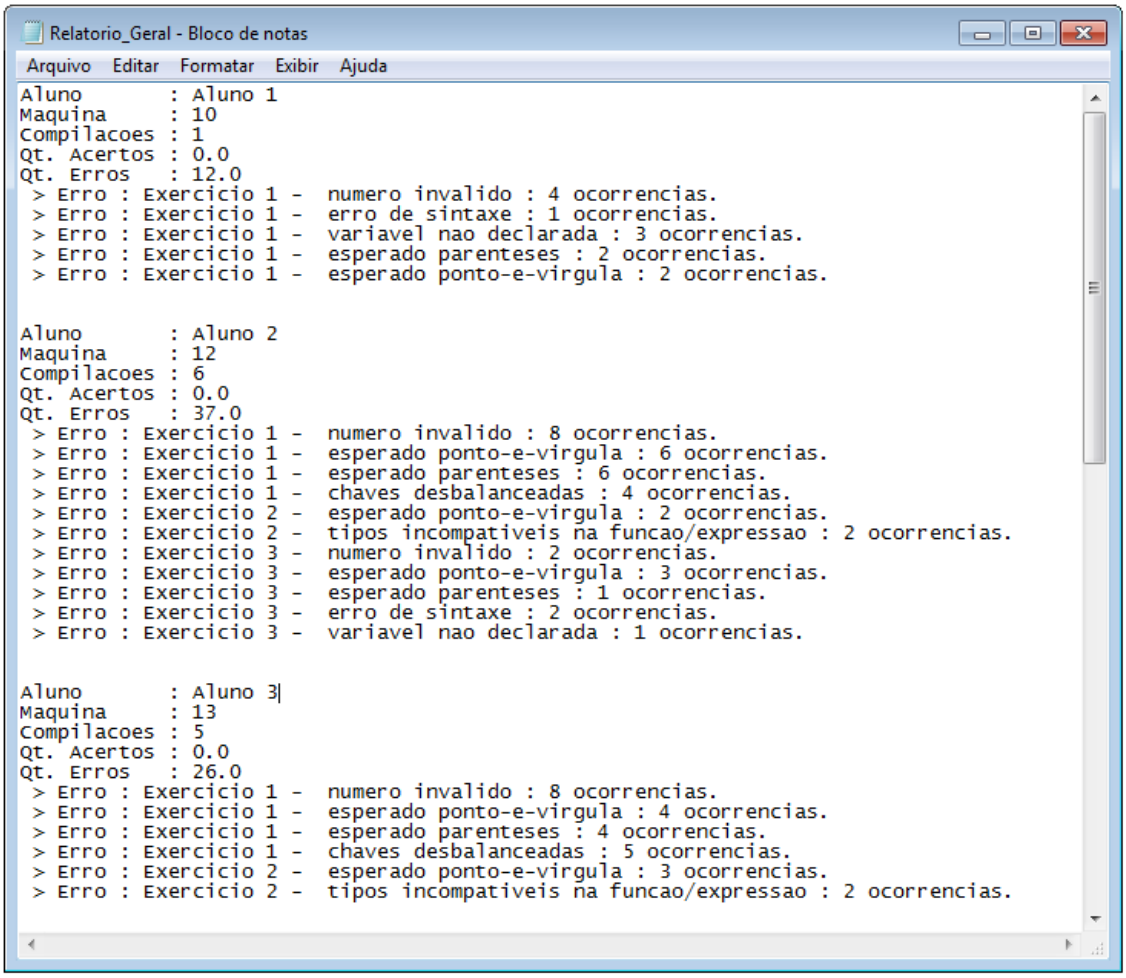


Fonte: Próprio autor (2018)

De posse deste relatório (Figura 39) o professor pode observar se os alunos estão tendo dificuldades e onde essas se encontram em cada um dos exercícios do conteúdo programático. Por meio das informações coletadas durante o

acompanhamento das atividades no sistema, o docente poderá identificar precocemente aqueles alunos que podem vir a não obter aprovação na disciplina, no intuito de melhorar o seu desempenho.

Figura 39 - Relatório Geral CFacil+



Fonte: Próprio autor (2018)

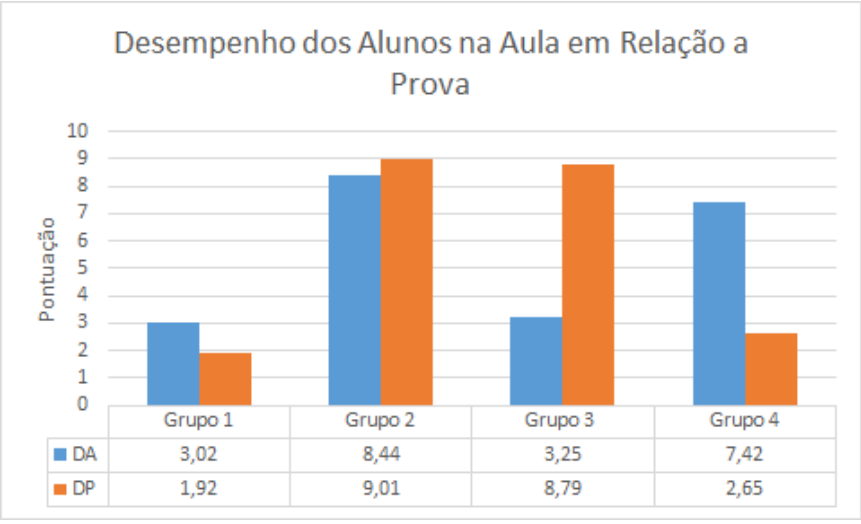
Ao identificar que um grande número de alunos apresenta dificuldades em um mesmo conteúdo, é possível trabalhar neste com mais profundidade ou com estratégias diferentes para propor novas formas de intervir nesta realidade.

4.4.3 Proposta de uma escala para classificação do desempenho dos alunos

Para implementação da escala de desempenho dos discentes, descrito na aplicação CFacil+, utilizaram-se métodos estatísticos para correlacionar a média de compilações e erros cometidos pelos alunos durante as práticas de programação com as médias da primeira avaliação da disciplina. O procedimento já relatado ajudou a identificar os estudantes que estão enfrentando dificuldades e a estabelecer parâmetros para validação da escala na ferramenta. Logo, este estudo foi aplicado para três turmas da disciplina de Algoritmos e Programação, onde foi distribuído um formulário para que os alunos pontuassem a quantidade de compilações e erros ao solucionar quatro exercícios, variando do nível mais fácil ao nível mais avançado. Para análise estatística dos dados, atribuiu-se pesos diferentes para cada uma das atividades, e calculou-se uma média aritmética ponderada para estabelecer uma pontuação de acordo com a quantidade de erros que foram cometidos.

A partir da análise estatística sobre a correlação das notas das avaliações com o desempenho dos alunos em sala de aula através da análise dos formulários, foi possível classificar os alunos em quatro grupos: 1) Os que apresentavam baixo desempenho em aula e um baixo desempenho na avaliação, 2) Um alto desempenho em aula e um alto desempenho na avaliação, 3) Baixo desempenho em aula e alto desempenho na avaliação e 4) Alto desempenho em aula e baixo desempenho na avaliação. O gráfico apresentado na Figura 40 mostra a relação entre o desempenho de cada um dos grupos em aula (DA) e na prova (DP).

Figura 40 - Comparação de Desempenho

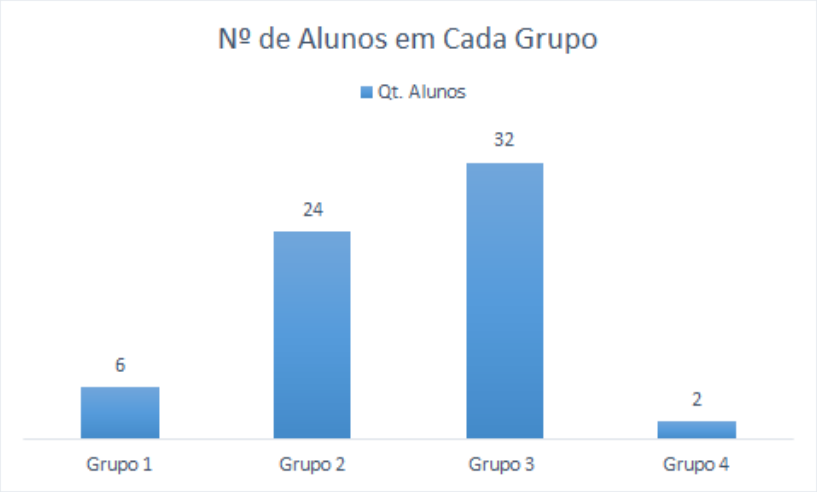


Fonte: Próprio autor (2018)

Aos alunos que faziam parte do grupo 1, a média de desempenho em sala de aula durante as atividades práticas foi de 3,92 pontos, enquanto que a média com relação a primeira avaliação foi de 1,92. Com relação aos alunos do grupo 2, a média de desempenho em sala de aula foi de 8,44 pontos, enquanto que a média na primeira avaliação foi de 9,41. Já os alunos que se enquadraram no grupo 3 a média de desempenho em sala de aula foi de 3,25 pontos, enquanto que na primeira avaliação a média observada foi de 8,79. Por fim, os alunos do grupo 4 apresentaram uma média de 7,42 pontos durante as práticas em aula e na avaliação, a média foi de 2,65.

Cabe salientar, que o universo amostral dos testes era de aproximadamente 20 alunos por aula de cursos de engenharia. O experimento teve duração de 3 aulas de 2 períodos por aula com a supervisão do professor. A Figura 41 apresenta a disposição dos alunos participantes em cada grupo.

Figura 41 - Quantidade de Alunos em cada Grupo



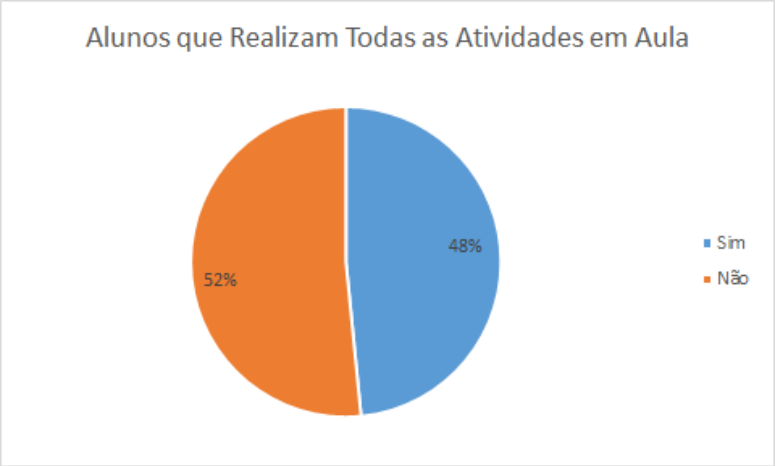
Fonte: Próprio autor (2018)

Uma quantidade de 6 alunos faziam parte do grupo 1, no grupo 2, continham 24 alunos, já no grupo 3, foi observada uma quantidade de 32 alunos, enquanto que o grupo 4 compreendia uma quantidade de 2 alunos.

Neste contexto, embora a maioria dos alunos se encontre no grupo 3, apresentando um alto desempenho na prova, é possível observar que os mesmos apresentam um baixo desempenho em aula, isso pôde ser comprovado pela análise dos formulários, onde os estudantes não pontuaram nenhuma compilação na maioria

dos exercícios propostos, conforme é apresentado no gráfico da Figura 42, onde identificou-se que 52% dos alunos não concluíram todas as suas atividades.

Figura 42 - Alunos que Realizaram as Atividades

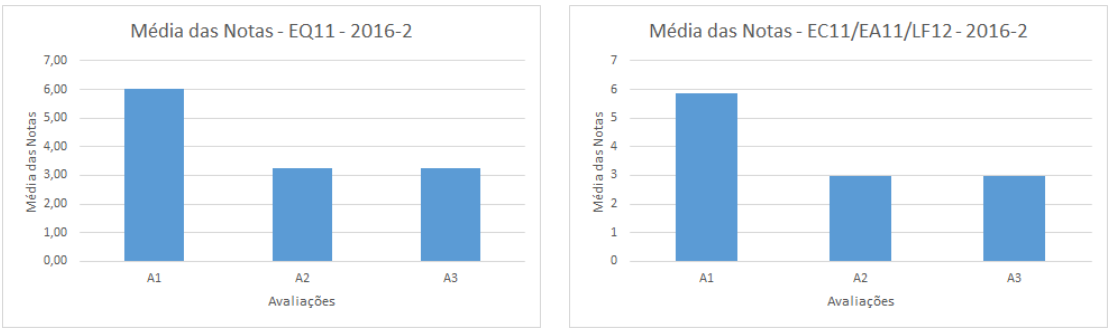


Fonte: Próprio autor (2018)

Com base na análise deste resultado, pode-se observar que a maioria dos estudantes não obtêm aproveitamento satisfatório diante das tarefas apresentadas. Ao concluir estas considerações, vale destacar que vários são os fatores que podem interferir no aprendizado destes alunos, um exercício em branco, por exemplo, pode tanto significar que o estudante não está interessado para o que foi tratado como indicar a incompreensão da proposta feita.

Outro aspecto importante a ser frisado e que foi apontado pelos professores das disciplinas de algoritmos e programação é que a média das notas com relação à primeira avaliação costuma sofrer quedas. Logo, buscou-se coletar evidências que demonstrassem esse desempenho em uma análise das avaliações dos alunos dessas mesmas disciplinas em outros semestres, conforme indicado nos gráficos a seguir.

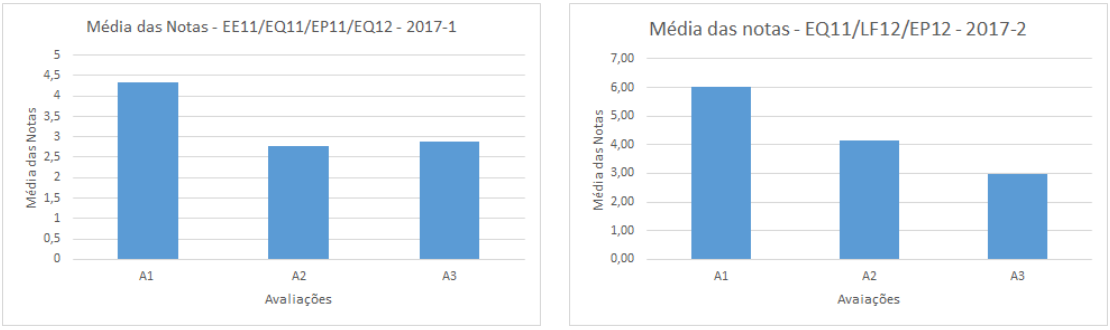
Figura 43 - Média do Desempenho dos alunos no ano de 2016



Fonte: Próprio autor (2018)

Os gráficos apresentados na Figura 43 ilustram a média de notas dos alunos considerando três avaliações realizadas durante os semestres de 2016 nos cursos de Engenharia Química, Engenharia de Computação, Engenharia de Alimentos e Licenciatura em Física. Pode-se verificar que os alunos atingiram uma média maior na avaliação 01 em relação à avaliação 02 e também em comparação à avaliação 03. A mesma situação também ocorreu no ano de 2017, nos cursos de Engenharia de Energias Renováveis, Engenharia Química, Engenharia de Produção e Licenciatura em Química, como pode ser observado nos gráficos da Figura 44.

Figura 44 - Média do Desempenho dos Alunos no Ano de 2017



Fonte: Próprio autor (2018)

A partir do levantamento dos dados sobre a média das notas nas disciplinas de Algoritmos e Programação no âmbito da Universidade Federal do Pampa no período de 2016 a 2017, é possível confirmar o que os docentes dessa área apontam acerca

do desempenho dos alunos, que, na sua maioria, costumam obter um desempenho satisfatório somente na primeira avaliação.

Esses resultados mostram que os alunos possivelmente não conseguem adquirir um conhecimento prévio necessário para obter um desempenho satisfatório nas últimas provas, resultando em um alto índice de reprovação na disciplina e, em alguns casos, pela desistência de um curso. Em parte, isto é decorrência da dificuldade encontrada pelos professores para acompanharem efetivamente as atividades laboratoriais de programação dado o grande número de estudantes geralmente sob sua supervisão (TOBAR, 2001).

Tendo como foco os aspectos apontados anteriormente, percebeu-se que, para validar a escala de desempenho do CFacil+, era preciso considerar o desempenho dos estudantes em sala de aula. Assim, de posse de estatísticas da turma, o professor poderá adotar estratégias que possibilitem promover maior desenvolvimento dos alunos com deficiências de aprendizagem. Auxilia-se dessa forma, na identificação daqueles estudantes que não apresentam avanços, permitindo a prática ou redirecionamento de intervenções mais precocemente.

Considerando a pontuação dos alunos em sala de aula, dentre os resultados obtidos, verificou-se que aqueles que apresentaram um melhor desempenho em sala de aula obtiveram uma pontuação entre 8,0 e 10,0 pontos, enquanto que aqueles que apresentaram um baixo desempenho, apresentaram uma pontuação entre 2,0 a 3,6 pontos. Analisando as pontuações dos alunos com a quantidade mínima e máxima de erros cometidos pelos estudantes durante a execução das atividades práticas de programação em sala de aula, chegou-se ao seguinte resultado apresentado na Tabela 2.

Tabela 2 - Escala de Desempenho CFacil+

Qt. de Erros	Pontuação	Status	Desempenho
23 ≤ Qt. de Erros < 25	2,0 ≤ Pontuação < 3,6	★	Muito Insatisfatório
8 ≤ Qt. de Erros < 23	3,6 ≤ Pontuação < 6,0	★★	Insatisfatório
5 ≤ Qt. de Erros < 8	6,0 ≤ Pontuação < 7,6	★★★	Satisfatório
0 ≤ Qt. de Erros < 5	7,6 ≤ Pontuação ≤ 10	★★★★	Muito Satisfatório

Fonte: Próprio autor (2018)

A quantidade de erros observada entre os alunos que obtiveram uma pontuação maior ou igual a 2 e menor do que 3,6 foi maior do que 23 erros, e menor do que 25 erros. Entre os estudantes que alcançaram uma pontuação maior ou igual a 3,6 e menor do que 6,0 a quantidade de erros foi maior do que 8 e menor do que 23 erros. Com relação aos que alcançaram uma pontuação maior ou igual do que 6,0 e menor do que 7,6 a quantidade de erros foi maior do que 5 e menor do que 8 erros. Enquanto isso, os alunos que apresentaram a melhor pontuação de 7,6 a 10 pontos, a quantidade de erros foi maior ou igual 0 e menor do que 5 erros. Essa classificação é representada, respectivamente, pelos seguintes conceitos: Muito Insatisfatório, Insatisfatório, Satisfatório, Muito Insatisfatório.

A partir desta análise, buscou-se fornecer ao professor uma forma de classificação que indicasse o progresso acadêmico dos alunos durante as práticas em laboratório. Aponta-se o nível de desempenho através de um conjunto de estrelas que são apresentadas na Tabela 2 nas cores vermelho, amarelo, laranja e verde. Essa informação foi gerada com base na frequência de erros que foram cometidos durante as práticas de programação em sala de aula.

Aos alunos que apresentaram uma quantidade de erros menor do que 5, eram exibidas 4 estrelas em cor verde na interface de monitoramento dos alunos. Aos alunos que apresentaram uma quantidade de erros maior do que 5 e menor do que 8, eram exibidas 3 estrelas em cor amarela na interface de monitoramento dos alunos. Aos alunos que apresentaram uma quantidade de erros maior que 8 e menor do que 23, eram exibidas 2 estrelas em cor laranja. Já aos alunos que apresentassem uma quantidade de erros maior do que 25 erros era exibida uma estrela em cor vermelha na interface de monitoramento dos alunos.

A interpretação dos intervalos da escala permite aos professores reconhecerem os alunos com dificuldades e, desta forma, a direcionarem a sua atenção aos alunos que apresentassem um elevado número de programas com erros. Estes casos foram classificados com uma estrela na cor vermelha na tela de monitoramento apresentada ao professor. No entanto, o resultado da aplicação da ferramenta em sala de aula chama atenção com relação às respostas dos questionários realizados com os alunos, visto que a taxa de erros utilizando-se o analisador CFacil foi maior do que a observada.

4.5 Verificação e validação

Com intuito de validar a proposta quanto às características técnicas utilizadas, seguiu-se um modelo de plano de testes, com base nos padrões estabelecidos pela norma da IEEE 829 (Apêndice II). Nesses testes, são apresentados os resultados da simulação da ferramenta a partir da verificação de cada uma das funcionalidades do projeto piloto que foi implementado. O modelo de plano de testes teve como finalidade verificar a pertinência da proposta quanto ao acompanhamento constante dos alunos pelos professores em aulas práticas de programação.

Na análise preliminar realizada, foi preparado um ambiente para testes, através da configuração de uma rede de computadores, no qual a ferramenta foi instalada nos laboratórios de informática geralmente utilizados pela disciplina. Nessa etapa foram aplicados diversos processos para verificar se os requisitos funcionais do sistema estavam sendo implementados corretamente. Como veremos nas próximas seções, essas rotinas consistem em testes de unidade, integração dos módulos e validação do sistema.

4.5.1 Testes de unidade

Os testes de unidade concentram-se seus esforços na verificação do funcionamento das menores unidades do projeto do sistema (Jacobson et al. 1997). Os casos de testes de unidades elaborados ficaram focados nos algoritmos implementados. Nesse processo, foram usadas duas metodologias de testes definidas por Peters e Pedrycz (2001): testes estruturais e testes funcionais.

Os testes estruturais, também chamados de testes “caixa-branca”, são processos que requerem a verificação das rotinas através do acompanhamento das instruções de programa fonte (Peters e Pedrycz, 2001). Essa rotina tem por finalidade principal testar a lógica implementada em baixo nível, verificando o funcionamento do programa. Esse teste foi realizado percorrendo todos os caminhos possíveis (ao menos uma vez), testando todas as condições lógicas e acessando as estruturas de dados. Essa tarefa foi de extrema importância, pois, através da varredura dos possíveis caminhos percorridos durante o processamento do sistema, pode-se

verificar a existência de erros, isto é, defeitos que afetariam diretamente a confiabilidade do sistema.

4.5.2 Testes de integridade

Os testes de integridade, segundo Gimenez (2001) consistem em verificar o funcionamento das diferentes unidades em conjunto. O objetivo desse processo consistiu em verificar o grau de conformidade do programa, isto é, se os resultados apresentados estavam de acordo com os requisitos funcionais especificados.

Inicialmente, era preciso rodar a aplicação cliente nas máquinas dos alunos e o servidor na máquina do professor. Quando um arquivo.c era compilado por um dos alunos, o sistema analisava periodicamente a cada 10 segundos o relatório de erros que era gerado, monitorando o aluno, pelo nome e pela numeração da máquina, apresentando a contagem do número de compilações, erros e acertos dos programas compilados e a numeração do exercício especificado pelo aluno durante as compilações.

O servidor, rodando em paralelo, identificava o arquivo, armazenava no diretório raíz da aplicação e apresentava os resultados na tela de interface de monitoramento para o professor.

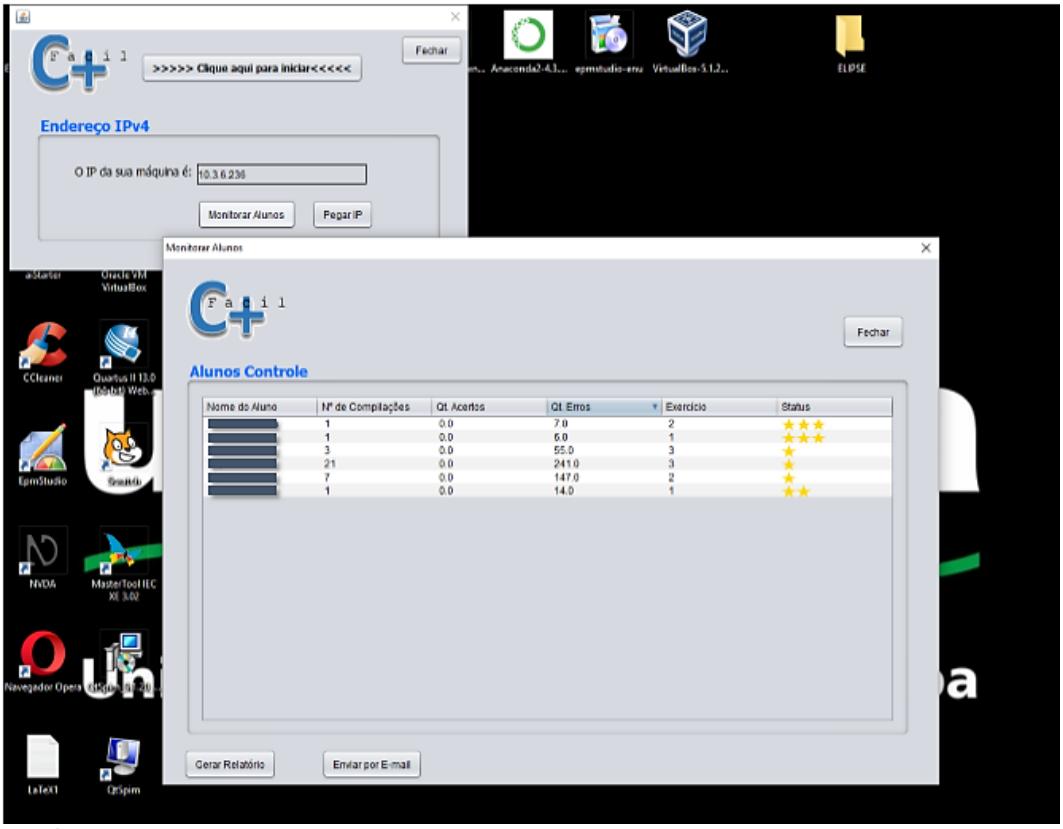
4.5.3 Testes de validação

Concluída a fase de testes de integração, deu-se início a etapa de testes de validação do sistema. Esse processo consistiu em testar o software através da ótica do usuário, onde verificou-se o funcionamento da ferramenta em um ambiente real. Com esse objetivo, o CFacil+ foi submetido a duas turmas de alunos da Universidade Federal do Pampa, na disciplina de algoritmos e programação, a fim de comprovar que a ferramenta implementa cada um dos requisitos corretamente.

As avaliações aconteceram com pessoas do meio acadêmico em sala de aula cedida pelos professores da disciplina, na Universidade Federal do Pampa. Os informantes eram alunos dos cursos de Licenciatura em Química, Engenharia de Produção, Engenharia de Alimentos, Engenharia de Computação e Engenharia de Energias Renováveis. Cada encontro, ocorrido nas diferentes turmas, teve a duração de dois períodos de 50 minutos subsequentes.

No primeiro teste com o CFacil+, foi proposto pelo professor que os alunos resolvessem 4 exercícios, variando do nível mais fácil ao mais difícil, sendo abordado o conteúdo de estruturas de repetição. Durante este primeiro teste, um total de 15 alunos estavam sendo monitorados, porém nem todos resolveram as atividades. Na interface apresentada ao docente, mostrada na Figura 45, é possível observar durante o experimento realizado, que os estudantes que conseguiram resolver até o terceiro exercício, apresentando em torno 5 ou mais compilações e um total de 55 ou mais erros utilizando-se o analisador CFacil para depuração e correção dos códigos implementados.

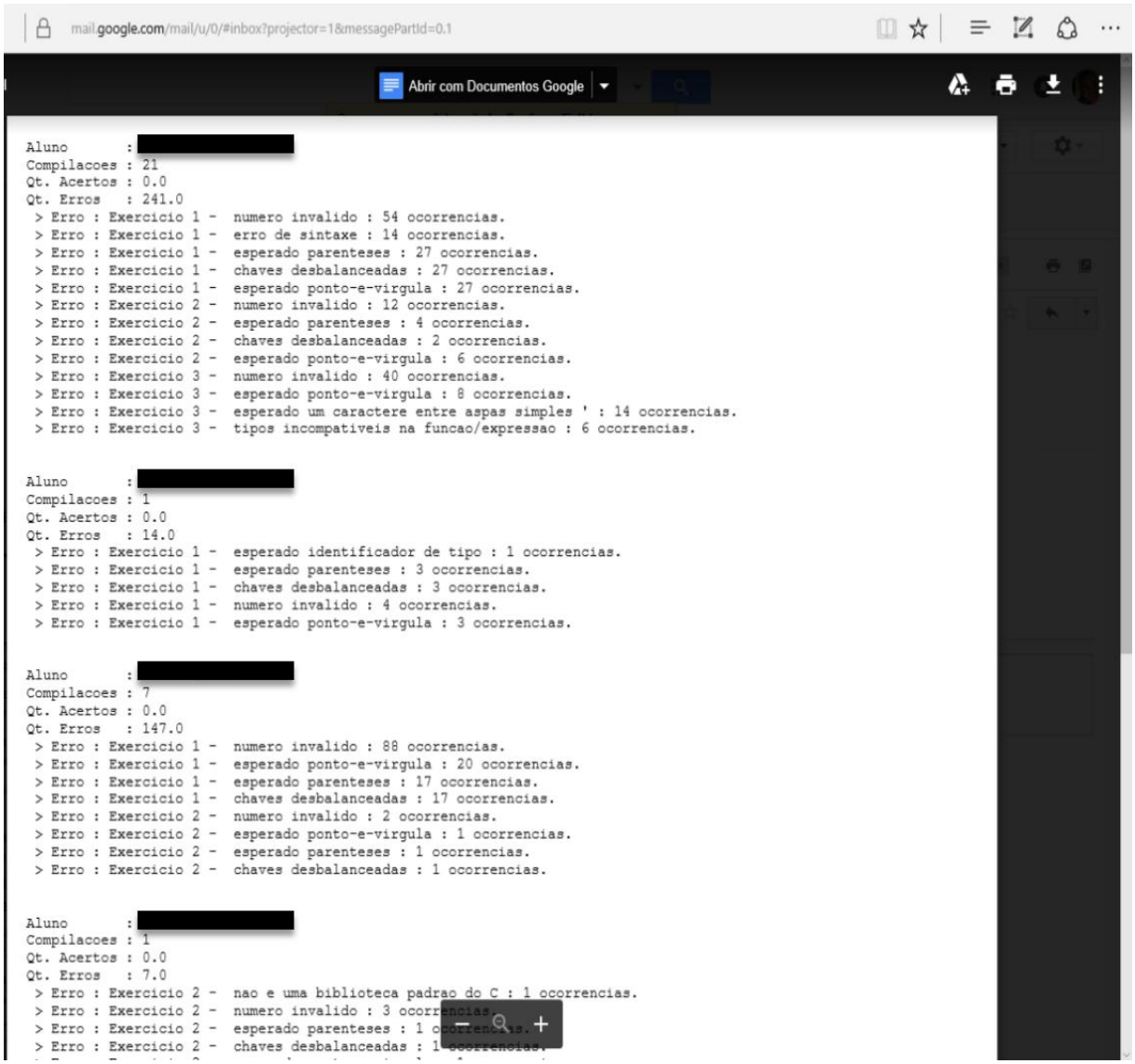
Figura 45 – Interface apresentada ao professor durante o primeiro teste com o CFacil+



Fonte: Próprio autor (2018)

Com base no relatório geral emitido via *e-mail* ao professor, apresentado na Figura 46, foi possível perceber que os erros relacionados ao conteúdo que foi abordado, se concentraram, na espera de *tokens* como abre e fecha chaves, abre e fecha parênteses e ponto e vírgula no final de comados.

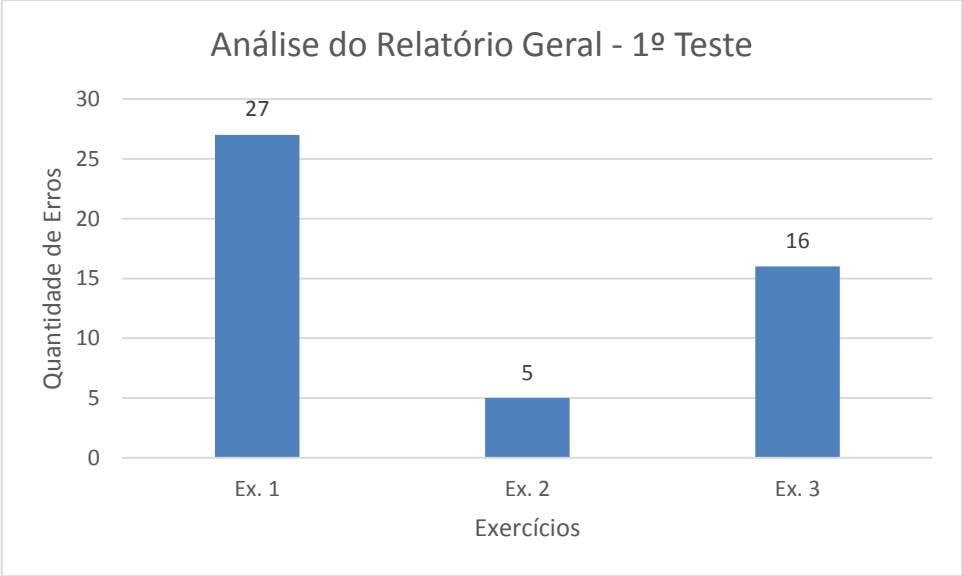
Figura 46 - Relatório geral enviado ao docente durante o primeiro teste da aplicação do CFacil+ em sala de aula



Fonte: Próprio autor (2018)

Além disso, foi possível identificar com base na análise deste relatório, a disposição da quantidade de erros que foram cometidos pelos alunos em cada um dos exercícios relacionados ao conteúdo de estruturas de repetição, conforme é apresentado no gráfico da Figura 47. No primeiro exercício constatou-se uma média de 27 erros. Enquanto que no segundo exercício a média total de erros encontrada foi de 6 erros e no terceiro foi de 16 erros.

Figura 47- Análise da quantidade de erros nos exercícios sobre estruturas de repetição



Fonte: Próprio autor (2018)

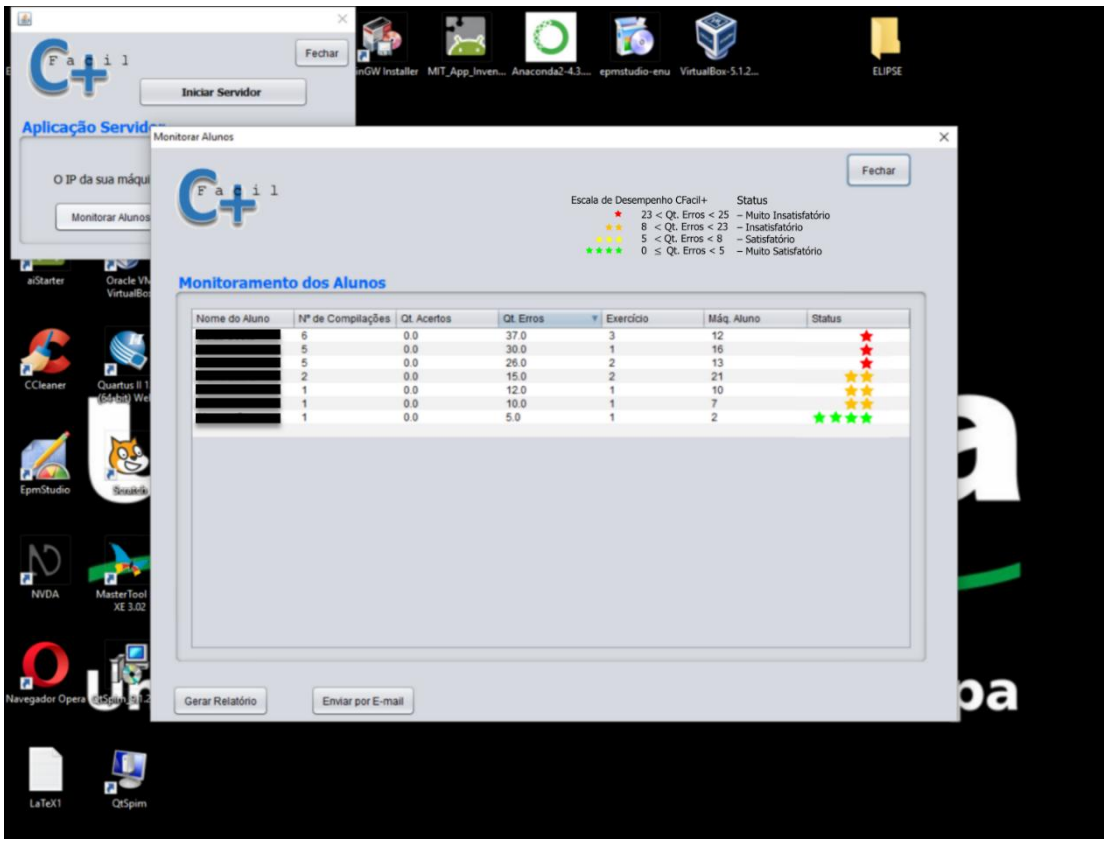
Com o uso do CFacil+, a medida que os alunos foram realizando os primeiros exercícios já tornou-se possível identificar as dificuldades de aprendizagem. Ao identificar que a maioria dos alunos apresentou dificuldade em um mesmo conceito surge a necessidade de trabalhar este com mais profundidade ou com estratégias diferentes.

Neste relatório, também foi possível identificar a descrição de cada um dos erros, permitindo ao docente identificar os erros mais comuns apresentados pelos alunos. Entende-se que as informações apresentadas não são suficientes para detectar e orientar todos os tipos de erros que os alunos cometem, mas certamente fornecem alguma referência para identificação da conformidade da solução.

Com base nestes resultados, foi possível identificar melhorias e organizá-las de forma que pudessem ser resolvidas de maneira viável, atentando-se aos requisitos ao qual a ferramenta foi proposta, pois durante o primeiro teste, não era possível visualizar de maneira ordenada a disposição dos alunos que apresentavam a maior quantidade de erros no topo da tabela e a classificação indicativa sobre o desempenho dos alunos dificultava a identificação imediata do aluno que possuía maior dificuldade em sala de aula pelo docente, também não estava sendo apresentado a numeração das máquinas as quais os alunos ocupavam durante a realização das atividades.

Logo, para alcançar os objetivos propostos pela ferramenta, procurou-se solucionar os problemas encontrados e aplicar o CFacil+ novamente em sala de aula. Neste segundo teste que foi realizado, um total de 18 alunos foram monitorados na realização de 4 exercícios sobre o conteúdo de vetores, também variando em nível de complexidade, como no teste anterior. No momento da realização deste experimento, era possível que o professor acompanhasse os alunos que realizavam as atividades, identificando dentre estes, aqueles que apresentavam maiores dificuldades em sala de aula através da interface de monitoramento que era apresentada pela ferramenta (Figura 48).

Figura 48 - Interface apresentada ao professor durante o segundo teste com o CFacil+



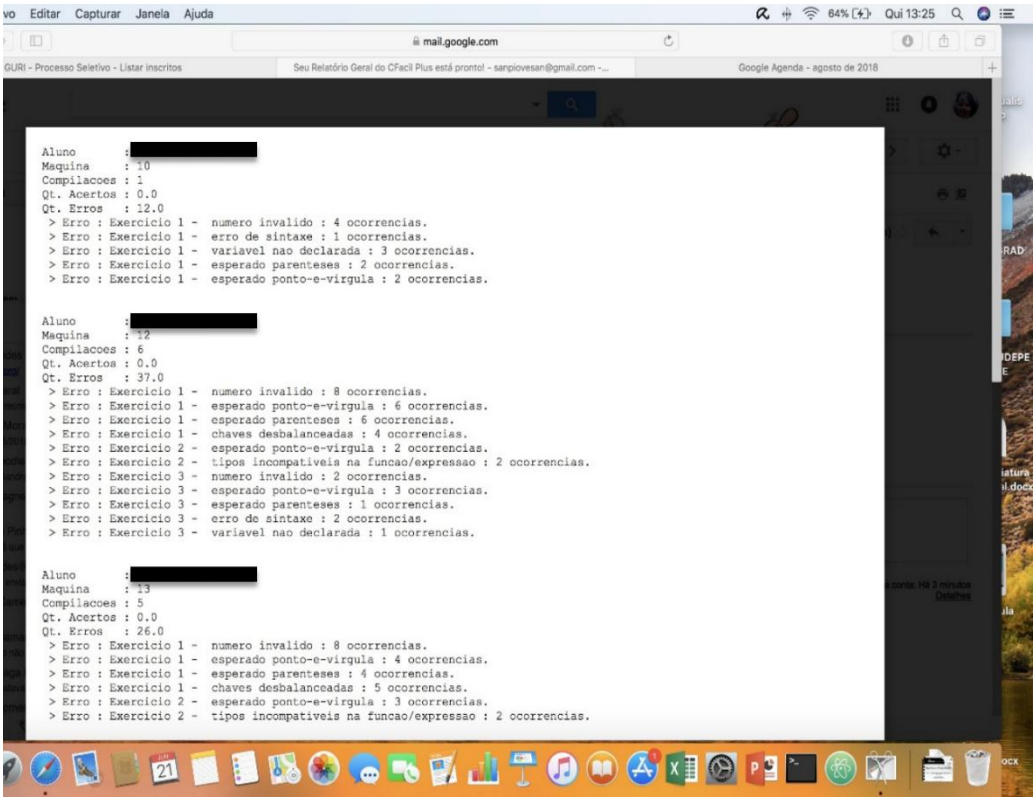
Fonte: Próprio autor (2018)

Na interface de monitoramento apresentada, os alunos que obtiveram um desempenho “Muito Insatisfatório” com uma quantidade de erros maior do que 25 erros foram classificados com uma estrela na cor vermelha e apareciam no topo da tabela. Enquanto que os alunos que apresentaram um desempenho “Insatisfatório”, com uma quantidade de erros entre 8 a 22 erros, eram classificados com 2 estrelas

na cor laranja, logo abaixo. Já o aluno que obteve um desempenho “Muito Satisfatório”, apresentando 5 erros durante a resolução das suas atividades, era classificado com 4 estrelas de cor verde. Esta classificação, serviu com um auxílio ao docente para que pudesse direcionar a sua atenção aqueles alunos que apresentavam maiores dificuldades na resolução das atividades propostas.

A partir da análise do relatório geral que foi enviado via *e-mail* pela ferramenta CFacil+ ao professor, como é apresentado na Figura 49, foi possível identificar a quantidade de erros e os erros mais comuns que foram cometidos pelos estudantes na resolução dos exercícios sobre o conteúdo de vetores.

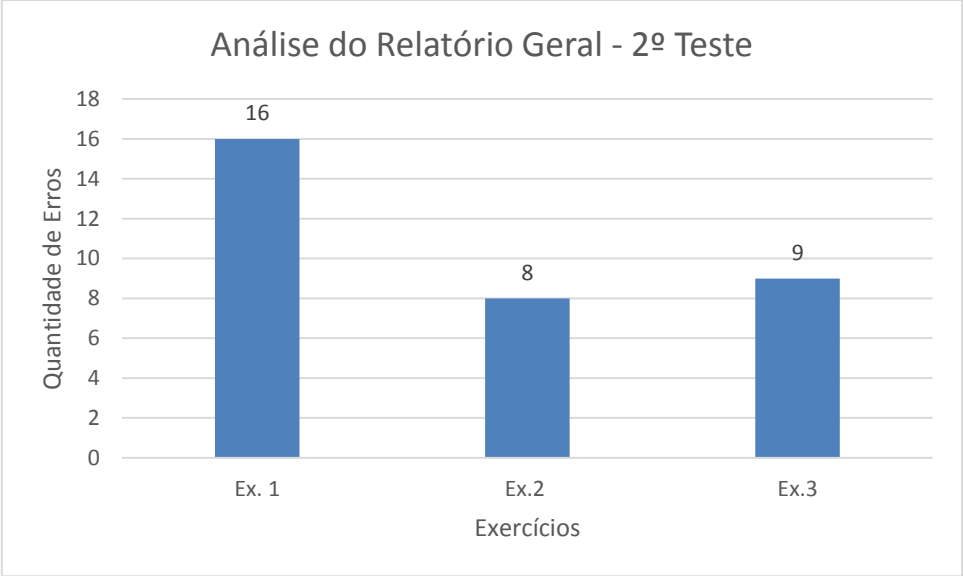
Figura 49 - Relatório geral enviado ao docente durante o segundo teste da aplicação do CFacil+ em sala de aula



Fonte: Próprio autor (2018)

A partir destas observações, através do gráfico mostrado na Figura 50, verificou-se que os alunos apresentaram uma maior dificuldade no primeiro exercício, apresentando uma média de 16 erros para resolução desta atividade. Ao passo que, a média de erros observadas para resolução do segundo exercício foi de 8 erros e do terceiro de 9 erros.

Figura 50 - Análise da quantidade de erros nos exercícios sobre estruturas de repetição



Fonte: Próprio autor (2018)

Considerando os resultados obtidos nos processos de avaliação da ferramenta, observou-se que o sistema (CFacil+) de monitoramento das atividades de programação dos alunos atende aos requisitos a que foi proposto. Os resultados comprovaram que através da ferramenta que foi desenvolvida é possível monitorar os alunos. Uma vez que proporciona uma visualização imediata do que está acontecendo durante as atividades, ela possibilita que o mediador observe a turma em uma posição privilegiada. A aplicação mostra o desempenho dos alunos nas tarefas propostas em sala de aula pelo docente, com base no relatório de erros que é gerado a partir da utilização do CFacil pelos estudantes, de maneira que as informações que trafegam entre os alunos são visualizadas de forma instantânea pelo docente.

4.5.4 Resultados e discussões

Com o intuito de coletar as impressões dos professores participantes da pesquisa, em relação a utilização do CFacil+, foi utilizada a plataforma Google docs. A referida ferramenta é *open source* e foi utilizada para realizar pesquisas através de um questionário padronizado (Apêndice III), com base no modelo de Lewis *et al.* (2002) para medir a usabilidade pós-teste do ponto de vista dos usuários.

Na elaboração do questionário procurou-se investigar aspectos de usabilidade e funcionalidade por parte dos professores. Para tal finalidade, decidiu-se utilizar

perguntas do tipo múltipla escolha, com a escala *likert*[3] de 1 a 5 (1 = concordo totalmente a 5 = discordo totalmente). Consideraram-se os resultados obtidos nos processos de avaliação da ferramenta nos quesitos de usabilidade e funcionalidade juntamente com a qualidade da ferramenta como recurso de ensino/aprendizagem. Por meio das respostas obtidas, foi possível perceber que os julgamentos foram favoráveis. Verificou-se, também, que ambos professores concordam que a aplicação do CFacil+, durante as práticas de programação, tornaria as aulas mais produtivas, visto que o sistema proporciona uma visualização imediata do que está acontecendo durante as atividades dos alunos. Quando questionados sobre a interface de monitoramento dos alunos, foi possível perceber que o *desing* do CFacil+, por ser simplificado, é fácil e intuitivo na visão dos professores, sendo possível realizar o acompanhamento dos alunos de forma efetiva.

Com o questionamento referente a apresentação das informações, foi constatado que estas permitem identificar os alunos que apresentam maior quantidade de erros com base no analisador CFacil. No entanto, identificou-se uma certa dificuldade na análise da descrição dos seguintes erros: erros de sintaxe, tipo inexistente de expressão e número inválido, onde não foi possível identificar exatamente o que ocorreu no código compilado pelos alunos, pois de acordo com as respostas auferidas por um dos professores, são descrições de erros bastante amplas e isto leva a crer novamente que alguns aspectos de funcionalidades precisam ser revistos na ferramenta CFacil. Porém, de acordo com a opinião dos docentes, as informações sobre a identificação do aluno, quantidade de erros, acertos e compilações, são apresentadas de forma clara e consistente e são relevantes para monitorar o desempenho dos alunos durante as aulas práticas de programação. No geral, ambos professores se sentiram satisfeitas ao aplicar a ferramenta proposta neste trabalho durante as suas aulas.

Baseado nos resultados obtidos no questionário observou-se que a ferramenta atende aos requisitos a que foram propostos, obtendo avaliações positivas. A partir da análise das respostas observa-se que o sistema de monitoramento proposto atingiu bons níveis de aceitação pelos professores. Assim, pode-se concluir que o CFacil+ conseguiu cumprir seus objetivos específicos, sendo uma boa alternativa de auxílio ao ensino de algoritmos e programação.

[3] Disponível em: https://www.britannica.com/topic/Likert-Scale

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi apresentada uma ferramenta para o acompanhamento das atividades dos alunos na disciplina de programação. O ferramental implementado foca em oferecer informações úteis sobre o acompanhamento das atividades dos estudantes a partir das compilações que os usuários realizam, numa visão global, mas dando oportunidade para análise individual.

O desenvolvimento do CFacil+ permitiu visualizar um relatório detalhado de cada estudante e de sua produção de exercícios de programação. Esta abordagem retrospectiva, permite que seja possível identificar quadros típicos de alunos que estão adotando comportamentos de risco, por exemplo, realizando poucos exercícios, realizando exercícios errados sem retomá-los, etc. Assim, a aplicação auxilia na orientação dos alunos de forma mais embasada e na reflexão de suas atitudes frente às atividades propostas em aula práticas de Algoritmos e Programação. Ao passo que é possível confrontá-los com as suas próprias dificuldades, o CFacil+ também permite o contraste com a produção da turma.

Dessa forma, pode-se concluir que, a partir do desenvolvimento da ferramenta, é possível o professor atuar de forma rápida nos alunos que mais necessitam de intervenção. Ao mesmo tempo, o professor pode rever práticas próprias de ensino, como por exemplo, ser capaz de se adequar a diferentes ritmos de produção de exercícios dos alunos.

5.1 Trabalhos futuros

Este trabalho buscou colaborar com um estudo anterior bem reconhecido, tendo ciência de que, como tudo que realizamos está sujeito a melhoras, o mesmo pode acontecer com a proposta aqui apresentada. Como trabalhos futuros, pretende-se promover a disponibilização da ferramenta CFacil+ para outras instituições. Visa-se, com isso, auxiliar consideravelmente no processo de monitoramento, não de erros simplesmente, mas da aprendizagem dos alunos de disciplinas de algoritmos e programação. Professores dessa área também se beneficiarão com o uso da ferramenta CFacil+.

REFERÊNCIAS

AMARAL, Érico Marcelo Hoff do. **Processo de ensino e aprendizagem de algoritmos integrando ambientes imersivos e o paradigma de blocos de programação visual**. 2015.

AMBER, M.; SOLOMON, S. ***Work-in-Progress : Leveraging Cloud Computing and Web Standards to Sup- port Learning Objectives in Multiple Classrooms Work-in-Progress : Leveraging Cloud Computing***. In: 122nd ASEE Annual Conference and Exposition. American Society for Engineering Education. 2015.

ANIDO, Ricardo. **Saci–ainda outro ambiente para o ensino de programação**. 2015.

BARBOSA, Leônidas da Silva. **Aprendizado significativo aplicado ao ensino de algoritmos**. Dissertação de Mestrado. Universidade Federal do Rio Grande do Norte. 2011.

BARCELOS, Ricardo; TAROUCO, Liane; BERCHT, Magda. **O uso de mobile learning no ensino de algoritmos**. RENOTE, v. 7, n. 3, p. 327-337, 2009.

BLATCHFORD, Peter; BASSETT, Paul; BROWN, Penelope. ***Examining the effect of class size on classroom engagement and teacher–pupil interaction: Differences in relation to pupil prior attainment and primary vs. secondary schools***. Learning and Instruction, v. 21, n. 6, p. 715-730, 2011.

DA SILVA, Bruno Siqueira; MONTENEGRO, Toni Ferreira. **O Ensino-Aprendizagem de Programação de Computadores: dificuldades e ferramentas de suporte**. 2016.

DA SILVA, Maria Natalia Santos; SANTOS, Maria Marlene. **As tecnologias de informação e comunicação no ambiente escolar**. Revista Educação & Tecnologia, v. 15, n. 15, 2017.

CAMPOS, R. L. B. L. **Lógica de Programação: Há como melhorar o aprendizado fugindo dos padrões estabelecidos nos livros didáticos e adotados pela maioria dos docentes?** In: XVII Congresso Iberoamericano de Educación Superior em Computacion (CLEI-2009-CIESC), p. 22-25, 2009.

CODEBUNK, ***Online Realtime Collaborative Editor and Compiler***, 2018. Disponível em: <https://codebunk.com/>. Acesso em 24 abril 2018.

CORREIA, Carlos Henrique; TAFNER, Malcon Anderson. **Análise orientada a objetos. Florianópolis**: Ed. Visual Books, 2006.

CODEBOARD, ***Codeboard.io***, 2018. Disponível em: <https://codeboard.io/>. Acesso em 03 maio 2018.

DA SILVA, Bruno Siqueira; MONTENEGRO, Toni Ferreira. **O Ensino-Aprendizagem de Programação de Computadores: dificuldades e ferramentas de suporte**. 2016.

DA SILVA, Maria Natalia Santos; SANTOS, Maria Marlene. **As tecnologias de informação e comunicação no ambiente escolar**. Revista Educação & Tecnologia, v. 15, n. 15, 2017.

DA SILVA, Idovaldo Cunha; FONSECA, Luís Carlos Costa; DA SILVA, Reinaldo de Jesus. **Proposta de um Ambiente Online para a Resolução e Correção Automática de Algoritmo**. 2015.

DE CAMPOS, Cassio P.; FERREIRA, Carlos E. **BOCA: um sistema de apoio a competições de programação**. In: Workshop de Educação em Computação. 2004. p. 885-895.

DEITEL, Harvey M.; DEITEL, Paul J. **Java, como programar**. 4ª Edição. São Paulo, 2003.

ESPAÑA-BOQUERA, Salvador et al. ***Analyzing the learning process (in Programming) by using data collected from an online IDE***. In: Information Technology Based Higher Education and Training (ITHET), 2017 16th International Conference on. IEEE, 2017. p. 1-4.

FACOM, **Maratona de Programação**, 2018. Disponível em: <http://maratona.wp.facom.ufms.br/2016/08/16/resultados-desafio-de-programacao-facom-2016-2/score/>. Acessado em: 26 Maio 2018.

FALCKEMBACH, Gilse A. Morgental; DE ARAUJO, Fabrício Viero. **Aprendizagem de algoritmos: dificuldades na resolução de problemas**. Anais Sulcomp, v. 2, 2013.

FONSECA, J. J. S. **Metodologia da pesquisa científica**. Apostila.Fortaleza: UEC, 2002.

GIMENEZ, I. **Processos do ciclo de vida: processos fundamentais – processo de desenvolvimento**. In: DA ROCHA, A.R.C.; MALDONADO, J.C.; WEBER, K.C. (Orgs.). Qualidade de software: teoria e prática. São Paulo: Prentice Hall, 2001, pp. 43-47.

GIRAFFA, Maria Martins; DA COSTA MOURA, Michael. **Evasão na disciplina de algoritmos e programação: um estudo a partir dos fatores intervenientes na perspectiva do aluno**. In: Congressos CLABES. 2016.

GOMES, Cledson Calaça Cavalcante et al. **Uma Proposta para Auxiliar Alunos e Professores no Ensino de Programação**: O Ambiente AIIP. 2011.

GOMES, Marina S.; DO AMARAL, Érico MH. **Simplificando a depuração de códigos na Linguagem C - Uma solução para alunos iniciantes**. RENOTE, v. 14, n. 1. 2016.

GUOLI, Zhang; WANJUN, Liu. ***The applied research of cloud computing platform architecture in the E-Learning area***. In: Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on. IEEE, 2010. p. 356-359.

IEEE Computer Society. ***IEEE Std 829: Standard for Software Teste Documentation***. 1998.

JACOBSON, I.; GRISS, M.; JONSSON, P. ***Software reuse: architecture process and organization for business success***. Harlow: Addison-Wesley, 1997.

JAVA. **Obtenha Informações sobre a Tecnologia Java**, 2017. Disponível em: <https://www.java.com/pt_BR/about/>. Acesso em: 24 Maio 2017.

JUNG, Inho et al. ***Interactive learning environment for practical programming language based on web service***. In: Information Technology Based Higher Education and Training (ITHET), 2016 15th International Conference on. IEEE, 2016. p. 1-7.

KOBRA, ***Realtime Collaborative Coding***, 2018. Disponível em: <http://kobra.io/>. Acesso em: 25 abril 2018.

KOC, Nizamettin; CELIK, Bekir. ***The Impact of Number of Students per Teacher on Student Achievement. Procedia-Social and Behavioral Sciences****, v. 177, p. 65-70, 2015.*

LEWIS, J. R. ***Psychometric Evaluation of the PSSUQ Using Data from Five Years of Usability Studies***. International Journal of Human-Computer Interaction, p. 462–488, 2002.

LOPES, Patrícia Padula et al. **Proposta de um Sistema para o Monitoramento das Atividades de Programação de Alunos Iniciantes.** In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. 2017. p. 942.

MOREIRA, Mireille Pinheiro; FAVERO, Eloi Luiz. **Um ambiente para ensino de programação com feedback automático de exercícios**. In: Workshop sobre Educação em Computação (WEI 2009). 2009.

NAZÁRIO, Débora Cabral; DE SOUZA, Allan. **BOCA-LAB: Corretor automático de Código adaptado ao Ensino de Linguagem de Programação**. Anais do Computer on the Beach, p. 42-46, 2010.

OCHOA, Xavier et al. **Need analyses of the students in programming courses in latin america.** 2011.

OLIVEIRA, Manassés Vitorino; RODRIGUES, Luciene Cavalcanti; QUEIROGA, Ana. **Material didático lúdico: uso da ferramenta Scratch para auxílio no aprendizado**

de lógica da programação. In: Anais do Workshop de Informática na Escola, p. 359, 2016.

PEARS, Arnold et al. ***A survey of literature on the teaching of introductory programming***. In: ACM SIGCSE Bulletin. ACM, 2007. p. 204-223.

PEREIRA, Alice Theresinha Cybis; SCHIMITT, Valdenise; DIAS, M. R. A. C. **Ambientes virtuais de aprendizagem. AVA-Ambientes Virtuais de Aprendizagem em Diferentes Contextos**. Rio de Janeiro: Editora Ciência Moderna Ltda, 2007.

PRESSMAN, Roger S. **Engenharia de software**. 6ª Ed. Porto Alegre: Bookman, 2006.

PETERS, J.F.; PEDRYCZ, W**. Engenharia de software: teoria e prática**. Rio de Janeiro: Campus, 2001.

RAABE, André Luís Alice; SILVA, JMC da. **Um ambiente para atendimento as dificuldades de aprendizagem de algoritmos.** In: XIII Workshop de Educação em Computação (WEI'2005). São Leopoldo, RS, Brasil. 2005.

RANTALA, Mikko. ***Real-time collaborative coding - technical and group work Challenges***. 2015.

RAPKIEWICZ, E.C; FALKEBACH G; SEIXAS, L; ROSA, N.S; CUNHA, V.V; KLEMANN, M. **Estratégias Pedagógicas No Ensino De Algoritmos e Programação Associadas Ao Uso De Jogos Educacionais**. RENOTE, Rio Grande do Sul, v.4 n.2**,** 2006.

SANTOS, Antunes et al. **A Importância do Fator Motivacional no Processo Ensino-Aprendizagem de Algoritmos e Lógica de Programação para Alunos Repetentes**, 2015.

SCHFFER, C. **Tecnologia computacional e desenvolvimento cognitivo: estudo de caso na formação de psicólogos**. São Paulo: Annablume; Belo Horizonte: FUMEC, 2004.

SCHULTZ, M. R. O. **Metodologias para Ensino de Lógica de Programação de Computadores. Monografia de Especialização**. Universidade Federal de Santa Catarina (UFSC), Florianópolis, SC, Brasi, 2003

SCOLARI, Angélica Taschetto; BERNARDI, Giliane; CORDENONSI, Andre Zanki. **O desenvolvimento do raciocínio lógico através de objetos de aprendizagem**. RENOTE, v. 5, n. 2, 2007.

SILVA, IFA; SILVA, Ivanda Maria Martins; SANTOS, Marizete Silva. **Análise de problemas e soluções aplicadas ao ensino de disciplinas introdutórias de programação. Universidade Federal Rural de Pernambuco**, Recife–PE, 2009.

SOMMERVILLE, Ian. **Engenharia de software**. 9 Ed. São Paulo: Pearson, 2011.

STAMOULI, Ioanna; HUGGARD, Meriel. ***Object oriented programming and program correctness: the students' perspective. In: Proceedings of the second international workshop on Computing education research***. ACM, p. 109-118, 2006.

TARDIF, M. **Saberes docentes e formação profissional**. 3. ed. Petrópolis: Vozes, 2002.

TOBAR, Carlos Miguel et al. **Uma arquitetura de ambiente colaborativo para o aprendizado de programação**. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE), p. 367-376, 2001.

VIEIRA, Carlos Eduardo Costa et. al. **Dificuldades no Processo de Aprendizagem de Algoritmos: uma Análise dos Resultados na Disciplina de AL1 do Curso de Sistemas de Informação da FAETERJ–Campus Paracambi**. Cadernos UniFOA, n. 27, p. 5-15, 2015.

VIELMO, Pauline. **A linguagem logo como alternativa lúdica de ensino**. Seminário Nacional de Pesquisa em Educação, 2016.

WAINER, Jacques et al. **Métodos de pesquisa quantitativa e qualitativa para a Ciência da Computação. Atualização em informática**, v. 1, p. 221-262, 2007.

APÊNDICE I

Documento de Requisitos

Sistema para o Monitoramento das Atividades de Programação para Alunos Iniciantes – CFacil+

Versão 1.0

1. Introdução

Este documento especifica os requisitos de um projeto piloto de Sistema para Monitoramento das Atividades de Programação para Alunos Iniciantes – CFacil+ Versão 1.0, fornecendo aos desenvolvedores as informações necessárias para o projeto e implementação, assim como para a realização dos testes e homologação do sistema.

Visão geral do documento

Além desta seção introdutória, as seções seguintes estão organizadas como descrito abaixo.

Seção 2 – Descrição geral do sistema: apresenta uma visão geral do sistema, caracterizando qual é o seu escopo e descrevendo seus usuários.

Seção 3 – Requisitos funcionais: relacionam a maneira de como o sistema deve operar, onde se especificam as entradas e saídas do sistema.

Seção 4 – Requisitos não-funcionais: especifica todos os requisitos não funcionais do sistema, divididos em requisitos de usabilidade, confiabilidade, desempenho, segurança, distribuição, adequação a padrões e requisitos de hardware e software.

Seção 5 – Referências: apresenta referências para outros documentos utilizados para a confecção deste documento.

Convenções, termos e abreviações

Identificação dos requisitos

A correta interpretação deste documento exige o conhecimento de algumas convenções e termos específicos, que são descritos a seguir.

Por convenção, a referência a requisitos é feita através do nome da subseção onde eles estão descritos, seguidos do identificador do requisito, de acordo com a especificação a seguir: [nome da subseção. Identificador do requisito]

Prioridades dos requisitos

Para estabelecer a prioridade dos requisitos, nas seções 4 e 5, foram adotadas as denominações “essencial”, “importante” e “desejável”.

Essencial é o requisito sem o qual o sistema não entra em funcionamento. Requisitos essenciais são requisitos imprescindíveis, que têm que ser implementados impreterivelmente.

Importante é o requisito sem o qual o sistema entra em funcionamento, mas de forma não satisfatória. Requisitos importantes devem ser implementados, mas, se não forem, o sistema poderá ser implantado e usado mesmo assim.

Desejável é o requisito que não compromete as funcionalidades básicas do sistema, isto é, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis podem ser deixados para versões posteriores do sistema, caso não haja tempo hábil para implementá-los na versão que está sendo especificada.

2. Descrição geral do sistema

Abrangência do sistema

O Sistema para Monitoramento das Atividades de Programação para Alunos Iniciantes – CFacil+ permite aos professores, monitorar, analisar e fornecer assistência aos alunos durante as aulas práticas em laboratório das disciplinas de Algoritmos e Programação.

3. Requisitos Funcionais

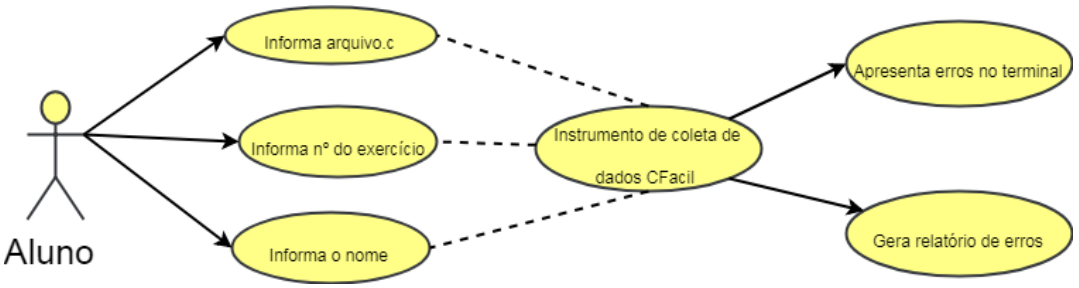
3.1. Aplicação Cliente

Requisito 1: [RF01] Instrumento de coleta de dados CFacil

- **Descrição:** Solicita ao aluno o nome do arquivo.c que ele deseja compilar, o nº do exercício que ele está executando e a matricula para compilação do seu código fonte.
- **Entradas:** Nome do arquivo.c, nº do exercício e nome.
- **Processo:** Captura os dados cadastrais da aplicação CFacil+ referente a matrícula do aluno e aos dados de compilação dos arquivos.c e identifica os erros mais comuns cometidos pelo aluno durante as compilações.
- **Saída:** Apresenta os erros ao aluno em um terminal e em um arquivo com o relatório de erros em formato txt. O relatório, armazena os erros cometidos pelo aluno e guarda data e hora de cada uma das compilações para análise. Este arquivo é salvo em um diretório.
- **Prioridade:**

x	Essencial		Importante		Desejável

Caso de uso:



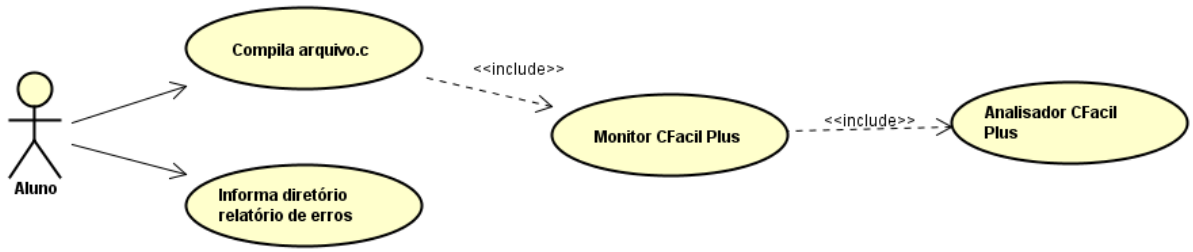
Requisito 2: [RF02] [Monitor CFacil+]

- **Descrição:** Realiza o monitoramento das máquinas dos alunos a partir da especificação do diretório que está sendo armazenado o relatório de erros.
- **Entradas:** Especificação do diretório da máquina do aluno que contém o relatório de erros.
- **Processo:** Realiza um monitoramento a cada 10 segundos do diretório da máquina do aluno que arquiva o relatório de erros. O processo é incremental e cada alteração no relatório (que se dá a partir de uma nova compilação de um arquivo.c pelo aluno) é detectada pelo monitor.
- **Saída:** A cada monitoramento, vai enviando as informações contidas no relatório de erros para o Analisador CFacil+.

- **Prioridade:**

x	Essencial		Importante		Desejável

Caso de uso:

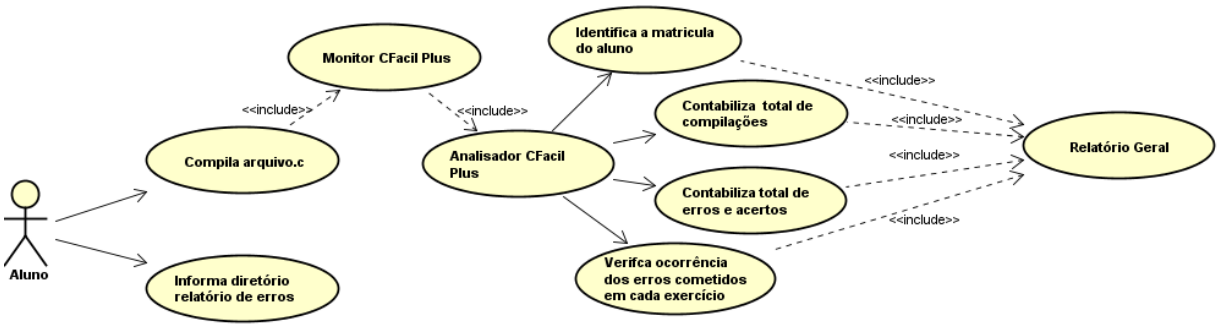


Requisito 3: [RF03] [Analisador CFacil+]

- **Descrição:** Verificar as informações contidas no relatório de erros enviadas pelo Monitor CFacil+ para contabilização dos erros e acertos do aluno.
- **Entradas:** Não é preciso especificar nenhuma informação ao sistema.
- **Processo:** Identifica a matricula do aluno, contabiliza a quantidade total de compilações realizadas, a quantidade total de erros e acertos e informa a descrição e a ocorrência dos erros cometidos em cada exercício a partir da análise das informações contidas no relatório de erros que vão sendo enviadas através do Monitor CFacil+.
- **Saída:** Encapsula todos os dados correspondentes a análise em um relatório geral a ser enviado via rede ao professor.
- **Prioridade:**

x	Essencial		Importante		Desejável

Casos de Uso:



Requisito 4: [RF04] [Socket cliente CFacil+]

- **Descrição:** Envia o relatório da análise para o Socket servidor CFacil+.
- **Entradas:** Especificação do endereço IP e número da porta do processo servidor.
- **Processo:** O cliente cria um socket conectando-se ao servidor através da porta e do endereço que foram especificados, prepara os dados do arquivo (em bytes) a ser enviado e solicita uma conexão para o envio do relatório da análise para o servidor. O cliente então, inicia o envio dos bytes dos dados sobre o socket

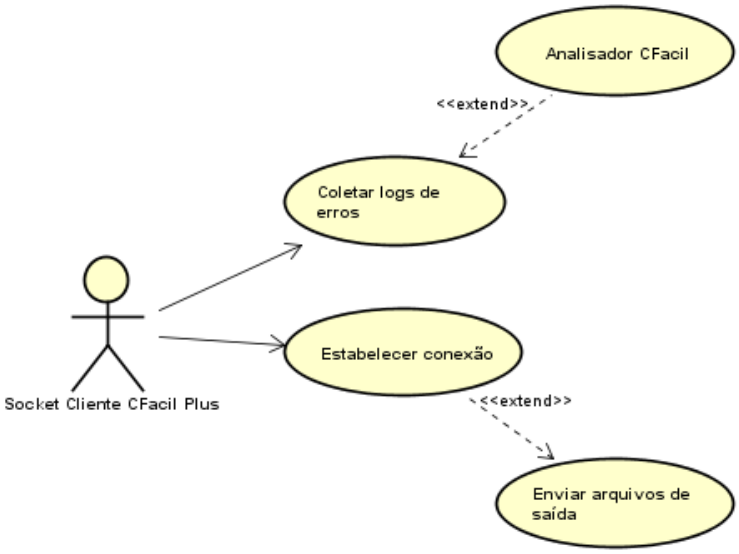
conectado. Neste ponto o servidor é notificado que uma conexão está sendo estabelecida.

- **Saída:** Estabelece conexão com o servidor e envia relatório da análise.

- **Prioridade:**

x	Essencial		Importante		Desejável

Caso de uso:

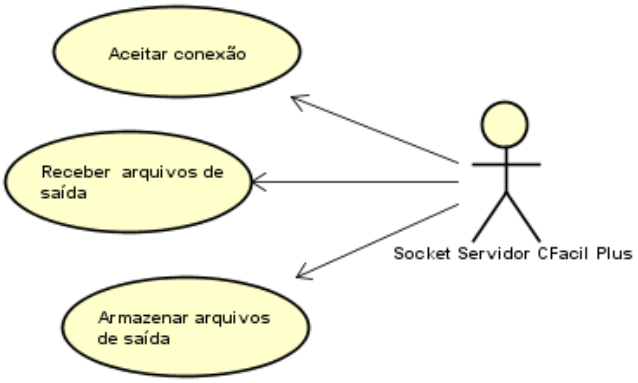


Requisito 5: [RF05] [Socket Servidor CFacil+]

- **Descrição:** O Servidor, quando iniciado, ficará aguardando o envio do relatório da análise CFacil+, quando a conexão for aceita, ele irá salvar o arquivo em seu diretório.
- **Entrada:** Especificação do diretório servidor onde deve ser salvo o relatório geral da análise do CFacil+.
- **Processo:** Recebe a solicitação do cliente e aceita. Uma vez que a conexão esteja estabelecida, o servidor cria um objeto *socket* que irá lidar com este cliente até que todos os pedidos de conexão terminem. O servidor então, inicia os preparativos para armazenar os *bytes* que chegam do cliente, recebe os *bytes* do arquivo de dados junto com o nome do arquivo e os armazena em um *array* de *bytes*, obtém o arquivo a partir dos *bytes* de dados recebidos e salva o arquivo em seu diretório.
- **Saída:** O arquivo da Análise CFacil+ deve ser salvo no diretório especificado na máquina do professor.
- **Prioridade:**

x	Essencial		Importante		Desejável

Casos de uso:



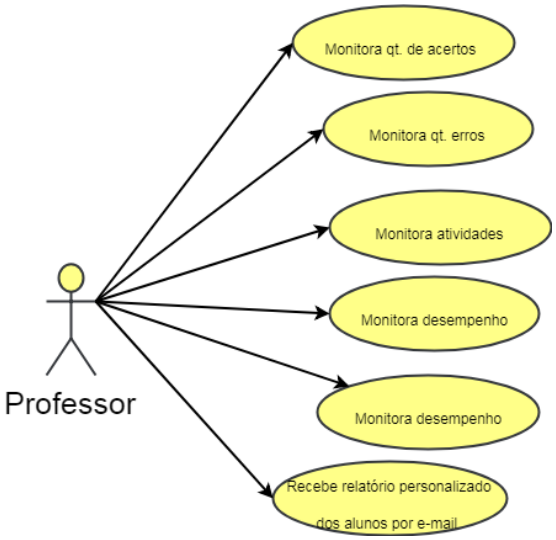
Requisito 6: [RF06] [Interface para acompanhamento das atividades dos alunos]

- **Descrição:** Permite que o professor monitore os alunos que estão compilando seus arquivos.c a partir de uma tabela que apresenta os relatórios da análise CFacil+.
- **Entradas:** Não é preciso especificar nenhuma informação ao sistema.
- **Processo:** A partir de uma tabela, é possível realizar o monitoramento das atividades que estão sendo desenvolvidas em sala de aula, através de uma tela que exibe ao professor: o nome do aluno, o nº total de compilações, a quantidade total de acertos e erros, bem como a descrição e a ocorrência dos erros cometidos em cada exercício pelos estudantes durante as atividades práticas de programação e um *status* de desempenho.
- **Saída:**

Prioridade:

x	Essencial		Importante		Desejável

Casos de uso:



4. Requisitos Não Funcionais

4.1. Usabilidade

A interface com o usuário é de vital importância para o sucesso do sistema. Principalmente por ser um sistema que não será utilizado diariamente, o usuário não possui tempo disponível para aprender como utilizar o sistema.

O sistema terá uma interface amigável e objetiva ao usuário.

Prioridade:

x	Essencial		Importante		Desejável

4.2. Desempenho

Embora não seja um requisito essencial ao sistema, deve ser considerada por corresponder a um fator de qualidade de software.

Prioridade:

	Essencial	X	Importante		Desejável

4.3. Hardware e Software

A linguagem principal adotada no desenvolvimento do trabalho será Java Script. Entretanto, outras linguagens também poderão ser usadas quando indicações técnicas recomendem.

Prioridade:

	Essencial	X	Importante		Desejável

5. Referências Bibliográficas

SOMMERVILLE, Ian. **Engenharia de software**. 9 Ed. São Paulo: Pearson, 2011

APÊNDICE II

Plano de Testes – CFacil +

Modelo de Roteiro de Testes

Projeto: CFacil+ **Autor: Patricia Padula Lopes**

Roteiro – v. 001.1	**Data: 22/05/2017**

Contador:	**001**
Criticidade:	Alta
Objeto de Teste:	Operações referentes ao Instrumento de Coleta de Dados da Ferramenta CFacil+
Caso de Teste:	Testar a compilação de arquivos.c com o CFacil+, verificar se ocorre a apresentação do relatório de erros em formato txt e através do terminal de compilação.
Pré - Condição:	**1.** Especificação do nome do arquivo.c, nº do exercício e nome do aluno.
Procedimento:	**1.** Entrar no terminal **2.** Informar o arquivo.c que se deseja compilar e a numeração do exercício. **3.** Clicar na tecla enter do teclado.
Resultado Esperado:	**1.** O sistema deve apresentar no terminal um relatório com os erros cometidos. **2.** As informações apresentadas no terminal devem ser salvas em um relatório .txt. **3.** O arquivo com o relatório de erros deve ser armazenado no diretório raíz da aplicação Cliente.

Contador:	**002**
Criticidade:	Alta
Objeto de Teste:	Operações referentes ao funcionamento do Monitor CFacil+
Caso de Teste:	Testar se o Monitor CFacil+ está realmente monitorando as máquinas dos alunos a partir da especificação do diretório que está sendo armazenado o relatório de erros.
Pré - Condição:	**1.** Especificação do caminho do diretório raiz da aplicação Cliente, onde está sendo armazenado o relatório de erros.
Procedimento:	**1.** Especificar no Monitor CFacil+ o diretório que contém o relatório de erros.
Resultado Esperado:	**1.** O sistema deve realizar um monitoramento a cada 10 segundos no diretório da máquina do cliente que arquiva o relatório de erros. **2.** Cada alteração no relatório que se dá a partir da compilação de um

	novo arquivo.c deve ser detectada pelo monitor. **3.** Após cada monitoramento vai enviando as informações contidas no relatório para o Analisador CFacil+.

Contador:	**003**
Criticidade:	Alta
Objeto de Teste:	Verificação do funcionamento do Analisador CFacil+
Caso de Teste:	Testar se após o envio das informações contidas no relatório de erros pelo monitor, o Analisador CFacil+ está contabilizando a quantidade total de erros e acertos de cada aluno, a ocorrência de cada um dos erros por exercício e armazenando estas informações juntamente com o nome do aluno em um arquivo de saída a ser apresentado no servidor.
Pré - Condição:	**1.** Especificação do diretório raiz da aplicação Cliente onde contém o relatório de erros.
Procedimento:	**1.** Especificar o caminho da máquina cliente onde o relatório de erros é armazenado e monitorado.
Resultado Esperado:	**1.** Capturar a nome do aluno. **2.** Contabilizar a quantidade total de compilações. **3.** Contabilizar a quantidade total de acertos. **4.** Contabilizar a quantidade total de erros. **5.** Descrever a ocorrência e a descrição de cada erro por exercício. 6. Salvar os dados da análise em um arquivo .txt

Contador:	**004**
Criticidade:	Alta
Objeto de Teste:	Socket Cliente CFacil+
Caso de Teste:	Testar o envio via rede do relatório após a sua análise, do socket cliente para o socket servidor.
Pré - Condição:	**1.** Especificação do endereço IP da máquina que hospeda o servidor.
Procedimento:	**1.** Informar o endereço IP do servidor.
Resultado Esperado:	**1.** O cliente deve criar um socket conectando-se ao servidor através da porta e do endereço que foram especificados. **2.** O cliente prepara os dados do arquivo (em bytes) a ser enviado. **3.** Deve ser iniciado o envio dos bytes dos dados sobre o socket conectado. **4.** O servidor é notificado que uma conexão está sendo estabelecida.

Contador:	**005**
Criticidade:	Alta
Objeto de Teste:	Socket Servidor CFacil+
Caso de Teste:	Testar o recebimento via rede do relatório enviado através do socket cliente para o socket servidor.
Pré - Condição:	**1.** Especificação do diretório raiz da aplicação Servidor onde o relatório deverá ser salvo.

Procedimento:	**1.** Informar o caminho do diretório raiz da aplicação Servidor.
Resultado Esperado:	**1.** O servidor irá receber solicitações de vários clientes para envio dos relatórios. **2.** O servidor deve aceitar simultaneamente a solicitação do envio dos relatórios da análise dos clientes conectados na rede. **3.** O servidor, a partir da implementação das threads deve redirecionar ambos clientes de uma porta para outra, liberando novamente sua porta inicial e permitindo que outros clientes se conectem novamente. **4.** É inicializados os preparativos para armazenar os bytes que chegam de cada um dos clientes. **5.** O servidor recebe os *bytes* do arquivo de dados junto com o nome do arquivo e os armazena em um *array* de *bytes*. **6.** O servidor obtém o relatório da análise, a partir dos bytes de dados recebidos. **7.** O arquivo deve ser salvo no diretório especificado.

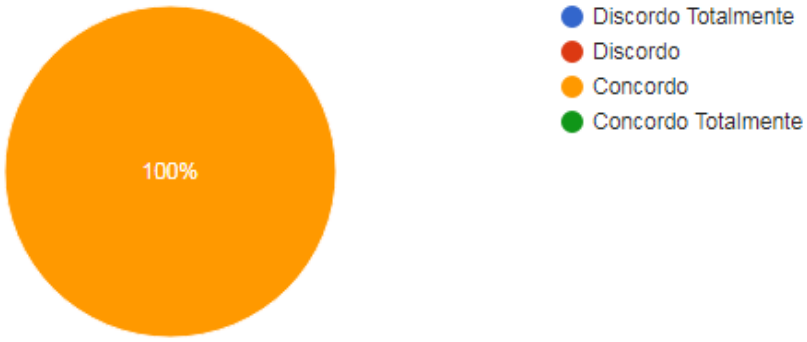
Contador:	**006**
Criticidade:	Alta
Objeto de Teste:	Interface para acompanhamento das máquinas dos cliente
Caso de Teste:	Testar a apresentação dos relatórios da análise CFacil+ de cada um dos clientes monitorados.
Pré - Condição:	**1.** Não há pré-condições estabelecidas.
Procedimento:	**1.** Verificar a apresentação das informações em uma tabela que contém a identificação do nome do aluno, a quantidade de compilações, erros e acertos cometido, bem como um *status* de desempenho.
Resultado Esperado:	**1.** Permitir a apresentação das informações referente ao monitoramento dos alunos.

APÊNDICE III

Questionário pós-teste CFacil+

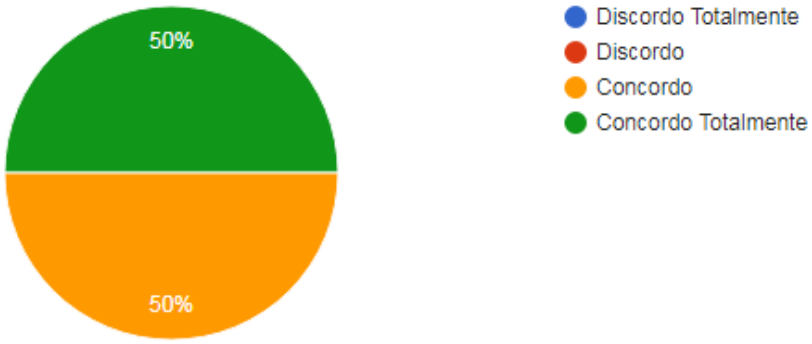
As aulas de algoritmos e programação poderiam tornar-se mais produtivas com a utilização deste sistema

2 responses



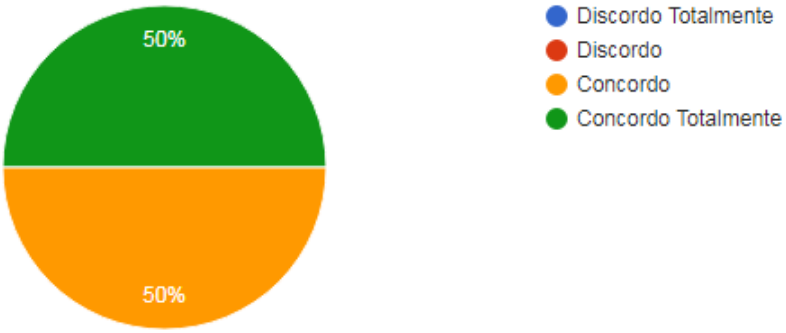
O sistema proporciona uma visualização imediata do que está acontecendo durante as atividades dos alunos

2 responses



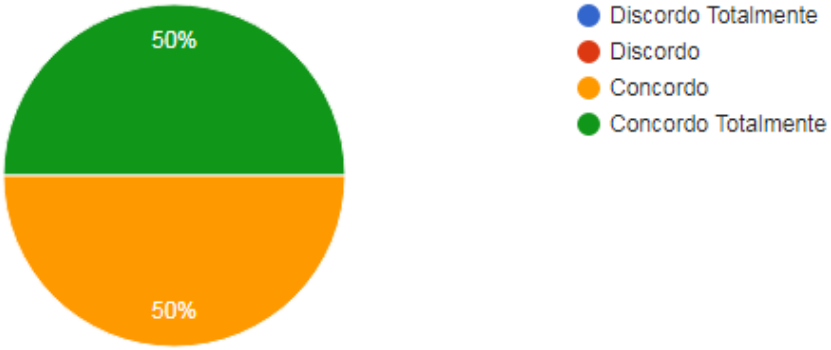
A interface da ferramenta é fácil e intuitiva

2 responses



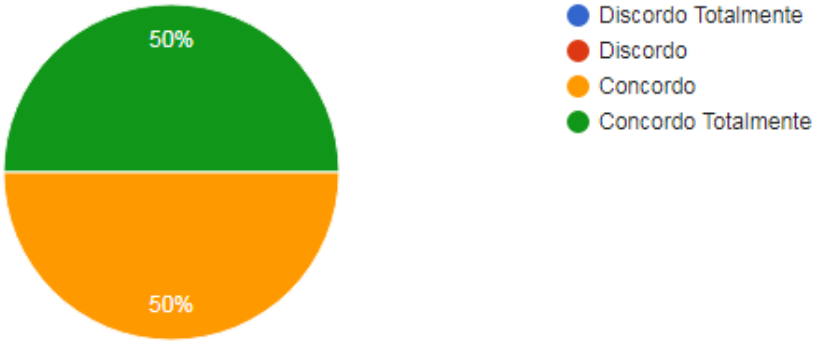
É possível realizar o monitoramento dos alunos na interface de monitoramento

2 responses



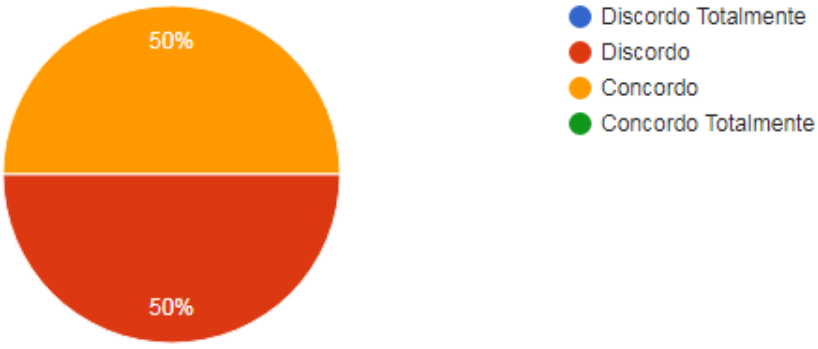
As informações fornecidas pelo sistema permitem identificar os alunos que apresentam maior quantidade de erros com base no analisador CFacil

2 responses



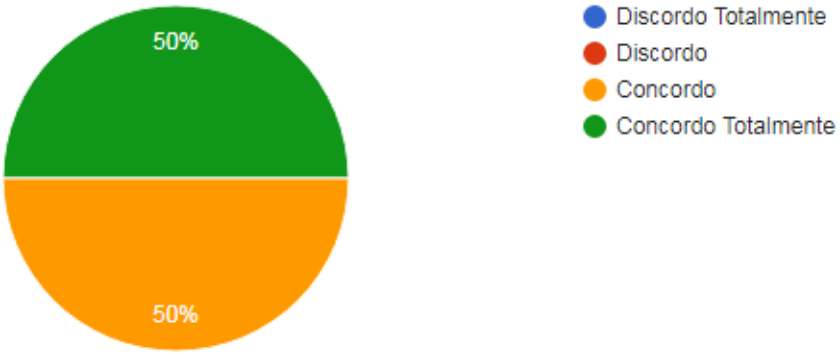
É possível analisar quais são os tipos de erros que são cometidos pelos estudantes a partir de relatórios detalhados que são atualizados periodicamente pela ferramenta

2 responses



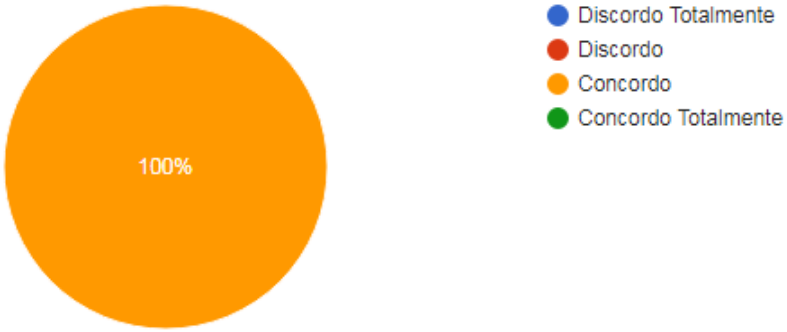
As informações sobre a identificação do aluno, quantidade de erros, acertos e compilações, são apresentadas de forma clara na interface de monitoramento

2 responses



O sistema apresenta informações relevantes para monitorar o desempenho dos alunos durante as aulas práticas de programação

2 responses



No geral, me senti satisfeito ao aplicar esta ferramenta durante minhas aulas

2 responses