Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Programação Orientada ao Problema: Uma Metodologia para Entendimento de Problemas e Especificação no Contexto de Ensino de Programação para Iniciantes

Andréa Pereira Mendonça

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linhas de Pesquisa: Modelos Computacionais e Cognitivos,
Engenharia de Software

Evandro de Barros Costa e Dalton D. Serey Guerrero
(Orientadores)

Campina Grande – Paraíba – Brasil

Ficha Catalográfica Elaborada pela Biblioteca Central da UFCG

M963p Mendonça, Andréa Pereira

Programação orientada ao problema: uma metodologia para entendimento de problemas e especificações no contexto de ensino de programação para iniciantes / Andréa Pereira Mendonça. – Campina Grande: UFCG, 2010.

187 f. : il. color.

Referências.

Tese (Doutorado em Ciência da Computação) Centro de Engenharia Elétrica e Informática, Universidade Federal de Campina Grande-UFCG.

Orientadores: Prof. Dr. Evandro de Barros Costa.
Prof. Dr. Dalton D. Serey Guerrero.

1. Programação de Computadores - Ensino 2. Engenharia de Software I. Título

CDU 004.42:37(043)

"PROGRAMAÇÃO ORIENTADA AO PROBLEMA: UMA METODOLOGIA PARA ENTENDIMENTO DE PROBLEMAS E ESPECIFICAÇÃO NO CONTEXTO DE ENSINO DE PROGRAMAÇÃO PARA INICIANTES"

ANDRÉA PEREIRA MENDONÇA

TESE APROVADA EM 30.11.2010

EVANDRO DE BARROS COSTA, D.Sc
Orientador(a)

DALTON DARIO SEREY GUERRERO, D.Sc
Orientador(a)

JORGE CESAR ABRANTES DE FIGUEIREDO, D.Sc
Examinador(a)

TIAGO LIMA MASSONI, Dr.
Examinador(a)

CHRISTINA VON FLACH GARCIA CHAVEZ, DRª
Examinador(a)

SERGIO CASTELO BRANCO SOARES, Dr.
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Tradicionalmente, as disciplinas introdutórias de programação enfatizam o desenvolvimento do raciocínio lógico e a construção de programas mais que qualquer outra habilidade. Nestas disciplinas, problemas bem definidos são recursos utilizados para auxiliar os estudantes na aprendizagem de programação. Entretanto, essa abordagem de ensino apresenta um problema: ela não expõe os estudantes a um processo de resolução de problemas realista, que contempla o espaço do problema (entendimento de problemas e especificação de requisitos) em conjunto com o espaço da solução (construção de programas). Em virtude disto, são freqüentes os relatos da literatura acerca das dificuldades dos estudantes iniciantes em lidar com entendimento de problemas mal definidos e especificação de requisitos. Para tratar o problema apresentado, nós defendemos que as disciplinas introdutórias de programação devem (i) utilizar estratégias de ensino que assumam a programação como parte da Engenharia de Software; e (ii) desenvolver atividades que permitam aos estudantes irem da especificação dos requisitos ao programa, por meio da resolução de problemas mal definidos. Baseados nisto, nós concebemos uma metodologia de ensino denominada Programação Orientada ao Problema (POP). POP caracteriza-se por um conjunto de atividades que podem ser inseridas na disciplina introdutória de programação. Ela é constituída por um ciclo de resolução de problemas que põe em prática atividades típicas da Engenharia de Software, tais como, elicitação e especificação de requisitos, testes e programação. Nós avaliamos POP por meio de estudos de caso e experimentos controlados. Nos experimentos, nós adotamos um conjunto de variáveis que consideramos representativas do bom desempenho dos estudantes para lidar com o espaço do problema em conjunto com o espaço da solução. Das variáveis observadas, em $83{,}3\%$ delas os estudantes POP foram mais eficientes que os estudantes não-POP. Para exemplificarmos, nos dois experimentos, os estudantes POP documentaram, em média, mais de 79% de requisitos que os estudantes não-POP. Além disso, implementaram 4 versões a menos do programa para atender a todos os requisitos, quando comparados aos estudantes não-POP. Embora os resultados não possam ser generalizados, o tratamento que fizemos das ameaças à validade nos permite aferir a qualidade dos resultados obtidos.

Abstract

Traditionally, introductory programming courses emphasize the development of logical reasoning and the construction of programs, more than any other skill. In theses couses, well-defined problems are resources used to assist students in their learning of programming. However, this teaching approach has a problem: it fails to expose students to a realistic problem solving process, involving the problem space (understanding the problem and requirements specification) along with the solution space (program construction). Thus, there have been too many reports in the literature concerning the novice students' difficulties in understanding ill-defined problems and requirements specification. To address the problem presented, we defended that introductory programming courses should (i) use teaching strategies that take programming as part of Software Engineering, and (ii) develop activities that enable students to go from requirements specification to the program by solving of ill-defined problems. Based on this, we have designed a teaching methodology called Problem Oriented Programming (POP). POP is characterized as a set of activities that can be inserted to the introductory programming courses. It consists of a cycle of problem solving that puts into practice typical activities of the Software Engineering, such as: requirements elicitation and specification, testing and programming. POP has been evaluated by means of case studies and controlled experiments. For the experiments, we have adopted a set of variables found to be most representative of the students' performance when dealing with the problem space together with the solution space. In 83.3% of all observed variables, POP students exhibited far greater efficiency than non-POP students. To exemplify this: in both experiments, POP students documented, on average, over 79% of the requirements that the non-POP students. Furthermore, POP students implemented four versions less of the program to meet all the requirements compared with the non-POP students. Despite the fact that the results cannot be generalized, the way we treated the threats to validity allow us to ensure the quality of our results.

Agradecimentos

A Deus que me permitiu aprender e progredir por meio do trabalho, perseverar mesmo diante dos obstáculos e encontrar suporte e colaboração de pessoas tão especiais, quanto as que cito aqui.

A minha mãe, avô, irmãs, sobrinhas e sobrinho por me ajudarem a manter o foco, a serenidade e a fé. Por serem minhas referências e meu porto seguro.

Aos meus orientadores Evandro Costa e Dalton Serey pela generosidade com que colaboraram para a minha formação e para o meu progresso na pesquisa. Especialmente, agradeço ao Dalton por ter me permitido participar das discussões e planejamento da disciplina de programação na UFCG e inserido POP como parte integrante desta disciplina. Sem essa valiosa contribuição muito pouco teríamos avançado em termos de investigações empíricas.

Agradeço a Verônica, esposa de Dalton, por permitir discussões de trabalho em sua casa e por me receber tão bem.

A Elloá Guedes pela companhia, colaboração e ajuda incontáveis. Pela presença constante nos bastidores da pesquisa, onde pude encontrar apoio nos momentos difíceis e comemoração fraterna nos momentos de progresso.

Ao meu amigo Francisco Neto, meu companheiro de Pilates e de discussões sobre experimentos.

A Danielle Chaves, um especial agradecimento, pelo suporte operacional dado a minha pesquisa e por me permitir orientá-la em alguns trabalhos.

A Cheyenne Ribeiro e Mariana Romão pela amizade, trabalho em grupo, troca de informações e ajuda mútua.

A professora Maria Cristina dos Santos, minha orientadora de PIBIC na graduação. Ali surgia o meu gosto pela pesquisa.

Ao professor Jorge Abrantes pelas avaliações que fez do meu trabalho, desde o meu primeiro WDCOPIN. Pelos trabalhos que fizemos em colaboração e pelas boas discussões que tivemos sobre ensino-aprendizagem de programação.

A professora Joseana Fechine pelos trabalhos em parceria e pelo incentivo constante. Ao pro-

fessor Bernardo Lula por ampliar meu conhecimento sobre teoria da computação e pelo trabalho colaborativo na condução do Laboratório de Inteligência Artificial.

Ao Expedito Lopes cuja amizade, companheirismo e "baianidade" me cativaram.

A Séfora Junqueira pelo incentivo, amizade e acolhimento nas minhas excursões de trabalho à Maceió.

A Ana Ely Sousa pelo incentivo constante.

Aos amigos da pós-graduação Ayla e Rodrigo Rebouças, Roberto Bittencourt, Álvaro Vinicios ("Degas"), Lívia Sampaio, Raquel Lopes, Matheus Gaudêncio, Katyusco Santos e Gilson Pereira. Amigos dessa longa caminhada que, mesmo diante de um trabalho tão solitário quanto o trabalho de tese, possibilitaram compartilhar conhecimento e, principalmente, palavras de incentivo e perseverança.

Aos professores Walter Pessoa e Gustavo Esteves pelo suporte na análise estatística dos dados e a professora Izabel (DME-UFCG) pelo livros compartilhados.

A Aninha pelo suporte constante na secretaria da pós-graduação, sempre nos atendendo com cordialidade e nos conduzindo, adequadamente, na solução dos problemas. A Vera pela paciência nos momentos de matrícula e na expedição de documentos e também pelas cobranças quando algo estava em atraso.

A Fatinha, uma amiga e incentivadora, que em nossas reuniões na sala da coordenação sempre "driblava" o povo para não atrapalhar.

Aos amigos do IFAM, Irlene Matias, Vicente Lucena, Juliana Lucena, Jucimar Brito e José Pinheiro. Grandes incentivadores e colegas de trabalho.

A Lílian Patrícia e família pela recepção e ajuda quando cheguei em Campina Grande.

A Edmary Dias, Luciana Souza, Rose Sobrinho, Sachie Yanai e Vívian Lane, amigas que, mesmo à distância, mantiveram contato freqüente.

Aos monitores da disciplina de programação da UFCG e aos estudantes que, voluntariamente, contribuíram para a minha pesquisa.

Aos professores Hamilton Soares, Raimundo Nóbrega e Ulysses de Oliveira e aos alunos Ana Paula Nunes e Glauco de Souza pelo suporte a pesquisa.

Aos membros da banca pela apreciação do meu trabalho e pelas sugestões de melhoria.

A FAPEAM pelo apoio financeiro e ao IFAM por prover a minha dispensa para cursar o doutorado.

Epígrafe

Se avexe não
Toda caminhada começa
No primeiro passo
A natureza não tem pressa
Segue seu compasso
Inexorávelmente chega lá

(*Música: A natureza das coisas. Compositor: Acioli Neto.*)

Dedicatória

À minha família, com todo o meu amor.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Capítulo 1

Introdução

A disciplina introdutória de programação é base para várias outras disciplinas do currículo de Computação [IEEE/ACM 2001; MEC/SESU 1999]. É nela que os estudantes iniciam seu contato com a resolução de problemas em Computação e desenvolvem habilidades que irão subsidiá-los no decorrer do curso.

Tradicionalmente, esta disciplina concentra-se muito mais no *espaço da solução*[1], isto é, na construção de programas, desenvolvimento do raciocínio lógico e aprendizagem da sintaxe de uma linguagem de programação [Falkner and Palmer 2009; Pears et al. 2007]. *Problemas bem definidos* são típicos recursos utilizados para auxiliar os estudantes na prática dos conceitos de programação [O'Kelly and Gibson 2006]. Estes problemas apresentam enunciados bem especificados, isto é, suas informações são suficientes para esclarecer o que deve ser feito. Assim, o entendimento do problema é facilitado e os estudantes podem concentrar seus esforços na elaboração e implementação de uma solução para o problema proposto.

Embora essa abordagem de ensino permita aos estudantes desenvolverem habilidades importantes e necessárias para o domínio de programação, nós acreditamos que ela conduz a uma série de restrições no processo de ensino-aprendizagem dos estudantes iniciantes. Uma delas, é que a disciplina não contempla problemas que retratem as características dos problemas do mundo real, os quais são mal definidos. *Problemas mal definidos* são aqueles cujo enunciado não é completo e pode conter ambigüidades, contradições, falta de informações, informações incorretas e/ou irrelevantes [Falkner and Palmer 2009; VanLehn 1989]. Contrariamente aos problemas bem definidos, os problemas mal definidos exigem que os estudantes mobilizem esforços para entender o problema antes de prover

[1]Estamos utilizando o termo *espaço da solução* para designar o conjunto das soluções possíveis para um dado problema. Portanto, o termo está associado ao "como" o problema será solucionado.

uma solução para o mesmo.

Outra restrição é que, ao utilizar exclusivamente problemas bem definidos, a disciplina introdutória de programação reduz o processo de resolução de problemas ao *espaço da solução*, perdendo a dimensão do *espaço do problema*[2]. Ao negligenciar o espaço do problema, os estudantes perdem a oportunidade de desenvolver habilidades para o entendimento de problemas e especificação dos requisitos do programa. Atividades que são inerentes ao processo de desenvolvimento de software.

É possível verificar também que essa abordagem fomenta uma separação entre programação e as demais atividades da Engenharia de Software, tais como, especificação e elicitação de requisitos. Em face dos aspectos citados, a disciplina introdutória de programação provê uma visão pouco realista do processo de desenvolvimento de software.

Os aspectos que mencionamos evidenciam o seguinte problema:

> As disciplinas introdutórias de programação, nos moldes tradicionais, não expõem os estudantes a um processo de resolução de problemas que contempla o *espaço do problema* em conjunto com o *espaço da solução*.

Evidências reforçam a existência deste problema. Diversos relatos na literatura apontam dificuldades dos estudantes iniciantes em lidar com situações que exigem habilidades para entendimento de problemas e especificação de requisitos.

McCracken et al. [2001] realizaram um estudo multi-nacional e multi-institucional para avaliar as habilidades de programação dos estudantes de Ciência da Computação no primeiro ano de curso. Eles observaram 216 estudantes de 4 universidades e os resultados apontaram que a maioria dos estudantes tinha dificuldades em abstrair do enunciado o problema que deveria ser resolvido. Nesse estudo, o enunciado do problema incluía detalhes irrelevantes e, na visão dos pesquisadores, este fato contribuiu para as dificuldades dos estudantes em identificar os objetivos do problema.

Blaha et al. [2005] também realizaram um estudo multi-nacional e multi-institucional com o objetivo de observar a capacidade dos estudantes iniciantes, graduandos e educadores em Ciência da Computação de reconhecer ambigüidades em uma especificação de software, descrita em linguagem natural. A tarefa dos participantes consistia em construir, a partir da especificação, uma solução inicial e decompor essa solução em partes. Os resultados desse estudo revelaram que apenas 63% dos estudantes iniciantes reconheceram ambigüidades na especificação e que 48% dos estudantes que reconheceram as ambigüidades, fizeram suposições, ao invés de explicitamente questionarem sobre as informações ambíguas.

[2]Estamos utilizando o termo *espaço do problema* para designar a descrição e caracterização do problema a ser resolvido. Portanto, o termo está associado à definição "do que" deve ser feito.

Nós também detectamos dificuldades similares ao realizarmos um estudo de caso com estudantes iniciantes de programação [Mendonça et al. 2009a]. Em nossa investigação, os estudantes tinham que resolver um problema mal definido, isto é, um problema cujo enunciado apresentava, intencionalmente, ambigüidades e falta de informações. Nesse estudo, nós verificamos que 80% dos estudantes, após lerem o enunciado do problema, iniciaram imediatamente a implementação do programa ou a escrita do algoritmo, sem perceber os "defeitos" do enunciado. Além disso, 50% dos estudantes necessitaram codificar mais de 4 versões do programa para atender a todos os requisitos. Os resultados desse estudo também apontaram dificuldades dos estudantes no que diz respeito à formulação de questões para esclarecer dúvidas, análise das restrições do programa e elaboração de testes para os casos não óbvios.

Há outros relatos que não resultam de investigações empíricas, mas da percepção dos professores devido a sua experiência com o ensino da disciplina introdutória de programação. Eastman [2003], por exemplo, afirma que os estudantes têm dificuldades em resolver problemas cujos enunciados são apresentados na forma narrativa. Para o autor, embora esses problemas se assemelhem aos problemas do mundo real, a existência de informações extra freqüentemente confunde os estudantes iniciantes, os quais comumente questionam: "Mas qual é o problema?". Similarmente, Muller et al. [2007] afirmam que os estudantes iniciantes têm dificuldades em identificar os elementos essenciais do problema e relacioná-los.

Essas dificuldades, no entanto, não estão restritas aos estudantes iniciantes. Eckerdal et al. [2006] realizaram um estudo multi-nacional e multi-institucional envolvendo 150 estudantes que estavam próximos de concluírem o curso de Ciência da Computação. O objetivo era investigar a capacidade destes estudantes de projetar software. Nesta investigação, os estudantes receberam uma pequena descrição das funcionalidades desejadas no sistema e poderiam questionar o investigador para sanar suas dúvidas. A especificação do projeto foi o artefato utilizado para análise dos dados. Segundo os autores, 21% das especificações eram meras transcrições das funcionalidades apresentadas aos estudantes, sem qualquer informação adicional e mais de 60% das especificações continham informações pouco significativas ao projeto. Segundo os autores, a maioria dos estudantes não entendia quais informações que um projeto deveria incluir e como elas deveriam ser comunicadas. Os autores justificaram o baixo desempenho dos estudantes argumentando que durante o curso de graduação os estudantes são acostumados a tratar com especificações tão claras quanto possíveis. Em suas conclusões, os autores escreveram: "*Talvez nós estejamos errados e devamos oferecer aos estudantes mais experiências com tarefas pouco especificadas*" [Eckerdal et al. 2006, p.407].

Segundo Conn [2002], essas dificuldades propagam-se até a prática profissional, na indústria de

software. Hall et al. [2002], por meio de estudos empíricos em 12 empresas, detectaram que do total de defeitos identificados no processo de desenvolvimento de software, 48% deles tinham origem na especificação de requisitos. Outras referências que indicam essas dificuldades dos estudantes e profissionais podem ser obtidas em De Lucena et al. [2006], Garg and Varma [2008] e Hofmann and Lehner [2001].

As dificuldades com entendimento de problemas e especificação de requisitos são tão comuns que chegam a ser tratadas de maneira jocosa, conforme ilustramos na Figura 1.1[3].

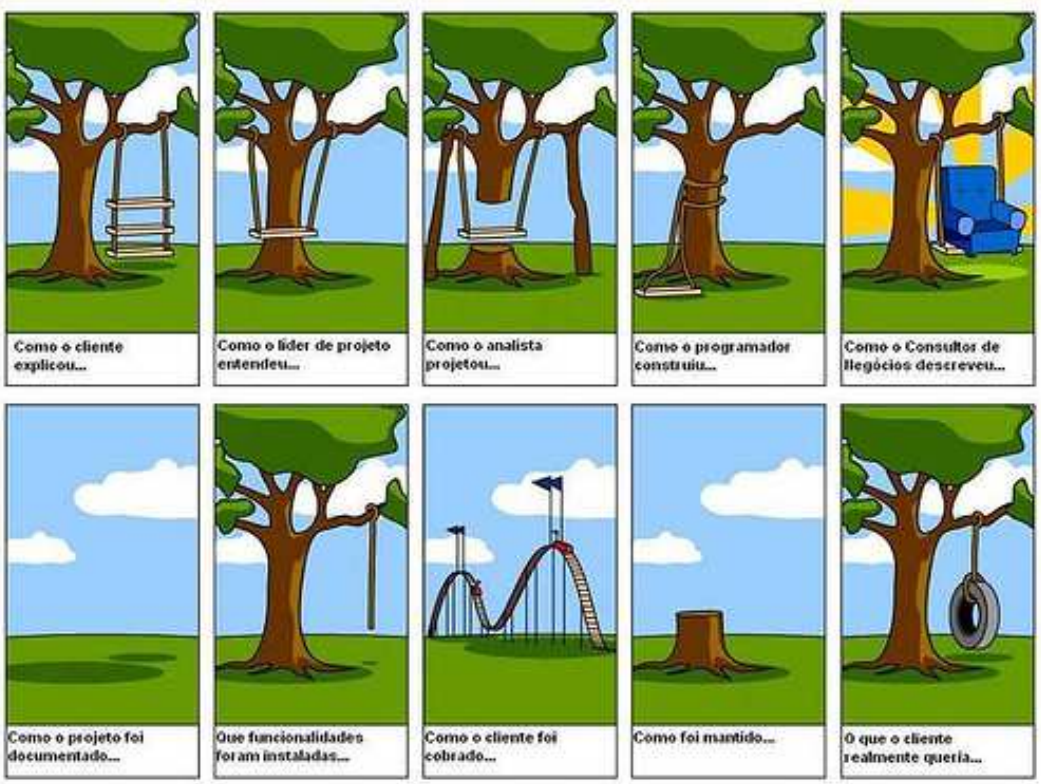

Figura 1.1: Dificuldades de entendimento do problema.

Retomando o problema que enunciamos, com respeito às disciplinas introdutórias de programação, podemos afirmar que prover recursos para mitigá-lo não é trivial. Ao contrário do que alguns podem pensar, não se trata apenas de mover as disciplinas de Engenharia de Software ou Análise de Sistemas para o início do curso, ministrando-as simultaneamente com a disciplina de programação. Isso acarretaria em uma incoerência na forma como os problemas são resolvidos em diferentes disciplinas. Por exemplo, seria incoerente resolvermos problemas na disciplina de Engenharia de Software partindo de estratégias mais amplas de resolução, na qual os estudantes vão da especificação dos requisitos ao programa; e na disciplina de programação reduzir o processo de resolução de problemas

[3]Fonte: `http://bit.ly/9VIPgk`

apenas ao espaço da solução (criação do programa).

Nós acreditamos que é necessário realizar mudanças na disciplina introdutória de programação para abranger também o espaço do problema. Porém, tais mudanças não devem comprometer a prática da construção de programas, nem tampouco requerer demasiadas alterações no conteúdo programático da disciplina. Deve ser possível para estudantes iniciantes realizarem atividades inerentes ao espaço do problema sem que isso exija deles uma carga excessiva de conceitos e conhecimentos prévios.

O ideal é que o processo de resolução de problemas de uma disciplina introdutória de programação se aproxime de um processo realista de desenvolvimento de software, contemplando clientes, *deadlines*, diferentes artefatos, várias iterações, etc. Apesar disso, é imprescindível que estas atividades possam ser dimensionadas no escopo da disciplina, ou seja, a complexidade precisa ser reduzida, mas sem descaracterizar o processo. Além disso, avaliar a efetividade dessas mudanças na disciplina é imprescindível, pois permite identificar as melhorias no processo ensino-aprendizagem e os ajustes que se façam necessários.

Com o intuito de provermos uma solução para o problema que enunciamos, nós assumimos alguns pressupostos. Primeiramente, o de que *programação é parte da Engenharia de Software*. Isso contrasta com algumas opiniões pré-estabelecidas na comunidade de ensino. Dijkstra [1976], ao tratar sobre ensino de programação, defende que a habilidade de raciocínio deve ser enfatizada mais que qualquer outra habilidade. Esse pensamento está presente nos cursos introdutórios de programação tradicionais, na medida em que eles evidenciam muito mais o espaço da solução. Nós defendemos que é necessário pensar na programação numa perspectiva mais ampla. Programação resulta em um produto de software e, como tal, produz artefatos que devem atender necessidades de um cliente e cuja qualidade deve ser aferida. Portanto, não há como dissociar programação da própria Engenharia de Software. Tal pressuposto é também suportado por Meyer [2003].

Problemas bem definidos são efetivos de muitas formas: para apresentar conceitos de programação, motivar a prática desses conceitos e avaliar a aprendizagem dos estudantes. Porém são recursos mais restritivos quando se deseja explorar o espaço do problema. Isso ocorre porque há, naturalmente, uma ênfase no espaço da solução. Em contraste, problemas mal definidos requerem dos estudantes a mobilização de esforços para esclarecer o problema antes de prover uma solução para o mesmo. Portanto, são recursos que, necessariamente, enfatizam o espaço do problema em conjunto com o espaço da solução. Assim, nós defendemos que, além dos problemas bem definidos, *os problemas mal definidos sejam também incorporados na disciplina introdutória de programação*.

Nosso terceiro pressuposto deriva do primeiro. Ao assumirmos que programação é parte de En-

genharia de Software, assumimos também que *é necessário estabelecer um ciclo de resolução de problemas que permita aos estudantes irem da especificação dos requisitos ao programa*, estabelecendo, com isso, similaridades com o que ocorre em um contexto real.

Em síntese, a proposição central desta tese para auxiliar na solução do problema apresentado é:

> Tratar o espaço do problema em conjunto com o espaço da solução no contexto da disciplina introdutória de programação demanda (1) a utilização de estratégias de ensino que assumam a programação como parte da Engenharia de Software e (2) o desenvolvimento de atividades que permitam aos estudantes irem da especificação dos requisitos ao programa, por meio da resolução de problemas mal definidos.

Baseados nisto, nós concebemos uma solução para o problema previamente apresentado. A solução consiste em uma metodologia de ensino para a disciplina introdutória de programação, denominada *Programação Orientada ao Problema (POP)*, que permite aos estudantes praticarem típicas atividades da Engenharia de Software, tais como, especificação e elicitação de requisitos, programação e testes.

Com POP os estudantes atuam tanto no espaço do problema quanto no espaço da solução por meio de um ciclo de resolução de problemas que os permite: (i) explorar problemas mal definidos; (ii) assumir o papel de desenvolvedor; (iii) produzir outros artefatos além do programa; (iv) refinar os artefatos produzidos; (v) atender a *deadlines*; e, (vi) trabalhar individualmente e também, cooperativamente, em pequenos grupos.

Para avaliarmos POP, nós realizamos dois tipos de investigação empírica: estudos de caso e experimentos controlados. Os estudos de caso tiveram por objetivo avaliar a aplicação de POP no contexto de sala de aula. Eles foram realizados nos dois semestres acadêmicos de 2009 com as turmas da disciplina introdutória de programação do Curso de Ciência da Computação da Universidade Federal de Campina Grande (UFCG). Os experimentos controlados também ocorreram nos dois semestres acadêmicos de 2009, próximo ao final de cada semestre letivo. O objetivo dos experimentos era avaliar o desempenho dos estudantes que utilizaram POP (estudantes POP) em contraste com estudantes que não adotaram POP (estudantes não-POP). Nos experimentos, os estudantes deveriam resolver um problema mal definido cujos requisitos deveriam ser esclarecidos por um cliente, papel desempenhado pelo investigador do estudo experimental. Para resolver o problema proposto, os estudantes deveriam implementar um programa que atendesse a 31 requisitos previamente estabelecidos.

Nestes experimentos, nós definimos um conjunto de variáveis que consideramos representativas para demonstrar o desempenho dos estudantes em lidar com o espaço do problema e da solução. Estas

variáveis foram: quantidade de requisitos documentados e de requisitos documentados corretamente, número de versões do programa, média de requisitos atendidos por versão do programa, porcentagem de questionamentos relevantes e tempo utilizado na resolução do problema.

Nos dois experimentos realizados, à exceção do tempo gasto na resolução do problema, em todas as demais variáveis os estudantes POP foram mais eficientes que os estudantes não-POP. No primeiro experimento, para as variáveis quantidade de requisitos documentados e número de versões do programa, por exemplo, os estudantes POP, em média, documentaram $29{,}6$ requisitos e precisaram de $1{,}6$ versões do programa para atender a todos os requisitos. Os estudantes não-POP, em média, documentaram $5{,}2$ requisitos e necessitaram criar $5{,}6$ versões do programa até que todos os requisitos fossem atendidos.

No segundo experimento, considerando essas mesmas variáveis, os estudantes POP, em média, documentaram $25{,}5$ requisitos e precisaram construir $1{,}7$ versões do programa para atender a todos os requisitos. Os estudante não-POP, em média, documentaram 1 requisito e necessitaram construir $7{,}9$ versões do programa para atender a todos os requisitos.

Em ambos os experimentos, esses resultados demonstraram que os estudantes POP foram mais efetivos que os estudantes não-POP na detecção das faltas de informações no enunciado do problema e na captura dos requisitos do programa.

A principal limitação de nosso estudo é o pequeno tamanho da amostra – 20 estudantes no primeiro experimento e 26 estudantes no segundo. Entretanto, é importante considerarmos o fato de que pesquisas envolvendo pessoas, especialmente com observações individuais, são complexas, caras e demandam muito tempo para serem executadas e avaliadas.

Embora não possamos generalizar os resultados, o tratamento que fizemos das ameaças à validade (interna, externa, de conclusão e de construção) nos permite aferir a qualidade dos resultados obtidos e concluir que POP: (1) expõe os estudantes a um ciclo de resolução que contempla o espaço do problema e da solução; e, (2) melhora o desempenho dos estudantes para tratar com entendimento de problemas e especificação de requisitos.

Nosso trabalho traz várias contribuições para a área de Educação em Computação. Nós disponibilizamos uma metodologia de ensino para disciplinas introdutórias de programação que prepara os estudantes para irem da especificação dos requisitos ao programa. Contribuímos também para a comunidade de Educação em Engenharia de Software que tem interesse em recursos educacionais que tenham ressonância na aprendizagem de Engenharia de Software [Lethbridge et al. 2007].

Identificamos também um conjunto de variáveis que permitiram, além de uma análise qualitativa, mensurar quantitativamente o desempenho dos estudantes com respeito a entendimento de problemas

e especificação de requisitos. Planejar avaliações quantitativas em pesquisas envolvendo aprendizagem de pessoas não é uma atividade trivial. Isto se deve a existência de aspectos abstratos, os quais são difíceis de mensurar.

Consolidamos um plano de investigação experimental cujos procedimentos podem ser observados, adaptados e/ou repetidos por outros pesquisadores da área. Essa é uma contribuição bastante significativa, principalmente se considerarmos que experimentos controlados são ainda pouco comuns na área de Educação em Computação [Valentine 2004; Hazzan et al. 2006].

Por meio dos resultados de nossas investigações empíricas, nós caracterizamos algumas das dificuldades dos estudantes com entendimento de problemas e ampliamos a discussão sobre os efeitos dos problemas bem e mal definidos na aprendizagem dos estudantes iniciantes de programação. Esses resultados são também bastante úteis para dar suporte a produção de livros e outros materiais didáticos. Atualmente, esses recursos não exploram o potencial dos problemas mal definidos.

Pesquisadores investigam a prática de testes por estudantes iniciantes de programação [Janzen and Saiedian 2006a; Desai et al. 2008; 2009] e nosso trabalho também contribui neste âmbito. Porém, com um diferencial: além de utilizarmos testes para melhorar a qualidade do programa construído, também utilizamos testes como um recurso para elicitar requisitos junto a um cliente.

Para apresentamos os aspectos teóricos que fundamentaram o nosso trabalho, bem como a caracterização e avaliação de nossa solução, nós organizamos este trabalho da seguinte forma: no Capítulo 2, nós apresentamos os conceitos teóricos que fundamentam a nossa pesquisa e demonstramos como eles estão relacionados a disciplina introdutória de programação. No Capítulo 3, nós descrevemos em detalhes a metodologia de Programação Orientada ao Problema (POP). No Capítulo 4, apresentamos o plano e execução dos estudos de casos realizados para avaliação de POP no contexto de sala de aula. Além disso, apresentamos uma análise quantitativa e qualitativa dos resultados obtidos. No Capítulo 5, nós descrevemos o plano e execução dos experimentos realizados. Apresentamos também uma análise estatística e qualitativa dos resultados obtidos. No capítulo 6, nós apresentamos os trabalhos relacionados, estabelecendo as diferenças e semelhanças entre eles e POP. Posteriormente, apresentamos nossas considerações finais e trabalhos futuros.

Capítulo 2

Fundamentos Teóricos e Contextualização no Ensino de Programação

Neste capítulo, nós apresentamos os conceitos que fundamentam a temática tratada nesta tese e aplicações destes conceitos na disciplina introdutória de programação.

No que diz respeito aos fundamentos teóricos, nós apresentamos, inicialmente, o conceito e classificação de problemas. Posteriormente, apresentamos o modelo de resolução de problemas de Pólya [1975], enfatizando a etapa de entendimento. Embora o trabalho de Pólya tenha sido voltado para o domínio da Matemática, seu modelo estabelece estratégias gerais que podem ser aplicadas em diferentes domínios de conhecimento e, por isso, é uma referência quando se trata de resolução de problemas [Pozo et al. 1998].

2.1 Definição de Problema

Não há um consenso na literatura sobre o conceito de problema. Porém, uma definição amplamente aceita identifica um problema como uma situação que apresenta dificuldades a um indivíduo para as quais não há solução imediata [Mazarío 1999]. Partindo desta definição, uma situação só será considerada como um problema quando o indivíduo não dispõe de conhecimento e procedimentos que o permitam solucioná-lo de forma imediata, ou seja, sem exigir reflexão ou tomada de decisão quanto às estratégias que devem ser empregadas para a sua resolução [Pozo et al. 1998].

Esta característica evidencia a diferença entre *exercício* e *problema*. Em um exercício, o indivíduo

dispõe e utiliza-se de mecanismos que o levam de forma rápida à solução, enquanto no problema isto não acontece [Pozo et al. 1998]. Esta diferenciação ocorre porque o exercício não representa algo novo e, portanto, o indivíduo pode resolvê-lo utilizando-se de estratégias e recursos que lhe são habituais.

Embora haja uma distinção na definição de problema e exercício, na prática esta definição não é tão exata. Isto acontece porque um problema para um indivíduo pode ser entendido como exercício para outro e vice-versa. Isto é fortemente evidenciado quando se trata das diferenças entre novatos e experientes. Nesta perspectiva, por exemplo, podemos dizer que resolver uma equação matemática, consertar um circuito elétrico ou criar um programa de computador pode representar problemas ou exercícios, a depender do nível de conhecimento dos estudantes que se deparam com tais situações.

De modo mais geral, podemos dizer que os professores utilizam em sala de aula execícios e problemas como recursos para favorecer a aprendizagem dos estudantes. Os exercícios são utilizados com o objetivo de fixar e praticar um determinado conceito ou procedimento. Por exemplo, solicitar que os estudantes solucionem uma equação matemática, cujas características são similares a uma equação apresentada anteriormente pelo professor. Ou, pedir que os estudantes construam um programa de computador que efetue a média aritmética de dois números, quando foi apresentado anteriormente um programa que efetuava a soma de dois números. Nestes casos, os estudantes irão exercitar a aplicação de conceitos e/ou procedimentos previamente aprendidos.

Quando os professores delegam um problema aos seus estudantes, por sua vez, apresentam alguma "novidade", ou seja, estabelecem uma maneira dos estudantes se depararem com uma situação nova a ser solucionada que inclui certo grau de dificuldade. Esta dificuldade acontece porque os exemplos e exercícios fornecidos anteriormente constituem um recurso instrumental necessário, mas não são suficientes para que eles alcancem a solução do problema.

Uma vez consolidada a definição de problema que tomaremos como referência ao longo deste trabalho, na próxima seção, apresentamos algumas classificações de problema.

2.2 Classificação de Problema

Existem várias classificações para problema. Pozo et al. [1998] afirmam que essa diversidade ocorre em virtude da área de conhecimento à qual o problema pertence, de seu conteúdo, do tipo de operação e de processos necessários para resolvê-lo, além de outras características.

Pólya [1975], por exemplo, classificou os problemas matemáticos em duas categorias: *problemas de determinação* e *problemas de demonstração*. *Problemas de determinação* possuem como partes

principais a incógnita, os dados e o condicionante. O objetivo nesse tipo de problema é determinar a incógnita. *Problemas de demonstração*, por sua vez, possuem como partes principais a hipótese e a conclusão do teorema, que deve ser demonstrada ou refutada.

Uma classificação de problema que é bastante adotada na literatura decorre dos trabalhos de Newell e Simon [Hayes and Simon 1974; Hong 1998; Jonassen 2000]. Nesta classificação, os problemas são divididos em *bem estruturados* (*well-structured*) e *mal estruturados* (*ill-structured*) ou, alternativamente, em *bem definidos* e *mal definidos*, respectivamente [VanLehn 1989; Lynch et al. 2006].

Problemas *bem estruturados* são aqueles que apresentam no enunciado todas as informações necessárias ao seu entendimento (estado inicial, objetivo e os operadores que transformam o estado do problema). Para a sua solução, requerem a aplicação de um limitado número de conceitos, regras e princípios, envolvendo procedimentos bem especificados e conhecidos. Segundo Jonassen [2000], problemas desse tipo são comumente trabalhados em escolas e universidades e, em geral, encontram-se nos livros textos no final de cada capítulo.

Problemas *mal estruturados*, por sua vez, são aqueles cuja descrição é menos clara. As informações disponíveis em seu enunciado são incompletas, ambíguas, contraditórias e/ou incorretas. Esses problemas não requerem, necessariamente, informações de uma única área de conhecimento, podem possuir múltiplas soluções todas igualmente válidas e há incerteza quantos aos conceitos, regras e princípios que devem ser empregados para a sua solução. Segundo Hong [1998], problemas desse tipo são comumente encontrados no cotidiano profissional.

Tomaremos como referência para o nosso trabalho esta última classificação de problemas: bem e mal estruturados ou, também chamados de bem ou mal definidos, respectivamente. Na próxima seção, apresentamos os procedimentos que devem ser empregados para a resolução de problemas, enfatizando os procedimentos definidos por Pólya [1975].

2.3 O Modelo de Resolução de Problemas de Pólya

Modelo de resolução de problema é um conjunto de procedimentos que orienta a resolução do problema [Pozo et al. 1998]. Existem vários modelos descritos na literatura. Deek [1997] apresenta uma síntese com os aspectos essenciais de alguns destes modelos, conforme apresentamos no Quadro 2.1.

Os modelos de Dewey e de Wallas são os mais antigos e representam abordagens opostas. O modelo de Dewey corresponde praticamente ao "método científico" aplicado à resolução de problemas. Neste modelo, o processo de resolução de problemas é dividido em quatro etapas: (1) definição do

Quadro 2.1: Síntese dos modelos de resolução de problemas.

	Entender e Definir o Problema	**Planejar a Solução**	**Projetar e Implementar a Solução**	**Verificar e Apresentar Resultados**
Dewey(1910)	Definir o Problema	Sugerir possíveis Soluções	Refletir sobre a Solução	Testar e Provar
Wallas(1926)	Preparação	Incubação, Iluminação		Verificação
Pólya(1945)	Compreender o Problema	Estabelecer um Plano	Executar o Plano	Examinar a Solução
Johnson(1955)	Preparação	Produção		Julgamento
Kingsley e Garry(1957)	Esclarecer e Representar o Problema	Procurar Pistas, Avaliar Alternativas	Aceitar uma Alternativa	Testar a Solução
Osborn e Parnes(1953 e 1967)	Descobrir Fatos, Descobrir o Problema	Descobrir a Idéia	Descobrir a Solução	Encontrar Aceitação
Simon(1960)	Inteligência	Projeto	Escolha, Implementação	
Rubinstein(1975)	Entender o quadro total	Recusar Julgamentos, Modelar	Mudar a Representação, Fazer Perguntas	Duvidar dos Resultados
Stepien, Gallagher e Workman(1993)	Analisar o Problema, Listar o que é conhecido, Desenvolver a definição do Problema	Listar o que é necessário, Listar possíveis Ações	Analisar Informações	Apresentar Resultados
Etter(1995)	Definir o Problema, Reunir Informação	Gerar e Avaliar Potenciais Soluções	Implementar e Refinar Solução	Verificar e Testar Solução
Meier e Hovde(1996)	Definir o Problema, Avaliar a Situação	Planejar Estratégias	Implementar Plano	Comunicar Resultados
Hartman(1996)	Identificar e Definir o Problema	Diagramar o Problema, Recordar Conteúdo, Explorar Estratégias Alternativas	Aplicar Conteúdo e Estratégias, Monitorar o Processo do Trabalho	Avaliar a Solução e o Processo
Deek(1999)	Formular o Problema	Planejar Solução	Projetar Solução, Traduzir Solução	Testar a Solução, Entregar a Solução

problema, que corresponde a extrair os dados e requisitos do problema; (2) sugerir possíveis soluções; (3) refletir sobre a solução; e (4) testar e provar, isto é, avaliar os resultados por meio da experimentação [Deek 1997; Deek et al. 1999].

Em contraste, o modelo de Wallas apresenta uma visão criativa, considerando o *insight*. O modelo de Wallas também é composto de 4 etapas: (1) preparação, isto é, ganhar informação sobre o problema; (2) incubação, que é pensar inconscientemente sobre o problema enquanto está engajado em outras atividades; (3) iluminação, que está diretamente relacionado ao *insight* sobre a solução do

problema; e (4) verificação, que diz respeito a avaliação da solução [Deek 1997].

Os modelos subseqüentes combinam, de alguma forma, elementos dessas duas abordagens. Em nosso trabalho, daremos ênfase ao modelo de resolução de Pólya [1975], visto que ele sintetiza as atividades dos demais modelos e estabelece procedimentos gerais que podem ser empregados em diferentes domínios de conhecimento.

O modelo de Pólya é constituído de quatro fases: (1) entender o problema; (2) estabelecer um plano; (3) executar o plano; e, (4) examinar a solução obtida. Cada uma dessas fases são descritas mais detalhadamente a seguir.

Entender o problema consiste em entender o enunciado do problema e identificar dados, incógnitas, objetivos e condicionantes, além de estabelecer uma notação mais adequada para a representação do problema. Nessa fase, é importante assumir uma postura de disposição para solucionar o problema proposto.

Estabelecer um plano consiste em relacionar dados e incógnitas, decompor o problema em partes, relembrar problemas correlatos, reformular o problema dado, utilizar estratégias (por exemplo, analogias) a fim de conceber um plano para alcançar a solução do problema.

Executar o plano consiste em transformar o plano em uma solução para o problema e checar a validade de cada passo para certificar-se de que não há erros.

Examinar a solução obtida consiste em fazer um retrospecto reconsiderando e re-examinando o resultado final e o caminho que levou até ele.

Tratando mais detalhadamente o entendimento de problemas, Pólya o divide em duas fases: *familiarização* e *aperfeiçoamento do entendimento*. A fase de familiarização consiste em conhecer o enunciado do problema e extrair dele informações que permitam uma visão geral do problema. A fase de aperfeiçoamento do entendimento é aquela na qual se deve reler o enunciado e conhecer com mais detalhes as partes principais do problema e como elas se relacionam, estabelecendo uma notação adequada para representá-las. Nós sintetizamos essas fases no Quadro 2.2.

Na fase de aperfeiçoamento do entendimento, Pólya enfatiza a importância do diálogo para auxiliar os estudantes no entendimento do problema. Esse diálogo pode ser do estudante consigo mesmo ou do professor com o estudante. Para exemplificarmos o tipo de diálogo que Pólya propõe, vamos supor que o professor apresente o seguinte problema aos seus estudantes (Quadro 2.3): *"Dado um triângulo retângulo cujas medidas de seus catetos são* $6cm$ *e* $8cm$*, determine o comprimento da hipotenusa."* A partir do enunciado do problema, o professor deve indagar os estudantes, fazendo-os identificar no enunciado as partes principais do problema e o objetivo que deve ser alcançado, conforme exemplificamos no diálogo do Quadro 2.3.

Quadro 2.2: Entendimento do problema segundo Pólya.

Familiarização
Descrição: Consiste em tomar conhecimento do enunciado do problema. **Procedimento:** Ler o enunciado e visualizar o problema como um todo.
Aperfeiçoamento do Entendimento
Descrição: Consiste de duas atividades principais - *extrair informações relevantes* e *adotar uma notação adequada*.
Extrair informações relevantes
Descrição: Consiste em reler o enunciado do problema buscando extrair as partes principais e a forma como elas estão inter-relacionadas. No caso de problemas de demonstração, a hipótese e a conclusão; e, nos problemas de determinação, as incógnitas, dados e condicionantes. **Procedimentos:** Indagar o resolvedor, partindo das questões mais gerais para as mais específicas, se necessário. Exemplos de indagações: • Qual a incógnita? Ou, fazer a mesma pergunta de diferentes maneiras: do que se precisa? o que é que se quer? o que é que se deve procurar? • Quais os dados? Qual é a condição? A condição é suficiente para determinar a incógnita? É suficiente? É contraditória?
Adotar uma notação adequada
Descrição: Adotar uma notação adequada para representar as partes principais do problema e seus inter-relacionamentos. **Procedimentos:** Representá-los à medida em que se destacam as partes relevantes do enunciado do problema.

2.4 Resolução de Problemas no Contexto de Programação

Nesta seção, nós apresentamos o nosso posicionamento sobre os fundamentos conceituais descritos anteriormente. Além disso, apresentamos a nossa perspectiva de como esses conceitos estão inseridos na disciplina introdutória de programação.

Primeiramente, concordamos com o conceito de problema apresentado na Seção 2.1. Para nós, um problema de programação deve apresentar alguma "novidade" para os estudantes. Deste modo, concordamos que os exercícios praticados na disciplina de programação constituem um recurso necessário, mas não suficiente para que os estudantes alcancem a solução de um problema proposto.

No que diz respeito a classificação de problemas, tomamos como referência aquela que categoriza

Quadro 2.3: Diálogo entre professor e estudantes.

	Problema
	Dado um triângulo retângulo cujas medidas de seus catetos são $6cm$ e $8cm$, determine o comprimento da hipotenusa.
	Diálogo
1	*Professor:* Qual é a incógnita?
2	*Estudantes:* O comprimento da diagonal do triângulo.
3	*Professor:* Quais são os dados?
4	*Estudantes:* As medidas dos dois catetos do triângulo.
5	*Professor:* Adotem uma notação adequada. Qual a letra que deve representar a incógnita?
6	*Estudantes:* x.
7	*Professor:* Quais as letras que vocês escolheriam para representar os catetos?
8	*Estudantes:* a e b.
9	*Professor:* Qual é a condicionante que relaciona a, b e x?
10	*Estudantes:* x é a diagonal do triângulo retângulo no qual a e b são os catetos.
11	*Professor:* É possível determinar a incógnita com esses dados?
12	*Estudantes:* Sim, como conhecemos os valores dos catetos, podemos aplicar o teorema de
13	pitágoras para determinar o valor da hipotenusa.

os problemas como bem e mal estruturados ou, equivalentemente, como bem e mal definidos (Seção 2.2). No entanto, salientamos que no domínio de programação os conceitos apresentados na Seção 2.2 não se aplicam de forma literal, pois problemas de programação, mesmo que bem estruturados, podem requerer informações de mais de uma área de conhecimento e possuir múltiplas soluções todas igualmente válidas. No escopo do nosso trabalho, os problemas serão classificados como bem ou mal estruturados de acordo com um aspecto, em particular, que é a *característica do enunciado do problema*, isto é, quão bem ou mal definido ele é. Para enfatizarmos essa característica, adotaremos no decorrer do trabalho as denominações – problemas bem e mal definidos.

2.4.1 Problemas Bem Definidos

Para nós, um problema de programação é bem definido quando seu enunciado apresenta as informações necessárias ao seu entendimento, deixando claro o que deve ser feito.

Na disciplina introdutória de programação é comum os professores adotarem problemas bem definidos [O'Kelly and Gibson 2006; Falkner and Palmer 2009]. São exemplos de típicos problemas

utilizados nessa disciplina: verificar se um número é primo, apresentar as raízes reais de uma equação de segundo grau, calcular o fatorial de um número e informar o tipo de um triângulo. No Quadro 2.4, nós apresentamos o enunciado deste último problema, que solicita um programa para determinar a classificação de um triângulo quanto aos lados.

Quadro 2.4: Exemplo de problema bem definido.

Problema do Triângulo

Faça um programa que leia da entrada padrão os valores correspondentes às medidas dos lados de um triângulo. Verifique se esses valores formam um triângulo e, se formarem, determine se ele é eqüilátero, isósceles ou escaleno. Na saída deve ser impressa uma mensagem informando o tipo do triângulo e, no caso, em que as medidas não formem um triângulo deve ser exibida a seguinte mensagem: "Nao eh triangulo".

Exemplos:

Entradas
lado 1 = 4
lado 2 = 9
lado 3 = 4
Saída
Nao eh triangulo
Entradas
lado 1 = 1.5
lado 2 = 2
lado 3 = 2.5
Saída
Triangulo escaleno

Problemas bem definidos são comuns em livros textos adotados nas disciplinas introdutórias de programação [Farrer 1999; Eckel 2006; de Oliveira 2008; Zelle 2004] e também em *sites* de maratonas de programação, tais como, o *site* disponível pelo SPOJ Brasil[1] e pela Universidade de Valladolid[2].

Nos enunciados dos problemas bem definidos, eventuais requisitos que não estão explícitos podem ser ignorados na solução elaborada pelos estudantes, ou aceitos pelo professor independentemente da forma que os estudantes os implementaram no programa. Por exemplo, no problema do

[1] `http://br.spoj.pl/`
[2] `http://online-judge.uva.es/p/`

triângulo (Quadro 2.4) não está especificado se o programa deve exibir a classificação do triângulo apenas uma vez, ou se deve permitir ao usuário verificar outras entradas. Também não está especificado se o programa deve tratar entradas não numéricas. Como estes requisitos não são o cerne do problema, são geralmente aceitas qualquer uma das alternativas implementadas pelos estudantes.

Para exemplificar essa situação, apresentamos dois exemplos de programas que foram escritos na linguagem Python e que solucionam o problema do triângulo apresentado no Quadro 2.4. Na primeira solução (Código Fonte 2.1), o estudante implementa o que é solicitado no problema e trata o caso do usuário digitar entradas alfabéticas, conforme pode ser observado nas linhas 3, 10 e 17 do código fonte do programa. Nesta solução, o programa não oferece opção para o usuário escolher se deseja ou não verificar a classificação de outro triângulo.

Código Fonte 2.1: Programa do Triângulo (Solução 1).

```
1  while True:
2      lado1 = raw_input("lado 1 = ")
3      if not (lado1.isalpha()):
4          if (float(lado1) > 0):
5              break
6      print "Digite novamente. Apenas numeros sao validos."
7
8  while True:
9      lado2 = raw_input("lado 2 = ")
10     if not (lado2.isalpha()):
11         if (float(lado2) > 0):
12             break
13     print "Digite novamente. Apenas numeros sao validos."
14
15 while True:
16     lado3 = raw_input("lado 3 = ")
17     if not (lado3.isalpha()):
18         if (float(lado3) > 0):
19             break
20     print "Digite novamente. Apenas numeros sao validos."
21
22 if (float(lado1) < float(lado2) + float(lado3)) and (float(lado2) < float
       (lado1) + float(lado3)) and (float(lado3) < float(lado1) + float(lado2
       )):
```

```
23      if float(lado1) == float(lado2):
24          if float(lado1) == float(lado3):
25              print "Triangulo equilatero"
26          else:
27              print "Triangulo isosceles"
28      else:
29          if float(lado2) == float(lado3):
30              print "Triangulo isosceles"
31          else:
32              print "Triangulo escaleno"
33  else:
34      print "Nao eh triangulo"
```

Na segunda solução (Código Fonte 2.2), o estudante implementa o programa utilizando funções e permite que o usuário escolha se deseja ou não verificar a classificação de outros triângulos. Essa opção é implementada nas linhas 44 a 52 do código fonte do programa. Diferentemente do código 2.1, essa segunda solução não verifica o caso do usuário digitar um valor alfabético para as entradas. É verificado apenas se as medidas dos lados do triângulo são positivas e maiores que zero, conforme as linhas 16 a 35 do código fonte.

Embora o *design* das duas soluções seja diferente e ambas incorporem uma ou outra funcionalidade adicional, as informações contidas no enunciado são suficientes para que os estudantes saibam o que deve ser feito.

Código Fonte 2.2: Programa do Triângulo (Solução 2).

```
1   def eh_triangulo(a,b,c):
2       return (a < b + c) and (b < a + c) and (c < a + b)
3
4   def classifica_triangulo(a,b,c):
5       if a == b:
6           if a == c:
7               return "Triangulo equilatero"
8           else:
9               return "Triangulo isosceles"
10      else:
11          if b == c:
12              return "Triangulo isosceles"
13          else:
```

```
14              return "Triangulo escaleno"
15
16  def entrada():
17      while True:
18          lado1 = raw_input("lado 1 = ")
19          if (float(lado1) > 0):
20              break
21          print "Digite novamente. Apenas numeros > 0 sao validos."
22
23      while True:
24          lado2 = raw_input("lado 2 = ")
25          if (float(lado2) > 0):
26              break
27          print "Digite novamente. Apenas numeros > 0 sao validos."
28
29      while True:
30          lado3 = raw_input("lado 3 = ")
31          if (float(lado3) > 0):
32              break
33          print "Digite novamente. Apenas numeros > 0 sao validos."
34
35      return float(lado1), float(lado2), float(lado3)
36
37  continua = True
38  while True:
39      lado1, lado2, lado3 = entrada()
40      if eh_triangulo(lado1, lado2, lado3):
41          print classifica_triangulo(lado1, lado2, lado3)
42      else:
43          print "Nao eh triangulo"
44      while True:
45          resposta = raw_input("Deseja ver a classificacao de outro
                triangulo [s/n]? ")
46          if (str.upper(resposta) in ['S','N']):
47              break
48          print"Digite apenas s ou n"
49      if str.upper(resposta) == 'S':
```

```
50          continua == True
51      else:
52          break
```

Entendimento de Problemas Bem Definidos

Problemas de programação bem definidos simplificam a fase de entendimento. Ao lerem o enunciado, os estudantes podem identificar facilmente as partes principais do problema e identificar também "o que" deve ser feito.

Admitindo a estratégia de diálogo de Pólya, descrita na Seção 2.3, vamos supor que o estudante ao se defrontar com o problema do triângulo, apresentado no Quadro 2.4, se disponha a entendê-lo de forma autônoma. Para isso, ele estabelece um diálogo consigo mesmo a fim de compreender o que está sendo solicitado. Nós exemplificamos esse diálogo no Quadro 2.5. Como podemos observar, baseado na leitura do enunciado, o estudante pode identificar o objetivo do programa, as entradas e saídas e suas restrições, os rótulos que devem ser exibidos para entrada de dados e a forma como as mensagens devem ser exibidas na saída (linhas 1 a 10 do diálogo).

Quadro 2.5: Diálogo do estudante consigo mesmo sobre o problema bem definido.

	Diálogo
1	*Estudante:* O que o meu programa precisa fazer?
2	*Estudante:* Humm... verificar se o triângulo é isósceles, equilátero ou escaleno.
3	*Estudante:* Será que é só isso?
4	*Estudante:* Eita, preciso dizer também se as medidas não formam um triângulo.
5	*Estudante:* Qual são as entradas?
6	*Estudante:* As medidas dos lados do triângulo.
7	*Estudante:* O que eu preciso imprimir na saída?
8	*Estudante:* Uma mensagem informando o tipo do triângulo ou se não é um triângulo.
9	*Estudante:* Como devem ser chamadas as medidas do triângulo?
10	*Estudante:* lado1, lado2, lado3.
11	*Estudante:* Como eu vou determinar se é triângulo sabendo apenas os lados?
12	*Estudante:* Eu já esqueci isso, vou pesquisar na Internet.
13	*Estudante:* Preciso de mais alguma coisa?
14	*Estudante:* Bom, eu pego o que diz a matemática e traduzo para a programação. Agora é
15	ralar na programação.

Há um aspecto que não está evidente no enunciado do problema (Quadro 2.4): como determinar o tipo do triângulo sabendo apenas as medidas dos lados? Essa informação pode ser facilmente esclarecida com base no conhecimento prévio do estudante, ou com uma simples consulta em livros ou na Web. Como apresentamos no diálogo (linhas 11 e 12 do Quadro 2.5), o estudante opta por consultar a Web. Uma vez que essa informação é obtida, caberá ao estudante focar no espaço da solução, isto é, traduzir os conceitos matemáticos para o contexto de programação e viabilizar a construção do programa. Esses aspectos são enfatizados nas linhas 13 a 15 do diálogo.

Desta forma, podemos concluir que problemas de programação bem definidos têm seu entendimento facilitado, pois o enunciado contém as informações que os estudantes necessitam para compreender o que deve ser feito.

A etapa de entendimento é, portanto, similar ao que foi proposto por Pólya (Seção 2.3): uma fase de *familiarização*, na qual os estudantes devem tomar conhecimento do enunciado do problema; e, uma fase de *aperfeiçoamento do entendimento*, na qual os estudantes vão extrair do enunciado as informações relevantes. Com pouco trabalho sendo requisitado no espaço do problema, os estudantes acabam concentrando seus esforços e atenção no espaço da solução.

2.4.2 Problemas Mal Definidos

Problemas de programação mal definidos são aqueles cujo enunciado não descreve o que deve ser feito de forma clara e completa. Isto ocorre porque o enunciado contém um ou um conjunto de defeitos[3], tais como, ambigüidades, contradições, falta de informações, informações incorretas e/ou irrelevantes.

No Quadro 2.6, apresentamos um exemplo de problema mal definido. Neste problema é solicitado um programa para simular investimento na poupança programada. Entretanto, o enunciado apresenta um conjunto de defeitos: falta de informações, ambigüidades e informações irrelevantes, também apresentados no Quadro 2.6. Em face de tais defeitos, os estudantes não são capazes de entender claramente o que deve ser feito e, por conseguinte, não conseguem construir um programa para solucionar o problema proposto.

Entendimento de Problemas Mal Definidos

Para representarmos as dificuldades dos estudantes com entendimento de problemas de programação mal definidos, vamos adotar novamente uma estratégia de diálogo na qual o estudante conversa con-

[3]Um *defeito* é uma falha na descrição do problema que pode levar a produção de uma solução incorreta.

Quadro 2.6: Exemplo de problema mal definido.

Problema da Poupança Programada

Em função da crise financeira mundial tem crescido os investimentos na poupança programada, pois é um investimento rentável e com baixíssimo risco. Um professor de administração financeira da UFCG deseja simular investimentos na poupança programada para ensinar seus alunos a driblarem a crise. A poupança rende 5% ao mês, sendo solicitado, por exemplo, tempo e capital programados pelo investidor com isenção total do imposto de renda.

Esse problema apresenta falta de informações, ambigüidades e informações irrelevantes.
Falta de informação: não são informadas quais são as entradas, saídas e como elas devem ser formatadas; quais os cálculos que devem ser efetuados e as restrições que o programa deve obedecer. Também não está claro o tipo de rendimento da poupança;
Ambigüidades: a expressão *"sendo tempo e capital programados pelo investidor com isenção total do imposto de renda"* refere-se ao fato de que a poupança programada é um investimento que tem isenção do imposto de renda? ou, que somente pode aplicar na poupança o investidor isento do imposto de renda?
Informações irrelevantes: a frase inicial que fala da crise mundial é desnecessária para o entendimento e resolução do problema. Ela foi inserida para tornar o problema mais realístico e também para motivar os estudantes a selecionarem informações úteis.

sigo mesmo na intenção de compreender o problema da poupança programada, descrito no Quadro 2.6. Nós exemplificamos este diálogo no Quadro 2.7.

Como podemos notar, o estudante tem uma série de dúvidas cujas respostas não estão presentes no enunciado do problema, conforme demonstramos nas linhas 1 a 11 do diálogo. Isto contrasta com os problemas de programação bem definidos, cujas respostas às dúvidas dos estudantes eram facilmente identificadas no próprio enunciado do problema.

Uma vez que as dúvidas não são sanadas pelo enunciado, o estudante conclui que não é possível construir o programa requerido. Além disso, ele percebe a necessidade de interagir com o professor para esclarecer as informações necessárias ao entendimento do problema (linhas 12 e 13 do diálogo). Nesse processo de esclarecimento, o estudante deve tomar nota das informações adquiridas com o professor, analisar se tais informações são suficientes e, se necessário, formular novos questionamentos e retomar as interações com o professor para adquirir as demais informações necessárias.

Portanto, o estudante terá que, naturalmente, trabalhar um pouco mais no espaço do problema (entendimento do problema e especificação dos requisitos do programa), antes de concentrar seus esforços no espaço da solução (construção do programa).

Quadro 2.7: Diálogo do estudante consigo mesmo sobre o problema mal definido.

Diálogo	
1	*Estudante:* O que o meu programa precisa fazer?
2	*Estudante:* Simular investimento na poupança programada.
3	*Estudante:* Mas, o que esse simular quer dizer? O que eu devo apresentar na saída do
4	programa?
5	*Estudante:* E essa poupança programada como ela funciona? Como é o cálculo?
6	*Estudante:* Aqui diz que a poupança rende 5% ao mês, mas não diz se é juros simples ou
7	composto.
8	*Estudante:* Parece que tempo e capital são as entradas do programa. Será que tem outras
9	entradas?
10	*Estudante:* E essa tal de isenção do imposto de renda ... será que essa poupança não
11	sofre desconto do imposto de renda? Acho que é isso ...
12	*Estudante:* Não posso fazer esse programa, eu tô cheio de dúvidas. Preciso esclarecer isso
13	com o professor urgente.

Uma vez que tratamos de problemas de programação bem e mal definidos, vamos apresentar na próxima seção uma relação entre o modelo de resolução de problemas de Pólya e o processo de desenvolvimento de software.

2.4.3 Pólya e o Processo de Desenvolvimento de Software

Como descrevemos na Seção 2.3, o modelo de resolução de problemas de Pólya é composto de quatro fases: (1) entender o problema; (2) estabelecer um plano; (3) executar o plano; e, (4) examinar a solução obtida.

Nós observamos uma relação entre estas fases do modelo de Pólya e o processo de desenvolvimento de software. De modo geral, podemos compreender o processo de desenvolvimento de software como sendo composto da definição dos requisitos, projeto de software, implementação, testes e manutenção [IEEE 2004].

Definir os requisitos que um projeto de software deve atender é a etapa inicial do processo de desenvolvimento de software e é uma fase significativa para o entendimento do problema. Nesta fase são realizadas a definição do escopo e objetivo do problema, assim como a definição dos requisitos que o software deve atender. Há, portanto, uma correspondência entre a definição dos requisitos e a primeira fase do modelo de Pólya – entender o problema.

O projeto de software diz respeito ao processo de definir a arquitetura, os componentes, interfaces e outras característica do sistema. O projeto de software tem correspondência com a segunda fase do modelo de Pólya que é estabelecer um plano para alcançar a solução.

A implementação diz respeito a construção do software por meio da utilização de uma combinação de recursos de desenvolvimento, tais como, linguagem e ambiente de programação, sistemas de controle de versões, entre outros. A implementação corresponde à fase de execução do plano, no modelo de Pólya.

Os testes permitem aferir a qualidade do software. A manutenção refere-se a modificações no software para corrigir defeitos, melhorar sua *performance* ou outros atributos que se façam necessários. Testes e manutenção têm correspondência com a última fase do modelo de Pólya, que é a de examinar a solução obtida.

No Quadro 2.8, nós apresentamos uma síntese das relações entre o modelo de Pólya e o processo de desenvolvimento de software. É importante destacarmos que essas fases são iterativas e não seqüenciais. Isto significa que as atividades em cada fase podem ser refinadas até que o objetivo seja alcançado.

Embora tenhamos feito uma descrição evidenciando todas as fases, nosso trabalho tem como ênfase a fase de entendimento do problema (segundo o modelo de Pólya) ou a definição de requisitos (se considerarmos o processo de desenvolvimento de software).

Quadro 2.8: Relação entre o modelo de Pólya e o processo de desenvolvimento de software.

Pólya	**Processo de Desenvolvimento de Software**
Entender o Problema: identificar os dados relevantes do problema.	*Definição de Requisitos*: identificar e definir os requisitos que o software deve atender.
Estabelecer um Plano: decompor o problema em partes e produzir estratégias para alcançar uma solução.	*Projeto de software*: definir a arquitetura, os componentes, interfaces e outras característica do sistema.
Executar o Plano: construir uma solução.	*Implementação*: construir o software.
Examinar a Solução: avaliar a solução obtida.	*Testes e Manutenção*: aferir a qualidade do software e proceder as correções que se fizerem necessárias.

Capítulo 3

Programação Orientada ao Problema

Programação Orientada ao Problema (POP) é uma metodologia de ensino que contempla um conjunto de atividades a serem inseridas em disciplinas introdutórias de programação a fim de auxiliar os estudantes da especificação dos requisitos ao programa.

Da perspectiva do ensino, POP combina atividades de Engenharia de Software e Programação; possibilita a realização de atividades em grupos e individuais; e pode ser utilizada desde a resolução de problemas simples até problemas mais complexos.

Da perspectiva da aprendizagem, POP proporciona aos estudantes uma visão mais realista do processo de desenvolvimento de software, permitindo-lhes conhecer e exercitar, em conjunto com a programação, outras atividades de Engenharia de Software, tais como, elicitação e especificação de requisitos e testes. Além da prática dessas atividades, os estudantes desenvolvem habilidades de diálogo, negociação e escrita.

Neste capítulo, nós descrevemos os princípios que fundamentam POP, seus participantes e papéis, e atividades desenvolvidas pelos estudantes para a resolução de problemas. Para orientarmos os docentes na aplicação de POP, nós disponibilizamos no Apêndice A os *guidelines* e material de suporte. Esses materiais também encontram-se disponíveis no *site* `http://sites.google.com/site/joinpop/`.

3.1 Princípios Pedagógicos e Técnicos

Os princípios pedagógicos e técnicos que fundamentam POP são:

- Assumir Programação como parte da Engenharia de Software;

- Adotar um ciclo[1] de resolução de problemas que vai da especificação dos requisitos ao programa;
- Produzir, como resultado do ciclo de resolução de problemas, especificação, programa e testes, ao invés de um único artefato – o programa;
- Adotar problemas mal definidos como parte integrante dos problemas propostos em disciplinas de programação;
- Adotar testes desde o início da disciplina, tanto para descobrir e/ou confirmar requisitos, quanto para aferir a qualidade do programa.

Como enfatizamos no Capítulo 1, programação resulta em um produto de software e, como tal, produz artefatos que devem atender necessidades de um cliente e cuja qualidade precisa ser aferida. Não há, portanto, como dissociá-la da própria Engenharia de Software.

Por concebermos programação como descrito acima, acreditamos que a forma de resolver problemas na disciplina introdutória de programação deve ser a mais realista possível, permitindo que os estudantes tenham uma compreensão de como os problemas são resolvidos no contexto de desenvolvimento de software. Neste contexto, os problemas são mal definidos e, para solucioná-los, os desenvolvedores precisam ir da especificação dos requisitos ao programa. Esta experiência de *desenvolvedor* deve ser vivenciada pelos estudantes de programação.

Os princípios de POP são apoiados pelas diretrizes curriculares para a área de Computação e Informática, as quais recomendam que o ensino de programação enfatize outras habilidades além do raciocínio lógico e da construção de programas. No Brasil, as diretrizes curriculares sugeridas pela Secretária de Educação Superior [MEC/SESU 1999] definem programação de computadores como sendo *"uma atividade voltada à resolução de problemas. Nesse sentido ela está relacionada com uma variada gama de outras atividades como especificação, projeto, validação, modelagem e estruturação de programas e dados, utilizando-se das linguagens de programação propriamente ditas, como ferramentas"* [MEC/SESU 1999, p. 5].

A *Joint Task Force on Computing Curricula* [IEEE/ACM 2001], ao tratar especificamente sobre cursos introdutórios de programação, afirma que *"estes cursos freqüentemente simplificam o processo de programação para torná-los mais acessíveis aos estudantes iniciantes, dando pouco peso*

[1] Ciclo é definido como um conjunto de fatos ou ações que se sucedem no tempo, marcando uma diferença entre o estágio inicial e o estágio conclusivo. Esta definição foi extraída do dicionário Houaiss da Língua Portuguesa, disponível em `http://houaiss.uol.com.br/` e acessada em outubro de 2010.

às atividades de análise, projeto e testes". Acrescenta ainda, que esta situação causa uma impressão superficial nos estudantes sobre o domínio das habilidades de programação e limita a capacidade deles de adaptar-se a diferentes tipos de problemas e contextos de resolução de problemas no futuro [IEEE/ACM 2001, p. 23].

3.2 Visão Geral do Ciclo de Resolução de Problemas de POP

POP é um conjunto de atividades voltadas para a disciplina introdutória de programação. Em POP, há ênfase na resolução de problemas mal definidos, concebidos e propostos pelo professor.

A resolução de problemas ocorre sob a forma de um ciclo que tem início com a entrega de um problema mal definido aos estudantes. Partindo deste problema, os estudantes devem consolidar uma especificação dos requisitos que o programa deve atender e também concretizar uma solução, com a entrega de um programa e casos de testes automáticos.

Assim, para esclarecer o problema mal definido proposto, os estudantes trabalham em pequenos grupos e cada grupo é acompanhado por um assistente. Os assistentes atuam junto aos grupos como uma espécie de cliente. Estes assistentes sabem exatamente o que querem, pois seguem uma especificação de requisitos documentada em um artefato de referência, previamente preparado pelo professor da disciplina.

A primeira interação dos grupos com seus respectivos assistentes é feita de forma presencial. Neste momento, os estudantes conduzem um diálogo com o assistente para esclarecer requisitos que o programa deve atender para resolver o problema proposto. Como resultado desta primeira interação, têm-se um documento de requisitos e um conjunto de casos de testes de entrada/saída, que são compartilhados, estritamente, entre os estudantes do grupo, assistente e professor.

Após esta primeira interação, utilizando como referência o documento de requisitos e os casos de testes de entrada/saída, os estudantes devem trabalhar individualmente na construção de um protótipo[2] do programa e de casos de testes automáticos. Caso haja dúvidas remanescentes da primeira interação, os estudantes devem comunicar-se com seu respectivo assistente. Esta comunicação ocorre por meio de uma lista de discussão, restrita aos membros do grupo, seu assistente e professor. Após o esclarecimento das eventuais dúvidas, os requisitos discutidos devem ser atualizados no documento de especificação criado pelo grupo.

[2]Versão preliminar do programa, construída para ser testada e aperfeiçoada.

Uma segunda interação presencial é realizada entre o grupo e o seu respectivo assistente. Nesta interação, os estudantes devem apresentar seus protótipos do programa e dos testes automáticos. Nesta oportunidade, os estudantes devem conduzir um diálogo com o assistente até se certificarem que todos os requisitos do programa foram esclarecidos. O grupo deve, também, atualizar o registro dos requisitos e dos casos de testes de entrada/saída, gerando uma versão final da especificação dos requisitos.

Neste ponto, os estudantes têm todos os requisitos esclarecidos junto ao assistente. Assim, após este segundo encontro, os estudantes devem voltar a trabalhar individualmente a fim de evoluir os seus programas e casos de testes automáticos para atender aos requisitos. Isto é feito até uma data pré-definida, quando os artefatos produzidos devem ser encaminhados ao professor para avaliação.

Em síntese, quando os estudantes entregam os programas e testes automáticos ao professor, isso consolida o encerramento do ciclo de resolução de problemas. Este ciclo dura uma semana, mas pode ser ajustado pelo professor quando perceber que isto se faz necessário.

Munido dos artefatos entregues pelos estudantes (documento dos requisitos, testes de entrada/saída, programa e testes automáticos), o professor deve proceder a avaliação dos estudantes.

Ao cumprirem esse ciclo de resolução de problemas, os estudantes terão trabalhado no espaço do problema e da solução, experimentando as "idas e vindas", naturais, no processo de desenvolvimento de software. De forma simplificada, terão praticado atividades da Engenharia de Software, tais como, elicitação e especificação de requisitos, programação e testes.

Na próxima seção, nós apresentaremos em detalhes os participantes de POP e os papéis que eles assumem no ciclo de resolução de problemas.

3.3 Participantes e Papéis

Participam de POP: professor, assistentes e estudantes. No ciclo de resolução de problemas de POP, o professor e os assistentes desempenham o papel de *clientes-tutores* e os estudantes assumem o papel de *desenvolvedores*.

Os *clientes-tutores* são assim denominados porque desempenham atividades de *cliente* e *tutor*. Enquanto *clientes*, eles devem responder aos questionamentos dos estudantes sobre os requisitos do programa. Como *tutores*, eles devem mediar as interações do grupo, administrar o tempo em sala de aula e orientar os estudantes na realização das atividades.

Os *desenvolvedores* têm a responsabilidade de cumprir o ciclo de resolução de problemas e produzir os artefatos esperados – documento de especificação, casos de testes de entrada/saída, programa

e testes automáticos. Para cumprirem as suas atividades, os desenvolvedores trabalham ora em *grupo*, ora *individualmente*. Em *grupo*, elicitam e especificam os requisitos do programa, produzindo o documento de especificação e os casos de testes de entrada/saída. *Individualmente*, os desenvolvedores trabalham na solução do problema, isto é, na criação de um programa e casos de testes automáticos que atendam aos requisitos dos clientes-tutores.

Nos Apêndices B e C, nós apresentamos orientações sobre como os clientes-tutores e desenvolvedores devem proceder no ciclo de resolução de problemas de POP. Na próxima seção, apresentaremos com detalhes o conjunto de atividades executadas por clientes-tutores e desenvolvedores, assim como os artefatos produzidos durante o ciclo de resolução de problemas.

3.4 Detalhamento do Ciclo de Resolução de Problemas de POP

O ciclo de resolução de problemas de POP compreende quatro atividades: *elaborar especificação inicial*; *iniciar implementação*; *concluir especificação*; e, *concluir implementação*. Estas atividades são descritas a seguir.

3.4.1 Elaborar Especificação Inicial

A elaboração da especificação inicial ocorre na primeira reunião entre o grupo de desenvolvedores e seu respectivo cliente-tutor. Nesta reunião, o cliente-tutor apresenta ao grupo de desenvolvedores o problema a ser resolvido. O enunciado deste problema é mal definido e é apresentado ao grupo na forma textual.

Os requisitos que devem ser atendidos pelos desenvolvedores para solucionar o problema proposto são de conhecimento dos clientes-tutores, e apenas deles. Esses requisitos foram previamente registrados em um *artefato de referência* que é utilizado pelos clientes-tutores para responder aos questionamentos dos desenvolvedores. Desta forma, para todos os grupos de desenvolvedores, independente de quem seja o cliente-tutor, os requisitos do programa são os mesmos.

O grupo deve escolher um dos desenvolvedores para fazer o registro escrito dos requisitos durante a reunião com o cliente-tutor. Para tanto, o desenvolvedor escolhido deve criar um arquivo, por meio de um editor de texto colaborativo, e compartilhá-lo com os membros do grupo e seu cliente-tutor. Neste arquivo será registrada a especificação dos requisitos. O desenvolvedor também é responsável por criar uma lista de discussão restrita ao seu grupo e cliente-tutor.

Após esses procedimentos iniciais, o grupo deve conduzir um diálogo com o cliente-tutor para elicitar os requisitos. Neste processo de interação, os desenvolvedores utilizam uma estratégia na qual o diálogo vai se tornando progressivamente mais estruturado, conforme exemplificamos no Quadro 3.1. Os desenvolvedores iniciam interagindo livremente, na forma de perguntas e respostas (a) e, posteriormente, na forma de casos de testes de entrada e saída (b). Esta estratégia é também incentivada pelo próprios clientes-tutores.

Quadro 3.1: Diálogo progressivamente estruturado.

Diálogo	
1	*Desenvolvedor:* Como é o cálculo do rendimento?
2	*Cliente-tutor:* Bom, o cálculo é feito com base no capital e no tempo de investimento.
3	*Desenvolvedor:* Há uma fórmula para isso?
4	*Cliente-tutor:* Sim. O rendimento é igual ao capital que multiplica o cálculo dos juros. E o
5	cálculo dos juros é feito somando um à taxa de juros e elevando esse resultado a quantidade
6	de meses de investimento.
7	*Desenvolvedor:* R = C vezes o quê?...
8	*Cliente-tutor:* R = C $*$ $(1 + taxadejuros)^{tempo}$
	(a)
9	*Desenvolvedor:* Então:
10	`Capital?` 30
11	`Tempo?` 12
12	`Rendimento:R$` 53.88
13	*Cliente-tutor:* Correto!
	(b)

Ao final da reunião, os desenvolvedores devem ter produzido um *documento de especificação inicial* e *casos de testes de entrada/saída*. O documento de especificação deve conter a descrição textual dos requisitos que o programa deve atender para solucionar o problema proposto. Nós não definimos em POP um padrão de especificação, mas oferecemos um exemplo de documento de especificação que mostra aos desenvolvedores os elementos que compõem uma especificação e como eles devem estar organizados e formatados. Esse exemplo está apresentado no Apêndice C.5.

Os casos de testes de entrada/saída também descrevem os requisitos que o programa deve atender. Eles devem estar registrados no documento de especificação e devem ser expressos em uma das duas formas: na forma de uma tabela com entradas e saídas, similar a tabela representada em (a), na Figura

3.1; ou de exemplos de entrada e saída, representados em (b), na Figura 3.1. Em ambos os casos, é importante notarmos que além dos valores das variáveis, as restrições de formatação das entradas e saídas são também apresentadas.

Entradas	Saídas (a)
Capital? 30 Tempo? 12	Rendimento: R$ 53.88

(b)

```
Capital? 30
Tempo? 12
Rendimento: R$ 53.88
```

Figura 3.1: Exemplos de testes de entrada e saída.

Em síntese, nesta atividade os desenvolvedores trabalham em grupo e interagem com o cliente-tutor para especificar os requisitos. Ao final da atividade, o grupo deve produzir uma versão inicial do documento de especificação, contendo a descrição dos requisitos e os casos de testes de entrada/saída, conforme apresentamos na Figura 3.2. Os artefatos produzidos irão subsidiar os desenvolvedores na atividade de iniciar a implementação.

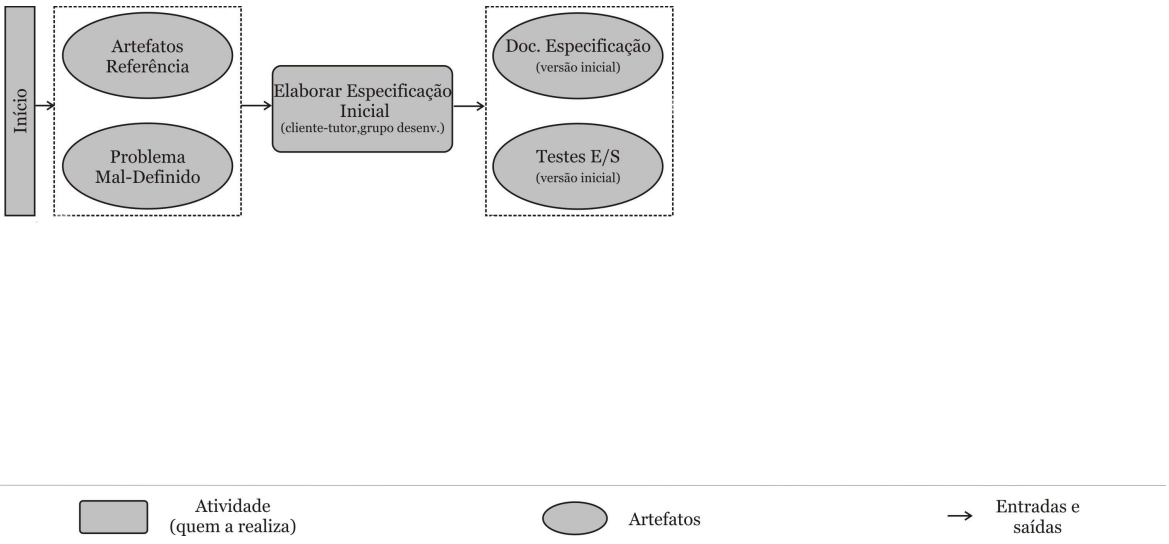

Figura 3.2: Atividade: Elaborar especificação inicial.

3.4.2 Iniciar a Implementação

Após a realização da atividade anterior, os desenvolvedores devem dar início à implementação. Para isto, eles devem trabalhar individualmente na construção de um protótipo do programa e de casos de testes automáticos que atendam aos requisitos especificados pelo grupo com o cliente-tutor.

O protótipo do programa deve ser escrito na linguagem de programação adotada na disciplina. Os testes automáticos devem implementar os casos de testes de entrada/saída, descritos no documento de

especificação, e também outros testes definidos pelo desenvolvedor no momento da implementação do programa. Com estes testes, os desenvolvedores podem detectar erros e verificar se o programa atende ao que foi especificado.

Durante a criação do protótipo do programa, é natural que surjam novas dúvidas sobre os requisitos. Para esclarecê-las, os desenvolvedores devem retomar o diálogo, por meio da lista de discussão (criada na atividade anterior, Seção 3.4.1), com seu respectivo grupo e cliente-tutor. Assim como na primeira atividade, o cliente-tutor deve basear-se no artefato de referência para responder aos questionamentos dos desenvolvedores. Os requisitos discutidos na lista devem ser atualizados no documento de especificação. Neste momento, a atualização do documento é de responsabilidade de todos os desenvolvedores.

Nesta atividade, o diálogo deve tornar-se um pouco mais estruturado, pois os desenvolvedores podem conversar sobre os requisitos utilizando também testes automáticos. Desta forma, os desenvolvedores podem criar testes automáticos com base nos testes de entrada/saída previamente documentados e também criar novos testes, conforme exemplificamos no Quadro 3.2 (c).

Portanto, nesta atividade, os desenvolvedores trabalham individualmente na criação de seus protótipos do programa e dos casos de testes automáticos, conforme ilustramos na Figura 3.3. Os desenvolvedores também atualizam o documento de especificação inicial, assim como os casos de testes de entrada/saída. Estes artefatos, em conjunto, serão utilizados como insumos na próxima atividade.

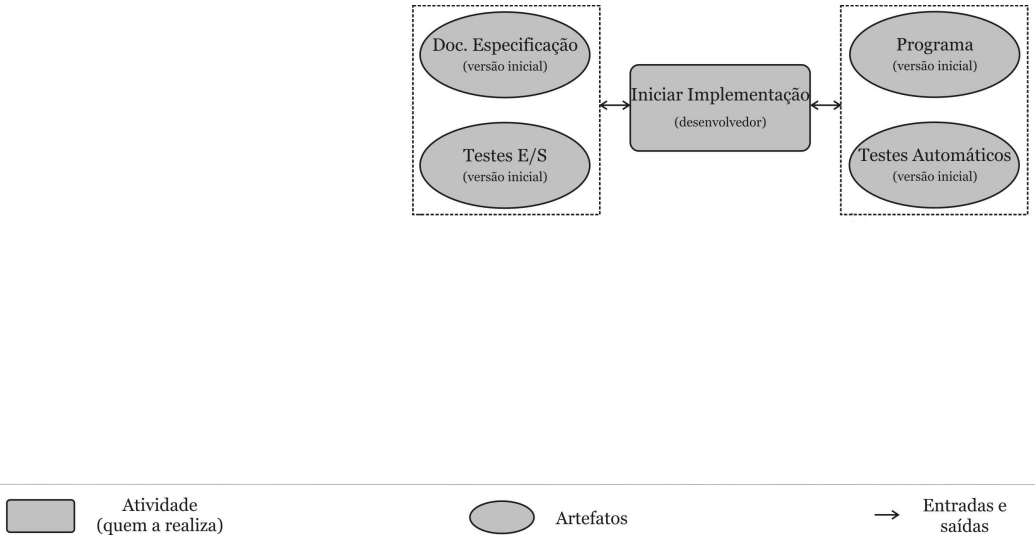

Figura 3.3: Atividade: Iniciar implementação.

Quadro 3.2: Diálogo progressivamente estruturado (cont.).

	Diálogo
1	*Desenvolvedor:* Como é o cálculo do rendimento?
2	*Cliente-tutor:* Bom, o cálculo é feito com base no capital e no tempo de investimento.
3	*Desenvolvedor:* Há uma fórmula para isso?
4	*Cliente-tutor:* Sim. O rendimento é igual ao capital que multiplica o cálculo dos juros. E o
5	cálculo dos juros é feito somando um à taxa de juros e elevando esse resultado a quantidade
6	de meses de investimento.
7	*Desenvolvedor:* R = C vezes o quê?...
8	*Cliente-tutor:* R = C $*$ $(1 + taxadejuros)^{tempo}$
	(a)
9	*Desenvolvedor:* Então:
10	`Capital?` 30
11	`Tempo?` 12
12	`Rendimento:R$` 53.88
13	*Cliente-tutor:* Correto!
	(b)
14	*Desenvolvedor:* `assert calcular_rendimento(30,12) == 53.88`
15	`assert calcular_rendimento(86.50,10) == 140.90`
16	*Cliente-tutor:* Ok!
	(c)

3.4.3 Concluir Especificação

Uma segunda reunião presencial é estabelecida entre cliente-tutor e grupo de desenvolvedores. Cada desenvolvedor deve estar munido de seu protótipo do programa e dos testes automáticos a fim de apresentá-los ao cliente-tutor.

O foco desta apresentação não é a análise do código fonte dos programas, mas sim retomar a interação sobre os requisitos. Os desenvolvedores devem testar seus protótipos do programa na presença do cliente-tutor e permitir que o cliente-tutor também os teste. Para testar os protótipos, os desenvolvedores devem usar os testes automáticos produzidos e também executar manualmente os testes que o cliente-tutor solicitar. Esta apresentação contribui para a identificação de novos requisitos que ainda não tenham sido especificados e também para corrigir e/ou detalhar requisitos já documentados.

Novamente, o grupo deve escolher um de seus desenvolvedores para atualizar os requisitos e ca-

sos de testes de entrada/saída. Nesta interação, os desenvolvedores devem evoluir a especificação inicial para uma especificação final contemplando todos os requisitos desejados pelo cliente-tutor (Figura 3.4). Como dissemos anteriormente, os clientes-tutores sempre respondem aos desenvolvedores baseando-se nos requisitos descritos no artefato de referência.

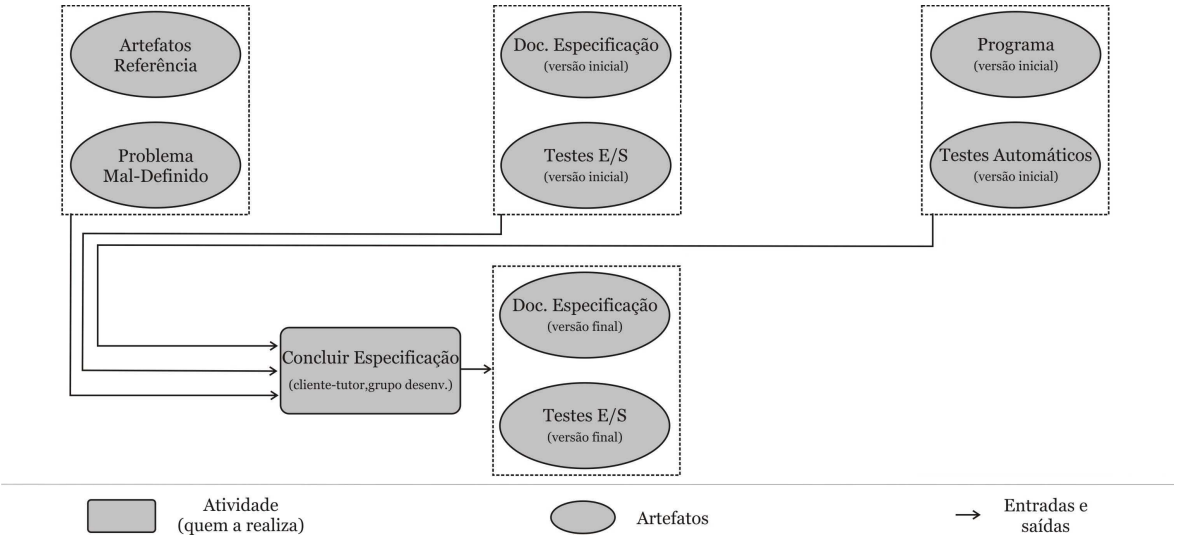

Figura 3.4: Atividade: Concluir especificação.

3.4.4 Concluir Implementação

Após a finalização da atividade anterior, os desenvolvedores têm conhecimento de todos os requisitos requeridos por seu cliente-tutor. Portanto, tomando como referência o documento de especificação final e os casos de testes de entrada/saída, os desenvolvedores devem, individualmente, evoluir seus protótipos do programa e casos de testes automáticos com o objetivo de finalizar a implementação, atendendo aos requisitos demandados (Figura 3.5). Nesta atividade, os desenvolvedores podem comunicar-se entre si, via lista de discussão ou presencialmente, para tirar dúvidas de implementação e prover ajuda mútua, como por exemplo, troca de material didático.

Esta atividade é desenvolvida até uma data pré-definida para a conclusão e entrega dos programas e casos de testes automáticos, finalizando, assim, o ciclo de resolução de problemas de POP. Este ciclo é apresentado de forma completa na Figura 3.6. Como podemos observar, os desenvolvedores iniciam com um problema mal definido e realizam um conjunto de atividades – elaborar especificação inicial, iniciar implementação, concluir especificação e concluir implementação. Com a realização destas atividades, os desenvolvedores constroem e evoluem a especificação dos requisitos, programa e testes.

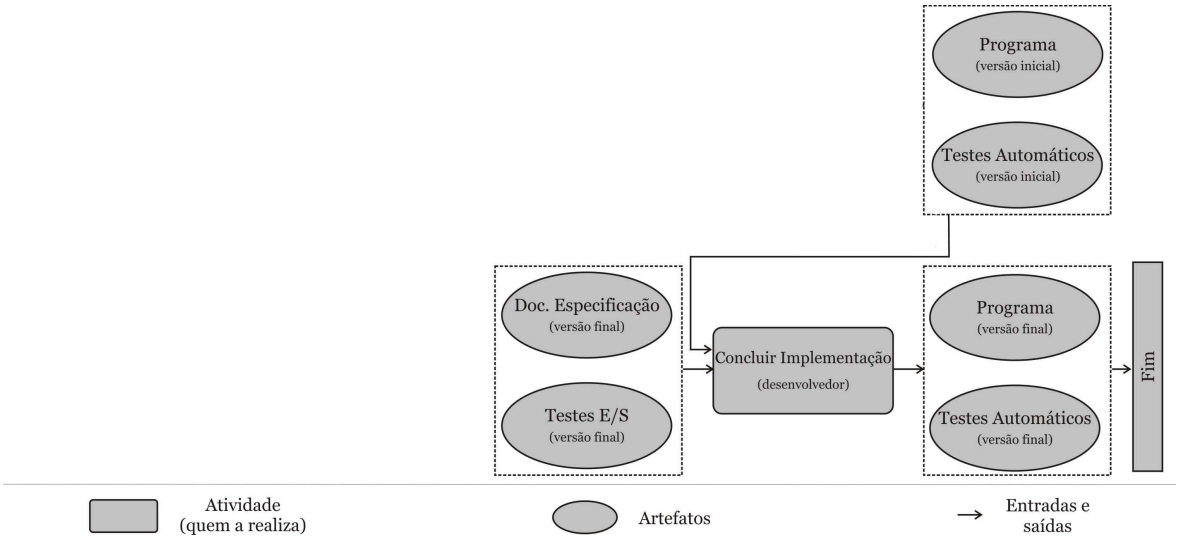

Figura 3.5: Atividade: Concluir implementação.

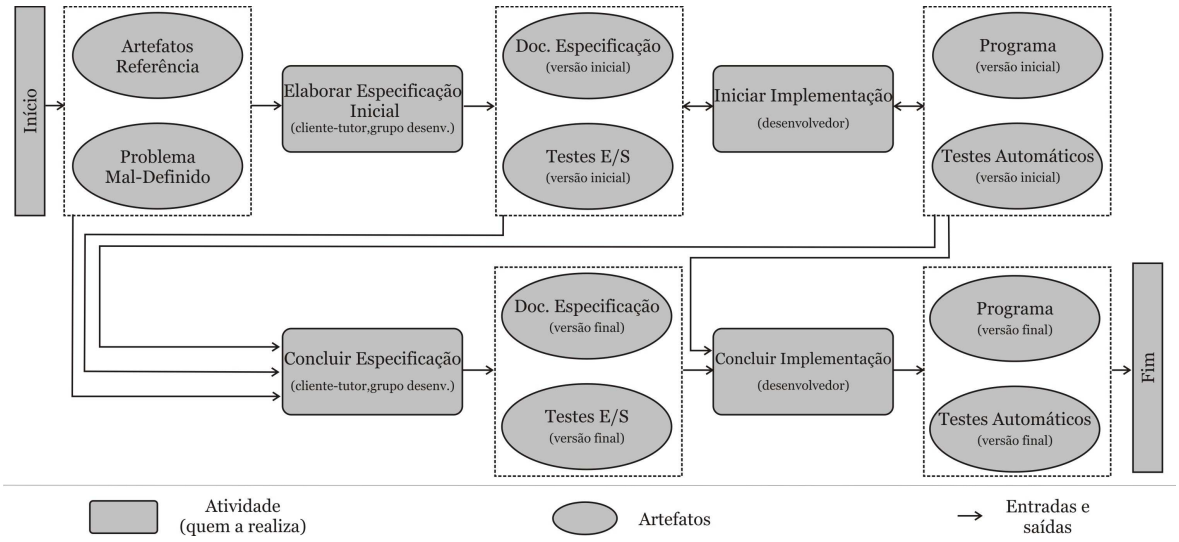

Figura 3.6: Ciclo de resolução de problemas de POP.

3.5 Considerações sobre o Ciclo de Resolução de Problemas de POP

O ciclo de resolução de problemas de POP possibilita aos estudantes experimentarem o papel de desenvolvedores, permitindo-lhes: (i) iniciar com uma especificação mal definida; (ii) interagir com um cliente; (iii) construir outros artefatos além do programa; (iv) prover melhorias nestes artefatos; e, (v) trabalhar tanto em grupo quanto individualmente.

O trabalho em grupo favorece o diálogo, a negociação e identificação dos requisitos. O trabalho

individual, por sua vez, permite aos estudantes desenvolverem o raciocínio lógico, aperfeiçoarem a aprendizagem da sintaxe da linguagem de programação e as habilidades para construção de programas e testes.

Ao perceberem que os artefatos produzidos em uma atividade podem sofrer modificações e ajustes em uma outra atividade, os estudantes têm a possibilidade de observar que a construção de um produto de software não é linear, havendo, portanto, uma característica iterativa.

Ao observarmos o ciclo de POP sob a perspectiva da resolução de problemas, veremos que os estudantes atuam tanto no espaço do problema quanto no espaço da solução. Observando este ciclo do ponto de vista da Engenharia de Software, veremos também que os estudantes, embora de forma simplificada, praticam atividades de elicitação e especificação de requisitos, testes e programação.

No que diz respeito a elicitação de requisitos, os testes são utilizados com uma estratégia de diálogo. Esta estratégia faz com que os estudantes percebam que testes não são recursos utilizados no fim do processo, apenas com o propósito de avaliar o programa, mas que podem ser utilizados desde o início para ajudar o desenvolvedor a compreender e extrair requisitos junto a um cliente.

Neste capítulo, nós enfatizamos o ciclo de resolução de problemas de POP, porque ele evidencia as atividades que os estudantes devem desempenhar para resolver problemas mal definidos, assim como os artefatos que eles devem produzir para atender as solicitações dos clientes-tutores. Entretanto, destacamos que POP compreende outras etapas além da execução do ciclo de resolução de problemas. Essas etapas serão apresentadas na próxima seção.

3.6 Apresentação das demais Etapas de POP

POP é composta pelas seguinte etapas: *planejamento*, *preparação*, *execução* e *avaliação*.

Planejamento. Etapa na qual o professor deve planejar a aplicação de POP na disciplina introdutória de programação, isto é, deve inseri-la no plano de aula da disciplina. Esta etapa é composta das seguintes atividades: (i) definir cronograma da disciplina e (ii) definir critérios de avaliação.

Preparação. Etapa que antecede cada execução de POP, isto é, etapa que antecede ao ciclo de resolução de problemas de POP. Esta etapa é composta das seguintes atividades: (i) elaborar o problema mal definido que será proposto aos estudantes; (ii) produzir os artefatos de referência; (iii) definir os deadlines; (iv) definir os recursos para interação do grupo e documentação dos requisitos; (v) dividir a turma em grupo; (vi) selecionar e orientar os clientes-tutores; e, (vii) preparar os estudantes para a execução de POP.

Execução. Etapa na qual há a execução do ciclo de resolução de problemas de POP na disciplina introdutória de programação. Esta etapa foi descrita neste capítulo.

Avaliação. Nesta etapa, o professor tem como atividade avaliar os estudantes e os procedimentos adotados na etapa de execução.

A exceção da execução, nós apresentamos no Apêndice A orientações sobre como o professor deve proceder nas demais etapas. Ainda para dar suporte ao professor na aplicação de POP, nós apresentamos no Apêndice A.5 uma estratégia para inserir testes automáticos na disciplina introdutória de programação.

No Apêndice B, nós apresentamos orientações aos assistentes que farão o papel de clientes-tutores e no Apêndice C, apresentamos aos estudantes orientações sobre como proceder enquanto desenvolvedores.

Capítulo 4

Estudos sobre Aplicação de POP em Sala de Aula

Neste capítulo, nós apresentamos dois estudos de caso sobre a aplicação de POP em sala de aula. Estes estudos tiveram por objetivo avaliar a execução de POP no contexto da disciplina introdutória de programação, com o propósito de caracterizar os efeitos da metodologia nos estudantes no que diz respeito a tratar com entendimento de problemas mal-definidos e especificação de requisitos. Outro propósito foi o de capturar informações úteis, junto aos participantes do estudo, que permitissem melhorar os procedimentos adotados em POP.

Nós realizamos os estudos nos dois semestres acadêmicos de 2009, com alunos iniciantes de programação do Curso de Ciência da Computação da Universidade Federal de Campina Grande (UFCG). A execução dos estudos de casos ocorreu após os estudantes terem aprendido o conteúdo referente a manipulação de *strings*. Estes estudos foram antecedidos por um estudo de caso piloto que nós realizamos no segundo semestre acadêmico de 2008 e cujos resultados nós reportamos em [Mendonça et al. 2009b].

Para o planejamento dos estudos de caso e descrição dos resultados, nós seguimos as orientações propostas em [Yin 1984], [Kitchenham et al. 1995] e [Runeson and Höst 2009].

4.1 Estudo de Caso 1

4.1.1 Planejamento

O planejamento de nosso estudo de caso inclui a descrição das questões de pesquisa, dos sujeitos que participaram do estudo, do objeto utilizado, das unidades de análise, dos artefatos avaliados e dos critérios de avaliação, assim como dos procedimentos empregados para coleta de dados.

Questões de Pesquisa

Para alcançar os objetivos propostos, nosso estudo de caso deve responder às seguintes questões de pesquisa:

1. Quando adotamos POP, os estudantes documentam requisitos?
2. Quais tipos de defeitos são mais comuns nos documentos de especificação produzidos pelos estudantes que adotam POP?
3. Ao concluir o ciclo de POP, os programas produzidos pelos estudantes atendem aos requisitos requeridos pelo cliente-tutor?
4. Quais as dificuldades apresentadas pelos estudantes ao praticarem as atividades de POP?
5. Quais as dificuldades apresentadas pelos clientes-tutores para a condução de POP em sala de aula?

Para responder às três primeiras questões de pesquisa, nós adotamos os critérios descritos na Seção *Artefatos Avaliados e Critérios de Avaliação*, descrita na página 41. Para responder às duas últimas questões, além da observação, nós adotamos dois questionários: um, aplicado aos estudantes; e outro, aos clientes-tutores. Nós apresentamos os questionários no Apêndice D.

Sujeitos

Nós aplicamos POP com os estudantes matriculados na disciplina de Laboratório de Programação I[1], período 2009.1, do curso de Ciência da Computação da UFCG. Esta disciplina possuía três turmas, sendo a terceira turma composta apenas por estudantes repetentes.

Em nosso estudo, nós consideramos apenas os estudantes iniciantes, que compunham as Turmas 1 e 2. Considerando que a divisão dos estudantes por turma ocorre de forma aleatória e que o mesmo

[1]Esta disciplina é oferecida no primeiro período do curso de Ciência da Computação.

tratamento (POP) foi aplicado, igualmente, em ambas as turmas, nós não faremos distinções entre as Turma 1 e 2. Portanto, nossos sujeitos foram 70 estudantes iniciantes de programação.

Objeto

O objeto usado no estudo de caso foi uma lista de problemas, contendo três problemas mal definidos, os quais deveriam ser especificados e solucionados pelos estudantes seguindo a metodologia POP. Nós utilizamos três problemas a fim de permitir que aprendizagem dos estudantes fosse reforçada pela repetição das atividades do ciclo de resolução de problemas.

Nos Quadros 4.1, 4.2 e 4.3, nós apresentamos os enunciados desses problemas. Para solucioná-los, os programas dos estudantes teriam que atender a 17, 12 e 26 requisitos, respectivamente. Nós descrevemos esses requisitos no Apêndice E.

Quadro 4.1: Problema 1 – Poupança programada.

Enunciado
Em função da crise financeira mundial tem crescido os investimentos na poupança programada, pois é um investimento rentável e com baixíssimo risco. Um professor de administração financeira da UFCG deseja simular investimentos na poupança programada para ensinar seus alunos a driblarem a crise. Faça um programa que atenda a solicitação desse professor, considerando que a poupança rende 5% ao mês, sendo tempo e capital programados pelo investidor com isenção total do imposto de renda.
Defeitos
Esse problema apresenta falta de informações, ambigüidades e informações irrelevantes. **Falta de Informações:** não são informadas quais são as entradas, saídas e como elas devem ser formatadas; quais os cálculos que devem ser efetuados e as restrições que o programa deve obedecer. Também não está claro o tipo de rendimento da poupança; **Informações Ambíguas:** a expressão *"sendo tempo e capital programados pelo investidor com isenção total do imposto de renda"* refere-se ao fato de que a poupança programada é um investimento que tem isenção do imposto de renda? ou, que somente pode aplicar na poupança o investidor isento do imposto de renda? **Informações irrelevantes:** a frase inicial que fala da crise mundial é desnecessária para o entendimento e resolução do problema. Ela foi inserida apenas para deixar o problema mais interessante.

Unidade de Análise

Conforme definido em POP, a especificação dos requisitos deve ser realizada em grupo e a implementação, individualmente. Assim, nosso estudo de caso possui duas unidades de análise: o *grupo*; e o

Quadro 4.2: Problema 2 – Freqüência das palavras.

Enunciado
Um professor de português precisa de um programa que dado um parágrafo, apresente com que freqüência as palavras se repetem.
Defeitos
Falta de Informações: não são informadas, por exemplo, as formatações de entrada e saída; se há limites para o tamanho do parágrafo; se caracteres especiais e números devem ou não ser aceitos; ou, se deve haver distinção de caracteres maiúsculos e minúsculos.

Quadro 4.3: Problema 3 – Jogo da forca.

Enunciado
Este mesmo professor de português gostaria de desenvolver uma atividade lúdica e educativa com seus alunos e por isso deseja que você faça um programa para adivinhação de palavras, similar ao jogo da forca.
Defeitos
Falta de Informações: não são informadas, por exemplo, as formatações de entrada e saída; se a forca deve ser jogada por dois jogadores humanos or entre um humano e a máquina; se deveria existir uma base de dados de palavras e se a escolha delas deveria ser feita de forma randômica; se haveria necessidade de armazenar login e pontuação dos jogadores.

estudante, individualmente.

Artefatos Avaliados e Critérios de Avaliação

Nós avaliamos a versão final de dois artefatos: *documentos de especificação* e *programas*. Para cada artefato, os procedimentos para avaliação, assim como os critérios adotados são definidos como segue.

Documento de Especificação. Para avaliação dos documentos de especificação nós adotamos duas variáveis: *requisitos documentados* (RD) e *requisitos documentados corretamente* (RC). Para analisá-las, nós realizamos inspeções nos documentos de especificação dos estudantes utilizando a técnica de leitura baseada em defeitos (*Defect-Based Reading* – DBR) [Porter et al. 1995]. Nós escolhemos esta técnica por considerá-la simples de ser aplicada e adequada ao nosso estudo, uma vez que era nosso propósito detectar os defeitos comuns nas especificações dos estudantes.

Na versão original de DBR, a detecção de defeitos era realizada em documentos de requisitos descritos em SCR [Heninger 1980], uma notação formal para sistemas de controle de processo dirigido a eventos.

No nosso caso, nós fizemos uma adaptação desta técnica. Nós utilizamos documentos de especificação escritos em linguagem natural e a técnica foi adotada para focar sobre uma classe de defeitos

em um conjunto pré-definido de requisitos. Este conjunto refere-se aos requisitos que os programas deveriam atender para solucionar os problemas propostos, conforme descrevemos no Apêndice E.

Para classificar os tipos de defeitos presentes nos documentos de especificação dos estudantes, nós adotamos uma versão simplificada da taxonomia de defeitos definida por Shull et al. [2000] que é apresentada no Quadro 4.4. Para aumentar a fidelidade da avaliação, os documentos foram inspecionados por duas pessoas, trabalhando juntas.

Quadro 4.4: Taxonomia de defeitos.

Tipo	Descrição
Requisito Omitido	Algum requisito que foi omitido no documento de especificação.
Requisitos Ambíguo	Algum requisito que foi descrito no documento de especificação de forma ambígua, causando duplo sentido.
Requisito Contraditório	Dois requisitos que se contradizem mutuamente ou expressão ações que não podem ser ambas corretas.
Requisito Incorreto	Algum requisito que consta no documento de especificação, mas não descreve corretamente o problema.
Requisito Irrelevante	Algum requisito que consta no documento de especificação, mas não é importante, nem necessário ao entendimento do problema.

Nós definimos as variáveis *requisitos documentados* (RD) e *requisitos documentados corretamente* (RC) como segue:

- *Requisitos Documentados (RD):* corresponde a porcentagem de requisitos documentados pelo grupo de estudantes.

$$\text{RD} = \frac{\text{QRD}}{\text{QRR}} \times 100\%$$

  Na qual:

  - QRD : Quantidade de requisitos documentados pelo grupo de estudantes;
  - QRR: Quantidade de requisitos de referência, estabelecidos na especificação de cada programa, conforme consta no Apêndice E.

- *Requisitos Documentados Corretamente (RC):* corresponde a porcentagem de requisitos documentados sem defeitos, pelo grupo de estudantes.

$$\text{RC} = \frac{\text{QRD} - \text{QRD}_f}{\text{QRD}} \times 100\%$$

Na qual:

- QRD : Quantidade de requisitos documentados pelo grupo de estudantes;
- QRD_f: Quantidade de requisitos documentados com defeito, contemplando requisitos ambíguos, contraditórios, incorretos e irrelevantes, conforme Quadro 4.4.

Programas. Nós avaliamos os programas dos estudantes levando em consideração a variável *requisitos atendidos* (RA) que é descrita como segue.

- *Requisitos Atendidos (RA):* corresponde a porcentagem de requisitos corretamente implementados (atendidos) pelo programa do estudante.

$$\text{RA} = \frac{\text{QRA}}{\text{QRR}} \times 100\%$$

Na qual:

- QRA : Quantidade de requisitos atendidos (implementados) pelo programa do estudante;
- QRR: Quantidade de requisitos de referência, estabelecidos na especificação de cada programa, conforme consta no Apêndice E.

Coleta de Dados

No estudo de caso, nós coletamos os seguintes dados: (i) documentos de especificação dos grupos; (ii) código fonte dos programas e testes automáticos dos estudantes; (iii) diálogos dos estudantes registrados nos grupos de discussão; (iv) respostas aos questionários aplicados aos estudantes e clientes-tutores (Apêndice D); e, (v) anotações sobre o comportamento dos estudantes durante a realização das atividades de POP.

4.1.2 Execução

Como proposto em POP, nós organizamos os estudantes em grupos e para cada grupo foi designado um cliente-tutor, papel exercido pelos monitores da disciplina de Laboratório de Programação I. Para motivar os estudantes a manterem o foco sobre o "o que fazer" e não sobre "como fazer", nós disponibilizamos uma nota de aula sobre *especificação de requisitos* para que os estudantes lessem antes de

iniciar o ciclo de resolução de problemas de POP. Os clientes-tutores eram os responsáveis por apresentar os *deadlines* aos estudantes e explicar quais artefatos deveriam ser entregues por eles em cada um dos *deadlines*.

Embora cada grupo fosse assistido por um cliente-tutor, a interação entre os estudantes e o cliente-tutor, o registro dos requisitos, a formatação e atualização do documento de especificação eram realizadas na base do "aprender fazendo", isto é, não houve treinamento prévio dos estudantes e o próprio grupo foi negociando e ajustando a forma de conduzir o trabalho para atender aos *deadlines*.

Nós adotamos o *Google Docs*, editor de texto colaborativo, para documentar os requisitos. Para permitir a interação entre os estudantes e seu respectivo cliente-tutor fora de sala de aula, foram criadas listas de discussão no *Google Groups*. Ambos os recursos, eram restritos aos membros de um grupo e seu respectivo cliente-tutor. Nós adotamos os produtos do *Google* por serem ferramentas estáveis, simples de usar e de conhecimento da maioria dos estudantes.

Após a leitura da nota de aula, cada grupo reuniu-se presencialmente com seu respectivo cliente-tutor. Esta primeira sessão teve duração de duas horas e resultou na criação da versão inicial do documento de especificação. Este documento continha a descrição textual dos requisitos e casos de testes de entrada/saída para os programas que deveriam solucionar os problemas propostos.

Após a primeira reunião, os estudantes de cada grupo foram orientados a construir, individualmente, protótipos dos programas e casos de testes automáticos para atender ao documento de especificação produzido por seu respectivo grupo. A implementação dos protótipos dos programas e dos casos de testes automáticos foi realizada pelos estudantes em um horário extra-classe.

Durante esta fase, surgiram novos questionamentos sobre os requisitos e estes foram tratados com o cliente-tutor por meio da lista de discussão, cabendo aos estudantes providenciarem a atualização do documento de especificação.

Uma segunda reunião presencial, também de duas horas, foi realizada entre cada grupo com seu respectivo cliente-tutor. O objetivo desta reunião era produzir a versão final do documento de especificação. Nesta reunião, os protótipos dos programas foram usados pelos estudantes para identificar novos requisitos, assim como detalhar requisitos já identificados.

Após essa segunda reunião, os estudantes ficaram responsáveis por construir a versão final dos programas e dos casos de testes automáticos com o objetivo de cumprir o último *deadline*. Os programas foram construídos utilizando a linguagem Python, que era adotada na disciplina. Os testes automáticos foram escritos usando o comando `assert`[2] do Python.

No final do estudo de caso, nós aplicamos, em versão digital, dois questionário (Apêndice D):

[2]Este comando de Python permite testar uma condição.

um, para os estudantes; e, outro para os clientes-tutores. O objetivo dos questionários era adquirir informações sobre as dificuldades na execução das atividades proposta em POP e sugestões de melhoria.

Em síntese, nós executamos POP obedecendo aos *deadlines* descritos no Quadro 4.5.

Quadro 4.5: Estudo de Caso 1 – *Deadlines*, atividades e *deliverables*.

Deadline	**Atividades e *Deliverables***
04/05/2009	1^a reunião presencial. Atividade: Elaborar especificação inicial. *Deliverable*: Versão inicial do documento de especificação e casos de testes de entrada/saída.
06/05/2009	Atividade: Iniciar implementação (atividade a ser concluída antes de iniciar a 2^a reunião presencial). *Deliverable*: Protótipo dos programas e dos casos de testes automáticos.
06/05/2009	2^a reunião presencial. Atividade: Concluir especificação. *Deliverable*: Versão final do documento de especificação e casos de testes de entrada/saída.
11/05/2009	Atividade: Concluir implementação. *Deliverable*: Programas e testes automáticos.

Cabe ressaltarmos que a execução do estudo de caso foi antecedida por uma fase de preparação dos clientes-tutores. Nessa fase preparatória, nós apresentamos aos clientes-tutores a metodologia, o cronograma de atividades, as instruções para realização das atividades de POP em sala de aula, e fornecemos material de apoio. O material de apoio era composto por três documentos que estabeleciam: (i) a divisão das turmas em grupos, especificando o nome dos componentes de cada grupo, assim como a indicação de cada respectivo cliente-tutor; (ii) orientações gerais sobre como conduzir as atividades de POP em sala de aula; e, (iii) os requisitos de referência para cada programa. Este último documento nós denominamos de *artefato de referência*.

4.1.3 Desvios

Quatro monitores da disciplina de Laboratório de Programação I, que fariam o papel de clientes-tutores, não puderam participar da primeira reunião presencial com os estudantes, em virtude de estarem envolvidos com outras atividades da graduação. Esta indisponibilidade foi comunicada previamente, e estes clientes-tutores foram representados por outros, apenas na primeira reunião presencial.

Os clientes-tutores substitutos eram alunos da graduação e pós-graduação. Eles também passaram por uma fase de preparação para atuarem junto aos estudantes. Esta foi a única situação que não previmos no planejamento original do estudo de caso e que contornamos antes da execução da pesquisa.

4.1.4 Avaliação da Validade

Em nosso estudo, nós levamos em consideração o tratamento de quatro tipos de validade: *interna*, *externa*, de *conclusão* e de *construção*.

Validade Interna. De acordo com depoimentos dos estudantes, nenhum deles havia tido experiência anterior com especificação de requisitos. Além disso, os estudantes repetentes foram mantidos em uma turma separada, garantindo que não haveria influência desses estudantes na produção dos demais alunos. Para evitar possíveis interferências dos clientes-tutores na escrita dos documentos de especificação e entrega deliberada dos requisitos aos estudantes, nós estabelecemos, na fase de preparação, procedimentos de conduta. Além disso, por adotarmos artefatos de referência que especificavam os requisitos dos programas, nós garantimos que todos os estudantes estavam sujeitos aos mesmos requisitos, independente de quem fosse o cliente-tutor. Com esses procedimentos, nós aumentamos a confiança de que fatores inerentes às interações humanas estavam sob controle.

Validade Externa. Neste estudo de caso, nós realizamos uma avaliação de POP em um contexto real de sala de aula. Esta avaliação nos deu suporte para as tomadas de decisão quanto ao *design*, execução e análise dos dados para as próximas pesquisas empíricas.

Validade de Construção. As variáveis e métricas que nós adotamos para avaliação dos documentos de especificação e programas dos estudantes eram baseadas nos artefatos de referência, previamente definidos para cada problema proposto. Assim, nós garantimos o estabelecimento de padrões na avaliação dos artefatos. Ao realizamos a avaliação dos documentos de especificação contando com o trabalho conjunto de duas pessoas, nós minimizamos as ocorrências de erros. Além disso, como nós utilizamos várias fontes de dados (documento de especificação, código fonte dos programas, respostas dos questionários, etc.), nós aumentamos a confiança nos resultados e evitamos erros e avaliações tendenciosas no estudo.

Validade de Conclusão. Neste estudo, nós não dispusemos de um grupo de controle, mesmo porque o objetivo não era fazer comparações com estudantes que não adotaram POP. Assim, não houve como estabelecermos relacionamentos estatísticos, dado o contexto da pesquisa. Para avaliação deste estudo de caso nós utilizamos dados quantitativos e qualitativos, que mesmo sem poder estatístico, contribuíram para a avaliação do fenômeno estudado (POP) levando em consideração o contexto, isto é, o ambiente de sala de aula.

4.1.5 Respostas às Questões de Pesquisa

Os sujeitos de nosso estudo (70 estudantes de programação) foram divididos em 15 grupos. As questões de pesquisa 1 e 2 têm como unidade de análise os grupos e a questão de pesquisa 3, os estudantes, individualmente.

Questão de Pesquisa 1: Quando adotamos POP, os estudantes documentam requisitos?

Como dissemos anteriormente, para solucionar os problemas propostos 1, 2 e 3, os programas dos estudantes deveriam atender a 17, 12 e 26 requisitos, respectivamente. No que diz respeito a documentação desses requisitos, nós apresentamos em um gráfico *dotplot*, mostrado na Figura 4.1, a porcentagem de requisitos documentados pelos grupos, considerando os três problemas.

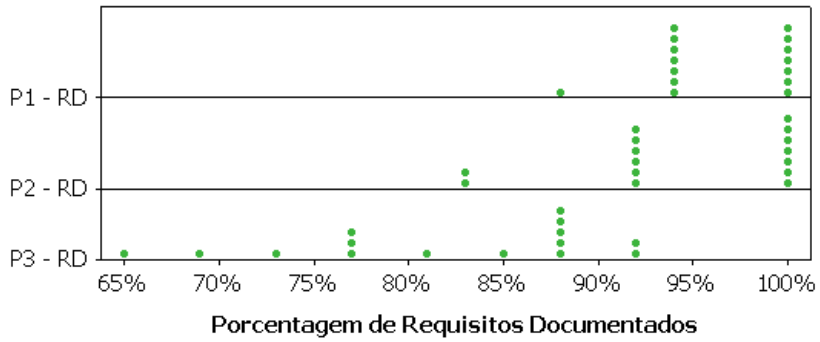

Figura 4.1: Requisitos Documentados (RD) para os três problemas.

Como podemos observar na Figura 4.1, para os três problemas, a maioria dos grupos documentou mais de 80% dos requisitos. Entretanto, com respeito aos requisitos documentados no problema 3, nós verificamos um desempenho inferior dos estudantes se comparados aos outros dois problemas. Para o problema 3, nenhum grupo documentou 100% dos requisitos e houve seis grupos que documentaram menos de 80%, sendo que o valor mínimo documentado foi de 65% dos requisitos.

Na Figura 4.2, nós apresentamos um gráfico de dispersão que mostra uma perspectiva comparativa dos requisitos documentados e documentados corretamente para cada grupo, considerando os problemas 1, 2 e 3, representados na Figura 4.2 por (a), (b) e (c), respectivamente.

Nesta figura, nós queremos chamar atenção para três situações: (1) os grupos que mesmo não tendo especificado todos os requisitos desejados pelo cliente-tutor, conseguiram para os requisitos especificados, documentá-los sem a ocorrência de defeitos; (2) grupos que especificaram todos os requisitos desejados pelo cliente-tutor, mas os documentaram com algum tipo de defeito; (3) grupos que conseguiram especificar todos os requisitos, documentando-os corretamente.

Para exemplificarmos o primeiro caso, observe que no problema 1 (Figura 4.2 (a)), o grupo 6,

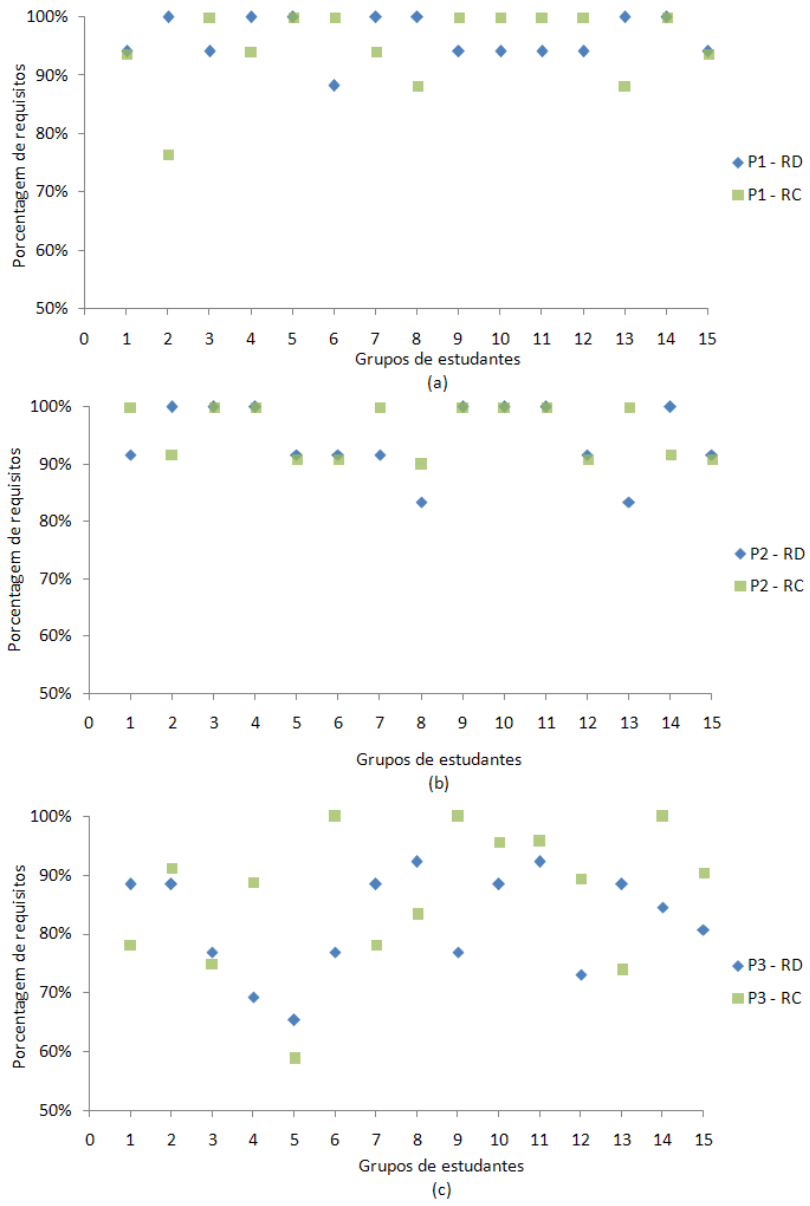

Figura 4.2: Requisitos Documentados (RD) e Requisitos Documentados Corretamente (RC).

documentou apenas $88{,}6\%$ dos requisitos, porém todos eles foram documentados corretamente.

Para exemplificarmos o segundo caso, observe que no problem 1, o grupo 2 documentou 100% dos requisitos, no entanto apenas $76{,}5\%$ desses requisitos foram documentados corretamente.

Considerando o terceiro caso, isto é, dos grupos que especificaram 100% dos requisitos, documentando-os sem qualquer defeito, nós citamos os seguintes exemplos: grupos 5 e 14, no problema 1 e os grupos 3 e 4, no problema 2.

No problema 3 houve a menor porcentagem de requisitos documentados corretamente – $58{,}8\%$ referente a documentação do grupo 5. A exceção deste grupo, todos os demais, considerando os três

problemas, documentaram mais de 70% dos requisitos corretamente.

Portanto, pelos dados obtidos, nós observamos que as atividades de POP favorecem a documentação dos requisitos por parte dos estudantes.

Questão de Pesquisa 2: Quais tipos de defeitos são mais comuns nos documentos de especificação produzidos pelos estudantes que adotam POP?

Na Tabela 4.1, nós apresentamos os tipos de defeitos encontrados nos documentos de especificação dos grupos de estudantes, considerando os três problemas propostos. Nesta tabela, cada tipo de defeito está associada a sua quantidade e porcentagem correspondente. Com o intuito de tornarmos a apresentação desses dados mais sumária, nós apresentamos na Figura 4.3, a porcentagem de cada tipo de defeito, considerando os três problemas propostos.

Tabela 4.1: Quantidade e porcentagem de defeitos na especificação.

Problemas	#/%	**Defeitos**					
		Omitido	**Ambíguo**	**Contraditório**	**Incorreto**	**Irrelevante**	**Total**
P1	#	9	1	0	8	3	21
	%	42,8	4,8	0	38,1	14,3	100
P2	#	10	1	0	4	2	17
	%	58,8	5,9	0	23,5	11,8	100
P3	#	70	2	1	39	0	112
	%	62,5	1,8	0,9	34,8	0	100

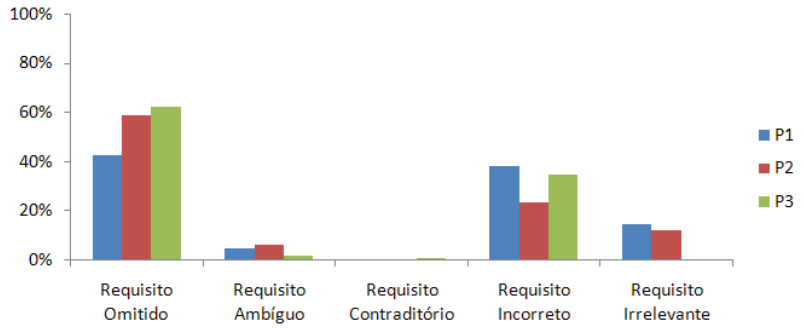

Figura 4.3: Tipos de defeitos.

Como podemos observar na Figura 4.3, os tipos de defeitos mais freqüentes foram *requisitos omitidos* e *requisitos incorretos*. Com relação ao tipo de defeito *requisitos omitidos*, nós observamos que ele estava mais relacionado às omissões das restrições do programa. Por exemplo, na descrição

dos requisitos do programa para solucionar o problema 2, alguns grupos omitiram o requisito que estabelecia que não deveria haver diferenciação entre caracteres maiúsculos e minúsculos (requisito 3, descrito no Apêndice E.2). Na descrição dos requisitos do programa para solucionar o problema 3, também foi comum os grupos omitirem o requisito que definia que os sinais de acentuação e apóstrofos, mesmo que digitados pelo usuário, deveriam ser ignorados na saída (requisito 6 e 7, descritos no Apêndice E.3).

Com relação ao tipo de defeito *requisitos incorretos*, nós observamos que foi mais freqüente os grupos documentarem erroneamente as mensagens que deveriam ser exibidas em caso de entradas inválidas. Na Figura 4.4, nós apresentamos um exemplo de especificação incorreta da mensagem que deveria ser exibida no caso de entrada inválida para o capital a ser investido (problema 1).

Mensagem esperada pelo cliente-tutor no caso de entrada de valores inválidos para a variável capital.

```
Investimento minimo de R$ 30,00.
Capital?
```

Mensagem de erro documentada pelo grupo.

```
Investimento minimo de 30 reais.
```

Figura 4.4: Requisito incorreto.

Pelas observações que fizemos nas especificações produzidas pelos grupos, nós acreditamos que o registro de casos de testes de entrada/saída favoreceu a documentação dos requisitos. Nossa observação se justifica pelo fato de que ao expressar casos de testes de entrada/saída, os grupos conseguem explicitar um conjunto de requisitos, de forma sucinta e sem precisar de textos adicionais. Por exemplo, na Figura 4.5, nós apresentamos dois casos de testes de entrada/saída documentados por um dos grupo. Como podemos verificar na Figura 4.5, os casos de testes especificam diferentes requisitos, como por exemplo, *label* da entrada, tratamento dos caracteres não-alfabéticos e mensagem de saída, no caso de não haver palavras válidas na entrada.

```
Paragrafo?
Nenhuma palavra foi digitada.

Paragrafo? 12354860&*(&@$*#@&*$(&#$(*{}{}{["
Nenhuma palavra foi digitada.
```

Figura 4.5: Casos de testes de entrada/saída.

Os casos de testes de entrada/saída foram adotados pela maioria dos grupos para representar

requisitos. Na Figura 4.6, nós apresentamos, por meio de um gráfico *boxplot*, a porcentagem de requisitos cobertos pelos casos de testes de entrada/saída documentados pelos grupos, considerando os três problemas. Como podemos observar na Figura 4.6, para os problemas 1 e 2, os grupos cobriram, em média ($\oplus$), mais de 45% dos requisitos por meio de casos de testes de entrada/saída. Em contrapartida, no problema 3, a média foi de apenas $29{,}49\%$ dos requisitos.

Esse resultados demonstram que, como proposto em POP, os testes de entrada/saída foram também utilizados pelos estudantes para especificar requisitos.

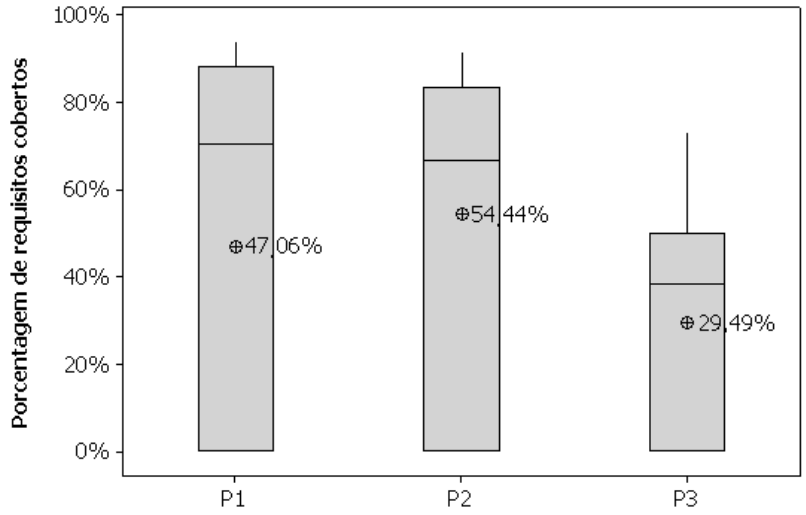

Figura 4.6: Porcentagem de requisitos cobertos pelos casos de testes de entrada/saída.

Questão de Pesquisa 3: Ao concluir o ciclo de POP, os programas produzidos pelos estudantes atendem aos requisitos requeridos pelo cliente-tutor?

Conforme estabelecido em POP, a implementação dos programas e casos de testes automáticos foi realizada individualmente. Na Tabela 4.2, nós apresentamos a porcentagem de estudantes que entregaram programas e testes automáticos, cumprindo ao último *deadline* do ciclo de resolução de problemas.

Como podemos observar na Tabela 4.2, a exceção dos programas para solucionar o problema 3, em todos os demais, a porcentagem de submissão foi superior a 65%. A situação mais crítica ocorreu no problema 3, no qual mais de 50% dos estudantes, não submeteram seus programas.

Com respeito aos teste automáticos, nós podemos observar na Tabela 4.2 que a porcentagem de estudantes que entregaram testes foi muito baixa, não chegando, para nenhum dos três problemas propostos, a 30% de submissões. Este valor foi verificado, mesmo para o problema 1, cuja solução era mais simples e teve a maior porcentagem de submissões de programas.

Na Figura 4.7, nós apresentamos um gráfico de barras que mostra a porcentagem de requisitos

Tabela 4.2: Porcentagem de estudantes que entregaram programas e testes automáticos.

Problema 1		**Problema 2**		**Problema 3**	
Programas	**Testes**	**Programas**	**Testes**	**Programas**	**Testes**
85,7%	27,1%	67,1%	20,0%	41,4%	5,7%

implementados pelos programas submetidos pelos estudantes. Como podemos observar, mais de 70% dos estudantes atenderam a uma faixa de 75% a 100% dos requisitos, considerando os três programas.

Cabe ressaltarmos, que 89,7% dos estudantes que submeteram o programa 3 atenderam a uma faixa de 75% a 100% dos requisitos. Portanto, mesmo tendo o menor número de submissões (41,4%, conforme Tabela 4.2), os programas submetidos para o problema 3 tiveram uma alta porcentagem de atendimento dos requisitos.

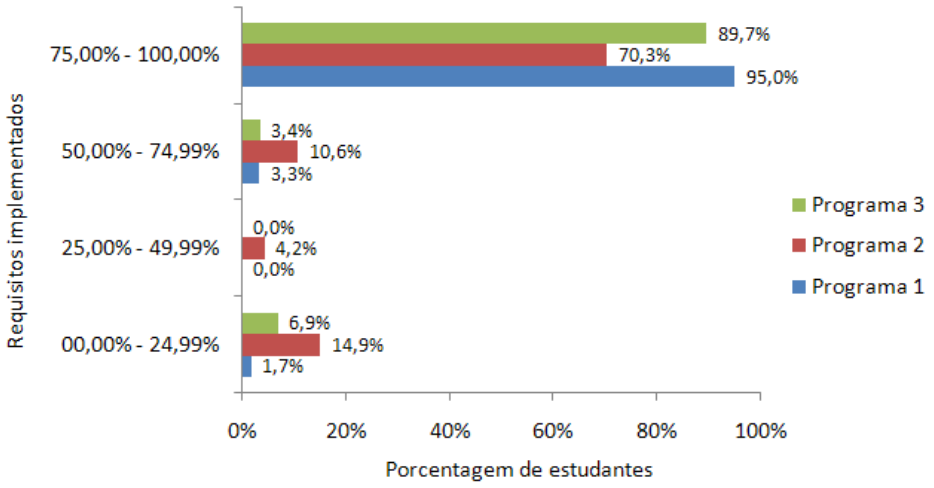

Figura 4.7: Requisitos implementados pelos estudantes.

Questão de Pesquisa 4: Quais as dificuldades apresentadas pelos estudantes ao praticarem as atividades de POP?

Para responder a esta questão de pesquisa, nós utilizamos as respostas dos estudantes para o questionário que foi aplicado após o estudo de caso. Este questionário é apresentado no Apêndice D e a resposta a ele era voluntária. Dos 70 estudantes que participaram do estudo, apenas 39 responderam ao questionário.

Nós trataremos das dificuldades dos estudantes considerando a especificação dos requisitos, implementação do programa e criação testes. Apresentamos também as lições aprendidas e sugestões de melhorias, segundo relatos dos estudantes.

Dificuldades na especificação dos requisitos.

Com relação à especificação dos requisitos, os estudantes destacaram três dificuldades principais: (i) questionar o cliente; (ii) documentar os requisitos; e (iii) interagir com o grupo.

No que diz respeito a *questionar o cliente*, os estudantes, inicialmente, tiveram dificuldades em assumir uma postura pró-ativa a fim de esclarecer o que deveria ser feito. Essa dificuldade é ilustrada pela resposta de um dos estudantes ao questionário, conforme apresentamos na Figura 4.8.

> "Inicialmente fiquei sem saber o que exatamente teria que perguntar ao cliente. Pensei até que o cliente poderia me dizer exatamente o que o programa iria fazer, mas depois vi que eu deveria perguntar tudo, numa tentativa de desvendar o que o cliente realmente queria."

Figura 4.8: Dificuldade na elicitação dos requisitos.

Com relação a *documentação dos requisitos*, os estudantes tiveram dificuldades para estabelecer uma estrutura de documento na qual os requisitos ficassem descritos de forma organizada e clara para todos os membros do grupo. Esta dificuldade é representada pela resposta de outro estudante ao questionário, conforme ilustramos na Figura 4.9.

> "Sentimos dificuldade na forma de escrever o documento de especificação, pois não tinhamos parametros de comparação para saber se aquela era a melhor estrutura/forma a se usar para a especificação."

Figura 4.9: Dificuldade na documentação dos requisitos.

Como a atividade de especificação era realizada em grupo e a maioria das atividades realizadas na disciplina de programação eram individuais, os estudantes tiveram também dificuldades em *interagir com os demais membros do grupo*. Uma das dificuldades dizia respeito à negociação da melhor forma de registrar e atualizar o documento de especificação. Essa negociação era importante para manter a organização e clareza do documento, independente de qual membro do grupo fizesse o registro ou atualização dos requisitos.

Outra dificuldade era a de garantir a participação igualitária de todos os membros do grupo nas discussões para o esclarecimento dos requisitos. Nesse sentido, cabe destacarmos que alguns fatores interferem nas interações do grupo, tais como timidez ou até mesmo apatia de alguns estudantes. Embora esse fatores interfiram no trabalho em grupo, $66{,}7\%$ dos estudantes avaliaram as interações com seu respectivo grupo como ótimo ou bom, conforme ilustramos na Figura 4.10.

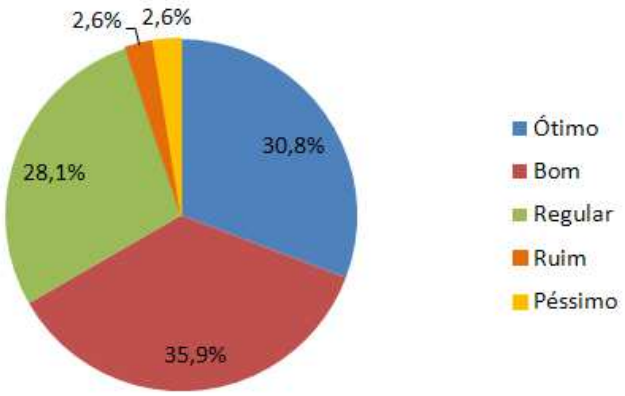

Figura 4.10: Avaliação das interações em grupo.

Dificuldades para implementar os programas.

Na Figura 4.11, nós apresentamos as opiniões dos estudantes quanto a complexidade para solucionar os problemas propostos. Como podemos observar na Figura 4.11, os estudantes atribuíram complexidades gradativas aos programas, sendo que $92{,}3\%$ dos estudantes consideraram o programa 1 fácil e $64{,}1\%$ dos estudantes consideraram o programa 3 difícil.

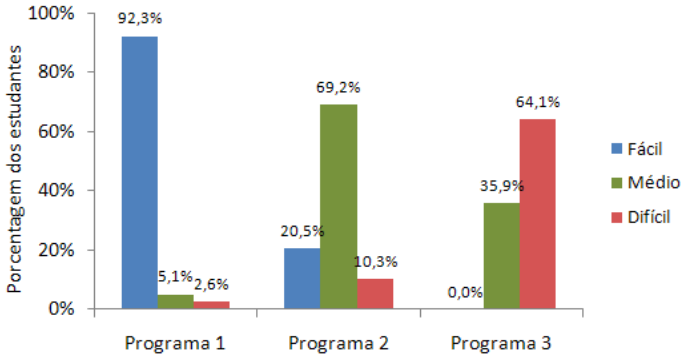

Figura 4.11: Complexidade para solucionar os problemas.

Na fase de implementação, os estudantes destacaram as seguintes dificuldades: organizar o código em funções; manipular *strings* e trabalhar com o comando para limpar o console, considerando os sistemas operacionais Windows e Linux. Além disso, os estudantes apontaram a necessidade de mais tempo para a fase de implementação.

Dificuldades em criar testes automáticos.

Do total de estudantes que responderam ao questionário, 55% deles relataram dificuldades em criar testes automáticos. Alguns estudantes relataram que a dificuldade em estruturar o código em

funções, tornou a prática de testes ainda mais árdua. Ao se auto-avaliarem com relação à criação de testes, 84,6% dos respondentes acreditavam que seu desempenho nesta atividade era considerado regular, ruim ou péssimo, conforme demonstramos na Figura 4.12. Estes números são também evidências que justificam a baixa porcentagem de estudantes que entregaram testes automáticos, conforme dados apresentados na Tabela 4.2.

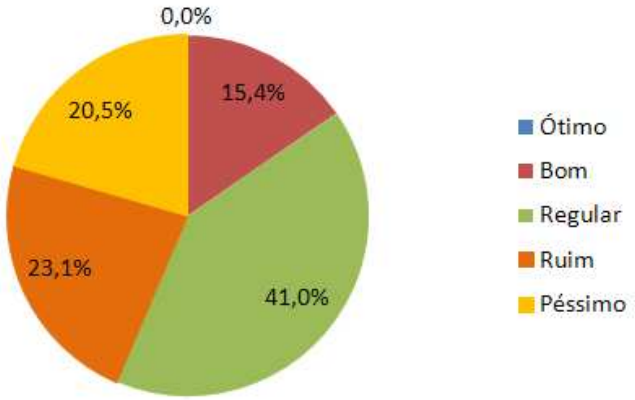

Figura 4.12: Auto-avaliação dos estudantes quanto a criação de testes automáticos.

Lições aprendidas.

Na Figura 4.13, nós apresentamos algumas respostas dos estudantes que descrevem as lições aprendidas com a execução de POP. Esses depoimentos reforçam a nossa convicção de que a aplicação da metodologia proporcionou aos estudantes experimentarem situações mais próximas daquelas vivenciadas no mundo real.

Que coisas novas você aprendeu?
"Como deve ser a relação entre o cliente e o programador, que perguntas devem ser feitas pelo programador para tentar compreender o que o cliente realmente deseja, e que nem sempre essa é uma tarefa fácil!"
"Descobri o quão difícil é saber o que uma pessoa quer."
"A importância das especificações e atenção a pequenos detalhes que podem comprometer um programa."
"A tentar entender o que o cliente deseja, ser bem detalhista para não implementar no programa algo que apenas eu tinha pensado e o cliente não gostaria. A documentar tudo que for solicitado. "Viver" um pouco da realidade do programador."

Figura 4.13: Lições aprendidas pelos estudantes.

Sugestões de melhorias.

Os estudantes apontaram duas sugestões de melhoria: (1) aumentar o prazo para a entrega dos programas; e (2) desenvolver uma atividade prévia que demonstre como interagir com um cliente e como produzir um documento de especificação.

Questão de Pesquisa 5: Quais as dificuldades apresentadas pelos clientes-tutores para a condução de POP em sala de aula?

Para responder a esta questão de pesquisa nós utilizamos as respostas dos clientes-tutores para o questionário que foi aplicado após o estudo de caso. O questionário é apresentado no Apêndice D e a resposta a ele era voluntária. Dos 13 clientes-tutores que participaram do estudo, apenas 8 responderam ao questionário.

Nós apresentamos as respostas obtidas nesse questionário de forma sumária e organizadas na forma de tópicos, conforme descrevemos a seguir.

Dificuldades observadas nos estudantes na fase de especificação dos programas.

Segundo os clientes-tutores as principais dificuldades foram:

- *Trabalhar em grupo*. Inicialmente, os estudantes apresentaram dificuldades em discutir o problema em grupo e envolver todos os participantes na discussão;
- *Dialogar com o cliente-tutor*. No início das atividades, os estudantes assumem a postura de aguardar o cliente-tutor definir os requisitos dos programas. Em virtude disso, os clientes-tutores, geralmente, iniciavam as interações incentivando os estudantes a questionarem;
- *Elicitar as restrições dos programas.* Os estudantes também apresentaram dificuldades em raciocinar sobre as restrições menos triviais dos programas e questionar sobre elas. Em geral, a elicitação destes requisitos ocorria após a primeira reunião com o cliente-tutor.

Para os clientes-tutores essas dificuldades eram mitigadas à medida que os estudantes progrediam de um problema para o outro. Para melhorar a elicitação das restrições dos programas, os clientes-tutores recomendaram aumentar a prática de testes na disciplina.

Adequação do material de apoio fornecido aos clientes-tutores.

Dos 8 cliente-tutores que responderam ao questionário, 7 deles consideraram o material fornecido totalmente adequado, e um deles considerou parcialmente adequado. A sugestão de melhoria era adicionar uma espécie de documento contendo *checklist* para cada requisito do programa. Esse recurso ajudaria o cliente-tutor a ter mais controle sobre os requisitos que iam sendo especificados pelos estudantes.

Dificuldades sentidas pelos próprios clientes-tutores na condução de POP.

Ao iniciar as atividades de POP, os clientes-tutores sentiram dificuldade em engajar todos os estudantes do grupo na discussão sobre os requisitos, assim como orientá-los a raciocinar sobre as restrições não triviais dos programas sem fornecer deliberadamente tais requisitos.

A fim de maximizar o engajamento dos estudantes no grupo, alguns cliente-tutores aumentaram o controle sobre os estudantes menos pró-ativos, contabilizando os questionamentos dos estudantes, solicitando o registro da contribuição de cada aluno no documento de especificação e requerendo, explicitamente, a participação de um ou outro estudante que se mostrasse menos ativo. Para motivar os estudantes a raciocinarem sobre os requisitos, os cliente-tutores, em geral, utilizaram a estratégia de questionar os estudantes sobre seu entendimento a respeito de como programa deveria funcionar para resolver o problema proposto.

Sugestões de melhoria.

Os cliente-tutores apontaram como sugestões de melhoria aumentar o tempo para a realização das atividades e aumentar a prática de testes na disciplina.

4.1.6 Discussão

O registro incorreto de mensagens de erros para o caso de entradas inválidas, tal como apresentada na Figura 4.4, foi um tipo de erro comum que observamos nos documentos de especificação dos estudantes. Essa situação nos chama atenção para a necessidade de melhor preparar os estudantes para tratar com requisitos não funcionais.

Nos casos em que os estudantes tinham um certo conhecimento prévio do domínio do problema, nós percebemos que eles eram menos efetivos na captura dos requisitos. Para exemplificar, citamos o caso do problema 3 que tratava do jogo da forca. Como os estudantes tinham um modelo mental prévio sobre o funcionamento do jogo, eles, confiando neste modelo, negligenciaram o esclarecimento e documentação de alguns requisitos. Essa situação, aliada ao fato de que o problema 3 tinha o maior número de requisitos a serem esclarecidos (26 requisitos, vide Apêndice E), contribuíram para a baixa porcentagem de requisitos documentados quando comparada aos problemas 1 e 2.

Dada as dificuldades dos estudantes com a criação de testes automáticos, a maioria deles assegu-raram a qualidade de seus programas executando o código e verificando, manualmente, os casos de testes previamente reportados em seu documento de especificação. A melhor preparação dos estudantes iniciantes para a prática de testes automáticos é um desafio, pois demanda outras habilidades, tais como, raciocinar sobre diferentes restrições e possibilidades de erros; e, estruturar o código em funções, o que implica na necessidade de uma maior percepção sobre *design*.

Janzen and Saiedian [2006a] e Desai et al. [2008; 2009] reportam também as dificuldades dos estudantes iniciantes com a criação de testes automáticos. De acordo com esses autores, os resultados têm sido mais promissores com estudantes em níveis mais avançados.

Cabe destacar, que não é objetivo de POP tornar os estudantes *experts* em testes, mas fazê-los praticar testes, percebendo-os não como uma atividade que está no fim do processo de desenvolvimento, mas que pode ser utilizado desde o início, inclusive como um instrumento para capturar requisitos.

Acreditamos que fortalecer o ensino de funções e prover, de forma mais sistemática, exemplos de testes para programas gradativamente mais complexos podem ajudar os estudantes na criação de testes automáticos.

No que diz respeito à entrega dos programas, uma outra lista de problemas de programação teve sua data de entrega estendida, sobrecarregado os estudantes no mesmo período do estudo de caso. Nós acreditamos que esse fato tenha colaborado para a diminuição no número de submissões dos programas, sobretudo para o programa 3 que era o mais complexo de ser implementado.

No que diz respeito ao trabalho em grupo, nós consideramos que algumas dificuldades dos estudantes são naturais. Embora, no estudo de caso tenhamos aumentado o controle sobre as interações dos estudantes, é muito difícil garantir que todos trabalhem igualitariamente. Nós observamos que as interações em grupo aumentaram as discussões sobre os requisitos. Entretanto, trouxeram alguns problemas que deveriam ser negociadas pelo próprio grupo, como por exemplo, administrar a atualização do documento de especificação.

Para melhorar o conhecimento dos estudantes sobre como interagir com o cliente-tutor e documentar requisitos, nós avaliamos como importante a sugestão dos estudantes de realizar uma atividade prévia na qual seja dado um exemplo de como proceder para esses casos. Assim, faremos essa melhoria na próxima execução de POP.

4.2 Estudo de Caso 2

Nesta seção, nós apresentamos o estudo de caso realizado no segundo semestre acadêmico de 2009. No que diz respeito ao planejamento e execução do estudo, destacaremos apenas os aspectos que foram modificados em relação ao estudo de caso anterior.

4.2.1 Planejamento

A exceção dos sujeitos que participaram deste estudo de caso, todos os demais aspectos adotados no planejamento do estudo anterior foram mantidos no segundo semestre acadêmico. Assim, nós apresentamos na próxima seção uma descrição mais detalhada dos estudantes que participaram do segundo estudo de caso.

Sujeitos

Novamente, nós aplicamos POP com os estudantes matriculados na disciplina de Laboratório de Programação I, período 2009.2, do curso de Ciência da Computação da UFCG. Assim como no semestre acadêmico anterior, esta disciplina possuía três turmas, sendo a terceira turma composta apenas por estudantes repetentes.

Neste estudo, nós consideramos também apenas os estudantes iniciantes, que compunham as Turmas 1 e 2. Estas turmas, originalmente, eram compostas por 70 estudantes, porém em virtude de desistências ou ausências na disciplina no período de realização do estudo, apenas 55 estudantes participaram da execução de POP.

Como descrevemos anteriormente, a divisão dos estudantes por turma ocorre de forma aleatória e o mesmo tratamento (POP) foi aplicado, igualmente, em ambas as turmas, por isso não faremos distinções entre as Turma 1 e 2. Assim, nossos sujeitos, neste estudo de caso, foram 55 estudantes iniciantes de programação.

4.2.2 Execução

Na execução deste segundo estudo de caso, nós incorporamos uma modificação, em relação ao estudo anterior, a fim de melhorarmos as habilidades dos estudantes para dialogar com o cliente-tutor e especificar requisitos.

Assim, nós realizamos na disciplina teórica de programação uma atividade prévia com os estudantes a fim de prepará-los para a execução de POP. Nesta atividade, nós falamos de forma sucinta sobre problemas bem e mal-definidos e sobre requisitos. Após essa breve conceituação, nós apresentamos um problema mal-definido aos estudantes e construímos em conjunto a especificação dos requisitos. Nesta atividade, nós assumimos o papel de cliente-tutor e todos os estudantes da turma exerciam o papel de desenvolvedores que deveriam questionar o cliente-tutor. Um estudante foi escolhido para ser o *escriba* e tinha como função documentar os requisitos no *Google Docs*. No processo de elicitação, os estudantes foram orientados a perceber os elementos que estavam mal definidos

no enunciado do problema, a questionar o cliente-tutor e a utilizar casos de testes para extrair requisitos. Ao final da aula, dois outros estudantes ficaram responsáveis por formatar o documento e compartilhá-lo com a turma. No processo de formatação, nós auxiliamos os estudantes respondendo aos questionamentos por meio da lista de discussão da disciplina.

Como no estudo de caso anterior, nós executamos POP na disciplina de Laboratório de Programação I, utilizamos os mesmos objetos, procedimentos e recursos. Os *deadlines* que nós estabelecemos para esse estudo de caso são apresentados no Quadro 4.6.

Quadro 4.6: Estudo de Caso 2 – *Deadlines*, atividades e *deliverables*.

Deadline	**Atividades e *Deliverables***
19/10/2009	1^a reunião presencial. Atividade: Elaborar especificação inicial. *Deliverable*: Versão inicial do documento de especificação e casos de testes de entrada/saída.
22/10/2009	Atividade: Iniciar implementação (atividade a ser concluída antes de iniciar a 2^a reunião presencial). *Deliverable*: Protótipo dos programas e dos casos de testes automáticos.
22/10/2009	2^a reunião presencial. Atividade: Concluir a especificação. *Deliverable*: Versão final do documento de especificação e casos de testes de entrada/saída.
28/10/2009	Atividade: Concluir implementação. *Deliverable*: Programas e testes automáticos.

4.2.3 Desvios

Assim como no primeiro estudo de caso, três monitores da disciplina de Laboratório de Programação I que fariam o papel de clientes-tutores não puderam participar da primeira reunião presencial com os estudantes e tiveram que ser representados por outros clientes-tutores. Os clientes-tutores substitutos foram os mesmos que participaram do estudo de caso anterior e, em virtude disso, não foi necessário submetê-los a uma fase de preparação.

4.2.4 Avaliação da Validade

A modificação feita nesse estudo de caso (preparação prévia dos estudantes) não alterou a avaliação da validade. Portanto, os aspectos descritos na Seção 5.1.4 são também pertinentes para este estudo de caso.

4.2.5 Respostas às Questões de Pesquisa

Os sujeitos de nosso estudo (55 estudantes de programação) foram divididos em 11 grupos. As questões de pesquisa 1 e 2 têm como unidade de análise os grupos e a questão de pesquisa 3, os estudantes, individualmente.

Questão de Pesquisa 1: Quando adotamos POP, os estudantes documentam requisitos?

Na Figura 4.14, por meio de um um gráfico *dotplot*, nós apresentamos a porcentagem de requisitos documentados pelos grupos, considerando os três problemas propostos.

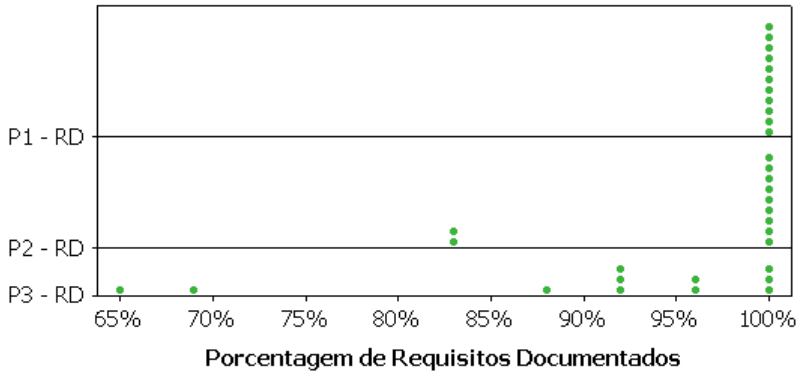

Figura 4.14: Requisitos Documentados (RD) para os três problemas.

Como podemos observar na Figura 4.14, a exceção de dois grupos, todos os demais documentaram mais de 80% dos requisitos referentes aos três programas. No programa 1, todos os grupos conseguiram documentar 100% dos requisitos. O mínimo documentado foi de $65{,}4\%$ dos requisitos, referente ao programa 3.

Na Figura 4.15, nós apresentamos um gráfico de dispersão que mostra a comparação entre os requisitos documentados e os documentados corretamente para cada grupo, considerando os problemas 1, 2 e 3, representados na Figura 4.15 por (a), (b) e (c), respectivamente.

Como podemos observar na Figura 4.15, dos requisitos documentados pelos grupos, mais de 80% dos requisitos foram documentados corretamente, considerando os três programas. Além disso, para os programas referentes aos problemas 1 e 2, seis grupos conseguiram especificar 100% dos requisitos, documentando-os sem qualquer tipo de defeito.

Portanto, pela repetição de nosso estudo, nós confirmamos que as atividades de POP favorecem a documentação dos requisitos por parte dos estudantes.

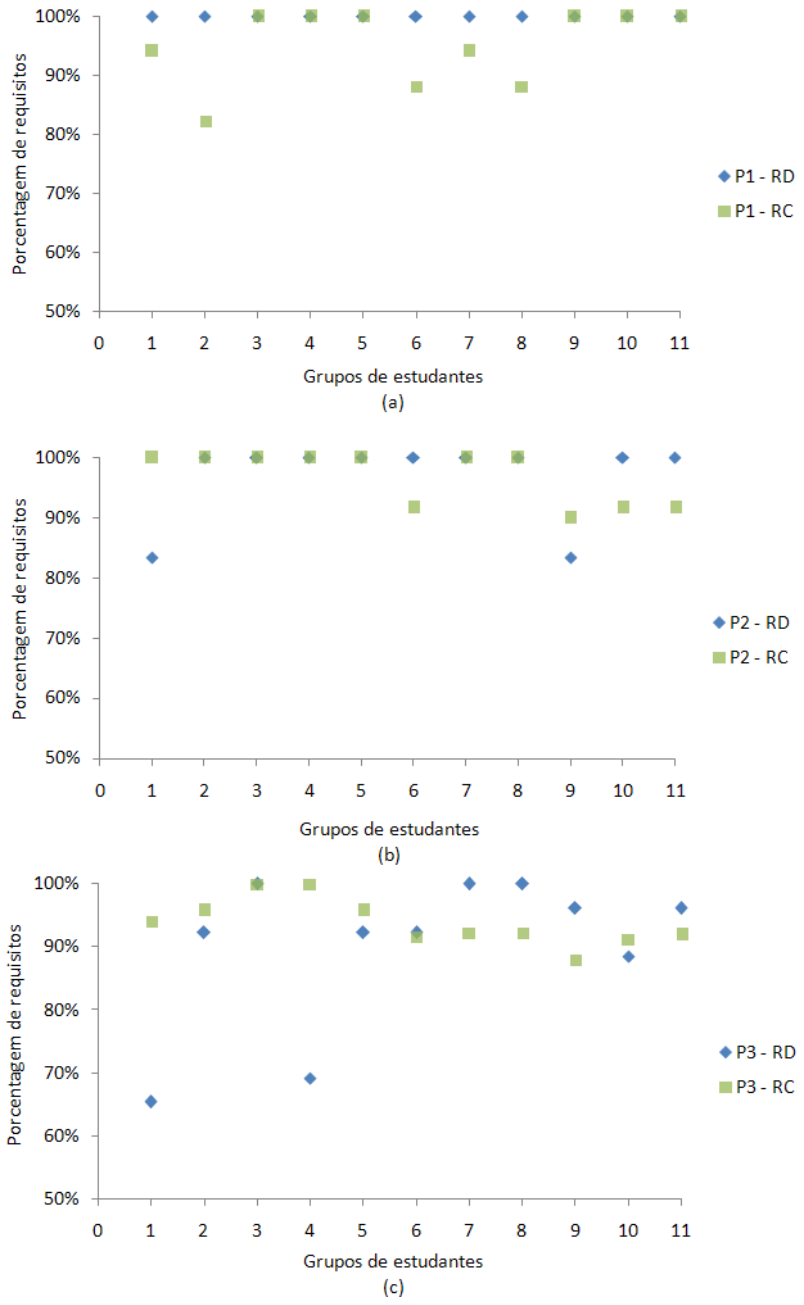

Figura 4.15: Requisitos Documentados (RD) e Requisitos Documentados Corretamente (RC).

Questão de Pesquisa 2: Quais tipos de defeitos são mais comuns nos documentos de especificação produzidos pelos estudantes que adotam POP?

Na Tabela 4.3, nós apresentamos os tipos de defeitos encontrados nos documentos de especificação produzidos pelos grupos, considerando os três problemas propostos. Nesta tabela, cada tipo de defeito está associada a sua quantidade e porcentagem correspondente. Nós apresentamos um

sumário dos tipos de defeito na Figura 4.16.

Tabela 4.3: Quantidade e porcentagem de defeitos na especificação.

Problemas	#/%	**Defeitos**					
		Omitido	**Ambíguo**	**Contraditório**	**Incorreto**	**Irrelevante**	**Total**
P1	#	0	0	4	5	0	9
	%	0	0	44,4	55,6	0	100
P2	#	4	0	4	0	0	8
	%	50	0	50	0	0	100
P3	#	28	2	5	9	0	44
	%	63,6	4,5	11,4	20,5	0	100

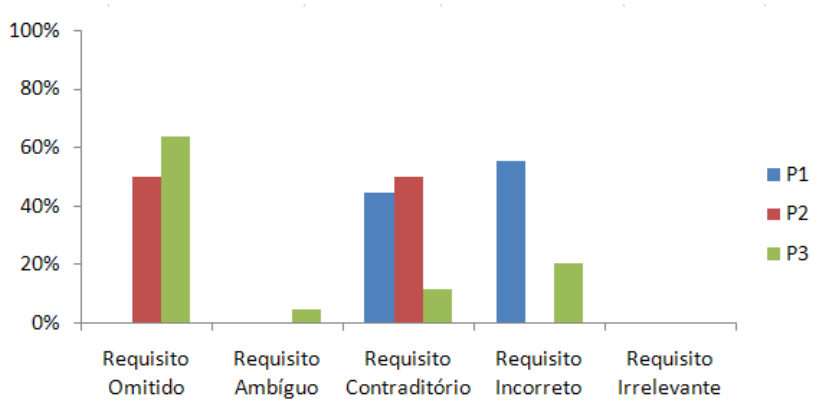

Figura 4.16: Tipos de defeitos.

Como podemos observar na Figura 4.16, os tipos de defeitos mais comuns foram requisitos *omitidos*, *contraditórios* e *incorretos*. O tipo de defeito *requisito omitido* estava mais relacionado as omissões das restrições e formatações de entrada e saída. A ocorrência deste tipo de defeito foi mais freqüente na especificação do programa 3 (jogo da forca). Por exemplo, alguns grupos omitiram as restrições referentes ao tratamento de palavras compostas, maiúsculas e minúsculas, com hífens e apóstrofos.

No que diz respeito ao tipo de defeito *requisito contraditório*, ele ocorreu com mais freqüência nos casos em que a descrição textual dos requisitos contradizia o caso de teste de entrada/saída que, em geral, era mais completo. Na Figura 4.17, nós apresentamos um fragmento de uma especificação na qual o grupo descreveu o que deveria ocorrer nos casos de entradas inválidas para tempo e capital, sem notificar que novas entradas deveriam ser solicitadas. Esta descrição contradiz o caso de teste de

entrada/saída fornecido como exemplo.

Entrada:
-"Capital? " # em R$ (reais), valor em float.
- O valor mínimo para capital deve ser R$ 30.00.
- Se capital for menor que o valor mínimo, emitir a mensagem: "Investimento minimo de R$ 30,00."

-"Tempo? " # em meses
- O tempo mínimo deve ser de 2 meses, e o tempo máximo de 48 meses.
- Se o tempo for menor que 2 meses ou maior que 48 meses, emitir a mensagem: "Periodo de tempo de 02 a 48 meses."

Exemplo:

```
Capital? -100
Investimento minimo de R$ 30,00.
Capital? 100
Tempo? 100
Periodo de tempo de 02 a 48 meses.
Tempo? 10
Rendimento: R$ 162.89
Capital Futuro: R$ 1162.89
```

Figura 4.17: Requisito contraditório.

Com relação ao tipo de defeito *requisito incorreto*, nós observamos que foi freqüente alguns grupos descreverem erroneamente os *labels* para entrada de dados e as mensagens que deveriam ser exibidas em caso de entradas inválidas. Na Figura 4.18, nós apresentamos um exemplo de especificação incorreta do *label* para entrada de palpites no jogo da forca.

***Label* para entrada de palpites esperado pelo cliente-tutor**
`Palpites?`

***Label* para entrada de palpites documentado pelo grupo**
`Palpite:`

Figura 4.18: Requisito incorreto.

Em se tratando de casos de testes de entrada/saída, podemos afirmar que eles foram adotados pela maioria dos grupos para representar requisitos. Na Figura 4.19, por meio de um gráfico *box-plot*, nós apresentamos a porcentagem de requisitos cobertos pelos casos de testes de entrada/saída documentados pelos grupos, considerando os três problemas.

Como podemos observar na Figura 4.19, apenas dois grupos não documentaram qualquer teste. Isto ocorreu para o problema 3 (P3), como pode ser visualizado no gráfico. Considerando os três problemas propostos, os grupos representaram, em média ($\oplus$), mais de 60% dos requisitos utilizando casos de testes de entrada/saída.

Portanto, pelos resultados obtidos, acreditamos que os estudantes compreenderam a necessidade da utilização de teste e fizeram uso dos mesmos para a especificação de requisitos, conforme proposto em POP.

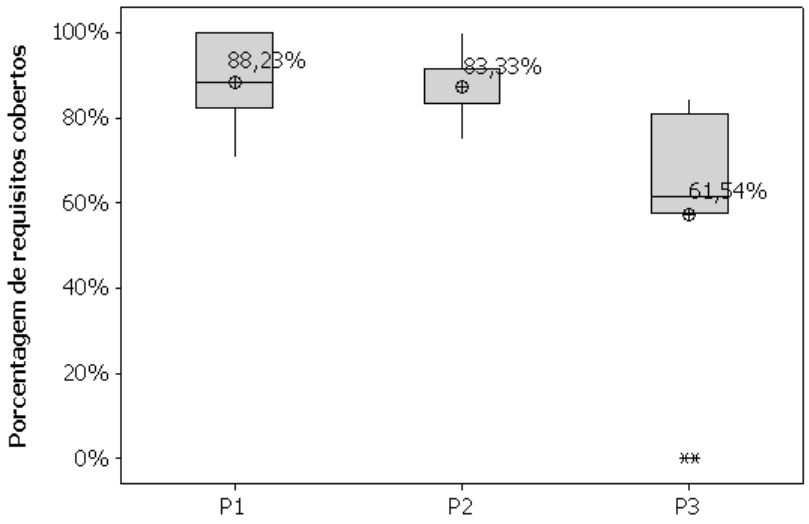

Figura 4.19: Porcentagem de requisitos cobertos pelos casos de testes de entrada/saída.

Questão de Pesquisa 3: Ao concluir o ciclo de POP, os programas produzidos pelos estudantes atendem aos requisitos requeridos pelo cliente-tutor?

Na Tabela 4.4, nós apresentamos a porcentagem de estudantes que entregaram programas e testes automáticos, ao fim do ciclo de resolução de problemas. Como podemos observar nesta tabela, a exceção do programa 3, os demais tiveram submissão acima de 80%. Novamente, a situação mais crítica ocorreu na submissão dos programa para solucionar o problema 3, na qual cerca de 50% dos estudantes não submeteram seus programas.

No que diz respeito aos testes automáticos, a submissão foi muito pequena, não chegando para nenhum dos três problemas propostos, a 15% de submissões.

Tabela 4.4: Porcentagem de estudantes que entregaram programas e testes automáticos.

Problema 1		**Problema 2**		**Problema 3**	
Programas	**Testes**	**Programas**	**Testes**	**Programas**	**Testes**
89,1%	12,7%	81,8%	5,4%	50,9%	5,4%

Na Figura 4.20, nós apresentamos a porcentagem de requisitos implementados pelos programas submetidos pelos estudantes. Nesta figura, podemos observar que mais de 70% dos estudantes atenderam a uma faixa de 75% a 100% dos requisitos, considerando os três programas. Apenas 13,7%

dos estudantes que submeteram seus programas implementaram menos que 50% dos requisitos, considerando os três programas.

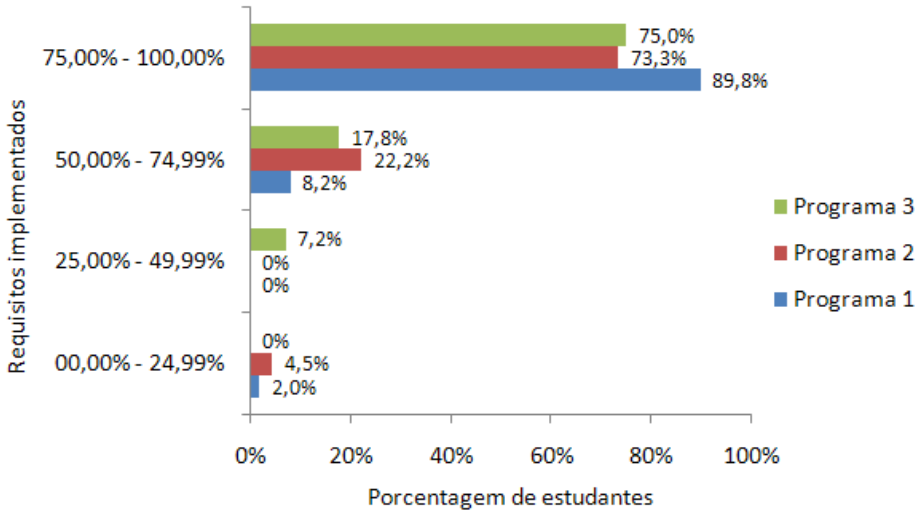

Figura 4.20: Requisitos implementados pelos estudantes.

Questão de Pesquisa 4: Quais as dificuldades apresentadas pelos estudantes ao praticarem as atividades de POP?

Para responder a esta questão de pesquisa, nós utilizamos as respostas dos estudantes para o questionário (Apêndice D) que foi aplicado após o estudo de caso. Dos 55 estudantes que participaram do estudo, apenas 38 responderam ao questionário.

Nós reportaremos as dificuldades dos estudantes com respeito a especificação dos requisitos, implementação do programa e criação de testes automáticos. Faremos também uma descrição das lições aprendidas e sugestões de melhorias, segundo relatos dos estudantes.

Dificuldades na especificação dos requisitos.

Com relação a especificação dos requisitos, os estudantes destacaram principalmente duas dificuldades: (i) questionar o cliente; (ii) interagir com o grupo. No que diz respeito a *questionar o cliente*, os estudantes, inicialmente, tiveram dificuldades em elaborar as perguntas que deveriam ser feitas ao cliente. Essa dificuldade é ilustrada pela resposta de um dos estudantes ao questionário, conforme apresentamos na Figura 4.21.

Outra dificuldade era o de *interagir com os demais membros do grupo*, mais especificamente, em manter todos os membros do grupo engajados na discussão e especificação dos requisitos. Mesmo notificando esta dificuldade, $55{,}3\%$ dos estudantes ao avaliarem as interações com seus respectivos grupos as consideraram como ótima ou boa, conforme apresentamos na Figura 4.22.

> "Não sabíamos o que perguntar exatamente, mas que no decorrer da entrevista com o cliente isso foi se desenvolvendo melhor ."

Figura 4.21: Dificuldade na elicitação dos requisitos.

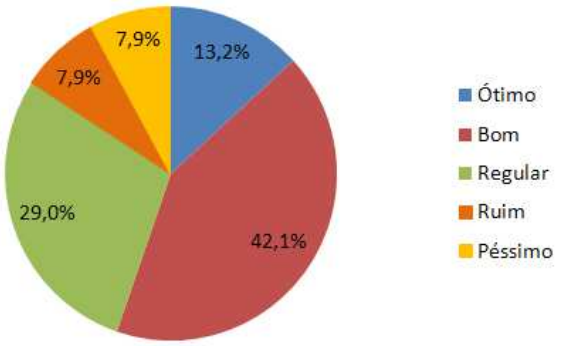

Figura 4.22: Avaliação das interações em grupo.

Dificuldades para implementar os programas.

Na Figura 4.23, nós apresentamos as opiniões dos estudantes quanto a complexidade para solucionar os problemas propostos. Os estudantes atribuíram complexidades gradativas aos programas, sendo que 92,3% dos estudantes consideraram o programa 1 fácil e 73,7% dos estudantes consideraram o programa 3 difícil.

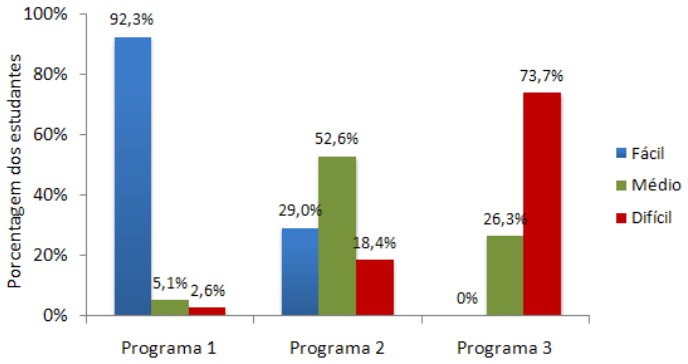

Figura 4.23: Complexidade para solucionar os problemas.

Na fase de implementação, os estudantes destacaram as seguintes dificuldades: organizar o código em funções e manipular *strings*. Além disso, os estudantes apontaram a necessidade de mais tempo para a fase de implementação, uma vez que nesse período eles tiveram avaliações de outras disci-

plinas.

Dificuldades em criar testes automáticos.

Ao se auto-avaliarem com relação a criação de testes automáticos, 76,3% dos estudantes acreditavam que seu desempenho foi considerado entre regular, ruim e péssimo, conforme demonstramos na Figura 4.24. Estes números são também evidências que justificam a baixa porcentagem de estudantes que entregaram testes automáticos, conforme apresentamos na Tabela 4.4.

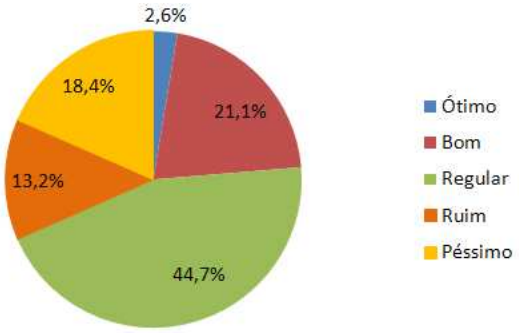

Figura 4.24: Auto-avaliação dos estudantes quanto a criação de testes automáticos.

Lições aprendidas.

Nós apresentamos na Figura 4.25, algumas respostas dos estudantes que descrevem as lições aprendidas na fase de especificação de requisitos. Além disso, os estudantes ressaltaram que a solução dos problemas propostos levaram-lhes a explorar a função `normalize`[3] de Python, aprimorar a manipulação de dicionários e a organização de programas utilizando funções.

Que coisas novas você aprendeu?
"Que não devo colocar minha opinião no projeto, não devo fazer o que eu achar melhor e sim fazer o que o cliente pediu."
"Percebi que o que o achamos ser o melhor para o cliente nem sempre é o que ele quer."
"Como escutar, dialogar, interagir com as propostas de trabalho, ter uma ideia de como seria no real do trabalho."

Figura 4.25: Lições aprendidas pelos estudantes.

[3]Esta função retorna cópia de uma string substituindo os caracteres acentuados por seus equivalentes não acentuados. Esta função também remove os caracteres gráficos não-ASCII.

Sugestões de melhorias.

Os estudantes apontaram como sugestões de melhoria aumentar o prazo para a entrega dos programas e melhorar o ensino para criação de testes automáticos.

Questão de Pesquisa 5: Quais as dificuldades apresentadas pelos clientes-tutores para a condução de POP em sala de aula?

Pelo fato dos clientes-tutores terem sido os mesmos que participaram do estudo de caso anterior, nós decidimos substituir o questionário por uma conversa direta com eles. A seguir apresentamos os aspectos principais.

Dificuldades observadas nos estudantes na fase de especificação dos programas.

Assim como no estudo de caso anterior, os clientes-tutores informaram que os estudantes tiveram dificuldades em envolver todos os participantes do grupo na discussão e elicitar os requisitos menos triviais.

Os clientes-tutores acreditam que, no contexto presencial, a interação entre os estudantes poderia ser maximizada se o laboratório tivesse um *layout* mais adequado para o trabalho em grupo.

Adequação do material de apoio fornecido aos clientes-tutores.

Os clientes-tutores consideraram o material adequado e não apontaram sugestões de melhoria.

Sugestões de melhoria.

Os cliente-tutores apontaram como sugestão de melhoria fortalecer a prática de testes automáticos na disciplina.

4.2.6 Discussão

Assim como no estudo anterior, os estudantes apresentaram dificuldades em especificar requisitos relacionados às mensagens de erros e às formatações de entrada e saída. Também nesse estudo, a especificação do programa para o jogo da forca obteve o maior percentual de defeitos. Entretanto, para todos os programas, a média de requisitos documentados no segundo estudo, foi maior que a média obtida pelos estudantes no semestre anterior. Observamos também que a estrutura e organização dos documentos foram similares às do documento construído em sala de aula, na atividade que antecedeu o estudo de caso.

O mesmo ocorreu com a média de requisitos representados pelos casos de testes de entrada/saída documentados pelos estudantes, que no segundo estudo foi superior à média obtida pelos estudantes no semestre anterior.

Nós acreditamos que a melhoria no desempenho ocorreu em virtude de termos realizado uma atividade prévia de especificação de requisitos. Além disso, o prazo pode também ter beneficiado a atuação dos estudantes, pois em relação ao estudo anterior, os estudantes tiveram um dia a mais para fechar a versão final do documento de especificação. Os estudantes também tiveram um dia a mais para entregar os programas e testes automáticos, o que pode ter colaborado para o maior número de submissões dos programas.

Em contrapartida, a porcentagem de estudantes que entregaram testes automáticos foi bem inferior à porcentagem obtida no estudo anterior. Novamente, verificamos as dificuldades dos estudantes em criar testes automáticos. Aliada a essa dificuldade, há o fato dos estudantes considerarem mais "prático" executarem os testes manualmente, pois eles têm poucos programas a serem testados, os quais também são relativamente pequenos e facilitam a execução de teste, detecção e correção de erros.

Uma limitação que percebemos é que os estudantes, individualmente, têm dificuldades em pensar em diferentes restrições e situações de erros que podem ocorrer no programa. No geral, essa situação foi minimizada pela interações em grupo.

Como descrito no estudo anterior, incentivar a prática de testes junto aos estudantes iniciantes não é uma tarefa simples e exige mudanças no escopo da disciplina de programação, tais como as mencionadas na Seção 5.1.6. Acreditamos também que atividades em grupo podem favorecer o melhor desempenho dos estudantes nesta atividade, como por exemplo, dividir a turma em grupos e cada grupo ser responsável pela criação de testes para um determinado programa.

Em síntese, consideramos que com relação à criação de testes automáticos os resultados de POP não foram conforme os almejados. O baixo desempenho na criação de testes automáticos não chegou a ser uma grave limitação para a metodologia, porque os estudantes exploraram a criação de testes de entrada/saída na fase de especificação dos requisitos. Esta atividade proporcionou aos estudantes: (i) elicitar requisitos; (ii) refletir sobre as restrições e possibilidades de erros nos programas, mesmo que de forma não exaustiva; e, (iii) garantir a qualidade de seus programas, ainda que por meio da execução manual dos testes.

4.3 Considerações Finais

Tomando como referência os estudos realizados com a aplicação de POP em sala de aula, nós faremos algumas considerações sobre aspectos pedagógicos e limitações de nosso estudo.

Aspectos Pedagógicos

Ao observarmos a melhoria dos estudantes no segundo estudo de caso, principalmente no que diz respeito a especificação dos requisitos, nós consideramos necessário preparar os estudantes antes de iniciar o ciclo de resolução de problemas de POP. Esta preparação provê os estudantes com um exemplo concreto de como interagir com um cliente e de como proceder na documentação dos requisitos, deixando-os mais seguros para a prática das atividades de POP.

No primeiro estudo, nós não procedemos desta forma. Em virtude disto, os estudantes tiveram que aprender fazendo. Em nosso entendimento, o fato dos estudantes não terem tido um exemplo de como proceder colaborou para as dificuldades na fase de interação com o cliente-tutor e especificação dos requisitos.

Ao elaborar e propor problemas mal definidos, é importante que os professores explorem também problemas nos quais os estudantes tenham algum conhecimento prévio do domínio. Isto permite que os estudantes aprendam a não se confiar apenas em seus modelos mentais sobre o problema e a não subestimar os requisitos do cliente-tutor. Nos dois estudos de caso que realizamos, o jogo da forca era o problema que os estudantes tinham bom conhecimento do domínio e, em contrapartida, foi o problema no qual os estudantes foram menos efetivos na especificação dos requisitos.

Vários estudantes, em ambos estudos de caso, consideraram que seria conveniente disponibilizar mais tempo para a resolução dos problemas. Estimar adequadamente o tempo é importante para não sobrecarregar os estudantes e, conseqüentemente, prejudicar a entrega dos artefatos. Entretanto, cabe notarmos que os estudantes, em geral, estão matriculados em outras disciplinas, cujos professores estipulam seus prazos e atividades de maneira independente. Assim, as requisições por maior prazo podem permanecer, independente de quão bem estimado foram os prazos na disciplina de programação.

No nosso primeiro estudo, uma lista de problemas de programação teve seu prazo de entrega adiado e isto pode ter comprometido a entrega dos programas. Entretanto, no segundo estudo de caso, nós acreditamos que as requisições por maior tempo ocorreram devido às atividades propostas nas outras disciplinas do curso.

Por fim, os dois estudos de caso realizados demonstraram a viabilidade da execução do ciclo de resolução de problemas de POP na disciplina introdutória de programação.

Limitações do Estudo

Em nossos estudos, nós pudemos observar a aplicação de POP com 125 estudantes iniciantes na resolução de três problemas de programação. Fizemos a análise das versões finais dos documentos de especificação, programas e testes produzidos pelos estudantes. Esta análise nos forneceu resul-

tados quantitativos e qualitativos sobre a produção dos estudantes durante o ciclo de resolução de problemas.

Entretanto, reconhecemos que mais dados deveriam ser apurados para dar uma perspectiva mais completa da evolução e dificuldades dos estudantes em cada atividade de POP. Por exemplo, a quantidade de requisitos que foram implementados no protótipo dos programas em comparação com a quantidade de requisitos que os estudantes evoluíram até a entrega da versão final; quantidade dos requisitos documentados corretamente, mas que não foram implementados na versão final do programa; ou, a quantidade de requisitos implementados corretamente, mas que não foram documentados.

O tempo necessário para apurar estas variáveis sugeridas extrapolaria o cronograma definido para a análise dos nossos estudos de caso. De todo modo, é importante evidenciarmos a complexidade de realizar a análise sugerida, principalmente quando levamos em consideração o número de estudantes e o número de problemas propostos.

Capítulo 5

Estudos sobre os Efeitos de POP em Estudantes Iniciantes de Programação

Nos estudos de caso, descritos no capítulo anterior, nós tivemos por objetivo observar POP no contexto de sala de aula e adquirir *feedback* dos participantes para melhorar a metodologia proposta. Além disso, era de nosso interesse observar os estudantes após a aplicação de POP. Neste caso, para avaliarmos os efeitos da metodologia sobre os estudantes iniciantes de programação no que diz respeito a sua capacidade de tratar com problemas mal definidos e especificação de requisitos. Para fazer esta avaliação, foi necessário observarmos individualmente os estudantes, comparando-os a estudantes de programação iniciantes que não adotaram POP.

Para isto, nós realizamos dois experimentos controlados com estudantes iniciantes de programação do Curso de Ciência da Computação de duas universidades: da Universidade Federal de Campina Grande (UFCG), cujos estudantes adotaram POP; e, de outra Universidade Federal do Nordeste, cujos estudantes não adotaram a metodologia proposta. Os experimentos foram realizados nos dois semestres acadêmicos de 2009, no final de cada período letivo.

Neste capítulo, nós apresentamos o planejamento, execução e descrição dos resultados destes experimentos. Para a realização deles, nós seguimos as orientações propostas por Wohlin et al. [2000], Kitchenham et al. [2002] e Jedlitschka and Pfahl [2005].

5.1 Experimento 1

5.1.1 Planejamento

O planejamento de nosso experimento inclui: a descrição dos objetivos e o objeto utilizado na pesquisa, os sujeitos que participaram do experimento, as métricas utilizadas para avaliação dos artefatos produzidos pelos estudantes, a definição da hipóteses, o tratamento aplicado à pesquisa e os procedimentos empregados para coleta de dados.

Objetivo da Pesquisa

No experimento, estudamos o impacto da adoção de POP sobre os estudantes de um curso inicial de programação. Nós avaliamos as seguintes métricas: quantidade de requisitos documentados, quantidade de requisitos documentados corretamente, número de versões produzidas do programa, média de requisitos atendidos por versão do programa, relevância dos questionamentos feitos e tempo total de resolução do problema. Os sujeitos de nosso experimento foram estudantes iniciantes de programação dos cursos de Ciência da Computação da UFCG e de outra Universidade Federal do Nordeste.

Objeto da Pesquisa

O objeto usado no experimento foi um problema mal definido sobre financiamento habitacional cujo enunciado nós apresentamos no Quadro 5.1.

Quadro 5.1: Problema do financiamento habitacional.

Enunciado
O Banco Imobiliário tem um plano de financiamento habitacional chamado "Lar doce Lar" que foi idealizado com o intuito de ajudar as pessoas a realizarem o sonho de adquirir a casa própria. Para colocá-lo em prática, o Banco deseja criar um simulador e deixá-lo disponível para a população. A idéia é que as pessoas que desejam adquirir crédito para a compra da casa própria possam simular os valores das parcelas do financiamento de forma rápida e fácil, bastando apenas informar o valor do imóvel que desejam adquirir e o número de parcelas em que pretendem pagar a dívida. As parcelas possuem taxa de juros fixas, assim todas as pessoas podem se beneficiar do financiamento habitacional. Manter valores de parcelas fixas é uma política do banco para evitar inadimplência dos solicitantes maiores de idade e tornar o financiamento acessível para a maioria da população. Suponha que você é o programador contratado pelo Banco Imobiliário e, portanto, deve entregar um programa que atenda às necessidades do referido Banco.

O enunciado do problema, intencionalmente, contém informação omitida, ambígua, contraditória,

incorreta e irrelevante, como por exemplo:

- **Informação omitida:** no enunciado do problema, não informamos, por exemplo, o valor da taxa de juros cobrada pelo banco, a forma como o financiamento e as políticas para evitar inadimplência são calculadas, nem detalhes sobre a formatação de entrada e saída dos dados;
- **Informação ambígua:** a frase "*(...) manter valores de parcelas fixas é uma política do banco para evitar inadimplência dos solicitantes maiores de idade* (...)", significa exatamente o quê? Que os menores de idade também podem fazer o financiamento e não estão sujeitos a estas políticas? ou, que apenas os maiores de idade podem fazer o financiamento?;
- **Informação contraditória:** a expressão "*(...) solicitantes maiores de idade (...)*" traz ao enunciado do problema uma contradição, pois anteriormente é dito que "*(...) todas as pessoas podem se beneficiar do financiamento habitacional (...)*". Neste caso, qual é a sentença verdadeira a de que todos podem ter acesso ao crédito, ou que apenas o crédito é fornecido aos maiores de idade?;
- **Informação incorreta:** a frase "*(...) bastando apenas informar o valor do imóvel que desejam adquirir e o número de parcelas que pretendem pagar a dívida (...)*" é incorreta, pois outras informações precisam ser coletadas, com por exemplo, idade e renda bruta dos solicitantes;
- **Informação irrelevante:** alguns trechos no enunciado do problema são irrelevantes, como por exemplo: "*(...) um plano de financiamento habitacional chamado"Lar doce Lar que foi idealizado com o intuito de ajudar as pessoas a realizarem o sonho de adquirir a casa própria (...)*". Do ponto de vista dos requisitos, essa sentença não provê os estudantes com informações relevantes. Nós a incluímos no enunciado para tornar o problema mais realístico e também para motivar os estudantes a selecionarem informações úteis.

Esses "defeitos" que nós, intencionalmente, incluímos no enunciado do problema tinham o propósito de incentivar os estudantes a dialogarem com o pesquisador[1] (que fazia o papel de cliente) e especificarem os requisitos do programa.

Para solucionar o problema mal-definido proposto, o programa dos estudantes deveria atender a 31 requisitos, conforme definidos no Quadro 5.2.

[1] Pessoa que conduzia o estudo experimental.

Quadro 5.2: Requisitos do programa para solucionar o problema de financiamento habitacional.

Entradas		17	Valor das parcelas = Valor final do imóvel/ número de parcelas
1	Renda Bruta	18	Valor final do imóvel = Valor do imóvel + (valor do imóvel * taxa de juros * número de parcelas)
2	Idade	**Políticas de Inadimplência**	
3	Valor do Imóvel	19	Valor do imóvel $\leq$ 25 $\times$ renda bruta
4	Numero de Parcelas	20	Valor das parcelas $\leq$ 25% da renda bruta
Formatação das Entradas		**Saídas**	
5	Renda Bruta?	21	Valor das parcelas
6	Idade?	22	Valor final do imóvel
7	Valor do Imovel?	Formatação das saídas	
8	Numero de Parcelas?	23	Valor das Parcelas: R\$ $< valor >$
Restrições das Entradas		24	$< valor >$ deve conter duas casas decimais
9	Renda Bruta $\geq$ R\$ 465,00	25	Valor Final do Imovel: R\$ $< valor >$
10	Idade $\geq$ 18	26	$< valor >$ deve conter duas casas decimais
11	Idade $\leq$ 57	**Mensagens de Erro para Entradas Inválidas**	
12	Valor do Imóvel $\geq$ R\$ 1500,00	27	Valor da renda fora do limite permitido. Digite outro valor. Renda Bruta?
13	Valor do Imóvel $\leq$ R\$ 220.000,00	28	Idade fora do limite permitido. Digite outro valor. Idade?
14	Numero de Parcelas $\geq$ 3	29	Valor do imovel fora do limite permitido. Digite outro valor. Valor do Imovel?
15	Numero de Parcelas $\leq$ 240	30	Numero de parcelas fora do limite permitido. Digite outro valor. Numero de Parcelas?
Cálculo do Financiamento		**Mensagem de Erro para o caso de Não Financiamento**	
16	Taxa de Juros = 0.5% a.m	31	Financiamento nao pode ser concedido.

Sujeitos

Os participantes de nosso experimento eram estudantes iniciantes de programação do curso de Ciência da Computação de duas universidades do nordeste brasileiro. O grupo experimental era composto por estudantes da Universidade Federal de Campina Grande (UFCG), que adotou POP; e o grupo de controle composto por estudantes de outra Universidade Federal do Nordeste, que não adotou POP.

Como o objetivo de coletarmos alguns dados pessoais dos estudantes, assim como caracterizarmos a *expertise* acadêmica deles, nós aplicamos um questionário no início do semestre acadêmico. Este questionário está descrito no Apêndice F e nos deu suporte na seleção da amostra. Na Seção 5.1.2, nós apresentamos as informações dos sujeitos selecionados para a execução do experimento.

Artefatos Avaliados e Critérios de Avaliação

Nós avaliamos os estudantes individualmente, levando em consideração os seguintes artefatos: *documento de especificação*, *programa* e *diálogos*. Para cada artefato as variáveis analisadas são descritas como segue.

Documento de Especificação

Assim como nos estudos de caso, no experimento nós também avaliamos o documento de especificação considerando as variáveis *requisitos documentados* (RD) e *requisitos documentados corretamente* (RC).

A inspeção dos documentos também foi realizada por dois inspetores que trabalharam juntos para garantir uma avaliação mais apurada. Para inspeção nós também adotamos a técnica de leitura baseada em defeitos (Defect-Based Reading – DBR) [Porter et al. 1995] e a versão simplificada da taxonomia de defeitos definida em [Shull et al. 2000] e apresentada no Quadro 4.4.

Para as variáveis RD e RC, nós consideramos apenas a última versão do documento de especificação por ela ser a versão mais completa. A métrica para essas variáveis é um valor numérico de 0 a 31 que corresponde aos requisitos estabelecidos no Quadro 5.2.

- **Requisitos Documentados (RD)**: correspondeu a quantidade de requisitos documentados pelo estudante.

- **Requisitos Documentados Corretamente (RC)**: correspondeu a quantidade de requisitos documentados sem defeitos pelo estudante.

Programa

Nós avaliamos o programa considerando duas variáveis: *número de versões do programa* (NV) e *requisitos atendidos* (RA).

- **Número de Versões do Programa (NV)**: correspondeu a quantidade de versões do programa construídos pelo estudante até que todos os requisitos do programa fossem atendidos ou a quantidade de versões do programa produzidos ao longo de três horas, tempo máximo que nós estabelecemos para a resolução do problema.

- **Requisitos Atendidos (RA)**: correspondeu a média aritmética de requisitos implementados por versão pelo estudante.

Considerando que a última versão do programa continha os requisitos atendidos nas versões anteriores, nós calculamos a média dividindo o número de requisitos atendidos na última versão do programa pelo número de versões do programa.

Diálogos

Para análise dos diálogos, nós estabelecemos a variável *relevância dos questionamentos* (RQ).

- **Relevância dos Questionamentos (RQ)**: correspondeu a porcentagem de questões sobre requisitos feitas pelo estudante.

Durante a resolução do problema foram comuns três tipos de questionamentos:

1. Sobre informações gerais: por exemplo, *o que é renda bruta?*, ou *o que é inadimplência?*. Esse tipo de questionamento, embora seja relevante para o estudante, não informa sobre os requisitos do programa;

2. Sobre implementação: por exemplo, *como eu faço para formatar a saída com duas casas decimais?*;

3. Sobre requisitos: neste caso, são perguntas que, de fato, tratam sobre requisitos. Por exemplo, *quais saídas são desejadas no programa?*; *como deve ser formatada essa saída?*; ou, *é necessário ler como entrada o nome do usuário?*.

Assim, o cálculo da variável *relevância dos questionamentos* foi realizado como segue:

$$\text{RQ} = \frac{\text{QQR}}{\text{QTQ}} \times 100\%$$

Na qual:

- QQR: Quantidade total de questionamentos sobre requisitos (considerando todas as versões dos programas);
- QTQ: Quantidade total de questionamentos (considerando todas as versões dos programas). Neste caso, o total de questionamento engloba questionamentos sobre informações gerais, implementação e requisitos.

Para demonstrarmos o cálculo da relevância dos questionamentos (RQ), tomemos como exemplo o diálogo do Quadro 5.3. Neste diálogo, o estudante faz cinco questionamentos, sendo que quatro deles tratam sobre requisitos (linhas 1 a 9 do diálogo) e um questionamento trata sobre informações gerais (linha 10 do diálogo). Neste caso, o cálculo de RQ é feito como segue:

$$\begin{aligned} \text{RQ} &= \frac{\text{QQR}}{\text{QTQ}} \times 100\% \\ &= \frac{4}{5} \times 100\% \\ &= 80\% \end{aligned}$$

Quadro 5.3: Exemplo de diálogo.

	Diálogo
1	*Estudante:* Além do valor do imóvel e do número de parcelas, é preciso o programa pedir
2	o nome da pessoa?
3	*Pesquisador:* Não.
4	*Estudante:* É preciso solicitar o endereço atual da pessoa?
5	*Pesquisador:* Não.
6	*Estudante:* É preciso ler o número de dependentes?
7	*Pesquisador:* Não.
8	*Estudante:* Que outras entradas são necessárias?
9	*Pesquisador:* Idade e renda bruta do solicitante.
10	*Estudante:* O que é renda bruta?
11	*Pesquisador:* É a renda mensal da pessoa sem incluir os abatimentos.

Além das variáveis anteriormente descritas, nós consideramos também o **Tempo de Resolução do Problema (TR)** que dizia respeito ao tempo que o estudante utilizou para resolver o problema proposto – especificar requisitos e implementar o programa, considerando todas as versões. O tempo foi contabilizado em minutos.

Definição das Hipóteses

Nesta seção, nós apresentamos a definição de nossas hipóteses. Há um conjunto de seis hipóteses nulas ($H0_{1..6}$), uma para cada variável definida no estudo. Nas hipóteses nulas, nós estabelecemos que não haveria diferença em adotar POP ou não adotar POP. Nós também definidos dois conjuntos de hipóteses alternativas ($H1_{1..6}$) e ($H2_{1..6}$), que são descritas como segue.

Hipóteses Nulas ($H0_{1..6}$): A quantidade de requisitos documentados (1), de requisitos documentados corretamente (2), a quantidade de versões do programa (3), a média de requisitos atendidos por versão do programa (4), a porcentagem de questionamentos relevantes (5), e o tempo usado na resolução do problema (6) adotando POP **não é diferente** de quando POP não é adotada.

$$
\begin{aligned}
H0_1 &: RD_{POP} = RD_{\text{não-POP}} \\
H0_2 &: RC_{POP} = RC_{\text{não-POP}} \\
H0_3 &: NV_{POP} = NV_{\text{não-POP}} \\
H0_4 &: RA_{POP} = RA_{\text{não-POP}} \\
H0_5 &: RQ_{POP} = RQ_{\text{não-POP}} \\
H0_6 &: TR_{POP} = TR_{\text{não-POP}}
\end{aligned}
$$

Hipóteses Alternativas ($H1_{1..6}$): A quantidade de requisitos documentados (1), de requisitos documentados corretamente (2), a quantidade de versões do programa (3), a média de requisitos atendidos por versão do programa (4), a porcentagem de questionamentos relevantes (5), e o tempo usando na resolução do problema (6) usando POP **é diferente** de quando POP não é adotada.

$$
\begin{aligned}
H1_1 &: RD_{POP} \neq RD_{\text{não-POP}} \\
H1_2 &: RC_{POP} \neq RC_{\text{não-POP}} \\
H1_3 &: NV_{POP} \neq NV_{\text{não-POP}} \\
H1_4 &: RA_{POP} \neq RA_{\text{não-POP}} \\
H1_5 &: RQ_{POP} \neq RQ_{\text{não-POP}} \\
H1_6 &: TR_{POP} \neq TR_{\text{não-POP}}
\end{aligned}
$$

Hipóteses Alternativas ($H2_{1..6}$): A quantidade de requisitos documentados (1), de requisitos documentados corretamente (2), a média de requisitos atendidos por versão do programa (4), a porcentagem de questionamentos relevantes (5) usando POP **é maior que** quando POP não é adotada; e, a quantidade de versões do programa (3) e o tempo usado na resolução do problema (6) usando POP **é menor que** quando POP não é adotada.

$$
\begin{aligned}
H2_1 &: RD_{POP} > RD_{\text{não-POP}} \\
H2_2 &: RC_{POP} > RC_{\text{não-POP}} \\
H2_3 &: NV_{POP} < NV_{\text{não-POP}} \\
H2_4 &: RA_{POP} > RA_{\text{não-POP}} \\
H2_5 &: RQ_{POP} > RQ_{\text{não-POP}} \\
H2_6 &: TR_{POP} < TR_{\text{não-POP}}
\end{aligned}
$$

Variáveis Independentes

Variáveis independentes são aquelas que podemos manipular e controlar em um experimento [Wohlin et al. 2000]. No nosso caso, as variáveis independentes são:

- Problema mal definido, cujo enunciado estava descrito textualmente;
- Implementação do programa usando a linguagem Python (grupo experimental[2]) e a linguagem C (grupo de controle[3]);

[2]Estudantes que adotaram POP.

[3]Estudantes que não adotaram POP.

- Características pessoais e acadêmicas dos estudantes;
- Metodologia de ensino.

Variáveis Dependentes

Variáveis dependentes são aquelas que desejamos observar o efeito em virtude da mudança de um ou mais fatores [Wohlin et al. 2000]. Em nosso experimento, queremos observar as seguintes variáveis:

- Requisitos documentados;
- Requisitos documentados corretamente;
- Número de versões do programa;
- Requisitos atendidos;
- Relevância dos questionamentos;
- Tempo de resolução do problema.

Tratamento

O tratamento aplicado em nosso experimento foi a metodologia de Programação Orientada ao Problema (POP). A ausência do tratamento nós chamamos de *não-POP*. Assim, nosso estudo experimental tem um fator com um único tratamento.

Dados Coletados

Neste estudo, nós coletamos os seguintes dados: (i) todas as versões do documento de especificação; (ii) código fonte de todas as versões do programa; (iii) áudio dos diálogos entre os estudantes e o pesquisador; (iv) anotações sobre o comportamento dos estudantes e de suas atividades durante a resolução do problema; e (v) respostas dos estudantes a entrevista realizada após o experimento.

5.1.2 Execução

Amostra

O questionário (Apêndice F) que nós aplicamos no início do semestre acadêmico para coletarmos alguns dados pessoais e acadêmicos dos estudantes, foi respondido por 56 estudantes da UFCG e por

40 estudantes da outra universidade. Os dados coletados nos auxiliaram no processo de seleção da amostra.

Nós excluímos da amostra os estudantes repetentes e aqueles que, próximo a execução do experimento, tinham desistido da disciplina de programação ou já estavam reprovados. Os demais estudantes, de ambas as universidades, nós dividimos em grupos e cada grupo foi classificado de acordo com certas características, como por exemplo:

- Grupo 1: Estudantes de 17 a 20 anos de idade, sem qualquer conhecimento prévio em programação, que cursaram ensino médio em escola pública, que não tinham concluído ou estavam cursando curso técnico ou de graduação;
- Grupo 2: Estudantes de 17 a 20 anos de idade, sem qualquer conhecimento prévio em programação, que cursaram ensino médio em escola privada, que não tinham concluído ou estavam cursando curso técnico ou de graduação;
- Grupo 3: Estudantes de 17 a 20 anos de idade, com algum conhecimento prévio em programação, que cursaram ensino médio em escola privada, que não tinham concluído ou estavam cursando curso técnico ou de graduação.

Para a seleção das amostras, nós procedemos da seguinte forma: respeitando a divisão dos grupos, nós selecionamos os participantes aleatoriamente sem reposição, isto é, se um estudante era selecionado e não desejava participar do estudo, este estudante era excluído e uma nova seleção era realizada. Se um estudante de uma universidade era selecionado de um grupo, outro estudante da outra universidade era selecionado do grupo com as mesmas características. Na seleção, nós priorizamos grupos compostos por estudantes sem conhecimento prévio em programação. O objetivo desse processo de seleção era formar amostras tão homogêneas quanto possível.

Como resultado, nós selecionamos duas amostras independentes com dez estudantes cada. As principais características dos estudantes em cada grupo (experimental e de controle), nós apresentamos no Quadro 5.4.

Procedimentos para Execução

Agendamento. Depois de selecionarmos a amostra e obtermos dos estudantes aceitação de participação no experimento, nós estabelecemos uma agenda de execução da pesquisa respeitando a conveniência e disponibilidade de horário de cada estudante. O agendamento de horário foi feito individualmente, pois o pesquisador precisava interagir com um estudante por vez. Nós realizamos o

Quadro 5.4: Características das amostras.

Características	Grupo Experimental	Grupo de Controle
Média de idade	18 anos	18 anos
Gênero	9 homens e 1 mulher	6 homens e 4 mulheres
Ensino médio	Todos de escola particular	Todos de escola particular
Computador e Internet em casa	Todos possuíam	Todos possuíam
Experiência prévia com programação	8 sem experiência; 1 com experiência em Pascal (4 meses) e 1 com experiência em C++ e Visual basic (1 ano)	7 sem experiência; 2 com experiência em Pascal (2 e 4 meses, respectivamente); 1 com experiência em Pascal, Java, PHP/JavaScript (1 ano)
Fazendo algum curso técnico	9 não; 1 – Curso de Montagem e Manutenção de Computador (Completo)	9 não; 1 – Curso de Redes de Computadores (Primeiro semestre – cursando)
Fazendo outro curso de graduação	9 não; 1 – Licenciatura em Informática (Primeiro semestre – Trancado)	9 não; 1 – Tecnologia em Sistemas para Internet (Primeiro semestre – cursando)

experimento no final do primeiro semestre acadêmico.

Recursos. Nós executamos o experimento em uma sala reservada, nas dependências de cada universidade. Cada estudante foi provido com um *notebook* contendo todos os recursos de *software* necessários a resolução do problema proposto. Os estudantes do grupo experimental utilizaram a linguagem de Programação Python versão 2.5 e o IDLE (*Python´s Integrated Development Environment*) versão 1.2. Os alunos do grupo de controle utilizaram a linguagem de programação C e o ambiente Dev-C++ versão 4.9. Em ambos os casos, os estudantes utilizaram a linguagem aprendida em seus respectivos cursos de programação. A adoção de diferentes linguagens de programação em nosso estudo é também discutida na Seção 5.1.4 que apresenta uma avaliação da validade da pesquisa.

Termo de confidencialidade. Cada estudante assinou dois termos – um de confidencialidade e outro autorizando o uso dos dados para esta pesquisa. Em contrapartida, nós nos responsabilizamos por preservar a identidade dos estudantes.

Sessão Experimental. Em nosso estudo, cada estudante realizou duas atividades: (i) tratar com um problema mal definido, tendo um tempo máximo de 3 horas; e, (ii) responder a uma entrevista não estruturada. A realização dessas duas atividades por cada um dos estudantes, nós denominamos de

sessão experimental.

Antes de iniciar cada sessão experimental, o pesquisador explicava ao estudante os procedimentos a serem adotados:

- O pesquisador faria o papel de cliente;
- O tempo usado para resolução do problema não deveria exceder 3 horas, contados a partir do momento que o enunciado do problema era entregue ao estudante;
- Perguntas poderiam ser realizadas a qualquer momento;
- O registro dos requisitos do programa deveria ser feito em um arquivo digital (documento de especificação);
- A linguagem e o ambiente de programação seriam os mesmos adotados pelo estudante em sua respectiva disciplina de programação;
- A documentação da linguagem de programação ou seu *help* poderiam ser consultados a qualquer momento;
- Todas as submissões e versões do programa e do documento de especificação seriam contabilizados;
- O estudante deveria submeter o programa e o documento de especificação, quando considerasse que o problema tinha sido resolvido;
- Uma vez que o programa fosse submetido, o pesquisador executaria testes sobre ele. No caso dos requisitos não serem completamente atendidos e o tempo limite não tivesse expirado, o estudante poderia retomar o processo de resolução do problema.

Uma vez que esses procedimentos eram explicados, o estudante iniciava a primeira atividade da sessão experimental – tratar com um problema mal definido. Essa atividade era composta de um conjunto de sub-atividades que exigiam a interação do estudante com o pesquisador e que nós denominamos de *protocolo de interação*. Nós utilizamos este protocolo como um padrão nas interações com os estudantes. O protocolo de interação é apresentado na Figura 5.1.

Como podemos observar na Figura 5.1, a interação da sessão experimental inicia quando o pesquisador *apresenta o problema* ao estudante, cujo enunciado estava descrito textualmente. Ao receber o enunciado do problema, o estudante era instruído a *ler o enunciado* antes de *trabalhar na*

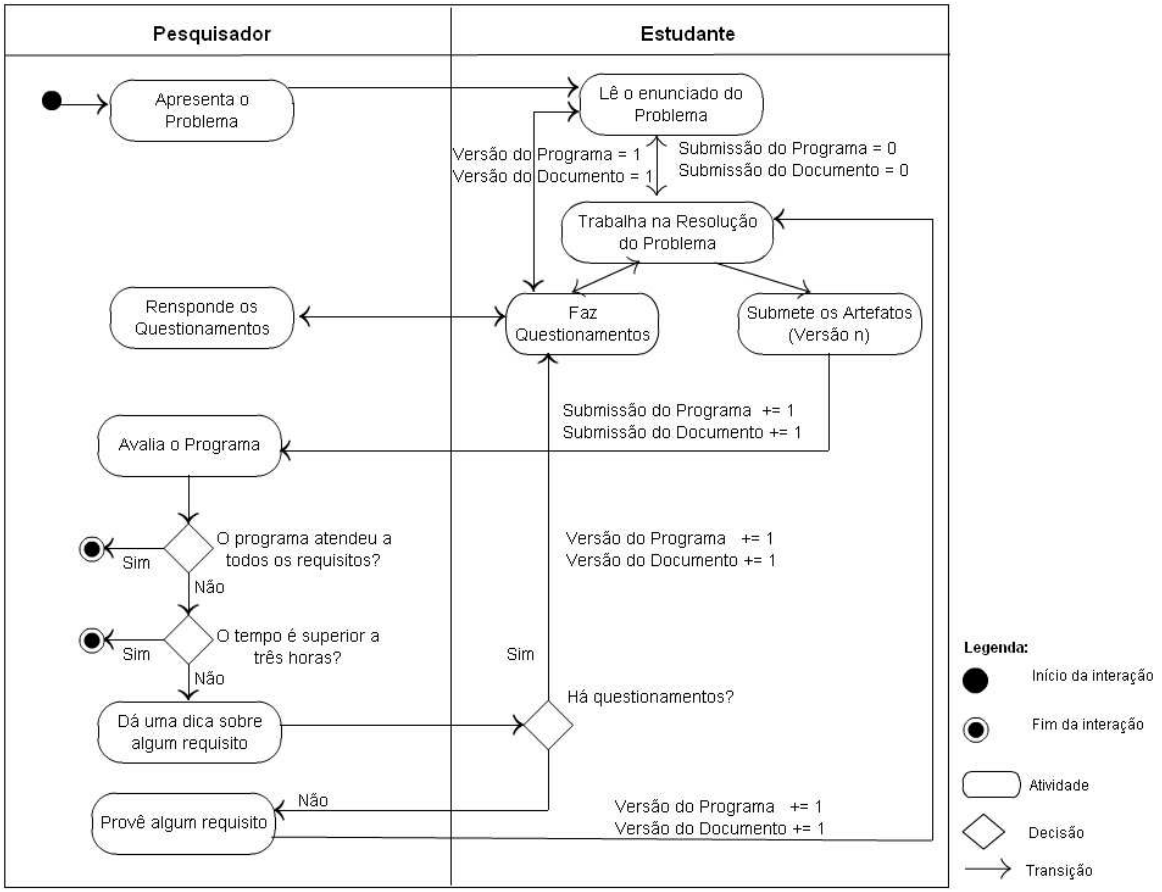

Figura 5.1: Protocolo de Interação.

resolução do problema. Como descrevemos anteriormente, em caso de dúvidas, o estudante poderia *fazer questionamentos* a qualquer momento.

Quando o estudante considerasse o problema resolvido, ele deveria *submeter os artefatos* (documento de especificação e programa) ao pesquisador e a submissão era contabilizada.

Uma vez que os artefatos eram submetidos, o pesquisador deveria *avaliar o programa*, isto é, averiguar se o programa atendia a todos os requisitos dentro do tempo pré-determinado. A interação na primeira atividade da sessão experimental *tratar com um problema mal definido* era finalizada em virtude da ocorrência de duas situações: (i) o programa atendia a todos os requisitos dentro do tempo estipulado (no máximo 3 horas); e, (ii) o programa não atende a todos os requisitos e o tempo não permite que a interação seja continuada.

Caso essas duas situações não ocorram, o pesquisador deve *dar uma dica sobre algum requisito* não atendido pelo programa. O propósito desta *dica* é incentivar o estudante a continuar especificando o programa. Se, com base nessa dica, o estudante *fizer questionamentos*, o pesquisador deve *responder ao questionamento*, e o estudante deve voltar a *trabalhar na resolução do problema*.

Entretanto, se mesmo depois de receber uma dica, o estudante não faz qualquer questionamento, então o pesquisador deve *prover algum requisito* ao estudante. Isso permite que os estudantes menos pró-ativos tenha a oportunidade de avançar na resolução do problema.

Sempre que o pesquisador avalia o programa e o estudante volta a trabalhar na resolução do problema, uma versão do documento de especificação e do programa é contabilizada.

A fim de ilustrarmos as atividades *dar um dica de algum requisito*, *prover algum requisito* e *trabalhar na resolução do problema*, nós apresentamos no Quadro 5.5, um fragmento do diálogo entre a estudante Caroline[4] (estudante não-POP) e o pesquisador. Como podemos observar nas linhas 1 e 2 do diálogo, o pesquisar fornece uma dica de um requisito referente a entrada. A estudante não questiona e o pesquisar precisa dizer, explicitamente, uma das entradas requeridas (linha 4 do diálogo). Somente após a descoberta do requisito é que a estudante questiona o pesquisador (linha 5). Neste momento, foi iniciada a segunda versão do programa e do documento de especificação. Na primeira versão, o programa da estudante tinha atendido apenas a cinco requisitos (3, 4, 16, 18 e 22, conforme Quadro 5.2).

Quadro 5.5: Diálogo entre estudante não-POP e pesquisador.

	Diálogo
1	*Pesquisador:* O programa só está pedindo na entrada o valor do imóvel e o número de
2	parcelas. Eu queria que tivesse outras entradas.
3	*Estudante:* (*não questiona, apenas olha para o pesquisador*).
4	*Pesquisador:* Por exemplo, eu quero saber a idade da pessoa.
5	*Estudante:* Certo ... só a idade ou mais alguma coisa?
6	*Pesquisador:* Eu quero a idade e a renda bruta.

Quando a interação para a atividade *tratar com um problema mal definido* é finalizada, dá-se início a segunda atividade da sessão experimental com a realização de uma *entrevista não estruturada*. Após a entrevista, a sessão experimental é finalizada.

Coleta de dados. Os dados foram coletados durante cada sessão experimental.

Desvios

Em virtude das restrições de horário de alguns estudantes do grupo de controle (não-POP), nós tivemos a necessidade de contar com a participação de um segundo pesquisador, para que a sessão expe-

[4]Nome fictício.

rimental pudesse ser realizada individualmente, nos casos em que não era possível agendar horários diferentes para cada estudante.

O pesquisador escolhido conhecia POP e participou como cliente-tutor nos estudos de caso realizados na UFCG. Para a realização do experimento, nós esclarecemos previamente os procedimentos a serem executados em cada sessão experimental, assim como instalamos todo o *software* necessário no *notebook* do pesquisador.

Ainda com relação ao grupo de controle, a maioria dos estudantes tinha resistência a tomar nota dos requisitos, especialmente, no arquivo digital que foi disponibilizado. Assim, para contabilizarmos os requisitos documentados e os documentados corretamente, consideramos também as anotações feitas em papel. Nós fizemos essa concessão apenas para o grupo não-POP.

Embora o pesquisador tenha se apresentado ao estudante como um cliente, ele teve que responder a perguntas de implementação (conforme descrevemos na Seção 5.1.1) e, algumas vezes, ajudar os estudantes na compreensão das mensagens de erros emitidas na fase de compilação do programa. Assim, no caso em que o estudante fizesse três tentativas sem sucesso para se recuperar do erro, o pesquisador interferia, ajudando-o a prosseguir na implementação do programa. Nós retomaremos esses aspectos nas Seções 5.1.4 (Avaliação da Validade) e 5.1.6 (Discussão).

5.1.3 Análise

Nós comparamos os dados coletados do grupo experimental e do grupo de controle a fim de observar se as hipóteses nulas poderiam ser rejeitadas. Para análise dos dados, nós utilizamos testes estatísticos paramétricos e não paramétricos. A utilização de ambos os tipos de testes foi necessário porque nem todos os dados apresentavam distribuição normal como é requerido para realização de testes paramétricos.

Para analisar as variáveis *requisitos documentados corretamente*, *relevância dos questionamentos* e *tempo de resolução do problema*, nós adotamos o teste paramétrico *t de Student* (*Student's t-test*), e para análise das demais variáveis nós adotamos o teste não paramétrico *Mann-Whitney* [Jain 1991; Siegel and Jr. 1988]. O teste Mann-Whitney é uma alternativa não paramétrica para o teste t de Student. Nós adotamos um nível de significância (α) de 0.05 que corresponde a um nível de confiança de 95%.

5.1.4 Avaliação da Validade

Nesta seção, nós discutimos quão válidos são os resultados pelo tratamento de quatro tipos de validade: *interna*, de *conclusão*, de *construção* e *externa*.

Validade Interna

Os sujeitos que participaram de nosso estudo eram estudantes iniciantes de programação do curso de Ciência da Computação que foram selecionados randomicamente. No processo de seleção, nós adotamos procedimentos a fim de formar amostras tão homogêneas quanto possível.

Embora o ideal fosse excluir da amostra estudantes com algum conhecimento prévio de programação, isto, na prática, foi difícil de realizar em virtude do pequeno número de estudantes dispostos a participar voluntariamente do estudo. Entretanto, o fato de estudantes com experiência prévia em programação terem sido selecionados para a pesquisa não representou uma ameaça a validade, pois a seleção de estudantes com essa característica ocorreu em ambos os grupos.

Ainda que os estudantes do grupo experimental (POP) e do grupo de controle (não-POP) tenham vindo de universidades diferentes, eles tinham em comum o fato de estarem matriculados em um curso de Ciência da Computação e, em geral, as disciplinas de programação para iniciantes possuem ementas similares, podendo variar a metodologia de ensino e a linguagem de programação (os estudantes POP adotaram Python e os estudantes não-POP adotaram C).

Nós acreditamos que a linguagem de programação teve pouca influência nos resultados, pois em nosso experimento os estudantes usaram a linguagem de programação que eles adotavam em seus respectivos cursos, evitando o esforço de aprender uma outra linguagem de programação. Além disso, embora a solução do problema seja expressa de diferentes formas em ambas as linguagens de programação, é importante notarmos que a solução requeria somente o uso de comandos básicos da linguagem, tais como, declaração de variáveis, entradas, saídas, execução de condicionais e iteração. Estes comandos são ensinados no início dos cursos de programação. Como o estudo experimental ocorreu somente no final do semestre acadêmico, os estudantes já estavam familiarizados com o uso desses comandos.

Com respeito a metodologia de ensino, nós ressaltamos que o experimento tinha por objetivo caracterizar o impacto de POP sobre os estudantes. Desta forma, o fator metodologia de ensino teve um valor particular para o grupo experimental, isto é, POP (tratamento) e outro valor para o grupo de controle, que nós denominamos de não-POP.

Com relação ao problema mal definido utilizado no estudo, nós levamos em consideração a apli-

cação dos conceitos básicos de programação, os quais já teriam sido aprendido pelos estudantes dos dois grupos no momento da execução do experimento. Além disso, os requisitos que deveriam ser atendidos pelo programa eram os mesmos para os dois grupos. Esses cuidados asseguravam que não haveria diferentes níveis de complexidade, nem a necessidade de aprendizagem adicional por parte dos estudantes de ambos os grupos.

Cabe ressaltarmos ainda que, para solucionar o problema proposto, os estudantes eram obrigados a interagir com o pesquisador a fim esclarecer os requisitos do programa. Com isso eliminamos a possibilidade de algum estudante solucionar o problema baseando-se apenas em seu conhecimento prévio.

Em nosso estudo, o pesquisador atuou como *observador* e *participante*. Para evitar qualquer tipo de interferência por parte do pesquisador, nós tomamos as seguintes precauções: (i) cumprir o protocolo de interação (Figura 5.1) a fim de garantir que todos os estudantes teriam o mesmo tratamento; (ii) não fornecer requisitos deliberadamente aos estudantes, garantindo que a aquisição dos requisitos fosse fruto do esforço do estudante para entender o problema e especificar o programa; e, (iii) entender a semântica das perguntas dos estudantes sem esperar perguntas em formatos pré-definidos ou com certo grau de formalidade. Assim, garantimos que as diferentes formas de expressão dos estudantes, desde que relacionadas ao problema, seriam respeitadas.

Em nosso estudo, nós consideramos que o único fator significativo de mudança era a adoção ou não de POP, permitindo o estabelecimento de relacionamentos causais.

Validade de Conclusão

Nosso planejamento experimental tentou garantir a correta execução do experimento, além da coleta e análise dos dados, pois deles dependiam a validade de conclusão do estudo. Para análise dos resultados, nós utilizamos o teste *t de Student* nos casos em que os dados apresentavam distribuição normal. Para testar se as amostras de dados vinham de uma população com uma distribuição normal, nós usamos o teste *Anderson-Darling*. Para análise da variância, nós utilizamos o teste de *Levene* [NIST/SEMATECH 2003]. O teste *Mann-Whitney*, um dos mais poderosos testes não paramétricos, foi utilizado nos casos em que os dados não apresentavam uma distribuição normal, pré-requisito do teste *t de Student*. Com isso garantimos a correta escolha e aplicação dos testes estatísticos.

Validade de Construção

Os estudantes pertencentes ao grupo experimental aprenderam a realizar as atividades de POP (tratamento) durante a disciplina de programação. Portanto, na execução do experimento, os estudantes de

ambos os grupos foram submetidos aos mesmos procedimentos e tinham conhecimento de como os dados seriam coletados.

Considerando que nós utilizamos várias fontes de dados (documento de especificação, código fonte dos programas, anotações, entrevistas e registros dos diálogo), nós aumentamos a confiança nos dados obtidos e minimizamos a possibilidade de opiniões subjetivas interferirem no estudo.

A concessão feita apenas ao grupo de controle, no que diz respeito a análise de anotações em papel ao invés de no arquivo digital, nos permitiu aproveitar, ao máximo, a produção dos estudantes não-POP.

Ainda no que diz respeito aos documentos de especificação, estes foram analisados por duas pessoas, levando em consideração que os estudantes não eram especificadores profissionais.

Validade Externa

Em nosso experimento, tivemos dificuldades em obter amostras com tamanhos significativos. Isto ocorreu por vários motivos: indisponibilidade dos participantes, tempo limitado para execução do estudo (principalmente quando se considera observações individuais), recursos financeiros limitados, entre outros. Assim, os resultados obtidos com nossas amostras não podem ser generalizados.

Embora os resultados sejam limitados, nós acreditamos que o plano e execução do experimento seja uma considerável contribuição que poderá ajudar na repetição do estudo, consolidando, no futuro, resultados mais gerais.

5.1.5 Resultados

Na Tabela 5.1, nós apresentamos os resultados obtidos para cada variável, considerando os dois grupos, POP e não-POP, respectivamente.

Como mencionamos na Seção 5.1.3, nós usamos o teste *t de Student* para analisar as hipóteses 2, 5 e 6, e o teste *Mann-Whitney* para analisar as demais hipóteses. Na Tabela 5.2, nós apresentamos o p-valor para o teste das hipóteses nulas, considerando a análise bilateral. Quando o p-valor é menor ou igual a α[5] (0,05), nós rejeitamos $H0$ com 95% de confiança; isto corresponde ao *status rejeitar* na Tabela 5.2. Por outro lado, se o p-valor é maior que α, então os dados não apresentam evidências suficientes para rejeitar $H0$, resultando no *status aceitar*, na Tabela 5.2.

[5]Nível de significância adotado em nosso estudo.

Tabela 5.1: Resultados obtidos no grupo POP e não-POP.

POP							**Não-POP**						
Nome	**RD**	**RC**	**NV**	**RA**	**RQ**	**TR**	**Nome**	**RD**	**RC**	**NV**	**RA**	**RQ**	**TR**
André	31	27	1	31	98,1	56	Camila	2	2	7	4,4	92,3	51
Antônio	29	29	2	15,5	97,2	90	Carlos	7	4	7	4,4	92,9	119
Amanda	31	25	1	31	94,3	97	Cássio	3	3	5	6,2	92,4	130
Arnaldo	22	16	1	31	95,6	72	César	2	0	5	6,2	95,0	41
Anderson	30	28	1	31	97,8	71	Cristiano	3	1	5	6,2	98,0	50
Augusto	31	25	2	15,5	95,5	63	Cristovão	13	11	6	3,7	90,9	177
Ayrton	30	30	2	15,5	98,3	86	Caroline	10	8	5	6,2	98,3	85
Andrey	31	18	2	15,5	95,0	54	Claúdio	1	1	6	5,2	93,2	79
Alberto	30	23	2	15,5	97,5	78	Carol	8	8	4	7,8	90,2	71
Alfredo	31	27	2	15,5	97,2	68	Cristina	3	1	6	5,2	98,2	73
Média	29,6	24,8	1,6	21,7	96,7	73,5	**Média**	5,2	3,9	5,6	5,6	94,1	87,6
DP	2,8	4,6	0,5	8,0	1,4	14,3	**DP**	4,1	3,8	1,0	1,2	3,1	42,5

Legenda:

Nome	Nome fictício dos estudantes.	RA	Média de requisitos atendidos por versão.
RD	Quantidade de requisitos documentados.	RQ	Porcentagem de questionamentos relevantes.
RC	Quantidade de requisitos documentados corretamente.	TR	Tempo, em minutos, gasto na resolução do problema.
NV	Quantidade de versões do programa.	Média	Média aritmética.
		DP	Desvio padrão.

Como podemos observar na Tabela 5.2, somente o tempo usado na resolução do problema pelos estudantes POP não foi significativamente diferente do tempo empregado pelos estudantes não-POP. Nesse caso, em que a hipótese nula não pôde ser rejeitada, nós calculamos o intervalo de confiança com nível de significância de $0{,}05$ a fim de possibilitar uma análise mais efetiva. O intervalo de confiança obtido foi de $(-45{,}317; 17{,}117)$. Dado que o zero está incluído neste intervalo, nós confirmamos que não há diferença no tempo de resolução do problema, considerando os dois grupos. Por ser um intervalo amplo, isto indica que o teste de hipótese para o tempo usado na resolução de problema (TR) foi estimado com um baixo grau de precisão.

Na Tabela 5.3, nós apresentamos o p-valor para o teste das hipóteses alternativas. No caso das

Tabela 5.2: p-valor para os teste das hipóteses nulas.

Hipóteses	$H0_1$	$H0_2$	$H0_3$	$H0_4$	$H0_5$	$H0_6$
p-valor	0,001	0,001	0,001	0,001	0,035	0,342
status	*rejeitar*	*rejeitar*	*rejeitar*	*rejeitar*	*rejeitar*	*aceitar*

hipóteses ($H1_{1..6}$), apresentamos o p-valor considerando a análise bilateral e no caso das hipóteses ($H2_{1..6}$), considerando a análise unilateral.

Tabela 5.3: p-valor para o teste das hipóteses alternativas.

Hipóteses	$H1_1$	$H1_2$	$H1_3$	$H1_4$	$H1_5$	$H1_6$
p-valor	0,001	0,001	0,001	0,001	0,035	0,342
status	*aceitar*	*aceitar*	*aceitar*	*aceitar*	*aceitar*	*rejeitar*
Hipóteses	$H2_1$	$H2_2$	$H2_3$	$H2_4$	$H2_5$	$H2_6$
p-valor	0,0005	0,0005	0,0005	0,0005	0,0175	0,171
status	*aceitar*	*aceitar*	*aceitar*	*aceitar*	*aceitar*	*rejeitar*

5.1.6 Discussão

Nós faremos uma análise descritiva e qualitativa dos resultados obtidos no estudo. A análise será organizada em tópicos, conforme descrevemos a seguir.

Primeira versão dos programas. A primeira versão dos programas revela o impacto dos estudantes ao lidar com problemas mal definidos. Nesta versão podemos avaliar a primeira percepção dos alunos sobre os "defeitos" do problema.

Na Tabela 5.4, nós apresentamos a média de três variáveis observadas no grupo POP e não-POP, considerando apenas a primeira versão do programa. Essas variáveis são: (i) tempo, em minutos, para iniciar a codificação; (ii) questionamentos feitos pelos estudantes; e (iii) requisitos atendidos pelo programa.

Tabela 5.4: Média de dados da primeira versão do programa.

Variáveis	**POP**	**Não-POP**
Tempo, em minutos, para iniciar a codificação	22	5
Questionamentos	30	4
Requisitos atendidos	92,5%	23,2%

Como podemos observar na Tabela 5.4, os estudantes POP, em média, levaram 22 minutos para iniciar a codificação, fizeram 30 questionamentos e atenderam a 92,5% dos requisitos. Os estudantes não-POP, por sua vez, em média, levaram 5 minutos para iniciar a codificação do programa, fizeram 4 questionamentos e atenderam a 23,2% dos requisitos.

No caso dos estudantes não-POP esses dados revelam: ansiedade para iniciar a implementação, como é comum em alunos iniciantes; pouca percepção dos "defeitos" presentes no enunciado do

problema; e, baixa capacidade de relacionar o problema proposto com situações do mundo real. Tais fatos colaboraram para que os estudantes não-POP fossem menos efetivos na elicitação dos requisitos e, conseqüentemente, tivessem um desempenho menor que os estudantes POP.

Relevância dos questionamentos *versus* Cobertura dos requisitos. Mesmo após a primeira versão do programa, os estudantes não-POP tiveram dificuldades para elicitar os requisitos do programa. Como podemos verificar na Tabela 5.1 (Colunas NV e RA), o grupo não-POP precisou em média de $5{,}6$ versões do programa para atender a todos os requisitos; e, implementou, em média, $5{,}6$ requisitos por versão. O grupo POP, por sua vez, precisou, em média, de $1{,}6$ versões do programa para atender a todos os requisitos; e, implementou, em média, $21{,}7$ requisitos por versão.

Embora o desvio padrão para a variável RA do grupo POP seja um valor alto (8), sugerindo que houve uma grande dispersão dos dados em relação ao grupo não-POP, uma observação mais atenta aos dados da coluna NV (Tabela 5.1) revela que os estudantes POP concluíram o programa em uma ou duas versões, tendo, portanto, um comportamento homogêneo. Essa homogeneidade é mascarada para a variável RA, em virtude do desvio padrão medir a dispersão em relação a média dos requisitos atendidos.

No que diz respeito a relevância dos questionamentos (Coluna RQ da Tabela 5.1), a média foi de $96{,}7\%$ para o grupo POP e de $94{,}1\%$ para o grupo não-POP, revelando que os estudantes de ambos os grupos concentraram-se em questionamentos que pudessem esclarecer requisitos. No entanto, se considerarmos a cobertura de requisitos ou, dito de outra forma, a descoberta de requisitos por versão, notaremos que o grupo não-POP, embora fazendo questionamentos relevantes, seus questionamentos revelaram menos requisitos por versão se comparado ao grupo POP. Para ilustrar essa afirmação, apresentamos na Figura 5.2 um gráfico *scatterplot* relacionando as variáveis questionamentos relevantes e número de versões do programa necessárias para atender a todos os requisitos.

Na Figura 5.2, vamos considerar, por exemplo, o caso dos estudantes não-POP que tiveram uma porcentagem de questionamentos relevantes maior ou igual a 97%. Apesar da alta porcentagem de questionamentos relevantes, eles necessitaram construir 5 ou mais versões do programa para atender a todos os requisitos. Por outro lado, os estudantes POP, com a mesma porcentagem de relevância dos questionamentos, precisaram de no máximo 2 versões do programa para atender a todos os requisitos.

Considerando que havia padrões nos requisitos (por exemplo, padrões para as mensagens de entrada e saída e para as mensagens de erro, conforme Quadro 5.2), a pouca cobertura de requisitos por versão, no grupo não-POP, revela também a dificuldade dos estudantes em abstrair novas informações a partir de situações prévias. Isto demonstra a necessidade de melhorar o nível de abstração dos estudantes iniciantes. Pesquisas desenvolvidas por Hazzan [2008] dão enfoque a esse assunto e

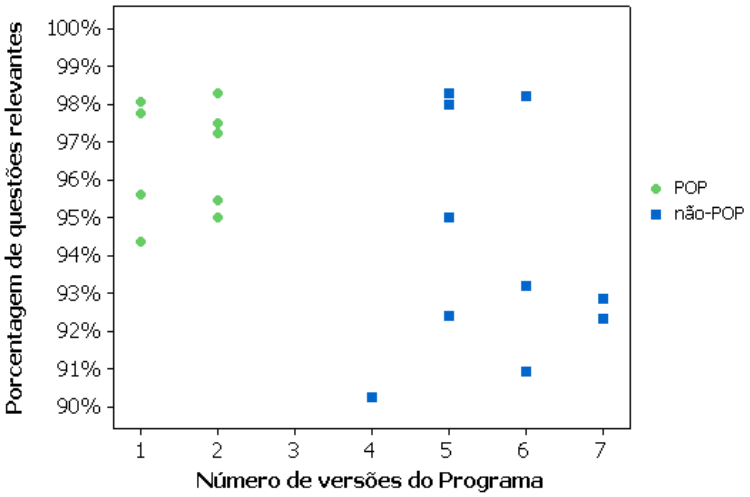

Figura 5.2: Relevância dos questionamentos *vs.* número de versões do programa.

confirmam nossa afirmação.

Documentação dos Requisitos e Tipos de Defeitos. No que diz respeito à documentação dos requisitos, a média de requisitos documentados e de requisitos documentados corretamente pelos estudantes POP (29,6% e 24,8%, respectivamente) foi bastante superior a dos estudantes não POP (5,2% e 3,9%, respectivamente), conforme apresentado na Tabela 5.1. Na Figura 5.3, por meio do gráfico *boxplot*, nós apresentamos os resultados das variáveis requisitos documentados (RD) e requisitos documentados corretamente (RC), considerando os grupos POP e não-POP.

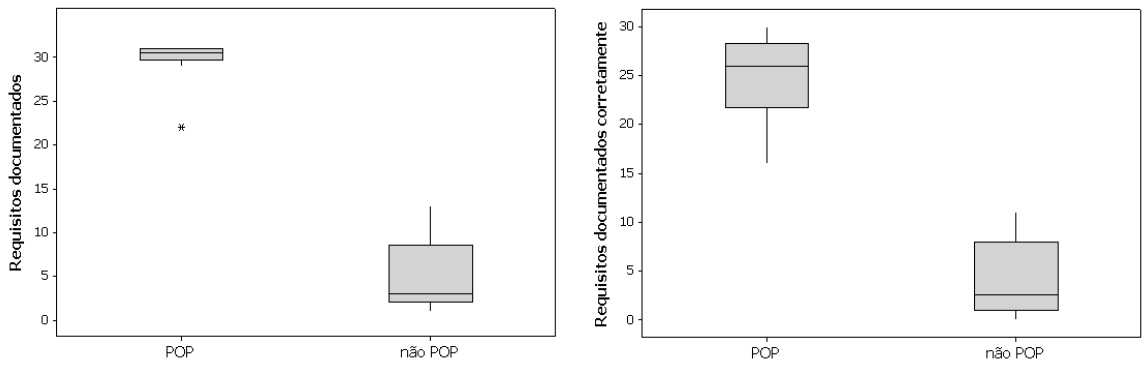

Figura 5.3: Distribuição dos RD e RC dos grupos POP e não-POP.

Como podemos observar na Figura 5.3, para a variável requisitos documentados, o grupo POP, a exceção do estudante Arnaldo (que documentou apenas 22 requisitos), os demais estudantes documentaram uma faixa de 29 a 31 requisitos. O grupo não-POP, por sua vez, documentou uma faixa de

2 a 8 requisitos, sendo 13 o maior número de requisitos documentados pelos estudantes desse grupo. Cabe ressaltarmos, que mesmo o estudante Arnaldo, que teve o desempenho mais baixo no grupo POP, documentou mais requisitos que qualquer um dos estudantes não-POP.

Com relação a variável RC, os estudante POP documentaram corretamente uma faixa de 23 a 29 requisitos, sendo 16 o menor número de requisitos documentados corretamente pelos estudantes POP. No grupo não-POP, os estudantes documentaram corretamente uma faixa de 1 a 8 requisitos, sendo 11 o maior número de requisitos documentados corretamente pelos estudantes não-POP. Em síntese, para as variáveis RD e RC, os estudantes POP tiveram um desempenho bem superior aos estudantes não-POP.

O tipo de defeito mais freqüente no grupo não-POP foi o de *requisitos omitidos*. Em geral, os estudantes não-POP faziam apenas rascunhos de algoritmos ou pequenas anotações, muitas vezes, mnemônicas e sem contexto.

No grupo POP nós iremos discutir os tipos de defeitos nas duas versões do documento de especificação, uma vez que os estudantes atenderam a todos os requisitos em no máximo duas versões do programa. Na primeira versão dos documentos, considerando todos os estudantes do grupo POP, totalizamos 70 defeitos, distribuídos como mostra a Figura 5.4.

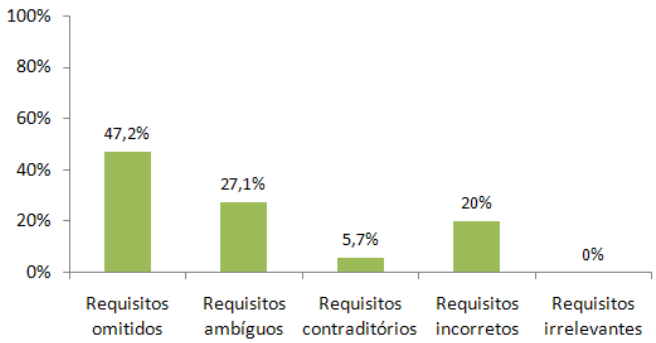

Figura 5.4: Grupo POP – Tipos de defeitos encontrados na primeira versão do documento.

Ao considerarmos apenas a totalização dos defeitos dos seis estudantes POP que fizeram as duas versões dos documentos, percebemos que houve uma diminuição no total de defeitos, de 42 na primeira versão, para 34 na segunda versão do documento. A distribuição dos tipos de defeitos, nós apresentamos na Figura 5.5.

Como podemos observar na Figura 5.5, na segunda versão do documento, houve uma significativa redução do tipo de defeito *requisitos omitidos*, o que significa que os estudantes aumentaram o número de requisitos documentados. No entanto, ao fazerem o registro de requisitos, os estudantes inseriram outros tipos de defeitos, como é caso de requisitos *incorretos* e *ambíguos*.

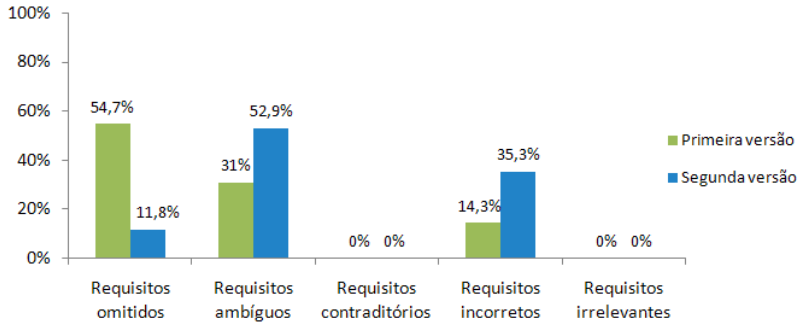

Figura 5.5: Grupo POP – Tipo de defeitos encontrados na primeira e segunda versões dos documentos.

Assim como observamos nos estudos de caso, o tipo de defeito *requisito incorreto* foi mais freqüente na descrição das mensagens de erro que deveriam ser exibidas ao usuário no caso de entradas inválidas. Nós apresentamos na Figura 5.6 um exemplo de requisito incorreto que foi extraído do documento de um estudante POP.

Mensagem de erro, esperada pelo pesquisador, para o caso de entrada de valores inválidos para a idade.
`Idade fora do limite permitido. Digite outro valor.`
`Idade?`

Mensagem de erro documentada pelo estudante.
`Idade fora do limite permitido. Digite outro valor.`

Figura 5.6: Requisito incorreto.

O tipo de defeito *requisitos ambíguos* foi mais freqüente na descrição das duas políticas de financiamento. Ao descrevê-las, muitos estudantes não deixavam explícitas se as duas políticas deveriam ser atendidas simultaneamente ou bastava atender apenas a uma delas. Nós apresentamos na Figura 5.7 um exemplo de requisito ambíguo, extraído do documento de um estudante POP.

Regra para não conseguir o financiamento:
Se o valor da parcela for > 25% da renda bruta
Se o valor do imóvel for > 25 x o valor da renda

Figura 5.7: Requisitos ambíguos.

Dificuldades em reconhecer ambigüidades em descrições de software foram também enfatizadas por meio de um estudo multi-nacional e multi-institucional envolvendo estudantes e professores de Ciência da Computação, conforme apresentado em [Blaha et al. 2005].

Reconhecer descrições ambíguas é uma habilidade importante, uma vez que requisitos ambíguos podem propagar erros indesejáveis em projetos de *software*. Em nosso estudo, nós observamos que, embora os requisitos fossem documentados de forma ambígua, os erros não se propagaram para os programas, porque os estudantes questionavam se eles deveriam usar o operador booleano "`and`" ou "`or`" para relacionar as duas políticas de financiamento do banco. A dificuldades dos estudantes foi a de documentar adequadamente essas políticas.

Para encerramos a discussão sobre os requisitos documentados e documentados corretamente, nossa percepção é que o fato de termos adotado uma estratégia de pesquisa com observação participante, na qual o pesquisador ficava próximo do estudante na sessão experimental, pode ter tido um impacto direto sobre o desempenho dos estudantes. Dada a proximidade do pesquisador, os estudantes podem não ter sentido tanta necessidade em documentar certos requisitos.

Nós acreditamos que a adoção de uma estratégia de pesquisa diferente, tal como, o pesquisador, à distância, respondendo verbalmente aos estudantes, poderia encorajá-los a antecipar questionamentos e melhorar a documentação dos requisitos. Entretanto, esta estratégia causaria um impacto sobre os custos da pesquisa, porque exigiria, por exemplo, mais de uma sala para realização do estudo; e, recurso tecnológico (como, por exemplo, câmera filmadora) ou humano para fiscalizar as atividades dos estudantes durante a sessão experimental.

Testes. Estudantes POP e não-POP testaram seus respectivos programas executando o código e verificando, manualmente, as saídas dos dados. Na Figura 5.8, nós apresentamos, por meio de um gráfico de barras, a média, por versão do programa, de testes executados pelos estudantes.

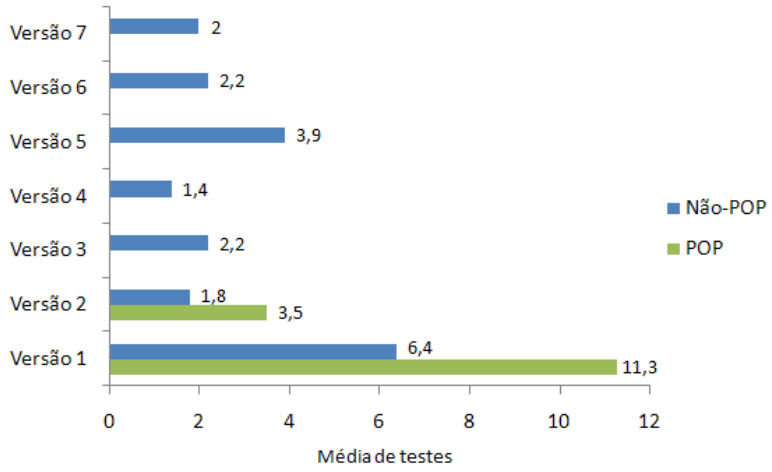

Figura 5.8: Média de testes executados por versão do programa.

A média de testes executados pelos estudantes não-POP foi, em todas as versões do programa,

mais baixa que a dos estudantes POP. Isto ocorreu em virtude de alguns estudantes não-POP submeterem seus programas sem fazer qualquer teste. Para exemplificar, na primeira versão, 2 estudantes não-POP não fizeram qualquer teste; na segunda versão, esse número aumentou para 4 estudantes; e, na quarta versão, aumentou para 5, o que representou 50% do grupo não-POP, uma vez que nenhum estudante desse grupo solucionou o problema em menos de 4 versões do programa.

Embora os estudantes POP não tenham criado testes automáticos, eles tiveram preocupação com as restrições das variáveis, formatações de entrada e saída e mensagens de erro, antecipando-se na descoberta desses requisitos. Nós acreditamos que esse comportamento está diretamente ligado a prática de testes durante a disciplina de programação.

Essa evidência foi confirmada na entrevista em que os estudantes disseram que o ato de ler `asserts`, e ter *feedback* de seus erros nas provas e exercícios os tornaram mais espertos na percepção das possibilidades de erros nos programas. Nas provas de laboratório dos estudantes POP, eles submetiam seus programas a um conjunto de casos de testes por meio de uma ferramenta denominada *Hoopaloo* e, de acordo com o *feedback* obtido, os estudantes poderiam corrigir seus programas e submetê-los novamente. No *feedback* dos execícios de programação, os monitores da disciplina indicavam os casos de testes nos quais os programas dos estudantes falhavam.

Tempo de resolução do problema. Com respeito ao tempo gasto na resolução do problema, não houve evidências suficientes de diferenças na média entre os grupos POP e não-POP, conforme apresentamos na Tabela 5.1. Nesta tabela, é possível observarmos que no caso do grupo não-POP, os valores são mais dispersos, haja vista que o desvio padrão para este grupo é quase o triplo do desvio padrão do grupo POP.

Embora, não tenha sido possível rejeitar a hipótese nula com respeito a essa variável, destacamos que os estudantes POP documentaram mais requisitos, executaram mais testes e solucionaram o problema em um menor número de versões do programa, se comparado com os estudantes não-POP. Mesmo sem força estatística, tais fatos demonstram um melhor uso do tempo para a resolução do problema por parte dos estudantes POP.

Do total de estudantes, apenas *Cristovão* (estudante não-POP, Tabela 5.1) não conseguiu atender a todos os requisitos, gastando 2 horas e 57 minutos para atender a 22 requisitos, na sexta versão do programa, ficando muito próximo de extrapolar o tempo estabelecido. Isto ocorreu porque o estudante teve mais dificuldades para (re)estruturar o programa, haja vista o acréscimo de requisitos em relação a primeira versão do programa feita por ele.

Dificuldades na implementação dos programas. Com respeito às dificuldades de implementação, alguns estudantes POP não lembravam como formatar as saídas com duas casas decimais.

Além disso, alguns estudantes tiveram dificuldades na implementação do requisito 20 (Quadro 5.2), que exigia o cálculo de 25% sobre a renda bruta, esquecendo-se que, para isso, teriam que tratar com coerção de tipos em Python.

Os estudantes não-POP tiveram dificuldades para: formatar as saídas com duas casas decimais; escrever os argumentos da função `scanf`, mais precisamente, o operador de referência (&); entender as mensagens de erro do compilador para, então, corrigir o programa. Com relação a esta última dificuldade, outras pesquisas [Nienaltowski et al. 2008; Hartmann et al. 2010] têm demonstrado que interpretar as mensagens de erro e de exceção dos compiladores é uma atividade difícil para os estudantes iniciantes e que o ensino desta habilidade merece mais atenção.

Ao questionarmos os estudantes sobre a complexidade de implementação, a maioria dos estudantes POP e não-POP, consideraram a implementação do programa *fácil* ou *muito fácil*, conforme apresentamos na Figura 5.9. Na entrevista, a estudante Camila (não-POP) justifica, como mostramos na Figura 5.10, o fato de ter necessitado construir 7 versões do programa até atender a todos os requisitos.

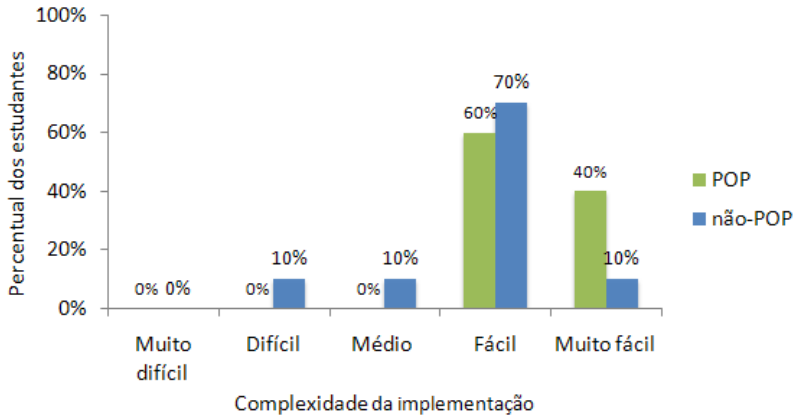

Figura 5.9: Complexidade da implementação segundo os estudantes.

> "Nunca havia me deparado com esse tipo de problema, então o mais difícil foi assimilar o que era requisitado. Achei muito importante desenvolver esse nível de raciocínio que vai mais além dos tipos de problemas que estamos acostumados a resolver."

Figura 5.10: Depoimento sobre dificuldade em especificar o programa.

5.2 Experimento 2

Nesta seção, nós descrevemos o experimento realizado no segundo semestre acadêmico de 2009. No que diz respeito ao planejamento e execução do estudo, nós iremos descrever apenas os aspectos que foram modificados em relação ao experimento anterior.

5.2.1 Planejamento

Os sujeitos que participaram desta pesquisa eram estudantes iniciantes de programação das mesmas universidades do primeiro experimento. Neste caso, os estudantes ingressaram nestas universidades no segundo semestre acadêmico de 2009. Nós mantivemos no segundo experimento todo o planejamento adotado no experimento anterior.

5.2.2 Execução

Este experimento foi executado no final do segundo semestre acadêmico de 2009 e nós adotamos os mesmos procedimentos utilizados na execução do experimento anterior. Nós descrevemos, nesta seção, apenas as informações sobre a amostra e sobre como contornamos as ocorrências não previstas no planejamento.

Amostra

Nós realizamos a seleção da amostra para o segundo experimento seguindo os mesmos procedimentos do experimento anterior. O questionário (Apêndice F), que nós aplicamos para coletar alguns dados pessoais e acadêmicos dos estudantes, foi respondido por 49 estudantes da UFCG e por 31 estudantes da outra universidade. Os dados coletados nos auxiliaram no processo de seleção da amostra.

Nesse estudo, nós fizemos a seleção de 15 estudantes para cada grupo a fim de aumentarmos o número de participantes na pesquisa. Entretanto, no grupo de controle nós conseguimos a efetiva participação de apenas 11 estudantes. Assim, para este segundo experimento tivemos duas amostras com diferentes tamanhos: 15 estudantes no grupo experimental e 11 estudantes no grupo de controle.

Nós apresentamos no Quadro 5.6 as principais características dos estudantes em cada grupo (experimental e de controle).

Quadro 5.6: Características das amostras.

Características	Grupo Experimental	Grupo de Controle
Média de idade	18 anos	19 anos
Gênero	13 homens e 2 mulheres	11 homens
Ensino médio	12 de escola particular; 2 de escola pública; 1 de escola pública/particular	9 de escolha particular; 2 de escola pública
Computador e Internet em casa	15 possuíam computador; 14 possuíam Internet	Todos possuíam computador e Internet
Experiência prévia com programação	10 sem experiência; 1 com experiência em Pascal (4 meses); 1 com experiência em C (3 meses); 2 com experiência em PHP (5 meses e 1 ano, respectivamente); 1 com experiência em Java/Delphi (2 meses)	8 sem experiência; 2 com experiência em Pascal (1 meses); 1 com experiência em Visual Basic (3 meses)
Fazendo algum curso técnico	13 não; 2 – Web Design (completo)	5 não; 2 – Montagem e manutenção de computadores (primeiro período – cursando); 2 – Montagem e manutenção de computadores (concluído); 1 – Automação Industrial (primeiro período – trancado); 1 – Suporte a Redes de Computadores (primeiro período – trancado)
Fazendo outro curso de graduação	14 não; 1 – Ciências Contábeis (segundo período – cursando)	10 não; 1 – Tecnologia em Sistemas para Internet (primeiro período – trancado)

Desvio

Alguns estudantes não-POP tiveram restrições de horário e, novamente, tivemos que contar com a participação de um segundo pesquisador. Neste caso, conseguimos a colaboração do mesmo pesquisador que havia participado do experimento anterior.

5.2.3 Análise

Neste experimento, nós também utilizamos testes estatísticos paramétricos e não paramétricos. Para analisar as variáveis *relevância dos questionamentos* e *tempo de resolução do problema*, nós adotamos o teste paramétrico *t de Student*, e para análise das demais variáveis, nós adotamos o teste não paramétrico *Mann-Whitney* [Jain 1991; Siegel and Jr. 1988]. Ambos os testes foram executados considerando um nível de significância (α) de 0.05 e um nível de confiança de 95%.

5.2.4 Avaliação da Validade

Dado que nós mantivemos os mesmos procedimentos na execução do segundo experimento, todo o tratamento da validade descrito na Seção 5.1.4 é igualmente válido para esse estudo.

5.2.5 Resultados

Na Tabela 5.5, nós apresentamos os resultados obtidos para cada variável investigada, considerando os dois grupos, POP e não-POP, respectivamente.

Na Tabela 5.6, nós apresentamos o p-valor para o teste das hipóteses nulas, considerando a análise bilateral. Como podemos verificar nesta tabela, somente o tempo usado na resolução do problema pelos estudantes POP não foi significativamente diferente do tempo empregado pelos estudantes não-POP. Nesse caso, em que a hipótese nula não pôde ser rejeitada, nós calculamos o intervalo de confiança com nível de significância de $0{,}05$. O intervalo de confiança obtido foi de $(-18{,}042; 15{,}787)$. Como podemos observar, o zero está incluído no intervalo, confirmando que não há diferença no tempo de resolução do problema, considerando os dois grupos. No entanto, por ser um intervalo amplo, isso indica que o teste de hipótese para a variável TR foi estimado com um baixo grau de precisão.

Tabela 5.5: Resultados obtidos no grupo POP e não-POP.

POP							**Não-POP**						
Nome	**RD**	**RC**	**NV**	**RA**	**RQ**	**TR**	**Nome**	**RD**	**RC**	**NV**	**RA**	**RQ**	**TR**
David	30	26	1	31	98,4	50	Fabiano	2	1	8	3,9	93,9	95
Daniel	9	0	2	15,5	97,4	82	Fábio	0	0	6	5,2	96,0	72
Daniele	24	14	2	15,5	98,2	117	Fabrício	1	1	9	3,4	92,5	114
Demétrio	29	18	2	15,5	96,0	96	Fausto	0	0	12	2,6	95,1	51
Danilo	30	24	1	31	98,9	71	Felipe	0	0	7	4,4	95,0	56
Diogo	30	28	1	31	97,6	102	Félix	0	0	8	3,9	95,8	67
Douglas	31	21	3	10,3	96,7	69	Fernando	1	1	7	4,4	98,0	80
Dimas	28	20	3	10,3	93,2	55	Frank	7	2	7	4,4	94,4	60
Diniz	30	28	2	15,5	95,7	78	Franklin	0	0	10	3,1	86,5	90
Débora	27	20	1	31	95,7	52	Fred	0	0	6	5,2	96,4	74
Davis	7	5	2	15,5	97,0	46	Frederico	0	0	7	4,4	89,4	91
Diego	19	13	1	31	96,1	77							
Dário	27	23	1	31	100,0	86							
Donato	31	25	2	15,5	94,4	108							
Domingos	31	25	1	31	95,9	87							
Média	25,5	19,3	1,7	22,0	96,7	78,4	**Média**	1,0	0,5	7,9	4,1	93,9	77,3
DP	7,8	8,2	0,7	8,8	1,8	21,8	**DP**	2,1	0,7	1,8	0,8	3,3	19,0

Legenda:

Nome	Nome fictício dos estudantes.
RD	Quantidade de requisitos documentados.
RC	Quantidade de requisitos documentados corretamente.
NV	Quantidade de versões do programa.
RA	Média de requisitos atendidos por versão.
RQ	Porcentagem de questionamentos relevantes.
TR	Tempo, em minutos, gasto na resolução do problema.
Média	Média aritmética.
DP	Desvio padrão.

Na Tabela 5.7, nós apresentamos o p-valor para o teste das hipóteses alternativas. No caso das hipóteses ($H1_{1..6}$), apresentamos o p-valor considerando a análise bilateral e no caso das hipóteses ($H2_{1..6}$), considerando a análise unilateral.

Tabela 5.6: p-valor para os teste das hipóteses nulas.

Hipóteses	$H0_1$	$H0_2$	$H0_3$	$H0_4$	$H0_5$	$H0_6$
p-valor	0,001	0,001	0,001	0,001	0,010	0,892
status	*rejeitar*	*rejeitar*	*rejeitar*	*rejeitar*	*rejeitar*	*aceitar*

Tabela 5.7: p-valor para o teste das hipóteses alternativas.

Hipóteses	$H1_1$	$H1_2$	$H1_3$	$H1_4$	$H1_5$	$H1_6$
p-valor	0,001	0,001	0,001	0,001	0,010	0,892
status	*aceitar*	*aceitar*	*aceitar*	*aceitar*	*aceitar*	*rejeitar*
Hipóteses	$H2_1$	$H2_2$	$H2_3$	$H2_4$	$H2_5$	$H2_6$
p-valor	0,0005	0,0005	0,0005	0,0005	0,005	0,446
status	*aceitar*	*aceitar*	*aceitar*	*aceitar*	*aceitar*	*rejeitar*

5.2.6 Discussão

Nós faremos uma análise descritiva e qualitativa dos resultados obtidos em nosso segundo experimento. Para esta análise, nós seguiremos a mesma organização de tópicos definida na seção de discussão do primeiro experimento (Seção 5.1.6).

Primeira versão dos programas. Na Tabela 5.8, nós apresentamos a média de três variáveis observadas no grupo POP e não-POP, considerando apenas a primeira versão do programa. Essas variáveis são: (i) tempo, em minutos, para iniciar a codificação; (ii) questionamentos feitos pelos estudantes; e (iii) requisitos atendidos pelo programa.

Tabela 5.8: Média de dados da primeira versão do programa.

Variáveis	**POP**	**Não-POP**
Tempo, em minutos, para iniciar a codificação	11	6
Questionamentos	19	2
Requisitos atendidos	76,3%	24,2%

Como podemos observar na Tabela 5.8, os estudantes POP, em média, levaram 11 minutos para iniciar a codificação, fizeram 19 questionamentos e atenderam a 76,3% dos requisitos. Os estudantes não-POP, em média, levaram 6 minutos para iniciar a codificação do programa, fizeram 2 questionamentos e atenderam a 24,2% dos requisitos. Novamente, os estudantes POP foram mais efetivos na captura dos requisitos que os estudantes não-POP.

Relevância dos questionamentos *versus* Cobertura dos requisitos. Os estudantes não-POP tiveram dificuldades para elicitar requisitos mesmo após a primeira versão do programa. Como observamos na Tabela 5.5 (Colunas NV e RA), o grupo não-POP precisou em média de 7,9 versões do programa para atender a todos os requisitos; e, implementaram, em média, 4,1 requisitos por versão. Os estudantes POP, por sua vez, precisaram, em média, de 1,7 versões do programa para atender a todos os requisitos; e, implementam, em média, 22 requisitos por versão.

Com relação a variável relevância dos questionamentos (Coluna RQ da Tabela 5.5), as médias foram 96,7% e 93,9% para os grupos POP e não-POP, respectivamente. Estes números indicam que os estudantes, de ambos os grupos, estiveram mais concentrados nos questionamentos referentes aos requisitos.

Apresentamos na Figura 5.11, um gráfico *scatterplot* que mostra a relação entre as variáveis questionamentos relevantes e número de versões do programa necessárias para atender a todos os requisitos. Nesta figura, vamos considerar, por exemplo, o caso dos estudantes não-POP que tiveram uma porcentagem de questionamentos relevantes maior ou igual a 95%. Apesar da alta porcentagem de questionamentos relevantes, eles necessitaram construir 6 ou mais versões do programa para atender a todos os requisitos. Em contrapartida, os estudantes POP, com a mesma porcentagem de relevância dos questionamentos, precisaram construir, no máximo, 3 versões do programa para atender a todos os requisitos. Novamente, observamos que os estudantes POP foram mais efetivos na captura dos requisitos que os estudantes não-POP.

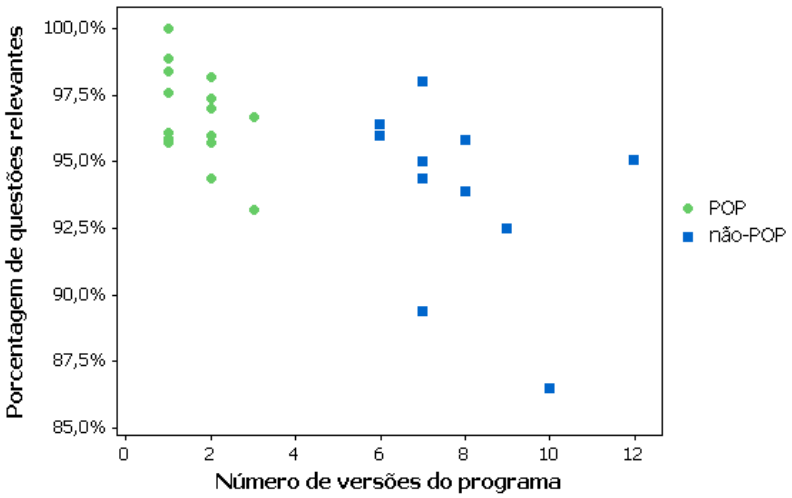

Figura 5.11: Relevância dos questionamentos *vs.* número de versões do programa.

Neste segundo experimento, uma situação recorrente no grupo não-POP nos chamou atenção. Os estudantes queriam adivinhar os requisitos, ao invés de questionar, objetivamente, o pesquisador. Para ilustramos esta afirmação, apresentamos no Quadro 5.7 um trecho de diálogo entre o estudante Fabrício e o pesquisador. Neste trecho, o estudante tenta elicitar a mensagem que deveria ser exibida no caso de entrada de valores inválidos para a idade. Embora todos os questionamentos do estudante sejam relevantes, pois estão diretamente relacionados ao esclarecimento da mensagem de erro, nós podemos notar o esforço do estudante para esclarecer um requisito.

Quadro 5.7: Esforço do estudante na elicitação de requisitos.

	Diálogo
1	*Estudante:* Eu coloco [referindo-se a mensagem de erro] *Você não é maior de idade,*
2	*tente outra vez*?
3	*Pesquisador:* Não. Eu não quero essa mensagem.
4	*Estudante: Digite novamente*?
5	*Pesquisador:* Não.
6	*Estudante: Não autorizado*?
7	*Pesquisador:* Não.
8	*Estudante: Simulação não autorizada*?
9	*Pesquisador:* Não, eu quero outra mensagem.
10	*Estudante: Digite dado maior que 18*?
11	*Pesquisador:* Não. A mensagem que eu quero é *Idade fora do limite permitido. Digite*
12	*outro valor.* Aí, você deve solicitar a idade novamente.

No grupo POP, $46{,}7\%$ dos estudantes não elicitaram todas as entradas na primeira versão. Isto ocorreu porque os estudantes exploraram pouco o enunciado do problema e fizeram questionamentos específicos sobre as entradas que estavam explícitas no enunciado – número de parcelas e valor do imóvel. No entanto, a exceção de um estudante, todos os demais conseguiram elicitar todas as entradas na segunda versão.

No grupo não-POP, $72{,}7\%$ dos estudantes não elicitaram todas as entradas na primeira versão. Além disso, 100% dos estudantes ao elicitarem a idade como entrada, não conseguiram fazer perguntas mais gerais para a descoberta de renda que também fazia parte da entrada do programa. Assim, estes estudantes precisaram de mais de duas versões do programa para elicitarem todas as entradas. 100% dos estudantes não-POP, à exceção da formatação das casas decimais, não questionaram sobre qualquer outra formatação para as entradas e saídas. Estas situações demonstram a dificuldade dos estudantes em inferir novos requisitos a partir dos requisitos descobertos.

Documentação dos Requisitos e Tipos de Defeitos. A média de requisitos documentados e de requisitos documentados corretamente pelos estudantes POP ($25{,}5\%$ e $19{,}3\%$, respectivamente) foi bastante superior a dos estudantes não-POP ($1{,}0\%$ e $0{,}5\%$, respectivamente), conforme apresentamos na Tabela 5.5. Na Figura 5.12, nós apresentamos em um gráfico *boxplot* os resultados das variáveis requisitos documentados (RD) e requisitos documentados corretamente (RC), considerando os grupos POP e não-POP.

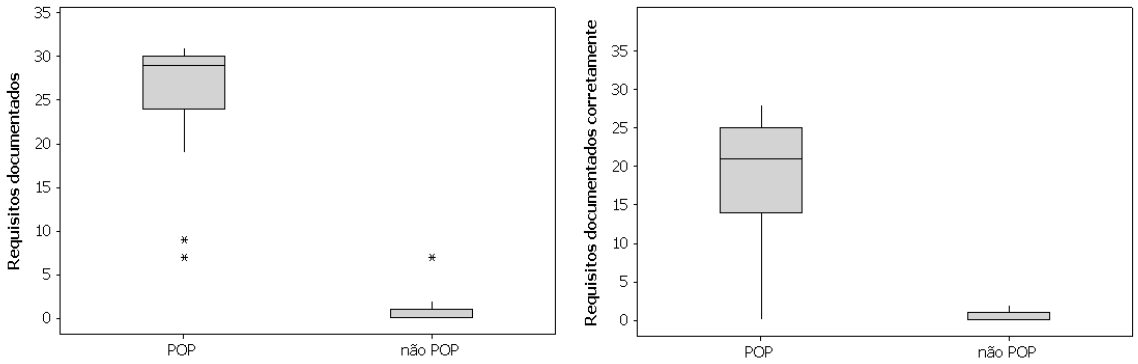

Figura 5.12: Distribuição dos RD e RC dos grupos POP e não-POP.

Como podemos observar na Figura 5.12, para a variável RD, o grupo POP, a exceção de dois estudantes, documentou uma faixa de 19 a 31 requisitos. O grupo não-POP, a exceção de um estudante que documentou 7 requisitos, os demais documentaram no máximo 2 requisitos.

Com relação a variável RC, a maioria dos estudantes POP documentaram uma faixa de 13 a 25 requisitos. Cabe, porém, salientarmos que neste grupo, houve um estudante que não teve qualquer requisito documentado corretamente. No grupo não-POP, os estudantes documentaram corretamente no máximo 2 requisitos. Novamente, para as variáveis RD e RC, os estudantes POP tiveram um desempenho bem superior aos estudantes não-POP.

Novamente, o tipo de defeito mais freqüente no grupo não-POP foi o de *requisitos omitidos*. A falta de registro dos requisitos não comprometeu a implementação do programa, pois o pesquisador estava próximo do estudante e disponível para perguntas. Entretanto, o fato de não documentarem requisitos e estarem voltados inteiramente para codificação, pode ter contribuído para os estudantes não-POP refletirem menos sobre o problema e, por conseguinte, precisarem construir um maior número de versões do programa para atender a todos os requisitos.

No grupo POP, nós iremos discutir os tipos de defeitos apenas nas duas versões do documento de especificação. A terceira versão do documento é pouco significativa, pois foi construída por apenas dois estudantes. Na primeira versão dos documentos, considerando todos os estudantes do grupo POP, totalizamos 208 defeitos, distribuídos como mostra a Figura 5.13.

Ao considerarmos a totalização dos defeitos dos oito estudantes POP que fizeram as duas versões dos documentos, percebemos que houve uma diminuição no total de defeitos, de 150 na primeira versão, para 125 na segunda versão do documento. A distribuição dos tipos de defeitos, nós apresentamos na Figura 5.14.

Como podemos observar nesta figura, na segunda versão do documento, houve uma significativa

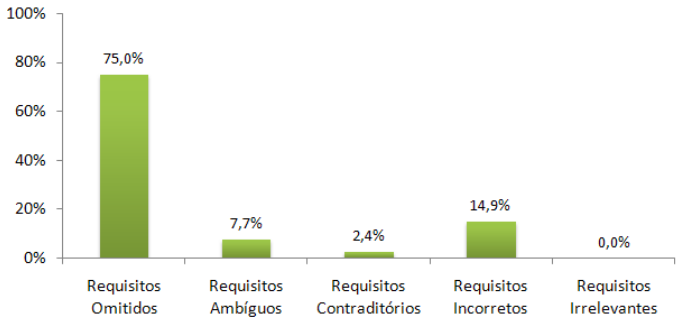

Figura 5.13: Grupo POP – Tipos de defeitos encontrados na primeira versão do documento.

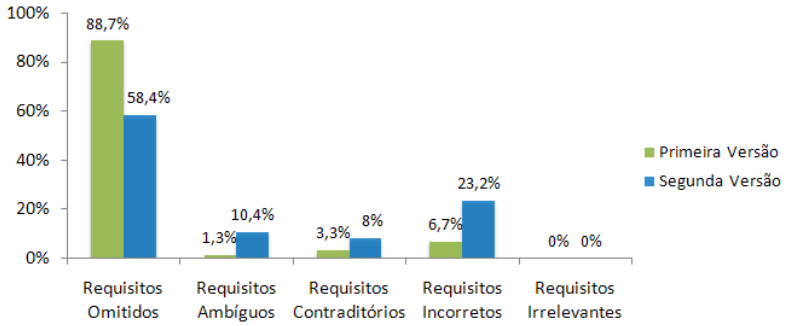

Figura 5.14: Grupo POP – Tipo de defeitos encontrados na primeira e segunda versões dos documentos.

redução do tipo de defeito *requisitos omitidos*. Em contrapartida, ao fazerem o registro de requisitos, os estudantes POP inseriram outros tipos de defeitos, tais como, requisitos *incorretos* e *ambíguos*.

O tipo de defeito *requisito incorreto* foi mais freqüente na descrição das mensagens de erro que deveriam ser exibidas ao usuário no caso de entradas inválidas e também na descrição das restrições das entradas. Este último, nós ilustramos na Figura 5.15.

Restrições para a entrada de idade
De 18 a 57 anos (18 ≤ idade ≤ 57)

Restrições para entrada de idade documentadas pelo estudante
➢ Idade
- A idade válida é entre 18 e 57 anos.

Figura 5.15: Requisito incorreto.

Assim como no primeiro experimento, o tipo de defeito *requisitos ambíguos* foi mais freqüente na descrição das duas políticas de financiamento.

Testes. Também neste experimento, estudantes POP e não-POP testaram seus programas executando o código e verificando, manualmente, as saídas dos dados. Na Figura 5.16, nós apresentamos a média, por versão do programa, de testes executados pelos estudantes.

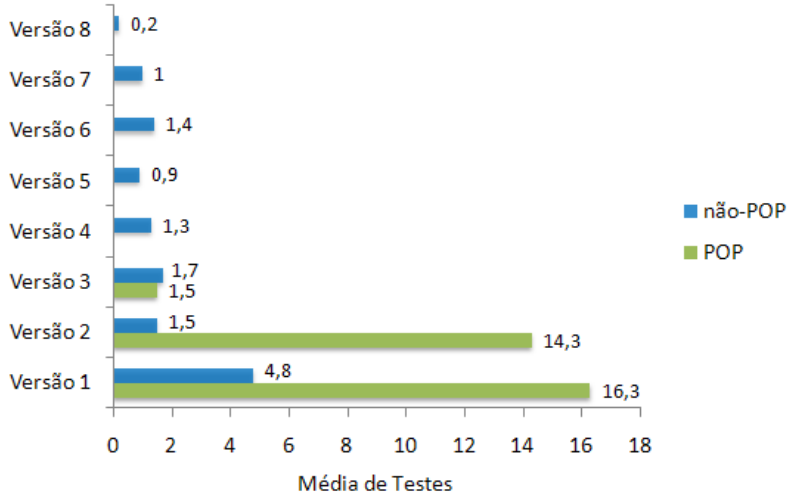

Figura 5.16: Média de testes executados por versão do programa.

A média de testes executados pelos estudantes não-POP foi, em todas as versões do programa, mais baixa que a dos estudantes POP. A partir da oitava versão, os estudantes não fizeram qualquer teste.

Tempo de resolução do problema. Assim como no experimento anterior, não houve evidências suficientes de diferenças na média entre os grupos POP e não-POP, conforme apresentamos na Tabela 5.5. Entretanto, é importante notarmos que os estudante POP fizeram um melhor uso do tempo. Embora eles tenham utilizado em média 1 minuto a mais que os estudantes não-POP, eles documentaram mais requisitos, solucionaram o problema em um menor número de versões e também executaram mais testes.

Dificuldades na implementação dos programas. Neste experimento, a dificuldade mais freqüente dos estudantes POP foram a de tratar com coerção de tipos em Python e formatação das saídas com duas casas decimais. Os estudantes não-POP apresentaram as mesmas dificuldades dos estudantes não-POP do experimento anterior: formatar saídas com duas casas decimais; escrever os argumentos da função `scanf`; e entender as mensagens de erro do compilador.

Ao questionarmos os estudantes sobre a complexidade da implementação, a maioria dos estudantes POP considerou a implementação do programa *fácil* ou *muito fácil*, conforme apresentamos na Figura 5.17. A maioria dos estudantes não-POP, por sua vez, considerou a implementação de *fácil* ou *média* complexidade. Na Figura 5.18, apresentamos algumas justificativas dos estudantes não-POP

para o fato de precisarem de várias versões do programa até atenderem a todos os requisitos.

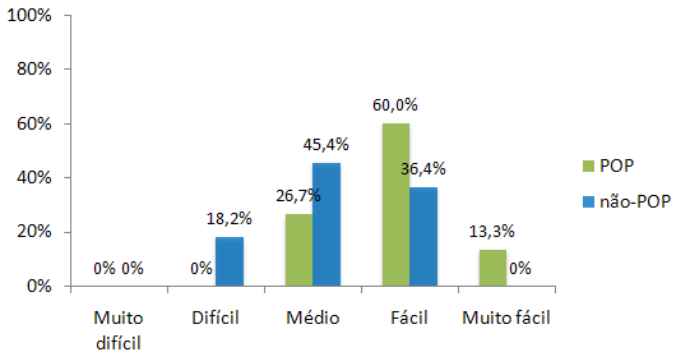

Figura 5.17: Complexidade da implementação segundo os estudantes.

"Eu achava que o enunciado tinha muita historinha."

"Como os dados da inadimplência não estavam explícitos pensei que não precisava implementar."

"Eu pensei em fazer logo a parte lógica do programa."

Figura 5.18: Justificativas para os vários números de versões do programa.

5.3 Considerações Finais

Ao considerarmos a análise dos dois experimentos, observamos que os estudantes POP tiveram melhor desempenho que os estudantes não-POP, em todas as variáveis, a exceção do tempo para resolução do problema.

De acordo com os dados observados, os estudantes do segundo experimento tiveram um desempenho inferior, para algumas variáveis, quando comparados aos estudantes do primeiro experimento. Neste caso, comparação feita entre os estudantes do mesmo grupo. Para exemplificar esta afirmação, analisaremos as variáveis requisitos documentados (RD) e requisitos documentados corretamente (RC). Na Tabela 5.9, nós apresentamos a média obtida pelos grupos nos dois experimentos. Como podemos observar nesta tabela, tanto os estudantes POP quanto os estudantes não-POP tiveram desempenhos inferiores nestas variáveis no segundo semestre acadêmico, se comparado aos seus respectivos grupos no primeiro semestre acadêmico.

A fim de demonstrarmos o desempenho por estudante, nós apresentamos nas Figuras 5.19 e 5.20, gráficos em linha que demonstram a comparação entre os estudantes de cada grupo, nos dois semestres acadêmicos. Nestes gráficos é perceptível o menor desempenho dos estudantes no segundo experimento, em ambos os grupos (POP e não-POP).

Tabela 5.9: Média de requisitos documentados e de requisitos documentados corretamente.

Grupos	RD	RC
POP – 2009.1	29,6	24,8
POP – 2009.2	25,5	19,3
não-POP – 2009.1	5,2	3,9
não-POP – 2009.2	1,0	0,5

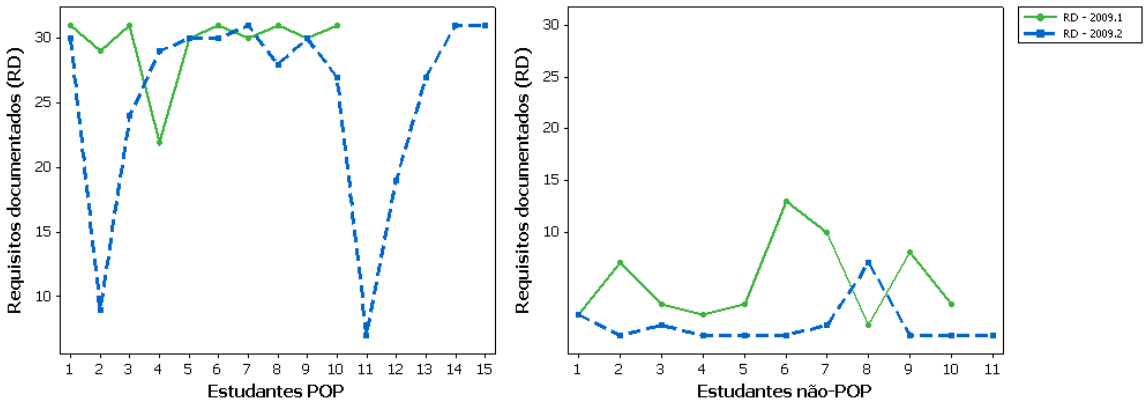

Figura 5.19: RD – Estudantes POP e não-POP nos dois experimentos.

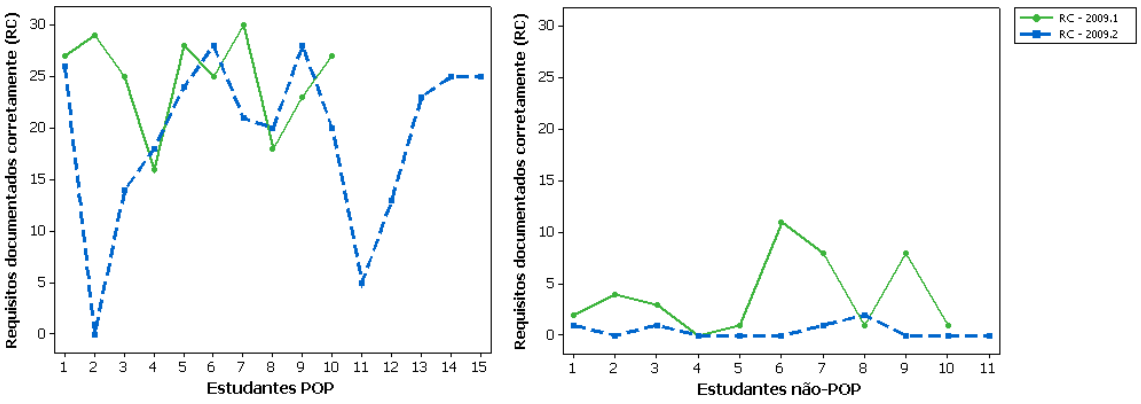

Figura 5.20: RC – Estudantes POP e não-POP nos dois experimentos.

No que diz respeito ao número de versões do programa, os estudantes POP, no primeiro experimento, precisaram construir no máximo 2 versões do programa para atender a todos os requisitos. Em contrapartida, no segundo experimento, dois estudantes POP precisaram construir 3 versões do programa para atender a todos os requisitos. Quando a média é considerada, esta diferença não se torna tão evidente (1,6 versões do programa no primeiro experimento e 1,7 versões do programa no segundo experimento).

No caso dos estudante não-POP, a diferença nos dois semestres foi maior. No primeiro experimento, os estudantes não-POP precisaram construir, em média, 5,6 versões do programa para atender a todos os requisitos. No segundo experimento, está média aumentou, pois os estudante precisaram construir, em média, 7,9 versões do programa para atender a todos os requisitos.

No caso dos estudantes POP, cabe destacarmos que nós melhoramos a aplicação de POP no segundo semestre acadêmico, uma vez que incluímos uma fase de preparação dos estudantes, antes de iniciarmos o ciclo de resolução de problemas. Além disso, os mesmos procedimentos e problemas utilizados no primeiro semestre acadêmico foram mantidos no segundo semestre.

Nós acreditamos que o menor desempenho dos estudantes do segundo experimento não está relacionado diretamente com a aplicação de POP, mas com fatores externos. Os estudantes que ingressam no segundo semestre acadêmico têm um desempenho inferior no vestibular em relação aos estudantes do primeiro semestre. Além disso, na disciplina de programação da UFCG, os estudantes do segundo semestre tiveram um percentual de aprovação menor que os estudantes do primeiro semestre acadêmico. Acreditamos que no contexto dos experimentos realizados, por tratarmos com estudantes iniciantes, esta distinção foi repercutida nas métricas coletadas, justificando a diferença de desempenho encontrada. Entretanto, acreditamos que estas diferenças são superadas ao longo da formação, não implicando em diferentes "níveis" de profissionais ao final do curso.

Por fim, com base nos resultados dos dois experimentos, observamos que os estudantes POP foram mais efetivos que os estudantes não-POP no que diz respeito a: (i) documentação dos requisitos; (ii) documentação correta dos requisitos; (iii) implementação dos requisitos utilizando um menor número de versões do programa; e, (iv) elaboração de questionamentos relevantes. Quanto ao tempo gasto na resolução do problema, os dados dos estudos não ofereceram evidências suficientes para estabelecermos diferenças entre os estudantes POP e não-POP. No entanto, considerando o desempenho dos estudantes POP com respeito aos artefatos produzidos, nós acreditamos que eles fizeram melhor uso do tempo. Estes resultados, embora não possam ser generalizados, fornecem evidências da efetividade de POP no que diz respeito às variáveis observadas.

Capítulo 6

Trabalhos Relacionados

Em nossa revisão bibliográfica, nós identificamos sete trabalhos relacionados ao tema tratado nesta tese. O trabalho de Brown [1988] apresentou procedimentos gerais para inserir especificação de requisitos na disciplina introdutória de programação. Deek [1997] propôs um modelo para resolução de problemas e desenvolvimento de programas. No trabalho de Reed [2002] é adotada uma abordagem baseada no método científico para explorar a resolução de problemas mal definidos. Eastman [2003] estabeleceu uma técnica para auxiliar os estudantes na identificação das informações relevantes de um problema. Rahman and Juell [2006] conceberam uma abordagem para enfatizar a prática de testes em cursos introdutórios de programação. No trabalho de Meyer [2003] é adotado um método de ensino de programação que expõe os estudante iniciantes à compreensão de especificações formais. Falkner and Palmer [2009], por sua vez, apresentam uma estratégia baseada no trabalho cooperativo para tratar com a resolução de problemas de programação.

Neste capítulo, nós apresentamos estes trabalhos e identificamos as semelhanças e diferenças entre eles e POP. Com base nas características principais de cada um, nós estabelecemos uma síntese dos trabalhos no Quadro 6.1.

6.1 Uma Estratégia para Especificação

A proposta de trazer a atividade de especificação para o contexto de estudantes iniciantes de programação foi também defendida por Brown [1988]. Segundo o autor, esta temática pode ser, naturalmente, introduzida no contexto da disciplina de programação, como uma forma de mostrar aos estudantes que no processo de resolução de problemas é necessário preocupar-se não apenas em "*como*" deve ser feito, mas também com "*o que*" deve ser feito. Brown [1988] descreve procedimentos gerais

que consistem, basicamente, em propor aos estudantes problemas para os quais devem ser feitas especificações mais detalhadas. O autor sugere que a partir de entrevista com o cliente (professor), o aluno detalhe a especificação em um documento padrão, composto por campos pré-definidos: descrição geral do problema, entradas, saídas e condições adicionais (manuais, velocidade de execução do programa, agenda de trabalho, etc).

Com POP, nós apresentamos procedimentos sistemáticos para orientar o professor na condução das atividades. Além disso, POP diferencia-se do trabalho de Brown por:

- Não estabelecer um documento padrão. Em nossa opinião, isso evita preocupações por parte dos estudantes em ter que ajustar a especificação para um formato pré-definido. Além disso, acreditamos que faz parte da aprendizagem dos estudantes descobrir a melhor forma de organizar e dar estilo ao documento de especificação. Isto, por conseguinte, possibilita aos estudantes expressar aspectos que são relevantes para eles e que poderiam não ter sido previstos no caso da adoção de um padrão;

- Defender o uso de testes como estratégia para auxiliar os estudantes no entendimento de problemas. Em POP, os testes são usados como estratégias para descobrir e confirmar requisitos. Isto ajuda os estudantes a refletir sobre o problema, planejar uma solução, reduzir erros no programa e praticar testes;

- Considerar o fato de que o entendimento do problema pode ser refinado na fase de codificação, ocasionando atualizações no documento de especificação. Embora pareça uma diferença simples, ela motiva os estudantes a criar estratégias para administrar as diferentes versões do documento e desenvolve uma visão mais realística do processo de desenvolvimento de software, na qual o entendimento do problema e a especificação dos requisitos não são atividades que se esgotam em uma única fase;

- Estabelecer estratégias de diálogo (Apêndices B7 e B8) para serem adotadas pelos clientes (clientes-tutores) e desenvolvedores (estudantes). No contexto de sala de aula, estas estratégias ajudam na melhoria da comunicação oral dos estudantes.

Além disso, nós estabelecemos estudos empíricos (Capítulos 4 e 5) para caracterizar a efetividade de POP. Este aspecto não foi tratado na proposta de Brown.

6.2 Um Modelo Comum para Resolução de Problemas e Desenvolvimento de Programas

Deek [1997] propôs um modelo para resolução de problemas e desenvolvimento de programas denominado *Common Model for Problem Solving and Program Development*. Este modelo é constituído por seis fases que lembram o ciclo tradicional de desenvolvimento de software – formulação do problema, planejamento da solução, projeto da solução, tradução da solução para uma sintaxe específica, testes e entrega da solução.

A fase de formulação do problema consiste em identificar e extrair do enunciado as informações relevantes para o problema. Deek sugere que os estudantes reflitam e se questionem sobre as seguintes informações: objetivos, dados, incógnitas, condições e restrições do problema. Estes elementos compõem os dados relevantes do problema e baseiam-se fortemente no trabalho de Pólya [1978]. Os testes são atividades localizadas na penúltima fase do modelo de resolução de problemas e são utilizados para avaliar a corretude do programa.

Para dar suporte as atividades estabelecidas no modelo, Deek concebeu um ambiente denominado SOLVEIT (*Specification Oriented Language in Visual Environment for Instruction Translation*). Em síntese, esse ambiente oferece editores para os estudantes documentarem suas atividades à medida que seguem nas fases do modelo proposto por ele. Para exemplificar, a Figura 6.1 mostra os recursos do SOLVEIT para a fase de formulação do problema. Para cada problema, os estudantes devem preencher as informações requeridas – *goal*, *givens*, *unknowns*, e assim sucessivamente. O SOLVEIT oferece um editor que contém templates para uma linguagem algorítmica especifica do SOLVEIT, consistindo de tipos básicos de dados, estruturas de controle, entre outros. No entanto, SOLVEIT não disponibiliza recursos para compilação do programa e execução de testes.

POP adota problemas mal definidos, o que não ocorre, ou pelo menos não está explícito, no modelo proposto por Deek. Diferente desse autor, nós consideramos que o processo de resolução de problemas é iterativo. Portanto, não obedece uma ordem seqüencial. Além disso, os comentários relacionados ao trabalho de Brown (Seção 6.1) são também pertinentes para marcar as diferenças entre o modelo proposto por Deek e POP.

No que diz respeito à avaliação, Deek realizou experimentos para avaliar a efetividade do SOLVEIT. Para a fase de formulação do problema, Deek definiu uma escala de pontos para avaliar o entendimento do estudante. Por exemplo, 4 pontos no caso de excelente representação do problema, isto é, o estudante descreveu todos os objetivos, dados, incógnitas e condicionantes; 3 pontos para o caso do estudante representar quase todas as informações relevantes e assim sucessivamente. Nos-

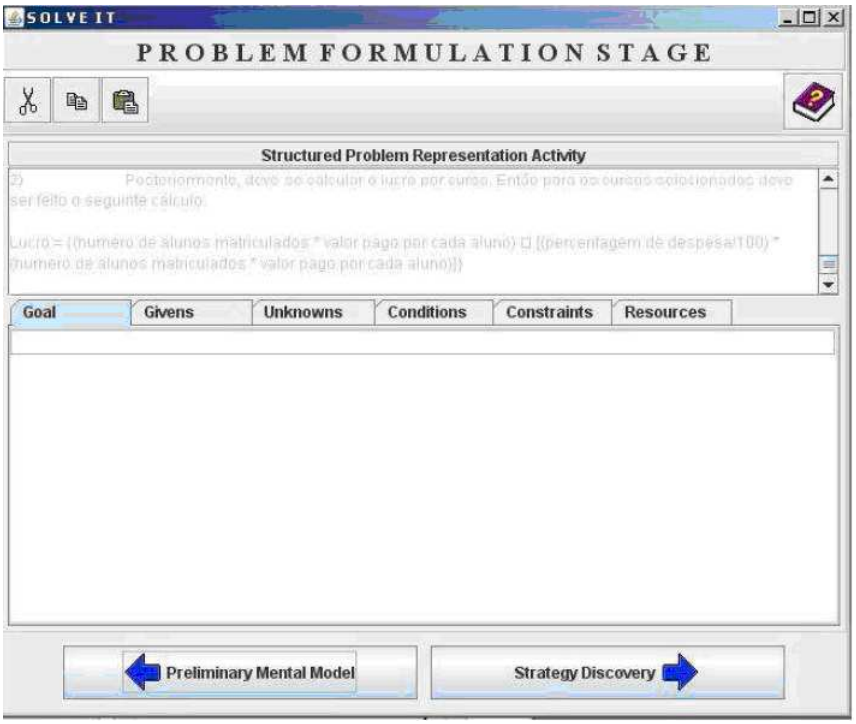

Figura 6.1: Editor do SOLVEIT para a fase de Formulação do Problema.

sos estudos empíricos envolveram a avaliação de múltiplas variáveis a fim de observarmos melhor o desempenho dos estudantes.

6.3 Uma Abordagem baseada na Investigação Científica

Reed [2002] explorou a resolução de problemas mal definidos com estudantes iniciantes de programação utilizando uma abordagem baseada no método científico. Segundo essa abordagem, um problema mal definido é entendido como um problema de *pesquisa* cuja resolução requer dos estudantes a formulação de hipóteses, projeto e implementação de experimentos para testar essas hipóteses, coleta e análise dos resultados.

Na aplicação desta abordagem com os estudantes, Reed utilizou como problema a mudança no sistema de pontuação em jogos de *volleyball* feminino. Para analisar este problema sob uma perspectiva científica, os estudantes juntamente com o professor levantaram algumas hipóteses, como por exemplo: dado um grande número de jogos, a média de tempo dos jogos será menor usando esse novo sistema de pontuação.

Após a definição das hipóteses, os estudantes planejaram um experimento e testaram as hipóteses levantadas, utilizando simulações em computador. Neste contexto, a atividade de programação é

compreendida como uma forma de criar ferramentas para testar as hipóteses formuladas. Portanto, o programa é visto como uma ferramenta para resolver problemas e não como um resultado de um exercício de programação. No caso do exemplo dado, os estudantes criaram um programa para simulação de jogos de *volleyball*.

A abordagem proposta por Reed foca no desenvolvimento de habilidades para a investigação científica. Em nossa opinião, esta abordagem poderia ser utilizada em conjunto com POP, ou seja, no momento em que os estudantes fossem criar o programa, as atividades de POP seriam postas em prática para que os estudantes especificassem os requisitos e criassem o programa para testar as hipóteses levantadas.

6.4 Identificação do Problema Baseado em Fato

Eastman [2003] estabeleceu uma técnica denominada *Fact-Based Problem Identification* para orientar os estudantes a identificarem as informações relevantes de um problema. A técnica deve ser aplicada nos casos em que o enunciado do problema é exposto em uma forma narrativa que contém além das informações relevantes, outras informações que estão presentes no enunciado apenas para deixá-lo mais interessante. Segundo o autor, essas informações extras, freqüentemente, confundem os estudantes, dificultando o entendimento do problema.

A técnica de Eastman consiste basicamente em identificar e registrar todos os *fatos* contidos no enunciado que são relevantes para o problema. O termo *fato* refere-se a palavras, nomes, números ou outras expressões que definem alguma propriedade de uma *entidade*. *Entidade* é alguma coisa que existe física ou abstratamente. À medida que os fatos são identificados, eles devem ser registrados e organizados em uma lista de fatos e com base nela os estudantes fazem a análise do que é solicitado no problema.

Esta estratégia de ler o enunciado, isolar as partes principais do problema e anotá-las adequadamente é fundamentada no trabalho de Pólya [1978]. Ela parte do pressuposto de que todas as informações estão contidas no enunciado do problema, bastando apenas isolar o que é relevante.

No caso de POP, o enunciado do problema é mal definido. Não há como descobrir o que deve ser feito sem que haja interação com um cliente. As considerações descritas na Seção 6.1 são igualmente válidas para destacar as diferenças entre POP e o trabalho de Eastman.

6.5 Testar antes de Codificar

Rahman and Juell [2006] conceberam uma abordagem para desenvolvimento de software denominada TBC (*Testing Before Coding*). Esta abordagem é composta por sete etapas, conforme ilustramos na Figura 6.2 e descrevemos a seguir.

1. Ganhar e analisar os requisitos: identificar as funções, comportamentos, restrições, *performance* e interface requeridos;

2. Desenhar o diagrama de alto nível: desenhar um diagrama de contexto contendo todas as entradas e saídas desejadas, ignorando os detalhes do programa;

3. Modelagem e especificação dos dados: definir os tipos de dados e restrições, além de identificar situações que podem causar erros;

4. Gerar casos de testes: criar casos de testes tomando como base os artefatos anteriores. Os autores sugerem utilizar uma tabela cujas colunas seguem o seguinte padrão: caso de teste, entradas, saídas esperadas, entrada válida/inválida, saída atual do programa e situação do teste (passou ou falhou);

5. Desenvolver o programa: escrever ou refatorar o programa;

6. Executar os casos de testes: executar os testes no programa e registrar os resultados. Nos casos de falha em algum caso de teste, o desenvolvedor deve criar casos de testes similares, corrigir o programa e executar todos os testes novamente;

7. Garantia de Qualidade e Avaliação do *Customer*: assegurar que o programa atende ao que foi solicitado. Caso algum requisito não seja atendido ou precise ser adicionado, o processo recomeça.

TBC e POP possuem alguns pontos de intersecção: antecipa conceitos da Engenharia de Software para o contexto de alunos iniciantes de programação, incentiva a especificação de requisitos e a prática de testes. Entretanto, há diferenças importantes: em POP os testes fazem parte da própria atividade de especificação de requisitos. Ao construirem o documento de especificação (descrição textual mais casos de testes de entrada/saída), os estudantes descobrem as entradas e saídas requeridas no programa, seus tipos de dados e restrições. Portanto, cumprem o propósito das atividades 1, 2, 3 e 4 de TBC – ganhar e analisar requisitos, desenhar digrama de alto-nível, modelar e especificar dados e gerar casos de testes, respectivamente. Acreditamos que simplificar etapas seja importante no

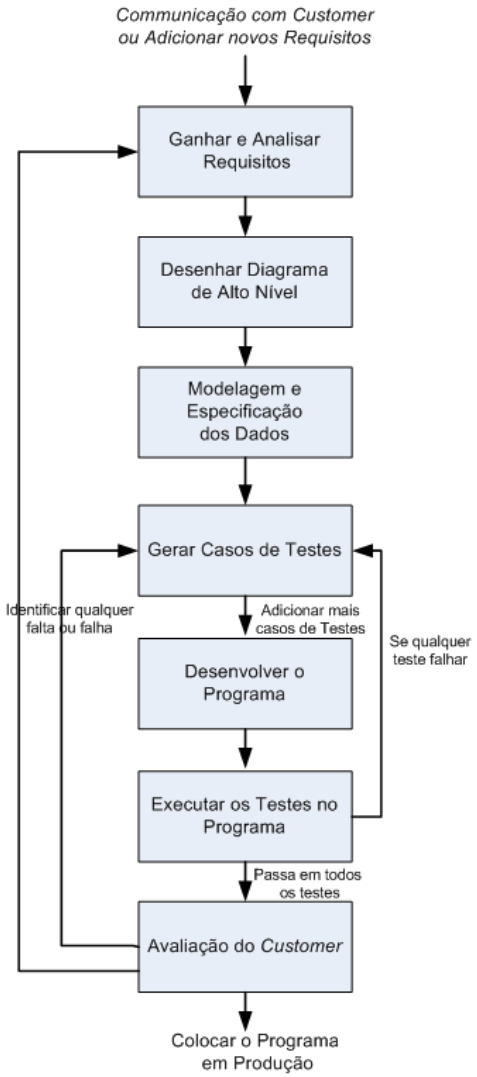

Figura 6.2: Etapas de TBC.

contexto de estudantes iniciantes, pois a exigência de atividades em muitas etapas pode desestimular a prática da abordagem proposta.

TBC não leva em consideração que durante a construção do programa os estudantes (desenvolvedores) pode ter dúvidas sobre os requisitos. Estas dúvidas, por sua vez, realimentam a interação entre desenvolvedores e *customer*. Em TBC, essa re-alimentação só ocorre após a entrega do programa ao *customer* (Figura 6.2). Também na Figura 6.2 fica claro que o artefato em foco é o próprio programa, pois ele é o único produto entregue ao *customer*.

Embora os autores digam que TBC tem sido aplicado nos cursos de programação para iniciantes do Departamento de Ciência da Computação da *North Dakota State University*, eles não detalham os resultados obtidos e não descrevem o plano de investigação empírica adotado.

6.6 Abordagem *Outside-In*

Meyer [2003] concebeu uma abordagem para ensino introdutório de programação denominada *Outside-In*. Esta abordagem baseia-se no paradigma de orientação a objetos, no uso da linguagem Eiffel, na utilização de bibliotecas e na prática do reuso.

A linguagem Eiffel implementa o conceito de *design by contract* que corresponde ao estabelecimento formal de como os componentes de *software* devem colaborar. *Assertions* são os mecanismos utilizados para descrever estas especificações [Meyer 1992]. Na abordagem proposta por Meyer, o conceito de contrato é utilizado para ensinar os estudantes a abstrair das especificações o comportamento dos módulos que eles utilizam e, por conseguinte, aplicar o mesmo procedimento para os módulos que eles irão produzir.

Na abordagem de Meyer, as questões referentes ao entendimento de problemas estão diretamente ligadas as habilidades dos estudantes de compreenderem a especificação do contrato, que está descrita em uma linguagem formal. O autor defende a exposição dos estudantes a métodos de desenvolvimento de software formais, diferenciando-se fortemente da proposta de POP.

Nós consideramos que tratar com especificações formais no contexto de estudantes iniciantes deve introduzir complexidade à disciplina de programação. Na literatura há relatos de que tratar com formalismo é difícil até mesmo para estudantes em níveis mais avançados [Finney 1996; Carew et al. 2005].

6.7 Uma abordagem baseada na Aprendizagem Cooperativa

Falkner and Palmer [2009] apresentaram uma abordagem para tratar com resolução de problemas mal definidos que consiste em envolver os estudantes na *observação* e *trabalho cooperativo* para a resolução de problemas. A abordagem foi utilizada na disciplina teórica de programação para iniciantes da *University of Adelaide*.

Na etapa de *observação*, o professor apresenta uma descrição incompleta do problema e com a turma trabalha na elaboração do problema e identificação dos principais elementos da solução, incluindo a identificação das classes e algoritmos necessários. Nesta atividade, duas ou mais potenciais soluções oferecidas pela turma são discutidas com respeito aos aspectos positivos e negativos.

Posteriormente, a turma é dividida em pequenos grupos que devem *trabalhar cooperativamente* na resolução de 2 ou 3 problemas. O grupo deve discutir sobre o problema, projetar e implementar

uma solução. Cabe ao *escriba* do grupo registrar as dúvidas e organizar a produção do grupo. O papel de *escriba* é rotacionado entre os membros do grupo, considerando cada problema proposto. O professor visita cada grupo, orientando os estudantes e respondendo aos seus questionamentos. Ao final, o professor sumariza as dúvidas e apresenta para a turma um ou dois exemplos de solução criados pelos próprios grupos.

A avaliação dessa abordagem foi realizada por meio da aplicação de questionário com os estudantes. O questionário tinha por objetivo capturar as opiniões dos estudantes sobre a abordagem, com respeito a motivação e suporte a aprendizagem (aprendizagem dos conceitos de programação e motivação para estudar a disciplina). Além disso, os autores avaliaram também o percentual de presença dos estudantes nas aulas teóricas comparando-as a presença nas aulas práticas.

Os autores enfatizam a importância de propor aos estudantes problemas pouco especificados e de fazê-los discutir sobre esses problemas, tirar suas dúvidas com o professor, antes de iniciar uma solução. Neste aspecto, há uma similaridade entre a abordagem de Falkner e Palmer e POP. Entretanto, há diferenças importantes: a abordagem dos autores é focada na aprendizagem dos conceitos de programação e na apresentação de múltiplas soluções para um mesmo problema. Isto beneficia a aprendizagem do estudantes, na medida em que eles passam a identificar diferentes caminhos, igualmente corretos, para solucionar o mesmo problema. Em POP, o foco é mais amplo, pois enfatiza um ciclo de resolução de problemas iterativo, com a produção de diferentes artefatos que são evoluídos durante o próprio ciclo. Na abordagem de Falkner e Palmer também não há uma caracterização de como os testes são adotados pelos estudantes.

No artigo dos autores [Falkner and Palmer 2009], não encontramos explicações sobre o trabalho individual dos estudantes. Embora o trabalho cooperativo seja importante, nós acreditamos que o trabalho individual é necessário para que os estudantes iniciantes reforcem sua aprendizagem de programação e ganhem auto-confiança na resolução de problemas.

A avaliação por meio de questionários, embora válida, é limitada, pois não permite uma observação mais apurada sobre o desempenho dos estudantes.

6.8 Uma Síntese

No Quadro 6.1, nós apresentamos uma síntese de cada trabalho apresentado neste capítulo. Há uma mudança conceitual na proposta de POP em relação aos demais trabalhos relacionados. POP "quebra" a perspectiva de que o entendimento do problema ocorre em uma fase isolada. Em POP, o entendimento do problema é refinado durante o processo de resolução, de acordo com níveis gradativos de

abstração. Por exemplo: inicialmente, o estudante trabalha com descrições textuais (alto nível de abstração) que são discutidas por meio do diálogo. Posteriormente, esse texto e diálogo são enriquecidos com casos de testes de entrada/saída, proporcionando um nível de abstração mais específico. Nesse nível, os estudantes enxergam com um pouco mais de profundidade o problema, detalhando melhor os requisitos relacionados às entradas, saídas e restrições, além de vislumbrarem a decomposição do problema. Um nível ainda mais específico de abstração ocorre quando os estudantes vão construir o protótipo do programa e implementar os testes automáticos (*asserts*), a fim de concretizar as atividades anteriores. Nesta fase, eles refinam seu entendimento do problema, podendo esclarecer requisitos ou reparar algum mal entendido, vivenciando, assim, "as idas e vindas" naturais no processo de resolução de problemas no domínio de programação.

Nós consideramos que transmitir esses princípios, já na série inicial, contribuirá para que o estudante, posteriormente, entenda o caráter iterativo do processo de desenvolvimento de software, principalmente, quando o contexto impuser problemas bem mais complexos. Isto também deverá favorecer a visão sobre a natureza transversal da Engenharia de Requisitos.

Outra diferença em relação aos demais trabalhos é que POP estabelece a atividade de testes como um diálogo. O princípio por traz desta decisão é tornar os testes tão naturais quanto é a própria comunicação. Essa pequena mudança faz com que o estudante reflita no fato de que os testes não têm apenas a função de verificar a existência de erros no programa, mas também a de extrair requisitos e minimizar os "ruídos" presentes na linguagem natural, facilitando o entendimento do problema entre os próprios estudantes e entre estudantes e clientes-tutores.

Ainda tratando dos aspectos de comunicação, ressaltamos a importância das estratégias de diálogo definidas em POP. Além delas estabelecerem cenários concretos de interação, minimizando os impactos negativos de possíveis condutas individuais, elas incentivam a adoção de boas práticas, como por exemplo, objetividade e expressar-se por meio de testes. Desta forma, enquanto alguns trabalhos expressam de forma geral que deve haver uma entrevista com o cliente, POP estabelece estratégias práticas para a realização desta atividade.

POP não está restrita a um relato de experiência. Nós descrevemos de forma sistemática as etapas e atividades de POP, demonstrando ao professor *como* proceder para inserir o ciclo de POP na disciplina de programação. Além disso, orientações a clientes-tutores e estudantes também são providas.

Vale ressaltar também que na avaliação de POP nós levamos em consideração dois contextos: um controlado, que permitiu a análise individual dos estudantes em comparação com estudantes que não utilizaram POP; e outro, que considerou o ambiente real de sala de aula. Estas avaliações ajudam

não apenas a caracterizar a efetividade de POP, como também oferecem a comunidade de educação em computação um plano de investigação empírica que pode ser repetido e/ou adaptado para outros contextos.

Portanto, em POP, nós definimos um ciclo de resolução de problemas no qual os estudantes trabalham, de maneira iterativa, no espaço do problema e no espaço da solução. Ao progredir no ciclo, os estudantes produzem artefatos que lhes dão suporte no entendimento do problema, especificação e implementação dos requisitos do programa, sendo os testes, recursos que os auxiliam na realização destas atividades. Desta maneira, o ciclo não se resume a produção de um único artefato – o programa. A inserção deste ciclo no contexto de sala aula foi observada por meio de estudos empíricos que possibilitaram a definição de orientações sistemáticas sobre como introduzir POP na disciplina introdutória de programação. Estes aspectos são inovadores quando comparados a todos os demais trabalhos citados anteriormente.

Quadro 6.1: Síntese dos trabalhos relacionados.

Trabalho	**Adota problemas mal definidos**	**Adota documento padrão**	**Adota testes para especificação**	**Adota estratégias de diálogo**	**A especificação pode ser refinada**	**Realizou avaliação empírica**
Uma Estratégia para Especificação	Sim	Sim	Não	Não	Não	Não
Modelo Comum Dual para Resolução de Problemas e Desenv. de Programas	Não	Sim	Não	Sim	Não	Sim
Investigação Científica	Sim	Não	Não	Não	Não	Não
Identificação do Problema baseado em Fato	Não	Sim	Não	Não	Não	Não
Testar Antes de Codificar	Sim	Não	Não	Não	Sim	Não
Outside-In	Não	Sim	Não	Não	Não	Não
Aprendizagem Cooperativa	Sim	Não	Não	Não	Sim	Não
POP	Sim	Não	Sim	Sim	Sim	Sim

Capítulo 7

Considerações Finais

Neste trabalho, nós evidenciamos um problema referente ao ensino da disciplina introdutória de programação que, nos moldes tradicionais, não expõe os estudantes a um processo de resolução de problemas que contempla o espaço do problema em conjunto com o espaço da solução.

Baseados no princípio de que programação é parte da Engenharia de Software, nós concebemos uma solução para o problema apresentado. Esta solução foi efetivada por meio de uma metodologia de ensino, denominada Programação Orientada ao Problema (POP), que motiva os estudantes a explorarem problemas mal definidos e resolvê-los por meio de um conjunto de atividades que vão da especificação dos requisitos ao programa.

A inserção de POP na disciplina introdutória de programação demanda: a inclusão de testes automáticos no conteúdo programático da disciplina, seleção e treinamento de clientes-tutores, criação de problemas mal definidos e dos artefatos de referência, e a correção dos artefatos produzidos pelos estudantes. Embora POP exija outras atividades além das que são requeridas por uma disciplina de programação tradicional, é preciso levar em consideração que este esforço representa uma melhor preparação dos estudantes. Além disso, não há custos financeiros adicionais e todos os artefatos produzidos podem ser aproveitados em aplicações posteriores de POP.

Para avaliar POP, nós executamos estudos de caso e experimentos que contabilizaram um total de 171 participantes, sendo 125 estudantes nos estudos de caso e 46 nos experimentos. Nos estudos de caso, nós avaliamos a aplicação de POP no contexto de sala de aula. Nos experimentos, avaliamos os efeitos de POP comparando o desempenho de estudantes que adotaram a metodologia com aqueles que não a adotaram. Os resultados destes experimentos demonstraram que os estudantes POP foram mais efetivos na documentação dos requisitos, na implementação dos requisitos utilizando um menor número de versões do programa e na elaboração de questionamentos para a elicitação dos requisi-

tos. Embora os resultados obtidos não possam ser generalizados, o fato de termos realizado dois experimentos nos permite prover mais confiança sobre a efetividade de POP com relação às variáveis observadas.

Os resultados de nosso trabalho contribuem diretamente para a área de Educação em Computação. Primeiramente, porque disponibilizamos uma metodologia de ensino para as disciplinas de programação introdutória. Em virtude dos princípios pedagógicos e técnicos de POP, esta metodologia tem ressonância também na comunidade de Educação em Engenharia de Software, uma vez que esta comunidade tem interesse em estratégias de ensino que possibilitem tratar os conceitos e princípios da Engenharia de Software de maneira progressiva no currículo [Lethbridge et al. 2007; ACM/IEEE 2004].

No plano de investigação empírica que nós concebemos e executamos a fim de avaliar POP, nós identificamos um conjunto de variáveis que permitem uma análise qualitativa e quantitativa do desempenho dos estudantes. Este plano de investigação é uma contribuição significativa, pois permite a repetição dos estudos e/ou adaptação deste plano para outros contextos. Além disso, estudos empíricos, embora sejam cada vez mais freqüentes em áreas como Engenharia de Software, eles são ainda pouco comuns na área de Educação em Computação [Valentine 2004; Hazzan et al. 2006].

Os resultados de nossos estudos contribuem também para ampliar as discussões sobre alguns temas ligados ao ensino de programação introdutória, dentre os quais, destacamos: utilização e efeitos de problemas bem e mal definidos no processo de ensino-aprendizagem, resolução de problemas de programação, antecipação de atividades de elicitação e especificação de requisitos e a prática de testes.

Nossas contribuições estão disponíveis neste documento da tese, nos artigos científicos publicados (Apêndice G) e no site de POP (`http://sites.google.com/site/joinpop/`).

Destacamos ainda um conjunto de trabalhos futuros, como por exemplo, investigar estratégias para tornar mais efetivo o ensino e a prática de testes automáticos no contexto da disciplina introdutória de programação. Nos estudos que realizamos, os alunos precisavam apenas testar o seu próprio código. Como os programas eram relativamente pequenos, os estudantes preferiam executar os testes manualmente, não vendo necessidade prática para a execução de testes automáticos. Assim, a prática de testes automáticos pelos estudantes POP não foi tão efetiva e outras estratégias devem ser investigadas a fim colaborar para o melhor desempenho dos estudantes neste aspecto.

É necessário também a realização de novos estudos de caso para avaliação de POP em sala de aula; neste caso, avaliando também as versões iniciais dos artefatos produzidos. Na realização destes novos estudos de caso, destacamos um conjunto de variáveis a serem observadas: quantidade de requisitos que foram implementados no protótipo dos programas em comparação com a quantidade

de requisitos que os estudantes evoluíram até a entrega da versão final; quantidade dos requisitos documentados corretamente, mas que não foram implementados na versão final do programa; quantidade de requisitos implementados corretamente, mas que não foram documentados; quantidade de requisitos cobertos pelos casos de testes automáticos; evolução dos casos de testes automáticos, considerando o que foi produzido para o protótipo do programa e, depois para a sua versão final.

Em nosso trabalho, nós observamos os diálogos dos estudantes para esclarecer requisitos e formulamos algumas estratégias de diálogo para orientar os clientes-tutores na condução das interações com os estudantes. Entretanto, nós não realizamos uma análise estatística destas interações a fim de evidenciar, por exemplo, o tipo de estratégia de diálogo mais freqüente, se há diferenças entre as estratégias utilizadas por estudantes do sexo feminino e masculino, o grau de generalidade ou especificidade das perguntas e qual a quantidade de questionamentos não precisos. Este tipo de análise de diálogo pode ser realizado de maneira interdisciplinar, envolvendo também pesquisadores das áreas de lingüística, linguagem natural e psicologia cognitiva [Weinberger and Fischer 2006; Pietarinen 2005; Peres and Meira 2008; Leitão 2000; Dillenbourg et al. 1996].

É importante também efetivar a realização de novos experimentos, adotando uma estratégia na qual o pesquisador apresenta o problema de forma verbal, ao invés de textual, e interage à distância e, por voz, com os estudantes. Ao adotar esta nova estratégia poderemos observar se o fato do pesquisador ficar à distância influencia o estudante a antecipar questionamentos, documentar requisitos e minimizar o número de submissões do programa. Estas observações seriam muito importante, principalmente, para melhor caracterizar os estudantes que não adotaram POP.

É necessário também realizar experimentos em um contexto mais amplo, levando em consideração três tipos de sujeitos – estudantes iniciantes, estudantes finalistas e profissionais. Os resultados deste estudo nos daria *feedback* sobre como os diferentes sujeitos reagem a resolução de um problema mal definido e quais as diferenças nas ações desses sujeitos.

Muitas universidades têm mobilizado esforços para oferecer cursos na modalidade à distância. Assim, é importante que as boas práticas aplicadas no ensino presencial, estejam também disponíveis no contexto virtual. No que diz respeito a POP, estudos devem ser realizados para verificar a adaptação desta metodologia no contexto virtual e as necessidades de recursos para mediação das interações entre os participantes.

Em nosso trabalho de pesquisa, nós avaliamos POP no contexto de estudantes iniciantes de programação. Entretanto, como o ciclo de resolução de problemas de POP é composto por atividades típicas da Engenharia de Software, nós acreditamos que ele possa ser utilizado de forma efetiva em outras disciplinas do currículo, como por exemplo, as disciplinas posteriores de programação, po-

dendo causar um efeito cascata positivo no decorrer do curso de graduação. Para que estas suposições sejam averiguadas é necessária a realização de estudos experimentais, os quais devem colaborar também para melhorias de POP e adaptações, se necessárias, para estas disciplinas.

As atividades de POP podem inspirar trabalhos também na área de processamento de linguagem natural, provendo recursos para automatizar o diálogo do estudante com o cliente-tutor, que passaria a ser um cliente-tutor virtual. Podem colaborar para esta pesquisa os trabalhos realizados em sistemas de tutoria inteligente baseados em linguagem natural, tais como, os desenvolvidos por [Lane 2004] e [VanLehn et al. 2007].

Sistemas *online* para submissão de programas e execução automática de testes são também úteis para auxiliar o professor na recepção e correção dos programas dos estudantes. No caso de POP, os estudantes poderiam submeter seus programas via sistema *online*. Esse sistema executaria testes automáticos para verificar se os requisitos solicitados pelos clientes-tutores foram atendidos. Os resultados auxiliariam o professor na avaliação dos programas. Cabe destacar, que este tipo de sistema daria suporte a submissão e correção dos programas, independente da adoção de POP.

Referências Bibliográficas

ACM/IEEE (2004). *Software Engineering 2004,Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*.

Blaha, K., Monge, A., Sanders, D., Simon, B., and VanDeGrift, T. (2005). Do students recognize ambiguity in software design? a multi-national, multi-institutional report. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 615–616, New York, NY, USA. ACM.

Brown, D. A. (1988). Requiring CS1 students to write requirements specifications: a rationale, implementation suggestions, and a case study. In *SIGCSE '88: Proceedings of the nineteenth SIGCSE technical symposium on Computer science education*, pages 13–16, New York, NY, USA. ACM.

Carew, D., Exton, C., and Buckley, J. (2005). An empirical investigation of the comprehensibility of requirements specifications. In *International Symposium on Empirical Software Engineering (ISESE 2005), Noosa Heads, Australia*, pages 256 – 265.

Conn, R. (2002). Developing software engineers at the c-130j software factory. *IEEE Softw.*, 19(5):25–29.

De Lucena, V. F. J., Brito, A., and Gohner, P. (2006). A germany-brazil experience report on teaching software engineering for electrical engineering undergraduate students. In *CSEET '06: Proceedings of the 19th Conference on Software Engineering Education & Training*, pages 69–76, Washington, DC, USA. IEEE Computer Society.

de Oliveira, U. (2008). *Programando em C Volume I: Fundamentos*. Ciências Modernas, 1 edition.

Deek, F. P. (1997). *An integrated environment for problem solving and problem development*. PhD thesis, New Jersey Institute of Technology, Newark.

Deek, F. P., Turoff, M., and Hugh, J. M. (1999). A common model for problem solving and program development. *IEEE Transactions in Education*, 42(4).

Desai, C., Janzen, D., and Savage, K. (2008). A survey of evidence for test-driven development in academia. *SIGCSE Bull.*, 40(2):97–101.

Desai, C., Janzen, D. S., and Clements, J. (2009). Implications of integrating test-driven development into cs1/cs2 curricula. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 148–152, New York, NY, USA. ACM.

Dijkstra, E. W. (1976). On the teaching of programming, i. e. on the teaching of thinking. In *Language Hierarchies and Interfaces, International Summer School*, pages 1–10, London, UK. Springer-Verlag.

Dillenbourg, P., Baker, M., Blaye, A., and O'Malley, C. (1996). The evolution of research on collaborative learning. *In E. Spada & P. Reiman (Eds) Learning in Humans and Machine: Towards an interdisciplinary learning science. Oxford: Elsevier.*, pages 189–211.

Eastman, E. G. (2003). Fact-based problem identification precedes problem solving. *Consortium for Computing Sciences in Colleges: Journal of Computing Sciences in Colleges*, 19(2):18–29.

Eckel, B. (2006). *Thinking in Java*. Prentice Hall, 4th edition.

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., and Zander, C. (2006). Can graduating students design software systems? *ACM: SIGCSE Bull.*, 38(1):403–407.

Falkner, K. and Palmer, E. (2009). Developing authentic problem solving skills in introductory computing classes. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 4–8, New York, NY, USA. ACM.

Farrer, H. (1999). *Pascal Estruturado*. LTC, 3a edition.

Finney, K. (1996). Mathematical notation in formal specification: Too difficult for the masses? *IEEE Trans. Softw. Eng.*, 22(2):158–159.

Garg, K. and Varma, V. (2008). Software engineering education in india: Issues and challenges. *IEEE 21st Conference on Software Engineering Education and Training (CSEET '08)*, pages 110–117.

Hall, T., Beecham, S., and Rainer, A. (2002). Requirements problems in twelve software companies: an empirical analysis. In *IEEE Proceedings Software*, volume Vol. 149.

Hartmann, B., MacDougall, D., Brandt, J., and Klemmer, S. R. (2010). What would other programmers do: suggesting solutions to error messages. In *CHI '10: Proceedings of the 28th international conference on Human factors in computing systems*, pages 1019–1028, New York, NY, USA. ACM.

Hayes, J. R. and Simon, H. (1974). *Knowledge and Cognition*, chapter Understanding written problem instructions, pages 167–200. Potomac, Maryland: Lawrence Erlbaum Associates.

Hazzan, O. (2008). Reflections on teaching abstraction and other soft ideas. *ACM: SIGCSE Bull.*, 40(2):40–43.

Hazzan, O., Dubinsky, Y., Eidelman, L., Sakhnini, V., and Teif, M. (2006). Qualitative research in computer science education. *ACM: SIGCSE Bull.*, 38(1):408–412.

Heninger, K. L. (1980). Specifying software requirements for complex systems: New techniques and their application. *IEEE Trans. Softw. Eng.*, 6(1):2–13.

Hofmann, H. F. and Lehner, F. (2001). Requirements engineering as a success factor in software projects. *IEEE Softw.*, 18(4):58–66.

Hong, N. S. (1998). *The relationship between well-structured and ill-structured problem solving in multimedia silulation*. PhD thesis, Pennsylvania State University.

IEEE (2004). *Guide to the Software Engineering Body of Knowledge – SWEBOK*. IEEE Computer Society, Los Alamitos, California.

IEEE/ACM (2001). Computing curricula 2001 computer science. Technical report, The Joint Task Force on Computing Curricula IEEE Computer Society Association for Computing Machinery.

Jain, R. (1991). *The art of computer systems performance analysis : techniques for experimental design, measurement, simulation, and modeling*.

Janzen, D. and Saiedian, H. (2008). Test-driven learning in early programming courses. *ACM: SIGCSE Bull.*, 40(1):532–536.

Janzen, D. S. and Saiedian, H. (2006a). Test-driven learning: intrinsic integration of testing into the cs/se curriculum. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 254–258, New York, NY, USA. ACM.

Janzen, D. S. and Saiedian, H. (2006b). Test-driven learning: intrinsic integration of testing into the cs/se curriculum. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 254–258, New York, NY, USA. ACM.

Jedlitschka, A. and Pfahl, D. (2005). Reporting guidelines for controlled experiments in software engineering. *International Symposium on Empirical Software Engineering*, 0:95 – 104.

Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational Technology: Research & Development*, 48(4):63–85.

Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., Emam, K. E., and Rosenberg, J. (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28:721– 734.

Kitchenham, B., Pickard, L., and Pfleeger, S. L. (1995). Case studies for method and tool evaluation. *IEEE Softw.*, 12(4):52–62.

Lane, H. C. (2004). *Natural Language Tutoring and the Novice Programmer*. PhD thesis, University of Pittsburg. Department of Computer Science.

Leitão, S. (2000). The potential of argument in knowledge building. *Human Development*, 43:332 – 360.

Lethbridge, T. C., Diaz-Herrera, J., LeBlanc, R. J. J., and Thompson, J. B. (2007). Improving software practice through education: Challenges and future trends. In *FOSE '07: 2007 Future of Software Engineering*, pages 12–28, Washington, DC, USA. IEEE Computer Society.

Lynch, C. F., Ashley, K. D., Aleven, V., and Pinkwart, N. (2006). Defining "ill-defined domains": A literature survey. *Proceeding of the Workshop on Intelligent Tutoring Systems for Ill-Defined Domains. 8th International Conference on Inteligent Tutoring Systems.*, pages 1–10.

Mazarío, I. (1999). La resolucion de problemas: un reto para la educacion matematica contemporánea. *Revista Cubana de Educación Superior. La Habana*, Vol. XIX(2).

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM: SIGCSE Bull.*, 33(4):125–180.

MEC/SESU (1999). Diretrizes curriculares de cursos da Área de computação e informática. Technical report.

Mendonça, A. P. (2008). Entendimento de problemas: Uma investigação exploratória com alunos iniciantes de programação. Technical Report DSC/005/2008, Departamento de Sistemas e Computação. Coordenação de Pós-Graduação em Ciência da Computação. Universidade Federal de Campina Grande.

Mendonça, A. P., de Oliveira, C., Guerrero, D. S., and de B. Costa, E. (2009a). Difficulties in solving ill-defined problems: a case study with introductory computer programming students. *Proceedings of the 39th IEEE international conference on Frontiers in education conference. San Antonio, Texas.*, pages 1171–1176.

Mendonça, A. P., Guerrero, D. S., and Costa, E. (2009b). An approach for problem specification and its application in an introductory programming course. *Proceedings of the 39th IEEE international conference on Frontiers in education conference. San Antonio, Texas*, pages 1529 – 1534.

Meyer, B. (1992). Applying "design by contract". *IEEE Computer Society Press: Computer*, 25(10):40–51.

Meyer, B. (2003). The outside-in method of teaching introductory programming. In *Ershov Memorial Conference, volume 2890 of Lecture*, pages 66–78. Springer-Verlag.

Muller, O., Ginat, D., and Haberman, B. (2007). Pattern-oriented instruction and its influence on problem decomposition and solution construction. *ACM: SIGCSE Bull.*, 39(3):151–155.

Nienaltowski, M.-H., Pedroni, M., and Meyer, B. (2008). Compiler error messages: what can help novices? In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 168–172, New York, NY, USA. ACM.

NIST/SEMATECH (2003). e-Handbook of Statistical Methods. http://www.itl.nist.gov/div898/handbook/, accessed on may 2010.

O'Kelly, J. and Gibson, J. P. (2006). Robocode & problem-based learning: a non-prescriptive approach to teaching programming. *ACM: SIGCSE Bull.*, 38(3):217–221.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. (2007). A survey of literature on the teaching of introductory programming. In *ITiCSE-WGR '07:*

Working group reports on ITiCSE on Innovation and technology in computer science education, pages 204–223, New York, NY, USA. ACM.

Peres, F. and Meira, L. (2008). Dialogismo: a idéia de gênero discursivo aplicada ao desenvolvimento de software educativo. *XIX Simpósio Brasileiro de Informática na Educação (SBIE). Fortaleza, Ceará.*

Pietarinen, A.-V. (2005). Relevance theory through pragmatic theories of meaning. *In: Proceedings of the XXVII Annual Meeting of the Cognitive Science Society. Lawrence Erlbaum, Alpha*, pages 1767–1772.

Pólya, G. (1975). *How to Solve It: A New Aspect of Mathematical Method*. Princeton University Press, 2 edition.

Pólya, G. (1978). *A Arte de Resolver Problemas: um novo aspecto do método matemático. Tradução Heitor Lisboa de Araújo (2a Reimpressão)*.

Porter, A. A., Votta, Jr., L. G., and Basili, V. R. (1995). Comparing detection methods for software requirements inspections: A replicated experiment. *IEEE Trans. Softw. Eng.*, 21(6):563–575.

Pozo, J. I., del Puy P. Echeverría, M., Castillo, J. D., Ángel G. Crespo, M., and Angón, Y. P. (1998). *A solução de problemas: aprender a resolver, resolver para aprender*. Porto Alegre: ArtMed.

Rahman, S. M. and Juell, P. L. (2006). Applying software development lifecycles in teaching introductory programming courses. In *CSEET '06: Proceedings of the 19th Conference on Software Engineering Education & Training*, pages 17–24, Washington, DC, USA. IEEE Computer Society.

Reed, D. (2002). The use of ill-defined problems for developing problem-solving and empirical skills in CS1. *J. Comput. Small Coll.*, 18(1):121–133.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164.

Shull, F., Rus, I., and Basili, V. (2000). How perspective-based reading can improve requirements inspections. *Computer*, 33(7):73–79.

Siegel, S. and Jr., N. J. C. (1988). *Nonparametric Statistics for The Behavioral Sciences*. McGraw-Hill.

Valentine, D. W. (2004). Cs educational research: a meta-analysis of sigcse technical symposium proceedings. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 255–259, New York, NY, USA. ACM.

VanLehn, K. (1989). *Foundations of Cognitive Science*, chapter Problem solving and cognitive skill acquisition, pages 526–579. Cambridge, MA: M. I. T. Press.

VanLehn, K., Graesser, A. C., Jackson, G. T., Jordan, P., Olney, A., and Rosé, C. P. (2007). When are tutorial dialogues more effective than reading? *Cognitive Science*, 31:3–62.

Weinberger, A. and Fischer, F. (2006). A framework to analyze argumentative knowledge construction in computer-supported collaborative learning. *Computers &. Education*, 46(1):71–95.

Wohlin, C., Runeson, P., Host, M., Ohlsson, C., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering: an Introduction*. Kluver Academic Publishers.

Yin, R. K. (1984). *Case Study Reseach Design and Methods*. Sage Publications.

Zelle, J. M. (2004). *Python Programming: An Introduction to Computer Science*. Franklin, Beedle & Associates, 1 edition.

Apêndice A

POP – Orientações para Professores

Neste capítulo, nós apresentamos de forma sistemática como o professor deve proceder para incluir POP na disciplina introdutória de programação. Para fazermos isso, organizamos as atividades em quatro etapas: *planejamento*, *preparação*, *execução* e *avaliação.*

Planejamento. Corresponde a etapa em que o professor planeja a inserção de POP na disciplina. Na descrição desta etapa, nós apresentamos ao professor as orientações necessárias para incluir as atividades de resolução de problemas de POP no plano de aula da disciplina;

Preparação. Etapa que antecede a execução do ciclo de resolução de problemas de POP. Na descrição desta etapa, nós apresentamos ao professor as orientações sobre como proceder para: (i) elaborar problemas mal definidos que serão propostos aos estudantes; (ii) produzir os artefatos de referência; (iii) definir os *deadlines*; (iv) definir os recursos para interação do grupo e documentação dos requisitos; (v) dividir a turma em grupo; (vi) selecionar e orientar os clientes-tutores; e, (vii) preparar os estudantes para a execução de POP.

Execução. Corresponde a execução do ciclo de resolução de problemas de POP, conforme apresentamos no Capítulo 3;

Avaliação. Na descrição desta etapa, apresentamos orientações sobre como o professor deve proceder na avaliação dos estudantes e dos procedimento adotados na execução de POP.

A.1 Planejamento

Como dissemos anteriormente, é nesta etapa que o professor planeja a inclusão do ciclo de resolução de problemas de POP na disciplina. O planejamento inclui as seguintes atividades: (i) definição do cronograma da disciplina e (ii) definição dos critérios de avaliação.

A.1.1 Definição do Cronograma

Cabe ao professor estabelecer o cronograma da disciplina e especificar dentro desse cronograma quantos e em quais momentos o ciclo de resolução de problemas de POP será executado.

Para isso, o professor deve levar em consideração as seguintes informações:

Disciplina Teórica e/ou Prática. As atividades de POP podem ser totalmente aplicadas em uma disciplina prática ou em conjunto com uma disciplina teórica. Neste último caso, a especificação dos requisitos deve ser feita na disciplina teórica e a implementação na disciplina prática;

Tempo requerido para executar POP. O professor necessitará apenas de duas semanas para executar POP na disciplina. Uma semana, para preparar os recursos materiais e humanos e a outra para executar o ciclo de resolução de problema de POP. Esse tempo pode ser ajustado de acordo com as necessidades do professor. No que diz respeito aos recursos materiais e humanos o professor precisará: (i) elaborar o problema mal definido que será proposto aos estudantes; (ii) produzir os artefatos de referência; (iii) definir os deadlines; (iv) definir os recursos para interação do grupo e documentação dos requisitos; (v) dividir a turma em grupo; (vi) selecionar e orientar os clientes-tutores; e, (vii) preparar os estudantes para a execução de POP. Todas estas atividades são descritas no Apêndice A.2.

No caso em que o professor deseje executar o ciclo de POP em diferentes momentos na disciplina, o tempo requerido para preparar as próximas execuções será minimizado, pois o professor precisará apenas elaborar o problema mal definido (i), os artefatos de referência (ii), definir *deadlines* (iii) e selecionar e orientar os clientes-tutores (vi), no caso em que ele não puder contar com os mesmos clientes-tutores que participaram anteriormente do ciclo de POP. Neste caso, estamos considerando que o professor vai reaproveitar os demais recursos estabelecidos anteriormente.

O momento de inserir POP na disciplina. POP deve ser introduzida na disciplina após os estudantes terem aprendido e praticado estruturas de repetição. Isso porque, a partir desse conteúdo o professor pode elaborar problemas que sejam mais interessantes;

Quantidade de problemas a serem utilizados. Para que os resultados de POP sejam mais efetivos, é importante que os estudantes resolvam, no mínimo, três problemas mal definidos. Essa quantidade se justifica porque é necessário que os estudantes repitam os procedimentos a fim de assimilarem

a forma de proceder nas atividades (elaborar especificação inicial, iniciar implementação, concluir especificação e concluir implementação). Os problemas podem ser propostos em momentos distintos na disciplina, ou podem fazer parte de uma mesma lista de problemas. Neste último caso, deve-se ter cuidado em estabelecer adequadamente os *deadlines*, a fim de não sobrecarregar os estudantes;

Natureza dos problemas. O professor deve utilizar problemas de natureza diferentes: matemáticos, jogos e sistemas de informação. Isso permite que os estudantes estejam atentos à especificação de diferentes requisitos. Por exemplo, em problemas matemáticos, os requisitos seguem, em geral, o seguinte padrão: entrada, processamento e saída. Em problemas envolvendo jogos, além desses requisitos, os requisitos não funcionais tornam-se bastante evidentes;

Complexidade dos problemas. O nível de complexidade dos problemas é um aspecto difícil de julgar. Problemas muito fáceis podem resultar em desinteresse por parte dos estudantes. Por outro lado, problemas muito complexos podem afetar a auto-estima dos estudantes e sua predisposição para resolvê-los, por se considerarem incapazes. O mais importante sobre esse aspecto, é que o professor proponha problemas cujo embasamento de conceitos e procedimentos já tenham sido contemplados na disciplina e que estes problemas permitam aos estudantes, gradativamente, aperfeiçoar sua aprendizagem de programação e tratar com entendimento e especificação de requisitos.

A.1.2 Definição dos Critérios de Avaliação

Não é nosso objetivo com POP impor uma forma de avaliação ou critérios de atribuição de nota, pois isso cabe ao professor da disciplina e depende também de suas crenças. No entanto, apresentamos orientações para que os professores possam refletir sobre elas quando forem determinar seus critérios de avaliação, considerando a inserção de POP em sua disciplina.

Como os estudantes produzem outros artefatos além do programa, é relevante que o professor considere na avaliação todos os artefatos produzidos. Isso possibilitará:

- Ao próprio professor, conhecer as dificuldades dos estudantes e reforçar o ensino dos conteúdos que se fizerem necessários;
- Aos estudantes, ter *feedback* sobre a qualidade dos artefatos produzidos; identificar os aspectos que precisam ser aprendidos ou aperfeiçoados; e, valorizar os produtos que resultam de cada fase da Engenharia de Software, por exemplo: da especificação do requisitos, o documento de especificação; da verificação, os testes; da codificação, o programa.

Além disso, ao considerar na avaliação todos os artefatos, o professor minimiza as chances dos estudantes elegerem um artefato em detrimento de outro, que não tinha atribuição de nota.

Segue uma descrição dos aspectos que consideramos relevantes avaliar em cada artefato.

Documento de Especificação

Organização do documento. Avaliar se o documento de especificação esta organizado de maneira que facilite a leitura e a identificação dos requisitos. Observações devem ser feitas com relação a formatação do documento: tipo e tamanho de fonte, espaçamento entre as linhas, cores, divisão das seções e sub-seções, manutenção de um formato padrão, etc;

Correção da escrita. Avaliar se o documento de especificação descreve corretamente os requisitos. Neste caso, deve-se observar se a especificação não contém erros grosseiros de ortografia, concordância e se os requisitos não apresentam defeitos, tais como, contradições, ambigüidades e informações incorretas;

Completude do documento. Avaliar se o documento contempla todos os requisitos que foram solicitados pelo cliente-tutor;

Descrição de Testes de Entrada/Saída. Avaliar a quantidade de requisitos cobertos pelos casos de testes de entrada e saída. Nesse caso, deve-se verificar a quantidade de requisitos representados por meio dos casos de testes de entrada e saída.

Programa

Atendimento aos requisitos. Avaliar a quantidade de requisitos implementados no programa, tendo como referência os requisitos desejados (estabelecidos nos artefatos de referência). O resultado dessa avaliação sinaliza ao professor algumas dificuldades dos estudantes com respeito a aprendizagem dos conceitos de programação e de implementação de programas;

Atendimento às boas práticas de programação. Os programas devem ser também avaliados com respeito a adoção das boas práticas de programação, por exemplo: nome significativos para as variáveis e funções, legibilidade do código e boa endentação. Esse *feedback* permite aos estudantes melhorarem suas habilidades de programação.

Testes Automáticos

Cobertura dos testes. Avaliar a quantidade de requisitos cobertos pelos casos de testes automáticos;

Qualidade dos testes. Avaliar se os testes exploram diferentes situações de erros, se ele não são redundantes, se testam apenas os casos óbvios.

É importante observarmos que esses critérios podem ser adotados na disciplina de programação até mesmo independente do uso de POP, como por exemplo, os critérios mencionados para avaliação do programa.

A.2 Preparação

Na etapa de preparação, o professor deve prover os recursos humanos e materiais necessários à execução do ciclo de resolução de problemas de POP. Nesta etapa, o professor deve realizar as seguintes atividades: i) elaborar o problema mal definido que será proposto aos estudantes; (ii) produzir os artefatos de referência; (iii) definir os *deadlines*; (iv) definir os recursos para interação do grupo e documentação dos requisitos; (v) dividir a turma em grupo; (vi) selecionar e orientar os clientes-tutores; e, (vii) preparar os estudantes para a execução de POP.

A.2.1 Elaborar o Problema Mal definido

Cabe ao professor elaborar o problema mal definido que será proposto aos estudantes. O enunciado do problema mal definido deve conter, intencionalmente, um ou uma combinação de "defeitos" – ambigüidades, contradições, falta de informações, informações incorretas e/ou irrelevantes.

Para elaborar o problema mal definido, o professor pode adotar duas estratégias: (i) partir de um problema bem definido, já conhecido, e re-elaborar o enunciado, inserido nele um ou uma combinação de "defeitos"; ou (ii) conceber o enunciado sem que haja um problema bem definido de referência.

Para exemplificamos o primeiro caso, vamos supor que o professor parta do problema do triângulo cujo enunciado bem definido é expresso no Quadro A.1 (a). Para construir uma versão mal definida, o professor pode omitir informações do enunciado bem definido e ainda introduzir ambigüidades, conforme apresentamos no Quadro A.1 (b).

Considerando o segundo caso, no qual não há um problema bem definido de referência, o professor deve obedecer os seguintes passos:

- Pensar em um domínio ou situação específica que daria um problema interessante, por exemplo, *financiamento habitacional*;
- Fazer um *brainstorming* de alguns requisitos que o programa deve satisfazer para resolver o problema;
- Refinar as idéias obtidas no *brainstorming*. Neste caso, o professor deve eliminar os requisitos que julga não oportunos para o momento e detalhar um pouco mais aqueles que pretende ver implementados nos programas dos estudantes;
- Elaborar o enunciado, inserindo nele alguns "defeitos".

Quadro A.1: Exemplo de problema bem e mal definido.

Problema do Triângulo – Bem Definido

Faça um programa que leia da entrada padrão os valores correspondentes às medidas dos lados de um triângulo. Verifique se esses valores formam um triângulo e, se formarem, determine se ele é eqüilátero, isósceles ou escaleno. Na saída deve ser impressa uma mensagem informando o tipo do triângulo e, no caso, em que as medidas não formem um triângulo deve ser exibida a seguinte mensagem: "Nao eh triangulo".

Exemplos:

Entradas
lado 1 = 4
lado 2 = 9
lado 3 = 4
Saída
Nao eh triangulo
Entradas
lado 1 = 1.5
lado 2 = 2
lado 3 = 2.5
Saída
Triangulo escaleno

(a)

Problema do Triângulo – Mal Definido

Dadas as medidas de um triângulo determine o tipo dele e apresente-o impresso na tela.

Esse problema apresenta falta de informações e ambigüidades.
Falta de informação: não são informados, por exemplo, se o triângulo deve ser classificado quanto aos lados ou quanto aos ângulos; a que se referem as medidas e também qual a formatação desejada na entrada;
Ambigüidades: por exemplo, a expressão *"apresentando-o impresso na tela"* está se referindo ao desenho do triângulo ou apenas a impressão de uma frase indicando o tipo de classificação do triângulo?

(b)

Nós apresentamos um exemplo de enunciado mal definido para o problema de financiamento habitacional, como segue:

> O Banco Imobiliário tem um plano de financiamento habitacional chamado "Lar doce Lar" que foi idealizado com o intuito de ajudar as pessoas a realizarem o sonho de adquirir a casa própria. Para colocá-lo em prática, o Banco deseja criar um simulador e deixá-lo disponível para a população. A idéia é que as pessoas que desejam adquirir crédito para a compra da casa própria possam simular os valores das parcelas do financiamento de forma rápida e fácil, bastando apenas informar o valor do imóvel que desejam adquirir e o número de parcelas em que pretendem pagar a dívida. As parcelas possuem taxa de juros fixas, assim todas as pessoas podem se beneficiar do financiamento habitacional. Manter valores de parcelas fixas é uma política do banco para evitar inadimplência dos solicitantes maiores de idade e tornar o financiamento acessível para a maioria da população. Suponha que você é o programador contratado pelo Banco Imobiliário e, portanto, deve entregar um programa que atenda às necessidades do referido Banco.

A.2.2 Produzir Artefatos de Referência

Após a criação do enunciado do problema, o professor deve elaborar os artefatos de referência – especificação, testes e/ou programa.

Especificação de Referência. É uma descrição textual que especifica os requisitos que o programa deve atender para solucionar o problema proposto. Estabelecer uma especificação de referência garante que o problema terá o mesmo grau de complexidade independente de quem será o cliente-tutor. A especificação de referência deve ser organizada de modo a permitir a rápida identificação dos requisitos pelos clientes-tutores. Nós apresentamos um exemplo de especificação de referência para o problema do financiamento habitacional no Apêndice A6.

Testes de Referência. É um conjunto de casos de testes que tem por objetivo detectar erros no programa, com relação ao atendimento dos requisitos. Os casos de testes reforçam a especificação de referência e também servem de base para avaliação dos programas. Eles podem ser disponibilizados pelo professor, na forma de testes automáticos, ou na forma de testes de entrada/saída, descritos na própria especificação de referência.

Implementação de Referência. É um programa que implementa todos os requisitos desejados e passa nos casos de testes de referência. O programa de referência funciona como uma espécie de "oráculo" no qual o cliente-tutor pode se basear caso tenha alguma dúvida sobre os requisitos.

O objetivo dos artefatos de referência é especificar os requisitos que os programas dos estudantes

devem atender para solucionar o problema proposto. Esses artefatos são utilizados pelos clientes-tutores para responder aos questionamentos dos estudantes sobre os requisitos do programa. Na maioria dos casos, a definição de uma *especificação de referência*, descrevendo também alguns casos de testes, são suficientes para esclarecer os requisitos. Portanto, o programa é imprescindível apenas nos casos em que os recursos anteriores deixam dúvidas sobre o que deve ser feito.

A.2.3 Definir *Deadlines*

O professor deve definir as datas para apresentação e entrega dos artefatos produzidos pelos estudantes em cada atividade do ciclo de resolução de problemas (Capítulo 3). Como apresentamos no Quadro A.2, no primeiro *deadline* os estudantes devem entregar a versão inicial da especificação dos requisitos e casos de testes de entrada/saída (artefatos produzidos na atividade *elaborar especificação inicial*). No segundo *deadline*, eles devem apresentar os protótipo de seus programas e testes automáticos e entregar a versão final do documento de especificação e dos casos de testes de entrada/saída (artefatos produzidos nas atividades *iniciar implementação* e *concluir especificação*). No terceiro *deadline*, os estudantes devem entregar a implementação dos programas e casos de testes automáticos, produzidos na atividade *concluir implementação*.

Quadro A.2: *Deadlines* e *deliverables*.

Deadline	***Deliverables***
1^0 *deadline*	Entrega da versão inicial do documento de especificação e dos casos de testes de entrada/saída. Esta entrega deve ser efetivada na primeira reunião com o cliente-tutor.
2^0 *deadline*	Apresentação dos protótipos e casos de testes automáticos e entrega da versão final do documento de especificação e dos casos de testes de entrada/saída. A apresentação e entrega dos artefatos deve ocorrer na segunda reunião com o cliente-tutor.
3^0 *deadline*	Entrega da implementação dos programas e testes automáticos.

A definição dos prazos deve levar em consideração a quantidade e complexidade dos problemas que os estudantes terão que resolver, a estimativa de tempo disponível dos alunos para a disciplina e a própria característica da turma. Em função desses aspectos, o professor pode ajustar os deadlines definidos no Quadro A.2.

A.2.4 Definir Recursos para Interação do Grupos e Documentação dos Requisitos

Além da linguagem e ambiente de programação, o professor deve definir um editor de texto colaborativo para os estudantes documentarem os requisitos e uma lista de discussão *online* para que os estudantes possam interagir com seu respectivo grupo e cliente-tutor após o horário de sala de aula.

Para a escolha do editor de texto colaborativo e da lista de discussão *online*, é importante que o professor leve em consideração os seguintes critérios: facilidade de uso, estabilidade das ferramentas e conhecimento das ferramentas por parte dos estudantes. Quanto mais próximo a ferramenta escolhida for daquelas que os estudantes conhecem, mais atrativo será o trabalho com elas.

A.2.5 Dividir a Turma em Grupos

Com POP, a especificação dos requisitos é realizada com os estudantes trabalhando em pequenos grupos. Desta forma, o professor deve estabelecer a divisão da turma em grupos, obedecendo os seguintes critérios: cada grupo deve ser composto de 5 ± 2 estudantes e um cliente-tutor deve ser nomeado para cada grupo. No primeiro dia do ciclo de resolução de problemas, a turma deve ser organizada de acordo com o estabelecido no plano de divisão da turma.

Caso o professor inclua POP em vários momentos na disciplina, é importante modificar a formação dos grupos para que os estudantes possam ter oportunidade de trabalhar com diferentes pessoas.

A.2.6 Selecionar e Orientar Clientes-Tutores

A divisão da turma em grupos dará ao professor a noção exata do número de clientes-tutores necessário para realizar o ciclo de resolução de problemas de POP. A partir disso, o professor deve selecionar e orientar os clientes-tutores para sua adequada atuação junto aos estudantes.

Para seleção dos clientes-tutores, o professor deve levar em consideração as seguintes informações:

Quem pode ser cliente-tutor. O papel de cliente-tutor pode ser assumido pelo professor, pelos monitores da disciplina e/ou por outros estudantes. Neste último caso, os estudantes já devem ter cursado a disciplina de programação. É uma boa oportunidade para engajar estudantes que estão cursando disciplinas como Análise e Projeto de Sistemas, Engenharia de Software e afins. No caso, em que os monitores da disciplina exercem a função de clientes-tutores, eles também devem auxiliar

os estudantes, mitigando suas dúvidas sobre programação.

Habilidades que o cliente-tutor precisa ter. A principal habilidade requerida em um cliente-tutor é ter boa comunicação. É importante que o cliente-tutor tenha facilidade de se comunicar e mediar as interações do grupo. Há pessoas que são tímidas para se comunicar em público, considerando uma grande platéia, mas que são bastante efetivas na comunicação em pequenos grupos. Outro aspecto importante é que o cliente-tutor deve estar consciente de que não deve entregar, deliberadamente, requisitos aos estudantes.

Quantidade de grupos que o cliente-tutor pode ser responsável. A situação ideal é que tenha um cliente-tutor para cada grupo. Entretanto, é importante considerarmos que nem sempre isso é possível, dado que as turmas de programação introdutórias, em geral, são numerosas. Assim, nos casos em que professor não dispuser de clientes-tutores em número suficiente, ele poderá designar um mesmo cliente-tutor para no máximo dois grupos. Nas datas agendadas para as reuniões, os clientes-tutores devem atender aos dois grupos, separadamente.

Para orientação dos clientes-tutores, o professor deve realizar uma reunião presencial e nela prover todas as informações necessárias à execução do ciclo de resolução de problemas: objetivo de POP, modo de trabalho, enunciado do problema mal definido, artefatos de referência, *deadlines* e *deliverables*, recursos de suporte as atividades, identificação do grupo para o qual o cliente-tutor foi designado e orientações para interação com o grupo.

Com relação a interação com o grupo, nós apresentamos no Apêndice B8 algumas estratégias de diálogo que fornecem orientações sobre como os clientes-tutores devem conduzir as interações com os estudantes. Para definição destas estratégias, nós observamos as interações de alunos e professores por meio das seguintes investigações: (i) estudo de caso realizado com alunos iniciantes de programação [Mendonça 2008]; e, (ii) lista de discussão da disciplina introdutória de programação da UFCG.

Cabe destacarmos que essas observações não tiveram por objetivo fazer análises estatísticas das interações, mas analisá-las para detectar distorções e gerar orientações mais sistemáticas para potencializar o diálogo entre os estudantes e entre os estudantes e seus respectivos clientes-tutores.

Por fim, cabe destacarmos que é muito importante que os clientes-tutores conheçam bem a especificação de referência e possam tirar suas dúvidas antes que o ciclo de resolução de problemas seja iniciado. Assim, após a reunião, caso os clientes-tutores apresentem outras dúvidas, o professor pode, dada a demanda, marcar uma nova reunião presencial ou esclarecer as dúvidas, virtualmente, por meio de uma lista de discussão.

A.2.7 Preparar Estudantes

No caso em que POP estiver sendo utilizada primeira vez na disciplina, o professor deve preparar os estudantes para a realização das atividades. Isso se faz necessário porque, com POP, incorporamos outras atividades diferentes daquelas que os estudantes estão acostumados, principalmente no que diz respeito ao uso de problemas mal definidos e a especificação de requisitos.

A melhor forma de preparar os estudantes é simularmos com eles a especificação de um programa. Para fazer isso, o professor deve:

- Escolher um problema mal definido e levar para sala de aula;
- Explicar o que são problemas mal definidos e qual a importância de esclarecer requisitos com o cliente;
- Fazer de conta que é um cliente e pedir para os estudantes da turma exercerem o papel de desenvolvedores de software;
- Escolher um estudante para ir registrando os requisitos discutidos em sala de aula;
- Estimular os estudantes a pensarem sobre o problema e a questionarem o cliente;
- Dar dicas aos estudantes sobre como documentar os requisitos;
- Disponibilizar, ao final, para toda a turma o documento de especificação que foi construído. Em geral, os estudante usam o documento construído em sala de aula como um exemplo a seguir, então é importante que ele esteja organizado e contenha os elementos necessários a especificação do programa. Nós apresentamos um exemplos de especificação no Apêndice C5;
- Discutir com a turma potenciais soluções para o problema.

A.3 Execução

Uma vez que o professor cumpriu as atividades da etapa de preparação, ele pode dar início ao ciclo de resolução de problemas de POP. Este ciclo está descrito detalhadamente no Capítulo 3 e deve ser aplicado de acordo com o planejamento da disciplina (Apêndice A1). No Quadro A.3, nós apresentamos uma síntese do ciclo de resolução de POP.

Quadro A.3: Síntese do ciclo de resolução de problemas de POP.

Atividade: Elaborar especificação inicial. Participantes: Cliente-tutor e grupo de desenvolvedores. Insumos: Problema mal definido e artefatos de referência. Produtos: Especificação inicial dos requisitos e casos de testes de entrada/saída.
Atividade: Iniciar implementação. Participantes: Desenvolvedores (trabalhando individualmente). Insumos: Especificação inicial dos requisitos e casos de testes de entrada/saída. Produtos: Protótipo do programa e casos de testes automáticos.
Atividade: Concluir especificação. Participantes: Cliente-tutor e grupo de desenvolvedores. Insumos: Problema mal definido, artefatos de referência, especificação inicial, casos de testes de entrada/saída, protótipo dos programas e testes automáticos. Produtos: Especificação final e casos de testes de entrada/saída.
Atividade: Concluir implementação. Participantes: Desenvolvedores (trabalhando individualmente) Insumos: Especificação final e casos de testes de entrada/saída, protótipo do programa e casos de testes automáticos. Produtos: Programa e testes automáticos.

A.4 Avaliação

Após a execução de POP na disciplina, o professor deve prover a avaliação dos estudantes, com respeito aos artefatos produzidos e também coletar informações de estudantes e clientes-tutores sobre as atividades desenvolvidas.

No que diz respeito a avaliação, o professor deve considerar os critérios definidos na etapa de planejamento (Apêndice A.1.2). Caso o professor queira delegar a atividade de correção do documento de especificação aos clientes-tutores, ele poderá fazê-lo tendo o cuidado de preservar a uniformidade nas avaliações. Nesse caso, as orientações devem ser dadas na reunião com os clientes-tutores (Apêndice A.2.6).

Com relação a coleta de informações sobre a execução de POP na disciplina, o professor pode ter *feedback* dos clientes-tutores e estudantes de maneira informal, por meio de conversas e observações durante a execução do ciclo de resolução de problemas, ou pode realizá-la de maneira mais formal, por meio de questionários, por exemplo.

Independente da forma adotada, o objetivo desta atividade é coletar informações sobre as dificuldades dos estudantes e clientes-tutores em realizar as atividades de POP, os conceitos de programação que precisam ser melhor entendidos e reforçados na disciplina. Esses elementos irão compor as sugestões de melhoria, que por sua vez realimentam as atividades de ensino-aprendizagem na disciplina.

A.5 Orientações para Inserção de Testes Automáticos na Disciplina de Programação

Para incentivar os estudantes à prática de testes, é importante utilizar uma forma sistemática de introduzir e aprofundar testes na disciplina de programação. Cabe destacarmos, que não é objetivo de POP tornar os estudantes *experts* em testes, mas fazê-los praticar testes, percebendo-os não como uma atividade que está no fim do processo de desenvolvimento, mas que pode ser utilizada desde o início, inclusive como um importante instrumento para capturar requisitos.

Para a prática de testes automáticos, POP inspirou-se na abordagem denominada *Test-Driven Learning* (TDL), concebida por David Janzen e colaboradores [Janzen and Saiedian 2006b; 2008; Desai et al. 2009]. Baseados em TDL, nós apresentamos, nessa seção, algumas estratégias que favorecem o ensino-aprendizagem de testes automáticos para estudantes iniciantes. Nossos exemplos serão apresentados na linguagem Python, utilizando o comando `assert`. Ao executar esse comando, Python testa a condição escrita. Se ela for verdadeira, o comando é finalizado sem mais nenhuma conseqüência. Se a condição for falsa, Python imprimirá uma mensagem de erro.

No trabalho de David Janzen e colaboradores são apresentados exemplos utilizando a linguagem JAVA. Caso o professor utilize uma linguagem de programação diferente de Python e JAVA, recomendamos explorar os recursos da linguagem a fim de utilizar as estratégias que nós descreveremos.

Primeiro Passo

Primeiramente, utilize testes desde o início da disciplina. Uma forma simples de fazer isso é ensinar usando testes. Por exemplo, demonstrando a funcionalidade dos métodos da linguagem. No Quadro A.4, nós utilizamos testes por meio de `asserts` para demonstrar as funcionalidades dos métodos `len()` e `list()` na manipulação de *strings*.

Quadro A.4: Manipulação de strings: métodos `len()` e `list()`.

```
s = 'casa'
assert len(s) == 4
assert list(s) == ['c', 'a', 's', 'a']
s = 'c a s a'
assert len(s) == 7
assert list(s) == ['c', ' ', 'a', ' ', 's', ' ', 'a']
```

Essa estratégia deve ser utilizada de forma recorrente na disciplina. Com isso os estudantes aprendem a ler `asserts` e familiarizam-se com seu uso.

Segundo Passo

Solicite que os estudantes escrevam testes (usando `asserts`) para confirmar seu entendimento sobre o comportamento de certas funções, por exemplo da função `soma()`, apresentada abaixo. Essa função retorna um número inteiro que corresponde ao valor do argumento somado a um, no caso em que o argumento passado é um número positivo.

Código Fonte A.1: Função `soma()`.

```
1 def soma(x):
2     if (x > 0):
3         x = x + 1;
4     return x
```

Neste caso, os estudantes devem adicionar `asserts` para a função `soma()`, conforme alguns exemplos que apresentamos no Quadro A.5.

Quadro A.5: `Asserts` para a função `soma()`.

```
assert soma(2) == 3
assert soma(4) == 5
assert soma(3) == 4
assert soma(7) == 8
assert soma(-2) == -2
assert soma(-3) == -3
assert soma(0) == 0
```

Outro exemplo, é solicitar a criação de testes para a função `eh_par()` (Código Fonte A.2). Esta função recebe um inteiro como parâmetro e retorna um `boolean` para indicar se o número é par ou não.

Código Fonte A.2: Função `eh_par()`.

```
1 def eh_par(x):
2     if x % 2 == 0:
3         return True
4     else:
5         return False
```

Novamente, os estudantes devem criar `asserts` para a função `eh_par()`, conforme os exemplos que apresentamos no Quadro A.6.

Quadro A.6: `Asserts` para a função `eh_par()`.

```
assert eh_par(4)
assert not eh_par(9)
```

Terceiro Passo

Solicite que os estudantes escrevam `asserts` para suas próprias funções. É importante notarmos que à medida que os programas vão se tornando mais complexos, por exemplo, agregando um conjunto de funções, os estudantes passam a sentir mais dificuldades. Então, é importante que o professor também apresente exemplos de testes para programas gradativamente mais complexos.

Quarto Passo

Quando os estudantes estiverem familiarizados com `asserts` (leitura e escrita), o professor poderá introduzir um *framework* para execução automática de testes, com por exemplo, o PyUnit no caso de adoção da linguagem Python.

É importante que o professor elabore também atividades nas quais os estudantes sejam responsáveis por criarem testes e testarem um conjunto de programas. Por exemplo, solicitar que um grupo de estudantes teste os programas de outro grupo de estudantes. Essa atividade é importante para prover maior percepção da importância dos testes.

Quando os estudantes testam apenas o seu próprio código, em geral, eles acham mais prático realizar testes manualmente. No entanto, ao terem que testar um conjunto de programas eles perceberão que esta alternativa é pouco efetiva.

A utilização de testes automáticos deve ocorrer durante toda a disciplina de programação, independente do uso do ciclo de resolução de problemas de POP.

A.6 Especificação de Referência

1. Enunciado do Problema

O Banco Imobiliário tem um plano de financiamento habitacional chamado "Lar doce Lar" que foi idealizado com o intuito de ajudar as pessoas a realizarem o sonho de adquirir a casa própria. Para colocá-lo em prática, o Banco deseja criar um simulador e deixá-lo disponível para a população. A idéia é que as pessoas que desejam adquirir crédito para a compra da casa própria possam simular os valores das parcelas do financiamento de forma rápida e fácil, bastando apenas informar o valor do imóvel que desejam adquirir e o número de parcelas em que pretendem pagar a dívida. As parcelas possuem taxa de juros fixas, assim todas as pessoas podem se beneficiar do financiamento habitacional. Manter valores de parcelas fixas é uma política do banco para evitar inadimplência dos solicitantes maiores de idade e tornar o financiamento acessível para a maioria da população. Suponha que você é o programador contratado pelo Banco Imobiliário e, portanto, deve entregar um programa que atenda às necessidades do referido Banco.

2. Especificação do Programa

2.1. Entradas e Restrições das Entradas

Os seguintes dados devem ser informados pelos solicitantes do financiamento:

- Renda Bruta da família

  Renda bruta $\geq$ salário mínimo

- Idade do solicitante

  $18 \leq$ Idade ≤ 57

- Valor do imóvel que pretendem comprar

  R$ 1500,00 $\leq$ Imóvel $\leq$ R$ 220000,00

- Número de parcelas que desejam dividir o financiamento

  3 <= Número de parcelas <= 240 (meses)

2.2. Formatação das Entradas

```
Renda Bruta?
Idade?
```

```
Valor do Imovel?
Numero de Parcelas?
```

2.3. Mensagens de Erro para Entradas Inválidas

Renda Bruta

`Valor da renda fora do limite permitido. Digite outro valor.`

`Renda Bruta?` (deve permitir que o usuário digite um novo valor)

Idade

`Idade fora do limite permitido. Digite outro valor.`

`Idade?` (deve permitir que o usuário digite um novo valor)

Valor do Imóvel

`Valor do imovel fora do limite permitido. Digite outro valor.`

`Valor do Imovel?` (deve permitir que o usuário digite um novo valor)

Número de Parcelas

`Numero de parcela fora do limite permitido. Digite outro valor.`

`Numero de Parcelas?` (deve permitir que o usuário digite um novo valor)

2.4. Cálculos e Restrições do Financiamento

Taxa de juros = 0,5% a.m (juros simples)

valor final do imovel = valor do imovel + (valor do imovel $*$ taxa juros $*$ numero de parcelas) valor da parcela = valor final do imovel $/$ numero de parcelas

Restrições do financiamento:

Se valor do imóvel > 25 $*$ renda bruta e valor das parcelas > 25% da renda bruta

Então emitir mensagem `Financiamento nao pode ser concedido.`

Obs.: A mensagem deve ser exibida pulando-se uma linha dos valores de entrada.

2.5. Saídas e Formatações das Saídas

O programa deve exibir na saída o valor das parcelas e o valor final do imóvel, obedecendo a seguinte formatação.

`Valor das Parcelas:` `R$` <valor> (<valor> com duas casas decimais)

`Valor Final do Imovel:` `R$` <valor> (<valor> com duas casas decimais)

Obs.: As saídas devem ser exibidas pulando-se uma linha dos valores de entrada.

2.6. Outras Restrições

O programa deve permitir apenas uma simulação dos valores.

Supor usuário esperto, portanto não é necessário tratar exceções, tais como, usuário digitar letras ao invés de números para as entradas.

2.7. Alguns Testes de Entrada/Saída

OBS.: Os testes abaixo levaram em consideração o valor do salário mínimo igual a R$ 465,00.

Teste 1

```
Renda Bruta? 300.00
Valor da renda fora do limite permitido.  Digite outro valor.
Renda Bruta? 18000
Idade? 10
Idade fora do limite permitido.  Digite outro valor.
Idade? 98
Idade fora do limite permitido.  Digite outro valor.
Idade? 35
Valor do Imovel? 221000.98
Valor do imovel fora do limite permitido.  Digite outro valor.
Valor do Imovel? 220000
Numero de Parcelas? 2
Numero de parcelas fora do limite permitido.  Digite outro valor.
Numero de Parcelas? 120

Valor das Parcelas:  R$ 2933.33
Valor Final do Imovel:  R$ 352000.00
```

Teste 2

```
Renda Bruta? 465.00
Idade? 57
Valor do Imovel? 11625.20
```

```
Numero de Parcelas? 210

Financiamento nao pode ser concedido.
```

Teste 3

```
Renda Bruta? 465
Idade? 18
Valor do Imovel? 1500
Numero de Parcelas? 3

Financiamento nao pode ser concedido.
```

Teste 4

```
Renda Bruta? 5300.90
Idade? 57
Valor do Imovel? 60000
Numero de Parcelas? 240

Valor das Parcelas: R$ 550.00
Valor Final do Imovel:  R$ 132000.00
```

Quadro A.7: Sumário dos Requisitos.

Entradas		17	Valor das parcelas = Valor final do imóvel/ número de parcelas
1	Renda Bruta	18	Valor final do imóvel = Valor do imóvel + (valor do imóvel * taxa de juros * número de parcelas)
2	Idade	**Políticas de Inadimplência**	
3	Valor do Imóvel	19	Valor do imóvel $\leq 25 \times$ renda bruta
4	Numero de Parcelas	20	Valor das parcelas $\leq 25\%$ da renda bruta
Formatação das Entradas		**Saídas**	
5	Renda Bruta?	21	Valor das parcelas
6	Idade?	22	Valor final do imóvel
7	Valor do Imovel?	Formatação das saídas	
8	Numero de Parcelas?	23	Valor das Parcelas: R\$ $< valor >$
Restrições das Entradas		24	$< valor >$ deve conter duas casas decimais
9	Renda Bruta $\geq$ R\$ 465,00	25	Valor Final do Imovel: R\$ $< valor >$
10	Idade ≥ 18	26	$< valor >$ deve conter duas casas decimais
11	Idade ≤ 57	**Mensagens de Erro para Entradas Inválidas**	
12	Valor do Imóvel $\geq$ R\$ 1500,00	27	Valor da renda fora do limite permitido. Digite outro valor. Renda Bruta?
13	Valor do Imóvel $\leq$ R\$ 220.000,00	28	Idade fora do limite permitido. Digite outro valor. Idade?
14	Numero de Parcelas ≥ 3	29	Valor do imovel fora do limite permitido. Digite outro valor. Valor do Imovel?
15	Numero de Parcelas ≤ 240	30	Numero de parcelas fora do limite permitido. Digite outro valor. Numero de Parcelas?
Cálculo do Financiamento		**Mensagem de Erro para o caso de Não Financiamento**	
16	Taxa de Juros = 0.5% a.m	31	Financiamento nao pode ser concedido.

Apêndice B

POP – Orientações para Clientes-Tutores

Neste capítulo, apresentamos orientações para os clientes-tutores sobre como proceder durante o ciclo de resolução de problemas de POP. Apresentamos também estratégias de diálogo para tornar mais efetiva a interação entre clientes-tutores e desenvolvedores.

B.1 Procedimentos Prévios

- Esteja presente na reunião em que o professor explicará os procedimentos a serem seguidos durante o ciclo de resolução de problemas de POP;
- Siga as orientações fornecidas pelo professor;
- Leia atentamente a especificação de referência para conhecer os requisitos do programa que serão solicitados aos desenvolvedores (estudantes);
- Tire dúvidas, previamente, com o professor sobre qualquer requisito não entendido;

B.2 Construção da Versão Inicial do Documento de Especificação

- Identifique o grupo no qual você irá atuar como cliente-tutor;
- Solicite ao grupo que escolha um de seus desenvolvedores para documentar os requisitos durante a reunião;

- Solicite ao desenvolvedor escolhido pelo grupo que crie um documento no editor de texto colaborativo, definido previamente pelo professor. Este documento deve ser compartilhado com os membros do grupo, com você e com o professor da disciplina;
- Solicite que seja criado também uma lista de discussão, restrito aos membros do grupo, você e o professor. Essa lista de discussão será utilizada para auxiliar na interação fora do horário de aula;
- Apresente ao grupo o problema que deve ser resolvido;
- Interaja com os desenvolvedores para esclarecer os requisitos. Na interação, esteja atento as estratégias de diálogo apresentadas nos Apêndices B7 e B8;
- Registre a participação dos desenvolvedores, identificando os desenvolvedores mais e menos participativos;
- Chame a atenção dos desenvolvedores menos participativos, aconselhando-os a serem mais pró-ativos nas interações com o grupo. Essa ação deve ser feita de forma educada e particular. Notifique o professor sobre essa situação;
- Garanta a entrega da versão inicial do documento de especificação e dos casos de testes de entrada/saída ao final da primeira reunião. Para orientar os estudantes quanto a documentação dos requisitos, siga as orientações descritas no Apêndice B4;
- Responda aos questionamentos do grupo na lista de discussão a fim de manter a interação, mesmo fora do contexto de sala de aula.

B.3 Construção da Versão Final do Documento de Especificação

- Leia o documento de especificação produzido pelo desenvolvedores antes de reunir com eles para a construção da versão final do documento de especificação;
- Chame atenção dos estudantes para aspectos que não estão claros ou não foram devidamente especificados;
- Solicite aos desenvolvedores que apresentem os protótipos do programa e testes automáticos criados por eles;

- Teste os protótipos trazidos pelos desenvolvedores, afinal você é um cliente que está ávido para ver seu problema solucionado;
- Discuta os requisitos a partir desses protótipos. Por exemplo, notifique os desenvolvedores caso alguma mensagem de erro esteja fora do padrão desejado, ou caso algum dado de entrada ou saída tenha sido omitido;
- Solicite atualização do documento de especificação, caso haja requisitos que mesmo discutidos, não tenham sido registrados;
- Garanta a entrega da versão final do documento de especificação e dos casos de testes de entrada/saída ao final da reunião.

B.4 Documento de Especificação

- Solicite aos desenvolvedores que mantenham atualizado o documento de especificação para refletir as necessidades do cliente-tutor;
- Lembre ao grupo que, embora um desenvolvedor tenha sido eleito em sala de aula para registrar os requisitos, a responsabilidade de mantê-lo atualizado é de *todos* os membros do grupo;
- Oriente os desenvolvedores na organização do documento de especificação, caso perceba que isto não está sendo realizado pelo grupo. Para fazer isso:
  - Lembre os estudantes que um documento mal organizado poderá confundi-los no momento da implementação e consumir mais tempo deles na identificação dos requisitos;
  - Garanta que os desenvolvedores estabeleçam uma organização para o documento de especificação. Essa organização pode seguir o exemplo fornecido pelo professor ou pode ser estabelecida pelo grupo. Caso os desenvolvedores solicitem a sua opinião, discuta com eles a organização do documento, levando em consideração a definição de seções, uso de marcadores, de espaçamento entre as linhas e um tipo de letra que torne o documento mais agradável para a leitura;
  - Solicite aos desenvolvedores que prestem atenção para não deixar informações ambíguas, contraditórias, incorretas ou informações irrelevantes no documento de especificação, pois isso poderá confundi-los no momento de construir o programa. Uma forma de chamar atenção dos desenvolvedores é questioná-los com relação a algum requisito que não está corretamente descrito.

B.5 Controle do Tempo

Oriente os desenvolvedores no uso adequado do tempo, mantendo-os engajados na discussão dos requisitos. Caso haja conversas paralelas, solicite a eles que voltem à atividade, por exemplo: "Pessoal, percebam que eu sou o cliente, vocês teriam esse comportamento se estivessem na empresa com um cliente?"; "Vocês precisam sair daqui com a descrição do que deve ser feito, vocês acham que o que vocês já descobriram é suficiente?"

No caso em que a reunião exija a descoberta de requisitos para mais de um programa, distribua o tempo equilibradamente. Por exemplo, supondo que o tempo de aula seja de 2 horas, e há três problemas para serem esclarecidos, então dedique em torno de 40 minutos para cada um problema.

B.6 Situações Não Planejadas

Caso os desenvolvedores discutam algum requisito que não foi contemplado na especificação de referência, parabenize-os e diga que você não deseja aquele requisito no programa.

Se acontecer de você se confundir com algum requisito na reunião com os desenvolvedores, você deve corrigir a situação. Para fazer isso: "Pessoal, eu andei pensando e acho melhor que a entrada seja desse jeito". Ou ainda, "Eu disse a vocês que só precisava disso, mas queria acrescentar também tal restrição". É importante que na reunião para gerar a versão final da especificação não ocorram mais erros.

B.7 Descoberta dos Requisitos

- Não forneça requisitos, deliberadamente, aos desenvolvedores. A descoberta dos requisitos deve ser feita a partir de perguntas dos próprios desenvolvedores. Para motivá-los, utilize algumas estratégias, tais como:
  - Solicite aos desenvolvedores que leiam com atenção o enunciado do problema e tirem suas dúvidas;
  - Incentive a interação, caso os desenvolvedores não questionem. Para fazer isso, indague-os: "Vocês têm alguma dúvida sobre o que deve ser feito para resolver esse problema?"; "As informações que vocês precisam para fazer o programa estão todas contidas nessa descrição?"

- Seja mais incisivo, caso os desenvolvedores continuem passivos. Incentive-os a pensar de forma mais pontual: “Pessoal, está claro para vocês quais são entradas do programa?”; “No entendimento de vocês, qual é(são) a(s) saída(s) esperada(s) do programa?”, “Está claro o tipo de cálculo que deve ser feito?”; “Mostrem pra mim um caso de teste de entrada e saída do programa?”. Com base no *feedback* dos desenvolvedores, responda os questionamentos e fomente novas interações;
- Solicite aos desenvolvedores que registrem os requisitos à medida que eles são descobertos. Ao tomarem nota dos requisitos, os estudantes acabam percebendo informações que precisam ser esclarecidas.

B.8 Estratégias de Diálogo

Para potencializar as interações com os desenvolvedores e evitar comportamentos inadequados, utilizem alguns estratégias de diálogo, conforme apresentamos a seguir.

Os diálogos apresentados são reais e foram observados e extraídos de um curso de programação para iniciantes.

B.8.1 Evite Interações Inadequadas

O cliente-tutor deve evitar a entrega de requisitos e informações, deliberadamente. A função do cliente-tutor é auxiliar no progresso do desenvolvedor, mas não solucionar o problema por ele. No Quadro B.1, apresentamos um tipo de interação que deve ser evitada. O desenvolvedor solicita uma dica sobre como realizar a multiplicação de um inteiro pela matriz (linhas 1 e 2 do diálogo). Na intenção de ajudá-lo, o cliente-tutor acaba fornecendo uma solução algorítmica para o problema, conforme pode ser verificado nas linhas 3 a 6 do diálogo.

Para evitar situações como a apresentada no Quadro B.1, o cliente-tutor deve conduzir o desenvolvedor a melhorar o seu *background* conceitual. Para fazer isso, deve aconselhar o desenvolvedor a pesquisar um pouco mais sobre o tema em questão ou indicar algum material de referência.

B.8.2 Faça o Desenvolvedor ser mais Preciso

Motive os desenvolvedores a serem mais precisos em seus questionamentos. Um exemplo de como proceder é apresentado no Quadro B.2. Inicialmente, o desenvolvedor faz uma pergunta imprecisa (linha 1 do diálogo) na expectativa de que alguém lhe forneça a especificação do programa. Isso

Quadro B.1: Interação Inadequada.

Diálogo	
1	*Desenvolvedor:* Alguém sabe como automatizar o processo de multiplicação do inteiro pela
2	matriz? Caso alguém saiba, por favor, me dê alguma dica.
3	*Cliente-tutor:* Multiplicando um número escalar por um vetor, é o mesmo que multiplicar
4	cada elemento do vetor pelo escalar: Sendo k um escalar e v = (a, b, c) um vetor, kv = (ka,
5	kb, kc). É só fazer um for pra varrer o vetor, e multiplicar cada um dos elementos pelo
6	número, armazenando o resultado num vetor produto, ou algo do tipo.

ocorre porque, muitas vezes, os desenvolvedores têm pressa para iniciar o programa e não lêem com atenção o enunciado. Note que o cliente-tutor não detalhou o problema e o aconselhou a refletir melhor sobre ele (linhas 2 e 3 do diálogo). Ao fazer isso, o cliente-tutor força o desenvolvedor a reler o enunciado do problema e a ser mais preciso na exposição de suas dúvidas, conforme pode ser observado nas linhas 4 a 6 do diálogo.

Quadro B.2: Motivando o desenvolvedor a ser mais preciso.

Diálogo	
1	*Desenvolvedor:* Eu não entendi o problema, alguém poderia me explicar?
2	*Cliente-tutor:* Para eu poder ajudá-lo é necessário que você explique melhor qual é a sua
3	dúvida. Reflete um pouco mais sobre o enunciado e tenta explicitar melhor sua dúvida.
Algum tempo depois ...	
4	*Desenvolvedor:* O programa deve ler um parágrafo e a saída deve ser a quantidade de vezes
5	que cada palavra aparece no parágrafo, é isso? E se o usuário não digitar nada? CASA e
6	casa contam como duas palavras diferentes?

B.8.3 Motive o Desenvolvedor a Questionar

Alguns desenvolvedores são menos pró-ativos, apresentam algumas resistências para iniciar um diálogo e até mesmo conformam-se com uma especificação superficial do programa, então cabe ao cliente-tutor motivá-los. Nesses casos, o cliente-tutor deve despertar o desenvolvedor para refletir sobre o problema e a fazer questionamentos. O Quadro B.3 apresenta um exemplo de como proceder. Neste caso, o cliente-tutor toma como referência para o diálogo o protótipo do programa.

Note que é o cliente-tutor quem inicia o diálogo, questionando sobre as saídas do programa para determinados valores de entrada (linhas 1 a 3 do diálogo). Somente após perceber que há um erro na

forma como o protótipo do programa estava funcionando, é que o estudante passa a questionar (linha 8).

Quadro B.3: Utilizando o protótipo do programa.

	Diálogo
1	*Cliente-tutor:* O que ocorre quando o usuário digita:
2	`Capital?` 20
3	`Tempo?` 12
4	*Desenvolvedor:* Ah, o programa vai calcular o rendimento e capital futuro.
5	*Cliente-tutor:* Certo... e qual seria a saída então?
5	*Desenvolvedor:* `Rendimento:R$` 35.92
6	*Desenvolvedor:* `Capital Futuro:R$` 275.92
7	*Cliente-tutor:* Esse cálculo está errado.
8	*Desenvolvedor:* Errado? Por quê?
9	*Cliente-tutor:* Porque em nosso banco há restrição quanto ao valor que pode ser investido.
10	*Desenvolvedor:* Qual é o valor mínimo então?
11	*Cliente-tutor:* O valor mínimo deve ser de 30 reais.

Outra forma de motivar os desenvolvedores é questioná-los com base no documento de especificação produzido pelo grupo. No Quadro B.4, nós apresentamos um diálogo em que o cliente-tutor pede explicações aos desenvolvedores sobre a formatação da saída, tomando por base o que foi documentado na especificação (linhas 1 a 7 do diálogo). Um dos desenvolvedores do grupo responde ao cliente-tutor, corrigindo a informação e demonstrado a formatação correta da saída (linhas 8 a 13).

B.8.4 Faça o Desenvolvedor Estruturar Progressivamente o Diálogo

Encoraje os desenvolvedores a estabelecerem um diálogo progressivamente estruturado, como mostramos no Quadro B.5. Os desenvolvedores devem iniciar a interação de maneira livre, na forma de perguntas e resposta (a). Posteriormente, eles devem ser incentivados a também dialogarem na forma de casos de testes de entrada/saída (b). Depois, na forma de testes automáticos (c), que devem ser expressos utilizando a sintaxe da própria linguagem de programação adotada. No caso do Quadro B.5 (c), os testes automáticos foram escritos utilizando o comando `assert` de Python.

Em particular, para o caso em que os clientes-tutores não estejam familiarizados com a sintaxe da linguagem de programação adotada, a recomendação é motivar os desenvolvedores a dialogarem entre si, de tal modo que eles possam construir testes automáticos e discutirem sobre a corretude

Quadro B.4: Utilizando o documento de especificação.

	Diálogo
1	*Cliente-tutor:* Vocês dizem no documento de especificação: "Deve haver **pelo menos um**
2	**espaço** entre a palavra e o valor da freqüência." Então a saída do seguinte exemplo poderia
3	ficar assim:
4	`Paragrafo?` Ligue agora!
5	
6	`agora  1`
7	`ligue  1`
8	*Desenvolvedor:* A saída não é essa. A coluna de freqüência deve ser alinhado com
9	SOMENTE um espaço com relação a maior palavra. Exemplo:
10	`Paragrafo?` Ligue agora!
11	
12	`agora 1`
13	`ligue 1`

desses testes. Isto permite que os desenvolvedores vivenciem todo o processo de estruturação do diálogo e também a prática de testes.

Quadro B.5: Diálogo Progressivamente Estruturado.

Diálogo	
1	*Desenvolvedor:* Como é o cálculo do rendimento?
2	*Cliente-tutor:* Bom, o cálculo é feito com base no capital e no tempo de investimento.
3	*Desenvolvedor:* Há uma fórmula para isso?
4	*Cliente-tutor:* Sim. O rendimento é igual ao capital que multiplica o cálculo dos juros. E o
5	cálculo dos juros é feito somando um à taxa de juros e elevando esse resultado a quantidade
6	de meses de investimento.
7	*Desenvolvedor:* R = C vezes o quê?...
8	*Cliente-tutor:* R = C $* (1 + taxadejuros)^{tempo}$
	(a)
9	*Desenvolvedor:* Então:
10	`Capital?` 30
11	`Tempo?` 12
12	`Rendimento:R$` 53.88
13	*Cliente-tutor:* Correto!
	(b)
14	*Desenvolvedor:* `assert calcular_rendimento(30,12) == 53.88`
15	`assert calcular_rendimento(86.50,10) == 140.90`
16	*Cliente-tutor:* Ok!
	(c)

Apêndice C

POP – Orientações para os Estudantes

Os estudantes em POP devem assumir o papel de *desenvolvedores* e, como tal, irão tratar com um cliente e resolver um problema que será apresentado por ele. Este cliente é também chamado de cliente-tutor.

Para resolver o problema apresentado pelo cliente-tutor, os desenvolvedores devem produzir os seguintes artefatos: (i) um documento de especificação contendo a descrição dos requisitos do programa e também o registro de casos de testes de entrada/saída; (ii) um programa que atenda aos requisitos do cliente; e, (iii) casos de testes automáticos que demonstrem a qualidade do programa produzido.

Para a produção destes artefatos é necessário: (i) elaborar uma versão inicial do documento de especificação; (ii) iniciar a implementação com o objetivo de criar um protótipo do programa e dos casos de testes automáticos; (iii) concluir a especificação dos requisitos; e, (iv) concluir a implementação do programa e testes automáticos. Para a realização adequada destas atividades, os desenvolvedores devem seguir as orientações descritas a seguir.

C.1 Para Criar uma Versão Inicial do Documento de Especificação

- Esteja disposto para trabalhar em grupo, pois a especificação dos requisitos será realizada em conjunto com outros desenvolvedores;
- Selecione, em comum acordo com o grupo, um dos desenvolvedores para registrar os requisitos que serão discutidos com o cliente-tutor. O desenvolvedor escolhido deve também criar um

documento em um editor de texto colaborativo e uma lista de discussão, adotando os recursos, previamente, definidos pelo professor. O cliente-tutor irá orientá-los nessa tarefa;

- Leia o enunciado do problema para todo o grupo;
- Identifique junto com o grupo as informações relevantes do problema. Identifique também as informações omitidas, ambíguas, contraditórias e/ou que sejam irrelevantes;
- Não tenha pressa para iniciar o programa, discuta com o grupo as questões que não estão devidamente claras;
- Questione o cliente-tutor para esclarecer os requisitos do programa: entradas, saídas, restrições das entradas e saída, formatações das entradas e saídas, mensagens de erro, procedimentos para os cálculos, etc;
- Seja objetivo nos seus questionamentos. Por exemplo, não pergunte ao cliente-tutor: "O que é para fazer"? e sim, "O objetivo do programa é calcular o rendimento da poupança?";
- Evite conversas que não tenham relação com o problema tratado;
- Certifique-se de que os requisitos estão sendo documentados;
- Registre casos de testes de entrada/saída no documento de especificação e verifique a corretude desses casos de testes com o cliente-tutor;
- Revise a especificação para verificar se os requisitos foram documentados corretamente;
- Observe a organização do documento, pois os requisitos precisam ser facilmente identificados. Além disso, o documento de especificação deve tornar a leitura agradável;
- Certifique-se de que os requisitos documentados dão suporte a criação de uma versão inicial do programa.

C.2 Para Iniciar a Implementação

- Trabalhe individualmente na construção de um programa e casos de testes automáticos;
- Utilize a linguagem e ambiente de programação adotados na disciplina para construir os artefatos citados acima;

- Construa um protótipo do programa para atender a versão inicial do documento de especificação e casos de testes de entrada/saída;
- Construa um conjunto de casos de testes automáticos para aferir a qualidade do programa;
- Interaja com o cliente-tutor e os demais desenvolvedores de seu grupo para esclarecer os requisitos, caso haja dúvidas. Esta comunicação deve ser realizada por meio da lista de discussão;
- Mantenha atualizado o documento de especificação. Esta é uma responsabilidade de todo o grupo;
- Mantenha o documento de especificação organizado;
- Utilize os casos de testes automáticos para discutir os requisitos com o grupo, isto também ajuda a fixar a aprendizagem e a prática de testes;
- Colabore para aprendizagem de programação dos desenvolvedores de seu grupo, trocando informações, tirando dúvidas e recomendando material de referência;
- Não cometa plágios;

C.3 Para Concluir a Especificação dos Requisitos

- Organize-se novamente em grupo;
- Apresente ao cliente-tutor sua versão inicial do programa e casos de testes automáticos. Nesta apresentação, execute o programa e peça para o cliente-tutor testá-lo. Utilize essa apresentação como um recurso para confirmar os requisitos e descobrir eventuais requisitos que não tenham sido especificados;
- Observe a apresentação dos demais desenvolvedores do grupo. Esteja atento para verificar se os requisitos também foram atendidos no programa deles;
- Questione o cliente-tutor para tirar dúvidas ou confirmar requisitos;
- Atualize o documento de especificação;
- Apresente ao cliente-tutor o documento de especificação a fim de confirmar se todos os requisitos desejados encontram-se documentados.

C.4 Para Concluir a Implementação

- Trabalhe individualmente para evoluir a implementação do programa e casos de testes automáticos;
- Construa a versão final do programa e dos testes automáticos. Para isso tome como referência a versão final do documento de especificação e casos de testes de entrada/saída;
- Colabore com a aprendizagem dos demais desenvolvedores do grupo, respeitando ao que foi mencionado na Seção C2;
- Entregue o programa e os casos de testes automáticos na data definida pelo professor.

C.5 Exemplo de Documento de Especificação

Cliente-Tutor: <Nome do cliente-tutor> <email do cliente-tutor>

Desenvolvedores: <Nome dos desenvolvedores> <email dos desenvolvedores>

1. Enunciado do Problema

O Banco Imobiliário tem um plano de financiamento habitacional chamado "Lar doce Lar" que foi idealizado com o intuito de ajudar as pessoas a realizarem o sonho de adquirir a casa própria. Para colocá-lo em prática, o Banco deseja criar um simulador e deixá-lo disponível para a população. A idéia é que as pessoas que desejam adquirir crédito para a compra da casa própria possam simular os valores das parcelas do financiamento de forma rápida e fácil, bastando apenas informar o valor do imóvel que desejam adquirir e o número de parcelas em que pretendem pagar a dívida. As parcelas possuem taxa de juros fixas, assim todas as pessoas podem se beneficiar do financiamento habitacional. Manter valores de parcelas fixas é uma política do banco para evitar inadimplência dos solicitantes maiores de idade e tornar o financiamento acessível para a maioria da população. Suponha que você é o programador contratado pelo Banco Imobiliário e, portanto, deve entregar um programa que atenda às necessidades do referido Banco.

Nota para os estudantes: Utilize o tipo de fonte `courrier new` para expressar: testes de entrada/saída, formatações das entradas e saídas e mensagens de erro. Observe que ao longo deste documento o tipo fonte `courrier new` foi utilizado para os ítens citados, diferindo do tipo de fonte empregado no restante do texto.

2. Especificação do Programa

2.1. Entradas e Restrições das Entradas

Os seguintes dados devem ser informados pelos solicitantes do financiamento:

- Renda Bruta da família

  Renda bruta $\geq$ salário mínimo

- Idade do solicitante

  $18 \leq$ Idade ≤ 57

- Valor do imóvel que pretendem comprar

  R$ 1500,00 $\leq$ Imóvel $\leq$ R$ 220000,00

- Número de parcelas que desejam dividir o financiamento

  3 <= Número de parcelas <= 240 (meses)

2.2. Formatação das Entradas

```
Renda Bruta?
Idade?
Valor do Imovel?
Numero de Parcelas?
```

2.3. Mensagens de Erro para Entradas Inválidas

Renda Bruta

`Valor da renda fora do limite permitido. Digite outro valor.`

`Renda Bruta?` (deve permitir que o usuário digite um novo valor)

Idade

`Idade fora do limite permitido. Digite outro valor.`

`Idade?` (deve permitir que o usuário digite um novo valor)

Valor do Imóvel

`Valor do imovel fora do limite permitido. Digite outro valor.`

`Valor do Imovel?` (deve permitir que o usuário digite um novo valor)

Número de Parcelas

`Numero de parcela fora do limite permitido. Digite outro valor.`

`Numero de Parcelas?` (deve permitir que o usuário digite um novo valor)

2.4. Cálculos e Restrições do Financiamento

Taxa de juros = 0,5% a.m (juros simples)

valor final do imovel = valor do imovel + (valor do imovel $*$ taxa juros $*$ numero de parcelas) valor da parcela = valor final do imovel / numero de parcelas

Restrições do financiamento:

Se valor do imóvel > 25 $*$ renda bruta e valor das parcelas > 25% da renda bruta

Então emitir mensagem `Financiamento nao pode ser concedido.`

Obs.: A mensagem deve ser exibida pulando-se uma linha dos valores de entrada.

2.5. Saídas e Formatações das Saídas

O programa deve exibir na saída o valor das parcelas e o valor final do imóvel, obedecendo a seguinte formatação.

`Valor das Parcelas:  R$` <valor> (<valor> com duas casas decimais)

`Valor Final do Imovel:  R$` <valor> (<valor> com duas casas decimais)

Obs.: As saídas devem ser exibidas pulando-se uma linha dos valores de entrada.

2.6. Outras Restrições

O programa deve permitir apenas uma simulação dos valores.

Supor usuário esperto, portanto não é necessário tratar exceções, tais como, usuário digitar letras ao invés de números para as entradas.

2.7. Alguns Testes de Entrada/Saída

OBS.: Os testes abaixo levaram em consideração o valor do salário mínimo igual a R$ 465,00.

Teste 1

```
Renda Bruta? 300.00
Valor da renda fora do limite permitido.  Digite outro valor.
Renda Bruta? 18000
Idade? 10
Idade fora do limite permitido.  Digite outro valor.
Idade? 98
Idade fora do limite permitido.  Digite outro valor.
Idade? 35
Valor do Imovel? 221000.98
Valor do imovel fora do limite permitido.  Digite outro valor.
Valor do Imovel? 220000
Numero de Parcelas? 2
Numero de parcelas fora do limite permitido.  Digite outro valor.
```

```
Numero de Parcelas? 120

Valor das Parcelas:  R$ 2933.33
Valor Final do Imovel:  R$ 352000.00
```

Teste 2

```
Renda Bruta? 465.00
Idade? 57
Valor do Imovel? 11625.20
Numero de Parcelas? 210

Financiamento nao pode ser concedido.
```

Teste 3

```
Renda Bruta? 465
Idade? 18
Valor do Imovel? 1500
Numero de Parcelas? 3

Financiamento nao pode ser concedido.
```

Teste 4

```
Renda Bruta? 5300.90
Idade? 57
Valor do Imovel? 60000
Numero de Parcelas? 240

Valor das Parcelas: R$ 550.00
Valor Final do Imovel:  R$ 132000.00
```

Apêndice D

Estudos de Caso – Questionários Aplicados aos Alunos e Clientes-Tutores

D.1 Questionário Aplicado aos Alunos

1 Quanto a complexidade, você considera o problema 1:

Considere a complexidade para solucionar o problema.

☐ Fácil

☐ Médio

☐ Difícil

2 Quanto a complexidade, você considera o problema 2:

Considere a complexidade para solucionar o problema.

☐ Fácil

☐ Médio

☐ Difícil

3 Quanto a complexidade, você considera o problema 3:

Considere a complexidade para solucionar o problema.

☐ Fácil

☐ Médio

☐ Difícil

4 Com relação as especificações dos problemas, como você avalia seu desempenho?

☐ Ótimo

☐ Bom

☐ Regular

☐ Ruim

☐ Péssimo

5 Quanto a elaboração de casos de testes, como você avalia seu desempenho?

Considere a elaboração de testes automáticos.

☐ Ótimo

☐ Bom

☐ Regular

☐ Ruim

☐ Péssimo

6 Quanto a construção dos programas, como você avalia seu desempenho?

☐ Ótimo

☐ Bom

☐ Regular

☐ Ruim

☐ Péssimo

7 Com você avalia a interação do grupo para especificação dos problemas? ☐ Ótimo

☐ Bom

☐ Regular

☐ Ruim

☐ Péssimo

8 Quais dificuldades você sentiu para entender/especificar o problema?

9 Que coisas novas você aprendeu nesse Roteiro?

10 Que sugestões ou comentários você faria para melhorar as atividades realizadas no Roteiro?

D.2 Questionário Aplicado aos Clientes-Tutores

Nome:

1 Quais as dificuldades dos estudantes na fase de especificação dos problemas?

2 O material de apoio fornecido para orientar na realização do roteiro foi adequado?

☐ Totalmente

☐ Parcialmente

☐ Não foi adequado

3 Que sugestões você daria para melhorar o material fornecido para orientação?

4 Quais dificuldades você sentiu para conduzir as atividades neste roteiro?

5 Que sugestões você daria para melhorar as atividades neste roteiro?

Apêndice E

Estudos de Caso – Requisitos dos Programas

Para solucionar os problemas mal definidos propostos nos estudos de caso (Capítulo 4), os programas dos estudantes teriam que atender aos requisitos apresentados nas seções seguintes.

E.1 Poupança Programada

Quadro E.1: Requisitos do Programa 1 – Poupança programada.

Entradas		9	Capital_Futuro = (Capital * tempo) + Rendimento
1	Capital	**Saídas**	
2	Tempo	10	Rendimento
Formatação das Entradas		11	Capital Futuro
3	`Capital?`	**Formatação das Saídas**	
4	`Tempo?`	12	`Rendimento:` <*valor do rendimento*>
Restrição das Entradas		13	<*valor do rendimento*> com duas casas decimais
5	Capital $\geq$ R$ 30,00	14	`Capital Futuro:` <*valor do capital futuro*>
6	Tempo $\geq$ 2 meses	15	<*valor do capital futuro* com duas casas decimais
7	Tempo $\leq$ 48 meses	**Mensagem de Erro para Entradas Inválidas**	
Fórmulas		16	`Investimento minimo de R$ 30,00. Capital?`
8	Rendimento = Capital * (1 + (taxa_juros/100)$^{\text{tempo}}$)	17	`Periodo de tempo de 02 a 48 meses. Tempo?`

E.2 Freqüência das Palavras

Quadro E.2: Requisitos do Programa 2 – Freqüência das palavras.

Entradas		
1	Parágrafo	O parágrafo é requerido na entrada e finalizado quando o usuário digitar `<enter>`
Formatação da Entrada		
2	`Paragrafo?`	
Restrições da Entrada		
3	Não diferenciar caracteres maiúsculos de minúsculos	Carro CARRO carro → carro
4	Caracteres não alfabéticos devem ser removidos	br%@as3il∗ → brasil
5	Sinais de acentuação devem ser ignorados	você → voce; maçã → maca
6	Considerar apóstrofos	d′agua → d′agua
7	Considerar hífens	guarda-chuva → guarda-chuva
Saídas		
8	Palavras válidas devem ser apresentadas na saída	
9	Para cada palavra válida no parágrafo deve ser apresentada sua respectiva freqüência	
Formatação da Saída		
10	A saída deve ser expressa em duas colunas. A primeira coluna deve conter as palavras válidas do parágrafo em ordem alfabética e minúscula	`Paragrafo? guarda guarda-roup@a guarda-guarda` `guarda-roupa`
11	A segunda coluna deve conter o número de aparições da palavras no parágrafo, justificada à direita, contendo um espaço da maior palavra	`Paragrafo? guarda guarda-roup@a guarda-guarda 2` `guarda-roupa 1`
Mensagem de Erro para Entradas Inválidas		
12	`Nenhuma palavra foi digitada.`	`Paragrafo?` %@432! `Nenhuma palavra foi digitada.`

E.3 Jogo da Forca

Quadro E.3: Requisitos do Programa 3 – Jogo da forca.

Entradas		
1	Palavra	A forca deve ser jogada entre dois jogadores. Cabe ao jogador cadastrar a palavra a ser adivinhada pelo outro.
2	Número de Chances	Refere-se ao número de chances que um jogador tem para adv-inhar a palavra.
Formatação das Entradas		
3	`Palavra?`	
4	`Chances?`	
Restrições da Entrada: Palavra		
5	Não diferenciar caracteres maiúsculos de minúsculos	CAMpiNENSE → campinense
6	Sinais de acentuação devem ser ignorados	forró → forro
7	Não Considerar apóstrofos	copo d′agua → copo dagua
8	Considerar hífens	guarda-chuva → guarda-chuva
9	Palavras compostas devem ser consideradas	Flávio José → favio jose
Restrições da Entrada: Número de Chances		
10	Número de chances ≥ 1	
Saídas		
11	`Palavra? _ _ _ _ _ _`	Após o label `Palavra?` cada caracter da palavra a ser adivin-hada deve aparecer substituída pelo símbolo "_" separada por um espaço.
12	Chances	
13	Palpites	Os palpites do jogador para adivinhar a palavra devem ser apre-sentados na saída
Formatações e Restrições da Saída		
14	Palavra com apóstrofo	`Palavra? copo d'agua` `Palavra? _ _ _ _ _ _ _ _ _`
15	Palavra com hífen	`Palavra? guarda-chuva` `Palavra? _ _ _ _ _ _ - _ _ _ _ _`
16	Palavra composta	`Palavra? Flávio José` `Palavra? _ _ _ _ _ _ _ _ _ _ _`. Deve conter três espaços entre uma palavra e outra
17	`Chances?`	O número de chances deve ser apresentada na saída seguida do label `Chances?` abaixo da saída `Palavra?`
18	`Palpites?`	Os palpites do jogador devem ser apresentada na saída seguida do label `Palpites?` abaixo da saída `Chances?`
19	O número de chances deve ser decrementado a cada palpite incorreto	

20	Todo palpite fornecido pelo jogador deve ser exibido na saída separados por um espaço	`Palavra? c _ _ p _ _ _ _ _ _`
		`Chances? 3`
		`Palpites? c p`

21	Cada palpite deve ser formado por apenas um caracter	

Quadro E.4: Requisitos do Programa 3 – Jogo da forca (cont.).

Mensagens e Formatações de Mensagens		
22	Quando a palavra é adivinhada pelo jogador uma mensagem de vitória deve ser exibida na saída	
23	Quando a palavra não é adivinhada pelo jogador dentro do número de chances que lhe foi dado uma mensagem informando a derrota deve ser exibida na saída	
24	`Voce venceu!`	Mensagem de vitória
25	`Voce perdeu!`	Mensagem informando derrota
26	`Chance maior ou igual a 1`	Mensagem que deve ser exibida quando o número de chances informado pelo jogador for < 1

Apêndice F

Questionário para Apoiar na Seleção da Amostra

Nome:

Telefone:

Email:

1 Idade?

2 Sexo?

☐ Masculino

☐ Feminino

3 Você cursou o ensino médio?

☐ Todo em escola particular

☐ Todo em escola pública

☐ Parte em escola pública e parte em escola particular

4 Você faz outro curso de graduação?

☐ Sim

☐ Não

Se a resposta for SIM, acrescente as informações abaixo:

Qual o curso?

Qual período está cursando?

5 Você faz ou fez algum curso técnico?

☐ Sim

☐ Não

Se a resposta for SIM, acrescente as informações abaixo:

Qual o curso?

Qual o estágio do curso:

☐ No início do curso

☐ Meio do curso

☐ Fim do curso

☐ concluído

6 É a primeira vez que cursa a disciplina de programação?

Responda mesmo que seu estudo tenha sido independente.

☐ Sim

☐ Não

7 Você já teve experiência com alguma outra linguagem de programação além da que você está aprendendo agora?

☐ Sim. Qual(is) linguagem(ens)?

☐ Não

8 Se você respondeu SIM para a questão 07 indique o tempo de experiência com as linguagens que você já estudou?

9 Com que freqüência você utiliza computador?

Linguagem	1 mês	1-3 meses	6 meses	6-1ano	>1 ano

☐ Diariamente

☐ Algumas vezes por semana

☐ Algumas vezes por mês

10 Você tem computador em casa?

☐ Sim

☐ Não

11 Você gosta de programar?

☐ Sim

☐ Não

☐ Mais ou menos

12 Para você, programar é uma atividade?

☐ Muito difícil

☐ Difícil

☐ Nem fácil nem difícil

☐ Fácil

☐ Muito fácil

Apêndice G

Publicações

Artigo em avaliação

Mendonça, Andréa P.; Guerrero, Dalton D. S.; Figueiredo, Jorge C.; Costa, Evandro B. (2010). *From Requirements Specification to the program: An Experimental Study with Novice Programming Students*. Submetido para ACM Transactions on Computing Education.

Artigos Publicados

Mendonça, Andréa P.; Chaves, Danielle; Guerrero, Dalton D. S.; Abrantes, Jorge; Costa, Evandro B. (2010). *Dealing with Requirements Specification: A Case Study with Novice Programming Students*. IEEE Multidisciplinary Engineering Education Magazine, 5(1):3-10.

Mendonça, Andréa P.; de Oliveira, Clara; Guerrero, Dalton D. S.; Costa, Evandro B. (2009). *Difficulties in Solving Ill-Defined Problems: A Case Study with Introductory Computer Programming Students*. Proceedings of the 39th IEEE International Conference on Frontiers in Education Conference. San Antonio, Texas, p. 1171-1176.

Mendonça, Andréa. P.; Guerrero, Dalton D. S.; Costa, Evandro B. (2009). *An Approach for Problem Specification and its Application in an Introductory Programming Course*. Proceedings of the 39th IEEE International Conference on Frontiers in Education Conference. San Antonio, Texas, p. 1529-1534.

Mendonça, Andréa P.; Chaves, Danielle; Guerrero, Dalton D. S.; Costa, Evandro B. (2009). *Tratando Especificação de Requisitos com Estudantes Iniciantes de Programação*. II Fórum de Educação

em Engenharia de Software (FEES). XXIII Simpósio Brasileiro de Engenharia de Software, p. 25-32.

Mendonça, Andréa P.; Guerrero, Dalton D. S.; Costa, Evandro B. (2009). *Problem Oriented Programming: An Approach to Teach Programming to Beginner Students*. VI International Conference on Engineering and Computer Education (ICECE' 09), Buenos Aires - Argentina.

Mendonça, Andréa P.; Costa, Evandro B.; Guerrero, Dalton D. S. (2008). *Elicitação de Requisitos - Evidências de uma Problemática na Formação dos Estudantes de Computação*. I Fórum de Educação em Engenharia de Software (FEES). Simpósio Brasileiro de Engenharia de Software. Monografia em Ciência da Computação N^0 43/08. ISSN 0103-9741, p. 65-73.