Compreendendo o processo de codificação de um exercício de programação em Python

Jéssika Renally Ribeiro Rodrigues, Matheus Gaudencio, Dalton S. Guerrero

[1]Universidade Federal de Campina Grande
Laboratório de Práticas de Software
Av. Aprígio Veloso, s/n, Prédio SPLab, Bodocongó
58.429-900, Campina Grande, PB

jessika.rodrigues@ccc.ufcg.edu.br, matheusgr@copin.ufcg.edu.br,

dalton@dsc.ufcg.edu.br

Abstract. *Programming courses often uses assessments that requires the student to create programs. But even with this practice, little is known about how students code. We evaluated character by character how 11 students coded two introductory programming exercises. We found that such analysis is a powerful tool that can improve teacher's perceptions about the skill and knowledge of their students. It is possible, for example, to identify different stages of a coding process and also to detect anomalies that may students with learning difficulties.*

Resumo. *Apesar da utilização de listas de exercícios no aprendizado de programação, pouco se sabe a respeito do processo de codificação do aluno. Nós avaliamos, caractere a caractere, como 11 estudantes resolveram dois exercícios num curso introdutório de programação. Nossos resultados sugerem que a análise do processo de codificação pode ser uma ferramenta poderosa e com potencial para ser realizada de forma automática para melhorar a percepção do professor sobre os conhecimentos e habilidades dos estudantes. É possível, por exemplo, discernir entre as várias etapas do processo de programação do estudante e detectar anomalias que são indícios de dificuldades.*

1. Introdução

A prática de estudantes de cursos introdutórios de programação é comumente baseada em exercícios [Chamillard e Braun 2000]. Depois de resolver questões, os alunos enviam a solução final ao professor. Com os resultados obtidos através da avaliação desses códigos, os professores avaliam o desempenho dos estudantes no aprendizado de programação e orientam sua intervenção no processo de aprendizado dos estudantes.

Embora o foco seja na solução final de cada estudante, o caminho utilizado durante a construção de uma solução pode apresentar, em si, uma grande quantidade de informações sobre o processo de aprendizado do aluno. Esta análise de uma grande quantidades de dados se encaixa dentro da área de Learning Analytics [Baker e Yacef 2009], na qual destaca-se o trabalho de Piech [Piech et al. 2012] que, ao analisar os códigos que eram compilados por alunos durante todo processo de resolução de exercícios, constatou que existem diferentes estratégias de se construir uma solução final.

Neste trabalho, nos concentramos em entender o processo de codificação dos alunos de um curso introdutório de programação focando na menor unidade

de edição, a construção do código caractere a caractere. Para nos auxiliar nesse processo, construímos o WebShell, um ambiente de programação semelhante ao BOCA [De Campos, C. P. e Ferreira, C. E. 2004] que funciona em um navegador de Internet e que, adicionalmente, coleta todos os passos na construção de uma solução – eventos como caracteres digitados, apagados ou modificados, tentativas de submissão e execução do código, bem como controle do foco da janela do navegador. O ambiente também permite que o estudante execute e teste o código desenvolvido.

Usando essa ferramenta, nós realizamos um experimento com onze estudantes de um curso introdutório de programação do programa de graduação em Ciência da Computação da UFCG, solicitando que resolvessem dois exercícios na linguagem de programação Python. Todo o processo de construção dessas soluções foi capturado. Com esses dados e informações realizamos uma análise qualitativa, no intuito de compreender os elementos presentes no processo de codificação dos alunos. Os principais resultados a que chegamos foram:

- Na maior parte das soluções foi característica a presença de três etapas básicas e bem definidas no processo de codificação: 1) compreensão do problema, 2) implementação e 3) verificação e teste da solução;
- Foi possível identificar alunos com dificuldades durante a construção de uma solução, seja pela grande alternância entre as operações de adição e remoção de código (caracterizando uma estratégia de tentativa e erro) ou pela demora e pouca alteração significativa durante a etapa de teste da solução (depuração da estratégia atual). Isto possibilita um melhor acompanhamento individualizado, especialmente em métodos de ensino construtivistas como a proposta de Borges [Borges, M. A. F. 2000];
- Apesar desse trabalho realizar uma análise qualitativa, há claros indícios de que vários processos de análise podem ser automatizados.

O referencial teórico utilizado pode ser encontrado na próxima seção. A metodologia utilizada nesse trabalho pode ser vista em detalhes na Seção 3. Na Seção 4, apresentamos os resultados apurados nesse artigo para, na Seção 5, discutirmos esses resultados. Por fim, na Seção 6 nós apresentamos as conclusões obtidas.

2. Referencial Teórico

Em 2011 ao conduzir um estudo que analisava o estado intermediário do código de alunos entre as tentativas de compilação, Piech percebeu que os alunos podiam ser caracterizados em dois grupos de comportamentos distintos. Enquanto que um deles apresentava, de tempos em tempos, períodos consideráveis de inatividade o outro grupo apresentava uma construção contínua quando se considera o número de linhas de códigos dos programas.

Também utilizando a perspectiva de investigar o processo de codificação Blikstein [Blikstein 2011] utilizou logs de programas para analisar o processo de codificação de estudantes do segundo ano de engenharia, a partir desse estudo ele percebeu alguns comportamentos, são eles: i) utilizar outro programa como base de parte do código; ii) parar de codificar para analisar os seus códigos ou para procurar soluções em outros códigos; iii) codificar de forma contínua a solução; iv) copiar e colar de código, e; v) corrigir detalhes no código, como formatação, endentação, nome de variáveis, etc.

3. Metodologia

Para coletarmos os dados do processo de codificação, nós construímos o *Webshell*, que é um ambiente de programação web com recursos como endentação automática, syntax

highlight, e execução de código. Através desse ambiente somos capazes de capturar diferentes eventos do processo de codificação do estudante, tais como: i) o momento e o valor de cada caractere digitado, apagado ou modificado; ii) o estado e as mudanças do foco da janela do editor no navegador (ativo ou inativo); iii) cada tentativa de execução do código do aluno, e; iv) cada tentativa de submissão da solução final para testes. Com estes dados coletados, é possível analisar todos os passos do processo de codificação de um aluno.

Para entender o processo de codificação, planejamos um experimento em ambiente controlado com 11 alunos escolhidos de forma representativa dentre os cento e dez estudantes, que cursavam a disciplina introdutória de programação do Curso de Ciência da Computação na Universidade Federal de Campina Grande no primeiro período de 2012. O experimento foi realizado 2 semanas antes do término das atividades daquele período.

Utilizando essa ferramenta, solicitamos que, em até 3h, os alunos resolvessem dois problemas de programação e submetessem a solução final. As questões envolviam o uso de listas, laços, funções e comandos para entrada e saída de dados. Na primeira questão, pedia-se explicitamente que o estudante construísse uma função com nome, parâmetros de entrada e retorno bem definidos. Na segunda questão, o enunciado era mais aberto e pedia apenas que um programa fosse desenvolvido, não exigindo explicitamente o uso de determinado tipo de elemento da linguagem. Formatação de entrada e saída, contudo, foram explicitamente definidas como condições a serem seguidas. A linguagem utilizada, Python, é a mesma utilizada no curso introdutório de programação. Importante registrar ainda que o estilo das questões também segue o padrão adotado no curso.

4. Análise dos Resultados

O tempo escolhido para a realização das questões foi mais do que suficiente para que alunos programassem em seu fluxo natural de codificação, sem se sentirem pressionados pelo tempo. A primeira e segunda questão foram resolvidas em média em 621 e 486 segundos, respectivamente (menos de 10 minutos). As medianas dos tempos foram de 290 e 256 segundos (menos de 5 minutos). A diferença se explica porque os tempos máximos foram de 2590 (43 minutos) e 1933 (32 minutos), respectivamente, o que elevou os tempos médios.

Para as duas questões e os onze alunos coletamos um total de 8121 eventos, sendo 142 eventos de execução de código, 33 de submissão da solução final, 229 de troca de foco e 7717 eventos de teclado. Todas as tentativas de execução e de submissão das soluções foram submetidas a testes, para identificar defeitos funcionais nos códigos submetidos. Para ambas as questões apresentadas, em todas as submissões ocorreu que se a solução satisfaz os asserts apresentados no enunciado da questão, então ela também satisfaz os casos de teste executados.

4.1. Curva de Codificação

Para simplificar a análise, optamos por avaliar a variação do número de caracteres ao longo do tempo durante o processo de resolução da questão. Para tanto, plotamos o número de caracteres ao longo do tempo para cada um dos estudantes e cada questão. Nessas curvas, que denominamos curvas de codificação, o eixo horizontal corresponde ao tempo, em segundos e o eixo Y ao número de caracteres do código. Para denotar os períodos em que o estudante retira o foco do editor, a curva é apresentada com dois tipos de linha: contínua para indicar que o foco está no editor e pontilhada para indicar que o foco não está no editor.

Os gráficos recebem ainda outras marcas. Linhas verticais indicam momentos em que o estudante tenta executar o código. Linhas de execução pontilhadas indicam que o código não passou nos testes e linhas tracejadas indicam que o código passou nos testes.

4.2. Cenário Comum

A Figura 1 apresenta as curvas de codificação de todos os alunos do experimento para a primeira questão. Para apresentar todos os gráficos na mesma escala (de 0 a 450 caracteres e de 0 a 1200 segundos) foi necessário cortar parte do gráfico da última linha e primeira coluna.

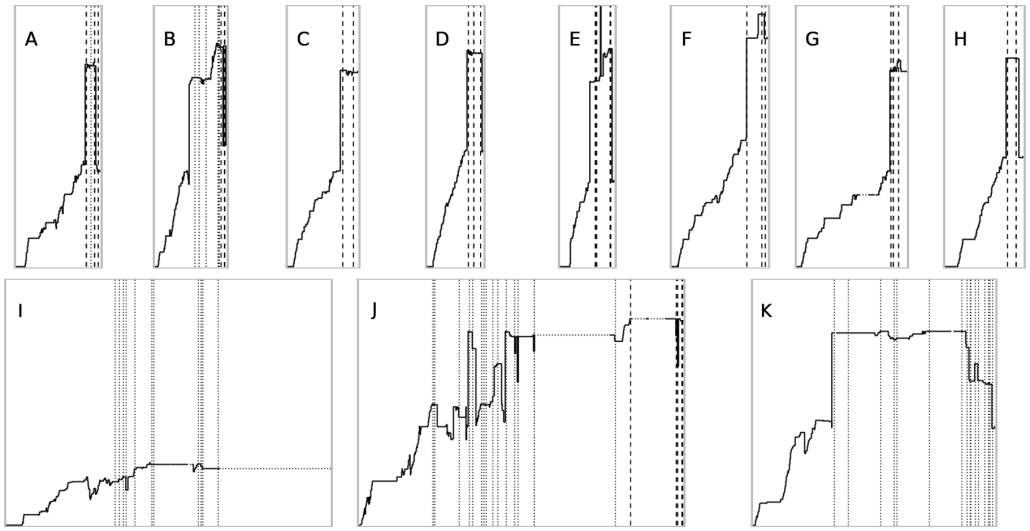

Figura 1. Curvas de Codificação dos Alunos na Questão 1

Pudemos constatar que 8 dos 11 gráficos seguem um padrão, em todos eles, há um momento inicial que associamos à leitura e compreensão do problema, em que não existe nenhuma alteração no número de caracteres. Após esse momento, percebemos uma etapa de crescimento linear no código, período em que a solução está sendo desenvolvida. Por fim, há a parte de depuração da solução que se inicia pela cópia e cola dos asserts disponibilizados no enunciado da questão, fenômeno perceptível pelo abrupto crescimento do número de caracteres do código. Na etapa de depuração, ocorrem execuções com falhas, que são rapidamente corrigidas, caracterizando o ajuste do código a situações de limite e/ou formatação de entrada e saída. No caso da segunda questão, o mesmo padrão de comportamento foi observado para 10 dos 11 alunos.

Além da cópia e cola de asserts, diagnosticamos outros comportamentos que se repetiram especialmente nas soluções que passaram nos testes automáticos, tanto na primeira questão, como na segunda. A lista de todos os comportamentos encontrados está definida a seguir:

1. Paradas após certos tokens da linguagem: em 21 processos de codificação houve paradas após tokens de definição de funções, laços e condicionais (def, for, if), essas paradas, na maior parte dos casos, tiveram 20 segundos de duração.
2. Paradas durante a definição de comandos: em 16 processos de codificação houve paradas durante a definição de comandos, tais como, paradas para escolher a lista de iteração do for ou parâmetros de entrada de uma função.
3. Paradas após a conclusão da solução: em 19 processos de codificação notamos que, após terminar de codificar a solução, os alunos param alguns segundos antes de testá-la.

4. Cópia e cola dos asserts disponibilizados no enunciado: em 19, das 22 curvas de codificação, notamos um crescimento acentuado depois que a solução estava pronta.
5. Pouca modificação no conteúdo da solução após a primeira execução do código: 18 dos 22 processos de codificação modificaram pouco seus códigos depois da primeira execução.
6. Troca da palavra assert por print: em 13 curvas de codificação os alunos trocaram a palavra-chave assert por print.
7. Os alunos não fizeram mais asserts e pararam de programar pouco tempo depois da primeira execução bem sucedida dos asserts. Isto foi verdade para 18 dos 22 casos avaliados.

4.3. Cenários Anômalos

Os três casos que tiveram curva de codificação diferente das demais, destoaram primeiramente pelo fator tempo, enquanto que esses processos de codificação demoraram 42, 15 e 20 minutos para terem uma versão final, todos os outros processos foram concluídos em cerca de cinco minutos ou menos. As curvas de codificação desses alunos podem ser vistas na Figura 1, são as curvas I, J e K.

É possível observar que o Aluno I codifica a solução, e faz várias tentativas de execução com falhas. Nela, o código não era nem sintaticamente correto. É interessante observar nesse aluno as sucessivas e espaçadas tentativas de execuções com falhas, bem como a troca de foco. O mesmo comportamento é observado no Aluno J. Neste caso, no entanto, há uma maior reescrita de código, e, apesar de um grande período fora de foco, o aluno retoma com uma alteração de sucesso em sua questão. Já o Aluno K apresenta uma construção de código bastante semelhante com as soluções de sucesso. Entretanto, durante a fase de testes, há diversas pequenas alterações no código seguidas por execuções que falharam.

5. Discussões

A presença dos comportamentos do cenário típico nos processos de codificação e a pouca variação entre eles, caracterizou um cenário comum para a maioria dos exercícios. Possivelmente, esses são os comportamentos que os professores não precisam se preocupar quando veem nos processos de codificação de seus alunos, pois eles não são indicativos de problemas. No entanto, quando esses comportamentos passam a variar muito, ou a apresentarem valores distantes da maior parte dos outros alunos, podem ser um indicador de intervenção para um aluno que está passando por algum problema no seu processo de codificação.

De forma geral, a análise dos comportamentos típicos, nos leva a pensar sobre algumas características dos processos de codificação, por exemplo, apesar da presença de asserts, somente três alunos fizeram asserts adicionais, além do mais, alguns alunos não tiveram segurança da ferramenta e do uso de assertivas de código (asserts), havendo, em alguns casos, a troca da palavra assert por print. Outra coisa a se perceber é que nenhum aluno começou com a cópia dos asserts, deixando esta etapa para o final. Por fim, é possível pensar sobre o próprio processo de programação quando se observa que há paradas presentes nas trocas de contexto, na mudança de bloco, por exemplo, ou onde se faz necessário alguma decisão, como na escolha de variáveis.

Nos cenário anômalos, é interessante observar que o comportamento dos três estudantes é claramente destoante dos demais apresentados e analisados, mas mesmo

que o código desses alunos tenha apresentado falhas durante as execuções, em nenhum momento eles desistem completamente da solução, o que poderia ser uma estratégia a ser adotada na presença de falhas.

6. Conclusões

Da análise dos resultados nós pudemos perceber a presença de dois grupos, no primeiro, o gráfico da curva de codificação apresenta claramente as etapas de entendimento do problema, implementação e verificação, em que a primeira etapa é marcada por um platô, a segunda etapa pelo crescimento linear do código e a terceira pela cópia e cola dos asserts com poucas execuções, que não geram quase nenhuma modificação no código. No outro grupo, no entanto, essas etapas não estão bem definidas, e nós encontramos comportamentos anômalos, como a presença de grandes períodos de inatividade no código, uma grande quantidade de execuções, várias mudanças no código após cada execução, execuções com erros sintáticos, variações constantes em eventos de adição/remoção de código e um tempo mais longo gasto para submeter os exercícios.

Os principais resultados ao qual chegamos foi a percepção clara das etapas de codificação e que para o nosso conjunto de dados, os comportamentos anômalos indicaram alunos que tiveram dificuldades para resolver os exercícios, e em 66% dos casos apresentaram respostas erradas. Enquanto que todos os alunos que apresentaram processos de codificação típicos acertaram aos exercícios. Outro fato importante, ao qual chegamos, é a possibilidade de utilizar métodos automáticos para realizar a análise, esses modelos automáticos são estratégias potenciais para ajudar a compreender melhor o processo de codificação, tanto para professores, quanto para alunos.

Referências

Baker, R. e Yacef, K. (2009). The state of educational data mining in 2009: A review and future visions. *Journal of Educational Data Mining*, 1(1):3–17.

Blikstein, P. (2011). Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, LAK '11, pages 110–116, New York, NY, USA. ACM.

Borges, M. A. F. (2000). Avaliação de uma Metodologia Alternativa para a Aprendizagem de Programação. In *Workshop de Educação em Computação, Congresso anual da SBC 2000*, Curitiba, Brasil. SBC.

Chamillard, A. T. e Braun, K. A. (2000). Evaluating programming ability in an introductory computer science course. *SIGCSE Bull.*, 32(1):212–216.

De Campos, C. P. e Ferreira, C. E. (2004). BOCA: um sistema de apoio a competições de programação (BOCA: A Support System for Programming Contests). In *Workshop de Educacao em Computacao (Brazilian Workshop on Education in Computing), Congresso anual da SBC 2004*, Salvador, Brasil. SBC.

Piech, C., Sahami, M., Koller, D., Cooper, S., e Blikstein, P. (2012). Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE '12, pages 153–160, New York, NY, USA. ACM.