# Desafios e oportunidades aos processos de ensino e de aprendizagem de programação para iniciantes

**Viviane Cristina Oliveira Aureliano[1,2]**
**Patricia Cabral de Azevedo Restelli Tedesco[2]**
**Lúcia Maria Martins Giraffa[3]**

[1]Instituto Federal de Pernambuco – Campus Belo Jardim
Belo Jardim, PE – Brasil

[2] Universidade Federal de Pernambuco – Centro de Informática
Recife, PE – Brasil

[3] Pontifícia Universidade Católica do Rio Grande do Sul – Escola de Humanidades
Porto Alegre, RS – Brasil

[vcoa, pcart]@cin.ufpe.br; giraffa@pucrs.br

***Abstract.*** *Introductory programming courses are intrinsically difficult, and are generally associated to high failure and evasion rates. It is up to the programming teacher to deal with which such difficulties through the organization of teaching activities that help learners to acquire the necessary knowledge. In this light, this paper discusses challenges faced by teachers when planning and executing such activities, and presents research opportunities in the area. Moreover, we propose an approach to minimize some of the problems faced by programming students. The use of this approach in a digital games course has contributed to the learning of novice programming students.*

***Resumo.*** *Disciplinas introdutórias de programação são intrinsecamente difíceis, comumente associadas a altos índices de reprovação e de evasão. Para lidar com essas dificuldades, cabe ao professor de programação organizar atividades de ensino e de aprendizagem que auxiliem os estudantes na aquisição do conhecimento necessário para a disciplina. Neste âmbito, este artigo discute desafios enfrentados pelos professores no planejamento e execução destas atividades, apresentando oportunidades de pesquisa na área. Além disso, apresentamos a proposta de uma abordagem para minimizar alguns dos problemas vivenciados pelos estudantes. A aplicação desta abordagem em um minicurso de jogos digitais demonstrou contribuir para a aprendizagem de estudantes iniciantes em programação.*

## 1. Introdução

A literatura na área de ensino e de aprendizagem em programação é unânime ao afirmar que começar a programar é considerado difícil pela maioria dos estudantes iniciantes nesta disciplina [CASPERSEN, 2007; DU BOULAY, 1989; GOMES; MENDES, 2007; GOMES; SANTOS; MENDES, 2012; MILNE; ROWE, 2002; ROBINS;

ROUNTREE; ROUNTREE, 2003]. Normalmente, as disciplinas introdutórias de programação estão relacionadas a altas taxas de reprovação, que podem chegar a um terço dos estudantes matriculados nestas disciplinas no mundo todo [BENNEDSEN; CASPERSEN, 2007; WATSON; LI, 2014]. Estas disciplinas também levam a altas taxas de evasão nos cursos de tecnologia [KINNUNEN; MALMI, 2006; SILVA FILHO et al., 2007]. Programar é considerada uma tarefa tão complexa que estudos apontam que, mesmo após passar por uma disciplina introdutória de programação, estudantes ainda apresentam sérios problemas ao aplicar os conceitos da disciplina [KURLAND et al., 1986; MCCRACKEN et al., 2001]. Pesquisas também apontam que a maioria dos estudantes têm seus conhecimentos de programação consolidados apenas ao final da segunda disciplina de programação [TEW; MCCRACKEN; GUZDIAL, 2005]. Por este motivo, ensinar programação é considerado um dos grandes desafios da área de educação em computação [CASPERSEN 2007].

Os obstáculos vivenciados pelos iniciantes em programação incluem problemas com a sintaxe de uma linguagem de programação (LP) [DU BOULAY, 1989], com os diferentes construtos de uma LP, tais como variáveis [PUTNAM et al., 1989; SAMURÇAY, 1989], estruturas condicionais e de repetição [CHERENKOVA; ZINGARO; PETERSEN, 2014; SPOHRER; SOLOWAY, 1989] e *arrays* [DU BOULAY, 1989]. Além disso, eles também encontram dificuldades para resolver os erros em seus programas e para dividir o programa em funções [LAHTINEN et al., 2005]. Por fim, eles encontram problemas ao lidar com conceitos mais abstratos de uma LP, tais como recursão e ponteiros [MILNE; ROWE, 2002].

Aliado a isso, os estudantes de programação adotam metodologias de estudo que são ineficientes. Eles decoram soluções para os problemas ao invés de entendê-las [GOMES; MENDES, 2007], estudam de maneira passiva ou superficial [BIGGS; TANG, 2011; CLARK; NGUYEN; SWELLER, 2005] e fazem poucos exercícios [GOMES; MENDES, 2007]. Além disso, os estudantes de programação possuem um tipo de conhecimento dito frágil, que é aquele conhecimento que eles possuem, mas não conseguem aplicar em novas situações ou problemas encontrados [PERKINS; MARTIN, 1986].

Contudo, o maior dos problemas vivenciados pelos iniciantes em programação não parece ser o entendimento dos conceitos básicos de uma LP, mas a combinação e a utilização adequada destes conceitos na construção de um determinado programa [LAHTINEN et al., 2005; ROBINS et al., 2003; SPOHRER; SOLOWAY, 1989; WINSLOW, 1996]. Sendo assim, a maioria dos erros cometidos pelos estudantes não resulta de falhas no conhecimento sobre os construtos da LP, mas resulta de dificuldades em combiná-los corretamente, ou seja, *"em colocar as peças do programa juntas"* [SPOHRER; SOLOWAY, 1989 p. 401].

Ser professor de uma disciplina introdutória de programação é uma tarefa tão complexa quanto ser estudante desta disciplina. Ou seja, o desafio de ensinar é proporcional ao de aprender. Razão pela qual muitas universidades escolhem como docentes destas disciplinas os professores mais experientes e com mais vivencia nas questões relacionadas ao ensino de programação. Programar vai requerer habilidades que são diferentes daquelas necessárias para ensinar a programar. A literatura relata algumas decisões que precisam ser tomadas pelo professor, tais como a LP e as

ferramentas associadas e as práticas pedagógicas do professor [PEARS et al., 2007]. No que diz respeito às práticas pedagógicas, as preocupações do professor devem ir além da apresentação do conteúdo aos estudantes em sala de aula ou da seleção de exercícios relacionados ao conteúdo. Estas preocupações devem incluir também a proposição de atividades que auxiliem e orientem os estudantes durante os seus processos de aprendizagem. Sendo assim, a atuação docente deve acontecer em pelo menos dois momentos, na definição das atividades de ensino e de aprendizagem dos estudantes que estejam alinhadas com os objetivos de aprendizagem definidos para a disciplina introdutória de programação [BIGGS; TANG, 2011].

Dentro deste contexto, o presente trabalho tem por objetivo discutir alguns dos desafios enfrentados pelos professores no planejamento e na execução de atividades de ensino e de aprendizagem e apresentar oportunidades de pesquisa na área de educação em programação para iniciantes. Além disso, apresentamos a proposta de uma abordagem para minimizar alguns dos desafios elencados. Para alcançar estes objetivos, o restante deste trabalho encontra-se organizado da maneira descrita a seguir. Na Seção 2 apresentamos os desafios e oportunidades elencados. Na Seção 3 descrevemos uma proposta para minimizar alguns dos desafios discutidos e apresentamos os resultados obtidos com um experimento realizado com a proposta. Por fim, na Seção 5 apresentamos as considerações finais deste trabalho.

## 2. Alguns desafios e oportunidades

Como mencionamos anteriormente, o papel do professor de programação deve buscar orientar o processo de aprendizagem dos estudantes na disciplina [BIGGS; TANG, 2011]. Sendo assim, discutimos nesta seção alguns dos desafios enfrentados pelos professores no planejamento e na execução destas atividades e apresentamos oportunidades de pesquisa na área de educação em programação para iniciantes.

### 2.1. Professor deve propor atividades de ensino

Uma das principais questões a serem consideradas pelo professor no ensino de programação diz respeito às suas práticas pedagógicas. A primeira coisa que um professor de programação pensa é a respeito de quais atividades de ensino adotar. Neste contexto, uma possibilidade é a utilização de exemplos trabalhados como parte do material instrucional proposto pelo professor.

Os exemplos trabalhados[1] (*worked examples*) são uma sequência de passos que demonstram a realização de uma tarefa em particular ou a solução de um problema específico [CLARK et al., 2005]. Os exemplos trabalhados auxiliam os estudantes iniciantes na construção de novos esquemas nas fases iniciais do processo de aquisição da habilidade cognitiva [ATKINSON et al., 2000; RENKL, 2002; VANLEHN, 1996]. Exemplos são considerados, tanto por estudantes quanto por professores, o tipo de material instrucional mais útil para a área de educação em LP [LAHTINEN et al., 2005].

[1] Utilizaremos os termos *exemplos trabalhados* e *exemplos* indistintamente no decorrer deste trabalho.

Dada a sua importância como ferramenta de ensino, é imprescindível que a escolha dos exemplos leve em consideração questões, tais como a sua qualidade e legibilidade [BÖRSTLER; CASPERSEN; NORDSTRÖM, 2015], a sua estruturação [CLARK et al., 2005], o formato por meio do qual ele é apresentado aos estudantes [BENNEDSEN; CASPERSEN, 2008] e o seu alinhamento com os objetivos de aprendizagem que se desejam alcançar [BIGGS; TANG, 2011].

### 2.2. Professor também deve propor atividades de aprendizagem

Para fazer com que os estudantes aprendam de maneira eficaz, as preocupações do professor devem ir além da apresentação do conteúdo aos estudantes em sala de aula ou da seleção de exercícios relacionados ao conteúdo. Neste sentido, é preciso que o professor faça com que os estudantes adotem metodologias de estudo mais eficazes. Portanto, é necessário que o professor de programação inclua em suas práticas pedagógicas atividades de aprendizagem que estimulem os seus estudantes a processarem o material instrucional mais profundamente.

Neste sentido, atividades de aprendizagem que promovam a reflexão [CHI et al., 1989], que façam com que os estudantes sejam sujeitos ativos e participantes do seu próprio processo de aprendizado [BRANSFORD; BROWN; COCKING, 2000] e que os encoraje a ser menos superficiais [BIGGS; TANG, 2011] têm demonstrado resultados positivos para o processo de aprendizado dos estudantes.

Dada a sua importância como ferramenta de aprendizado do estudante, é imprescindível que as atividades propostas estejam de acordo com os objetivos de aprendizagem que se deseja alcançar [BIGGS; TANG, 2011].

## 3. Uma proposta de solução para os desafios identificados

Tendo em vista os desafios relacionados aos processos de ensino e de aprendizagem de programação para iniciantes, uma abordagem é apresentada para minimizar os problemas elencados. Trata-se da abordagem *Stepwise self-explanation* [AURELIANO; TEDESCO, 2015] para o aprendizado de programação, proposta com o objetivo de orientar os estudantes iniciantes enquanto eles estudam para uma disciplina de programação.

### 3.1. *Stepwise self-explanation*

A abordagem *Stepwise self-explanation* lança mão de dois referenciais teóricos principais, o framework do *Stepwise Improvement* (FSI) [CASPERSEN, 2007; CASPERSEN; KÖLLING, 2009] e a teoria das auto-explicações (AE) [CHI et al., 1989]. O FSI é um framework conceitual que descreve a atividade de programação como um processo sistemático e incremental que engloba três atividades diferentes, extensão, refinamento e restruturação [CASPERSEN, 2007; CASPERSEN; KÖLLING, 2009]. A atividade de extensão ocorre quando a especificação é estendida de maneira a cobrir mais casos de uso. A atividade de refinamento ocorre quando o código abstrato é modificado de maneira a se construir código executável que implemente a especificação correspondente. A atividade de restruturação ocorre quando é realizada uma melhoria dos aspectos não funcionais do programa, no entanto essa modificação não envolve uma mudança no comportamento aparente do programa. O FSI oferece uma contribuição

significante à área de educação em programação fornecendo orientação nas três atividades que engloba, oferecendo orientação na maneira que materiais instrucionais (e.g., livro-texto, exercícios, aulas e exemplos) podem ser estruturados e na construção de programas propriamente dita.

Apesar do FSI fornecer orientação na estruturação e organização de materiais instrucionais, ele não oferece nenhum tipo de orientação na maneira que os estudantes podem estudar e aprender programação a partir destes materiais instrucionais. Em virtude disso, propomos que este tipo de orientação aconteça através da AE [CHI et al., 1989]. A AE consiste em *"um diálogo mental que aprendizes possuem enquanto estudam um exemplo trabalhado e que os ajuda a entender o exemplo e a construir um esquema a partir deste exemplo"* [CLARK et al., 2005]. De acordo com Chiu e Chi (2014), a atividade de auto-explicar, ou explicar para si mesmo, promove o aprendizado através da elaboração da informação que está sendo estudada, da associação dessa nova informação com a conhecimento prévio que o aprendiz possui, da construção de inferências e da conexão dos diferentes pedaços de informação.

Para esta abordagem, escolhemos exemplos como parte das atividades de ensino. Os exemplos trabalhados são apresentados normalmente de maneira estática, mas decidimos apresentá-los no formato de vídeo. Adotamos os exemplos em vídeo, pois eles possuem o formato ideal para exibição de programas para iniciantes em uma disciplina de programação, podendo combinar a apresentação dos programas com todo o seu processo de construção [BENNEDSEN; CASPERSEN, 2008]. Como atividades de aprendizagem, escolhemos estimular e orientar o processo de reflexão dos estudantes através da inclusão de questões para que eles auto-expliquem os exemplos que estão estudando. Para que pudéssemos incluir as questões para AE nos exemplos em vídeo, estruturamo-os de forma que eles tivessem duas partes distintas. A primeira delas é a apresentação do enunciado do problema seguido por exemplos de execução do programa com entradas específicas e as saídas correspondentes (atividade "Estende especificação" na Figura 1). Em seguida, a segunda parte consiste na apresentação de uma possível sequência de passos que soluciona o problema apresentado. Para facilitar o processo de aprendizado dos iniciantes em programação, decidimos construir vídeos que apresentam somente sequências de atividades de extensões e refinamentos. Assim, a cada novo passo da solução executado no vídeo, teremos uma extensão intermediária (atividade "Estabelece especificação intermediária" na Figura 1) e um refinamento (atividade "Refina implementação" na Figura 1).

As questões, por sua vez, foram adicionadas aos vídeos em quatro momentos distintos de sua execução: (i) um após a apresentação do enunciado do problema e de seus exemplos de entrada e saída (atividade "Auto-explica extensão principal" na Figura 1); dois a cada novo passo da solução apresentada: (ii) um após se informar a funcionalidade que será implementada no passo e antes de se apresentar a implementação para o referido passo (atividade "Antecipa refinamento" na Figura 1) e (iii) outro após a implementação para o passo ser apresentada (atividade "Auto-explica refinamento" na Figura 1); e (iv) o último após apresentar a solução completa do problema (atividade "Avalia código entradas específicas" na Figura 1).

As questões apresentadas no momento (i) tem por objetivo fazer com que o estudante auto-explique o propósito geral do programa apresentado no exemplo. As

questões apresentadas no momento (ii) tem por objetivo fazer com os estudantes reflitam a respeito de uma possível solução para o passo de resolução do problema antes da solução do professor ser apresentada e, depois comparem a solução pensada por eles com a solução apresentada pelo professor. As questões apresentadas no momento (iii) tem por objetivo fazer com que os estudantes façam inferências sobre a implementação apresentada pelo professor através da proposição de modificações no código apresentado. As questões apresentadas no momento (iv) tem por objetivo fazer com os estudantes calculem mentalmente a saída do código a partir de uma determinada entrada, de forma que eles reflitam sobre a execução do programa apresentado.

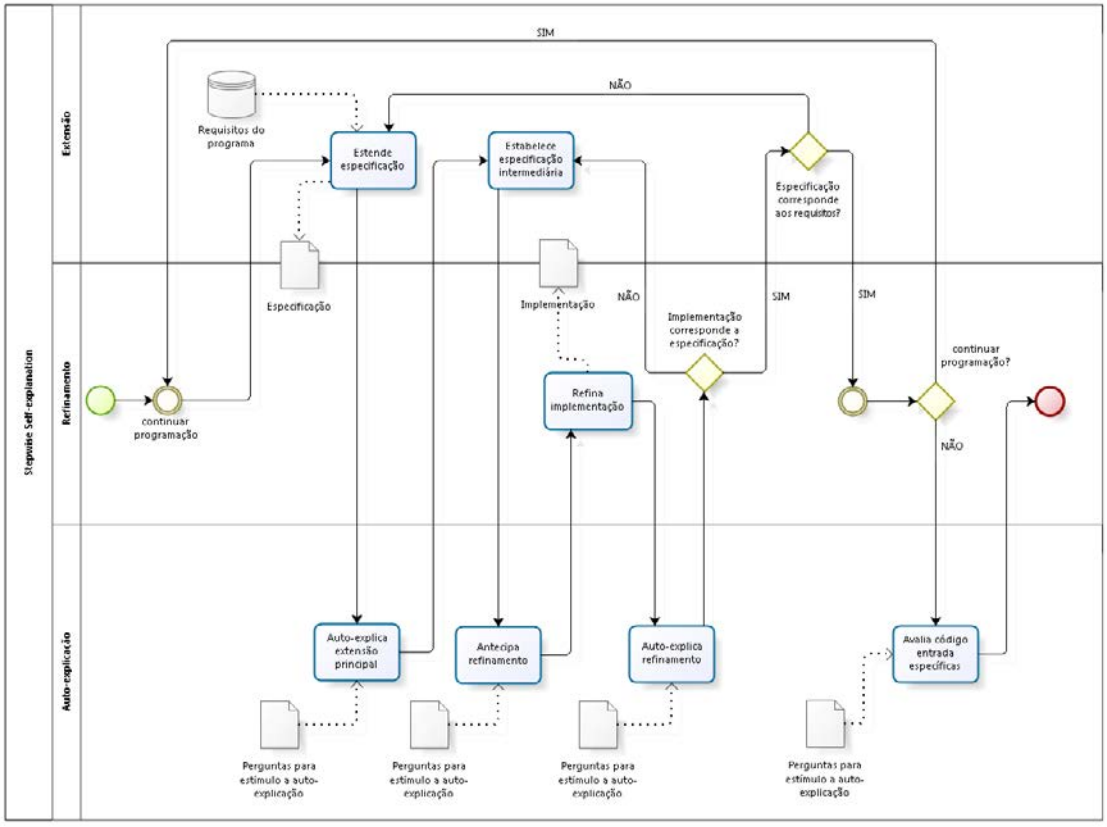


**Figura 1. Abordagem *Stepwise self-explanation*.**

Para ilustrar este procedimento, apresentamos na Figura 2 um trecho de um dos exemplos em vídeo produzidos segundo a abordagem, onde utilizamos a ferramenta *Scratch* para a construção de um jogo. Este trecho apresenta o passo "fazer com que o gato movimente-se para cima quando a seta para cima for pressionada" que é parte do problema "fazer com que o gato se movimente através das quatro setas direcionais". Juntamente com o trecho de vídeo (Figura 2 (c)), apresentamos também as questões para os momentos anterior (Figura 2 (a)) e posterior (Figura 2 (b)) à implementação do passo de resolução do problema. A cada questão apresentada, o vídeo pausa; em seguida, a questão é carregada e o estudante continua o vídeo após a submissão de sua resposta.

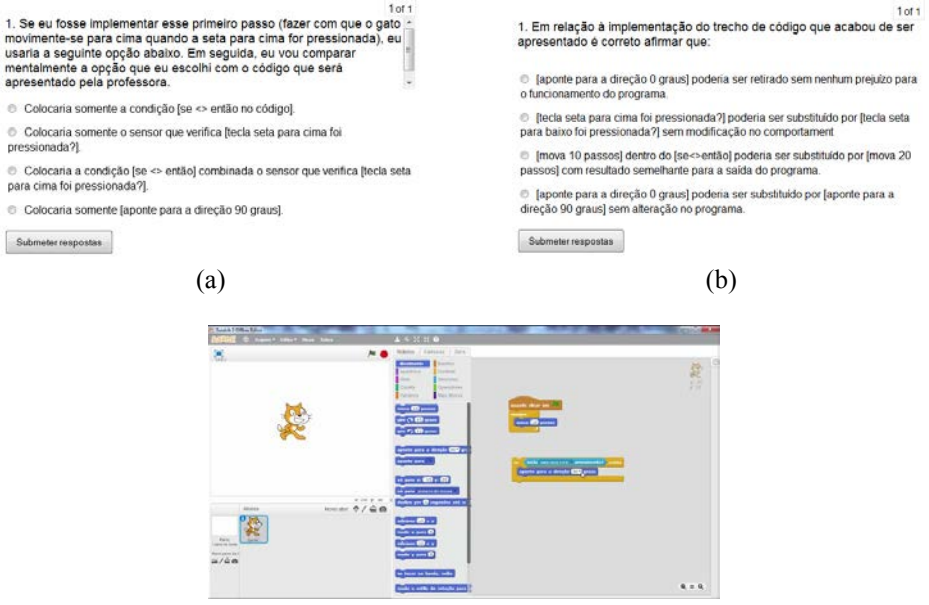


**Figura 2. Trecho de exemplo em vídeo, apresentando passo da implementação do problema (c) juntamente com as questões anterior (a) e posterior (b) à implementação do passo do problema.**

### 3.2. Experimentando a abordagem *Stepwise self-explanation*

Como forma de avaliar o potencial da abordagem *Stepwise self-explanation*, realizamos um experimento com estudantes de uma instituição de ensino que atua na educação de nível médio, técnico e superior no Estado de Pernambuco, no primeiro semestre de 2015. Este experimento foi realizado como um minicurso de introdução à programação através da construção de jogos digitais através da ferramenta *Scratch*. Portanto, o minicurso teve como objetivo de aprendizagem:

[OA1] Aplicar os conceitos de variáveis, estruturas de repetição e de decisão para a construção de um jogo usando a ferramenta *Scratch*.

Para alcançar este objetivo, a atividade de ensino do minicurso combinou períodos de aula expositiva seguidos por períodos de estudo individual dos exemplos em vídeo. As atividades de aprendizagem, por sua vez, consistiram em um conjunto de questões para promover a reflexão dos estudantes enquanto eles a partir dos exemplos em vídeo apresentados. Sendo assim, os materiais instrucionais produzidos para este experimento foram: (i) apresentação utilizada para guiar a aula expositiva; (ii) um conjunto de quatro vídeos que exibiam a aplicação dos conceitos apresentados na aula expositiva; (iii) pré e pós-testes para avaliar o desempenho dos estudantes antes e após o minicurso ministrado; (iv) um conjunto de questões para a AE.

Durante a execução do experimento, dois grupos foram organizados, o grupo controle (GC) e o grupo instrucional (GI). O GC fez uso de exemplos trabalhados em vídeos, enquanto que o GI fez uso dos mesmos exemplos em vídeo em conjunto com as questões de AE desenvolvidas para estes vídeos. Os resultados obtidos por ambos os grupos no pós-testes podem ser observados na Tabela 1. Com base nestes resultados, obtidos a partir das estatísticas descritivas e no âmbito do experimento realizado,

podemos dizer que aprender programação a partir da abordagem *Stepwise self-explanation* trouxe resultados superiores quando comparados à condição controle.

**Tabela 1. Médias e desvios padrão (entre parênteses) dos resultados dos pré e pós-testes dos GC e GI.**

| | GC (n=5) | GI (n=7) |
|---|---|---|
| Pós-teste | 8.2 (5.12) | 10.57 (2.94) |

Como complemento aos resultados positivos obtidos, os estudantes do GI e do GC tiveram uma opinião positiva a respeito da qualidade e utilidade dos exemplos e das explicações fornecidas pela professora nos exemplos em vídeo. Além disso, os estudantes do GI tiveram uma opinião positiva a respeito da qualidade e utilidade das questões empregadas para promover a AE dos vídeos.

## 4. Considerações finais

Neste trabalho, discutimos alguns desafios enfrentados pelo professor no planejamento e execução de atividades relacionadas aos processos de ensino e de aprendizagem de disciplinas introdutórias de programação, bem como apresentamos algumas oportunidades de pesquisa na área de Educação em Computação. Apresentamos a abordagem *Stepwise self-explanation* como uma proposta alternativa auxiliar a compor o projeto pedagógico e a definição das práticas pedagógicas para minimizar os desafios elencados. Além disso, demonstramos o potencial da abordagem através da realização de um experimento realizado como um minicurso de Introdução à Programação por meio da construção de jogos digitais e do feedback positivo fornecido pelos estudantes integrantes do experimento.

Cientes das restrições inerentes do estudo e, resgatando as dificuldades associadas ao trabalho de campo envolvendo docentes e discentes em ambiente escolar, pretende-se replicar a experiência em outras escolas e buscar trabalhar com segmentos específicos de alunos, organizando por faixa etária, tipo de curso e formação prévia. Entendemos que o trabalho realizado nos permitiu elencar indicadores de que o uso da auto-explicação pode colaborar para o aluno organizar seu pensamento e processos cognitivos, hábitos e atitudes. Entretanto, tudo isto funcionará se estiver devidamente contextualizado no planejamento do docente. Esta é uma das técnicas e atividades a serem consideradas no conjunto de possibilidades que temos para organizar o ensino de programação para iniciantes.

## Referências

ATKINSON, R. K. et al. Learning from Examples: Instructional Principles from the Worked Examples Research. **Review of Educational Research,** v. 70, n. 2, p. 181-214, 2000.

AURELIANO, V. C. O.; TEDESCO, P. C. D. A. R. **Aprendendo linguagem de programação através da auto-explicação de exemplos em vídeo**. Proceedings of the IV Congresso Brasileiro de Informática na Educação. Maceió - Alagoas - Brasil 2015.

BENNEDSEN, J.; CASPERSEN, M. E. Failure rates in introductory programming. **SIGCSE Bull.,** v. 39, n. 2, p. 32-36, 2007.

_____. Exposing the Programming Process. In: BENNEDSEN, J.;CASPERSEN, M., *et al* (Ed.). **Reflections on the Teaching of Programming**: Springer Berlin Heidelberg, v.4821, 2008. cap. 2, p.6-16. (Lecture Notes in Computer Science). ISBN 978-3-540-77933-9.

BIGGS, J.; TANG, C. **Teaching For Quality Learning At University**. McGraw-Hill Education, 2011. ISBN 9780335242757.

BÖRSTLER, J.; CASPERSEN, M.; NORDSTRÖM, M. Beauty and the Beast: on the readability of object-oriented example programs. **Software Quality Journal**, p. 1-16, 2015/02/14 2015.

BRANSFORD, J. D.; BROWN, A. L.; COCKING, R. R. **How People Learn: Brain, Mind, Experience and School**. Washington, DC: {N.R. Council} National Academy Press, 2000.

CASPERSEN, M. E. **Educating Novices in the Skills of Programming**. 2007. 323 DAIMI Aarhus University

CASPERSEN, M. E.; KÖLLING, M. STREAM: A First Programming Process. **Trans. Comput. Educ.,** v. 9, n. 1, p. 1-29, 2009.

CHERENKOVA, Y.; ZINGARO, D.; PETERSEN, A. **Identifying challenging CS1 concepts in a large problem dataset**. Proceedings of the 45th ACM technical symposium on Computer science education. Atlanta, Georgia, USA: ACM**:** 695-700 p. 2014.

CHI, M. T. H. et al. Self-explanations: How students study and use examples in learning to solve problems. **Cognitive Science,** v. 13, p. 145-182+, 1989.

CLARK, R. C.; NGUYEN, F.; SWELLER, J. **Efficiency in Learning: Evidence-Based Guidelines to Manage Cognitive Load**. Wiley, 2005. ISBN 978-0-7879-7728-3.

DU BOULAY, B. Some difficulties of learning to program. In: (Ed.). **Studying the Novice Programmer**, 1989. p.431-446. ISBN 1317786203.

GOMES, A.; MENDES, A. Learning to program-difficulties and solutions. **International Conference on Engineering Education–ICEE,** v. 2007, 2007.

GOMES, A. J.; SANTOS, A. N.; MENDES, A. J. A study on students' behaviours and attitudes towards learning to program. Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, 2012. ACM. p.132-137.

KINNUNEN, P.; MALMI, L. Why students drop out CS1 course? , Proceedings of the second international workshop on Computing education research, 2006. ACM. p.97-108.

KURLAND, D. M. et al. A study of the development of programming ability and thinking skills in high school students. **Journal of Educational Computing Research,** v. 2, n. 4, p. 429-458, 1986.

LAHTINEN, E. et al. **A study of the difficulties of novice programmers**. Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education. Caparica, Portugal: ACM**:** 14-18 p. 2005.

MCCRACKEN, M. et al. **A multi-national, multi-institutional study of assessment of programming skills of first-year CS students**. Working group reports from ITiCSE on Innovation and technology in computer science education. Canterbury, UK: ACM**:** 125-180 p. 2001.

MILNE, I.; ROWE, G. Difficulties in learning and teaching programming—views of students and tutors. **Education and Information technologies,** v. 7, n. 1, p. 55-66, 2002.

PEARS, A. et al. A survey of literature on the teaching of introductory programming. **SIGCSE Bull.,** v. 39, n. 4, p. 204-223, 2007.

PERKINS, D. N.; MARTIN, F. **Fragile knowledge and neglected strategies in novice programmers**. Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers. Washington, D.C., USA: Ablex Publishing Corp.**:** 213-229 p. 1986.

PUTNAM, R. T. et al. A summary of misconceptions of high school Basic programmers. In: (Ed.). **Studying the novice programmer**, 1989. p.301-314. ISBN 1317786203.

RENKL, A. Worked-out examples: instructional explanations support learning by self-explanations. **Learning and Instruction,** v. 12, n. 5, p. 529-556, 10// 2002.

ROBINS, A. V.; ROUNTREE, J.; ROUNTREE, N. Learning and Teaching Programming: A Review and Discussion. **Computer Science Education,** v. 13, n. 2, p. 137-172, / 2003.

SAMURÇAY, R. The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. **Studying the novice programmer,** v. 9, p. 161-178, 1989.

SILVA FILHO, R. L. L. E. et al. A evasão no ensino superior brasileiro. **Cadernos de Pesquisa,** v. 37, p. 641-659, 2007.

SPOHRER, J. C.; SOLOWAY, E. Novice mistakes: are the folk wisdoms correct? In: (Ed.). **Studying the novice programmer**, 1989. p.401-416. ISBN 0805800034.

TEW, A. E.; MCCRACKEN, W. M.; GUZDIAL, M. Impact of alternative introductory courses on programming concept understanding. Proceedings of the first international workshop on Computing education research, 2005. ACM. p.25-35.

VANLEHN, K. Cognitive skill acquisition. **Annu Rev Psychol,** v. 47, p. 513-39, 1996.

WATSON, C.; LI, F. W. B. **Failure rates in introductory programming revisited**. Proceedings of the 2014 conference on Innovation & technology in computer science education. Uppsala, Sweden: ACM**:** 39-44 p. 2014.

WINSLOW, L. E. Programming pedagogy—a psychological overview. **SIGCSE Bull.,** v. 28, n. 3, p. 17-22, 1996.