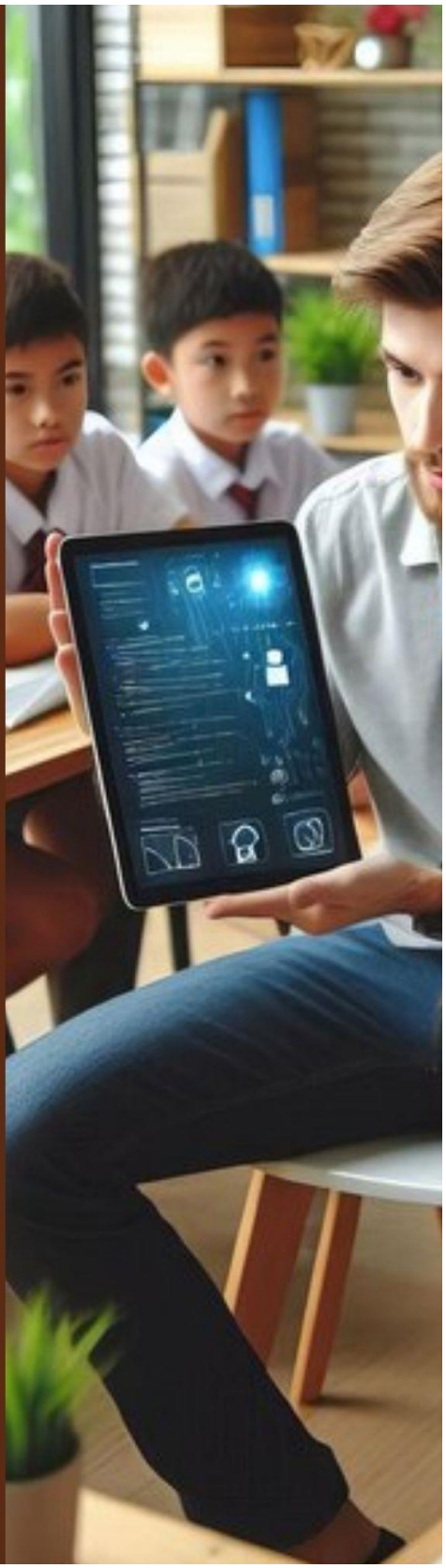
ISSN : 2455-9164

International Journal of EDUCATION TEACHING AND LEARNING

www.trpubonline.com/journals.php
trpub.online@gmail.com
or ijedtl@trpubonline.com



INTERNATIONAL JOURNAL OF EDUCATION TEACHING AND LEARNING

https://ijetl.com/index.php/ijetl
Vol. 9, No. 1, 2024
© Thomson & Ryberg Publications. All Rights Reserved

A Systematic Literature Review on Teaching and Learning Introductory Programming In Higher Education

E.Muralidhar ReddyB.Venkateshwarlu,N.Ch.Raju,Dr.M.Sreenivasulu
Department of Education teaching,Kakatiya university Warangal india

Abstract—Contribution: **Thispaperaddstotheresultsofprevious systematic literature reviews by addressing a more con-temporarycontextofintroductoryprogramming.Itproposesa categorization of introductory programming challenges, andhighlights key issues for a research roadmap on introductoryprogramminglearningandteachinginhighereducation.**

Background: **Despite the advances in methods and tools forteaching and learning introductory programming, dropout andfailure rates are still high. Published surveys and reviews eithercover papers only up to 2007, or focus on methods and tools forteachingintroductoryprogramming.**

Research Questions: **1) What previous skills and backgroundknowledgearekeyforanovicestudenttolearnprogramming? 2) Whatdifficultiesdonovicestudentsencounterinlearninghow to program? 3) What challenges do teachers encounter inteachingintroductoryprogramming?**

Methodology: **Followingaformalprotocol,automaticandman-ualsearcheswereperformedforworkfrom2010to2016.Of100 papers selected for data extraction, 89 were retained afterqualityassessment.**

Findings: **Themostfrequentlycitedskillsnecessaryforlearningprogramm ingwererelatedtoproblemsolvingandmath-ematical ability. Problem solving was also cited as a learningchallenge, followed by motivation and engagement, and difficul-ties in learning the syntax of programming languages. The mainteaching challenges concern the lack of appropriate methods andtools,aswellasscalingandpersonalizedteaching.**

***Index Terms*—Achievement, faculty development, higher edu-cation, introductory programming, STEM, student experience,systematicreview.**

I. Introduction

INTRODUCTORY programming courses are part of vari-ousundergraduatecurricula,particularlyinSTEMdegrees (Science, Technology, Engineering, and Mathematics). In thispaper, "introductory programming" (commonly called CS1 inthe United States) refers to a course for novice students thattypically covers problem-solving skills, basic programmingconcepts, the syntax and semantics of a programming lan-guage, and the use of this programming language in order to address the problem. Several subsequent classes are affected by a failing grade in an introductory programming course [1]. There has been gradual incorporation of programming basics into high school curriculum [2] and improvements in techniques and tools for teaching and learning introductory programming [2], [3], but there is still a high incidence of dropout and failure in these courses [4,5]. Nobody seems to agree on the most pressing problems[1,3–5] or have a firm grasp on how to classify them [6, 7]. Prior studies and evaluations of the literature have either concentrated on resources and techniques for teaching basic computer science concepts [11] or included works published up to 2007 [1, 8–10]. Therefore, a comprehensive literature study is required that does the following: (a) clarifies and organises basic programming challenges; (b) incorporates the most current research; and (c) takes into account the views of educators and students. The following is the outline of the paper: Attempts to organise research findings on introductory programming are discussed in Section II. Section III explains the systemic approach that was used, and Section IV explains how it was applied to the work that is being shown here. The study questions' responses are presented and analysed in Section V. Section VI delves further into the responses, pinpointing potential critical concerns with teaching and understanding basic programming. This research is compared to others in Section VII, which focusses on qualitative aspects. In Section VIII, we detail our findings and plans for the future.

II. PreviousStudies

Most of the surveys and systematic reviews on introduc-tory programming teaching and learning do not cover researchresults after 2008 [1], [8]–[10], and focus more on teachingmethods and tools than on student problems. A more recentsystematicreviewofintroductoryprogrammingexaminedsuc-cessful teaching practices [11]. Evaluating tools, methods, andpractices for teaching introductory programming is essential,but given that dropout and failure rates are still high [4], [5],and that the

literature shows no consensus on the main prob- lemsinvolved[1],[3],[4],thereisaneedforunderstanding

and categorization. Indeed, a better understanding of challengesinintroductoryprogrammingmayhelptodevelopbetterteachingtools,methods,andpractices.

III. METHOD

Theprotocoladoptedinthisreviewfollowstheguidelinesforsystematicliteraturereviewspresentedby KitchenhamandCharters[12]andPetticrewandRoberts[13].

A. *ResearchQuestions*

RQ1: What previous skills and background knowledge arekeyforanovicestudenttolearnprogramming?

RQ2:Whatchallengesdonovicestudentsencounterinlearninghowtoprogram?

RQ3: What challenges do teachers encounter in teachingintroductoryprogramming?

Theboundariesbetweenlearning(RQ1andRQ2)andteaching(RQ3)areobviouslyfuzzy,especiallyinascomplexasubjectasprogramming.Nevertheless,thisreviewkeepsthisdistinctiontoprovideabetterunderstandingofchallengesinintroductoryprogramming,alreadyproposedinaprogrammingcontext[6],[7].

B. *SearchProcess*

The search process started with a manual search in specificjournals chosen for their relevance to the subject: the ACMTransactionsonComputingEducation(TOCE),theIEEETransactions on Education, and Computer Science Education.A fourth source of data entry for the manual search was a sys-tematic literature review written in 2014 and often cited in theliteratureofthearea[11].

Amanualsearchisfrequentlycompletedbyanautomaticsearchonscientificdatabasesusingasearchstring.Recurringkeyterms—suchas“programming”,“programminglanguage”,“programmingteaching”,“computerprogramming”,“coding”,“CS1”,and“noviceprogrammers”—were identified in the papers from the manual search. Thesearch string for the automatic search was built from thesetermsand,afterpreliminarysimulations,wasdefinedas:(“learning programming” OR “teaching programming”) AND(“noviceprogrammers”OR“CS1”).

Thedatabaseschosenfortheautomaticsearchwere:ACMDigitalLibrary,IEEEXploreDigitalLibrary,SpringerLink,ScienceDirectandERIC(theEducationResources Information Center, sponsored by the Institute ofEducation Sciences). Before doing the automatic search, thesearch string was validated by being used in the IEEE andACMdatabases.Thepapersretrievedwerecross-checkedwith the papers selected manually (by applying inclusion andexclusion criteria - see below) from IEEE and ACM journalsrespectively.

C. *SelectionCriteriaandProcedure*

Papers included in the review had to be written in English,and published in conferences or journals, or as book chapters,between2010and2016,onthethemeofteachingandlearningintroductoryprogramminginhighereducation.

Paperswereexcludedifthey:

1) Didnotaddresstheresearchquestions;
2) Weretooshort,suchasworkshoppapers;
3) Werepublishedinlocalconferences;
4) Were written by the same research group with the samedata(inwhichcaseonlythemostrecentwaskept).

Becauseofthelargenumberofpapersretrievedinthesearch,atwo-stepprocedurewasperformed.First,apre-selectionstepappliedtheselectioncriteriaonthebasisoftitle, keywords and abstract. Next, the full text of the pre-selectedpaperswasanalyzed,andduplicates(criterion4)wereexcluded.

Two researchers carried out the selection process, both ofwhom independently analyzed each paper. The reasons forinclusionorexclusionwerecarefullynoted,andmeetingswereheld to resolve any disagreements, with the help of a thirdresearcher.

D. *Snowballing*

The results from both manual and automatic searches, afterthe application of the selection criteria, were “snowballed”,that is, the bibliographical references of all the selected paperswere considered as potential studies, and were then analyzedaccordingtotheinclusionandexclusioncriteria.

E. *QualityAssessment*

Thequalityoftheselectedpaperswasassessedagainstcriteria selected and adapted from Kitchenham & Charters’guidelines [12], focusing on the methods adopted and theirscientificrigor.

For papers based on field studies (qualitative and quantitativeapproaches),thecriteriawere:

1) Howwellwasthedatacollectioncarriedout?
2) Howwellwastheapproachto,andformulationof,theanalysisconveyed?
3) Howwellwerethecontextsanddatasourcesretainedandportrayed?
4) Howclearandcoherentwerethelinksbetweendata,interpretation,andconclusions?

Fortheoreticalpapers,thecriteriawere:

1) Howwelldidtheanalysisaddressitsoriginalaimsandobjectives?
2) Howhasknowledgeorunderstandingbeenextendedbytheresearch?
3) Howwellwasdiversityofperspectiveandcontextexplored?

Theassessmentwasperformedindependentlybytworesearchers, and each paper was scored on the scale (for eachcriterion): 0 - very poorly; 1 - poorly; 2 - reasonably; 3 - well;4 - very well. For scores that differed between the researchersby two or more points, a meeting was held with the thirdresearcher to resolve the conflict. A quality threshold wasadoptedtodecidewhethertokeepthepapersintheanaly-sis. The scores obtained for each question were averaged togivethefinalscoreforeachpaper.

F. DataCollectionandAnalysis

The following data were extracted from each study andrecordedinthecodebook(inspreadsheetformat):

- Search mechanism (manual, automatic or snowballing)and source (journal name, database name and article title,respectively);
- Title, keywords, authors, publication venue, type (journalorproceedings),andyear;
- Objectiveandparticipants;
- Scientific method (qualitative and/or quantitativeapproach,andspecificmethodsadopted);
- Summaryofthestudy;
- Answerstoeachresearchquestion;
- Qualityassessment;

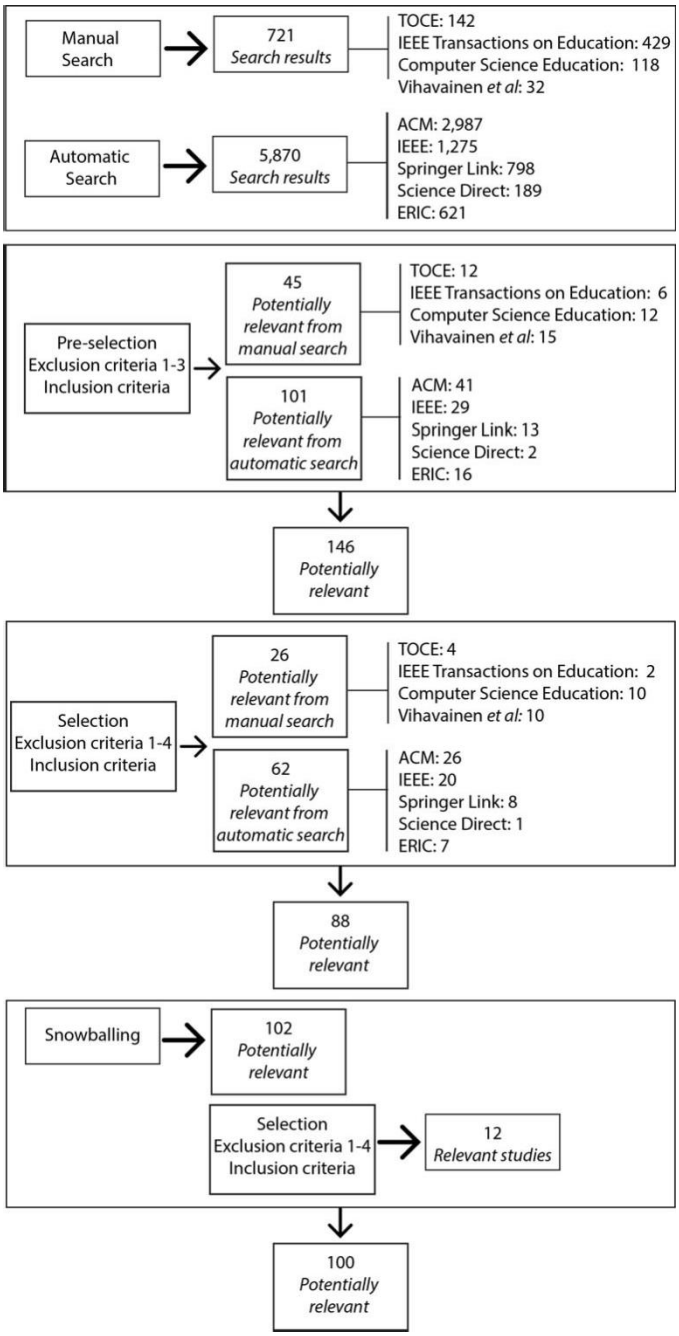
Oneresearcherextractedthedata,withtheotherchecking the extraction. Any disagreements were discussed untilresolved;ifnecessary,thethirdresearcherwasinvolved.

To analyze the answers to the research questions, a bottom-up inductive approach was adopted. Firstly, key terms werecollectedexactlyasusedineachpaper.Differenttermsusedtorefertosimilarconceptswereidentified,showingthatauthorschoosewordsratherfreelyindiscussingatopic,rather than following a standard terminology (perhaps drawnfrompriorexperienceorpreviousexperienceinprogramming).Suchsynonymswereconsolidatedunderthemostfrequently-used terms. Then related concepts were groupedinto categories under each research question, to clarify thepresentation and discussion of the results, Section V. Thisgrouping also drew on the literature where applicable (forexampleontheoriesonComputationalThinking[14],[15]),as discussed in Section V. Some terms were used by someauthorswith slightly different meanings;these are discussedinSectionV.Thefirsttworesearchersperformedtheinductiveanalysistogether,thenthethirdrevisedandvalidatedit.

IV. Method Application

A. SearchandSelectionResults

ThesearchforpapersrelevanttothereviewfollowedtheprocesslaidoutinFig.1.Themanualsearchpro-vided 721 potentially relevant papers, from which 45 werepre-selected. The automatic search yielded 5,870 potentiallyrelevantpapers,fromwhich101werepre-selected.Thisgaveatotalof146pre-selectedpapers.Inthesecondphaseoftheselectionprocess,26papersfromthemanualsearchand62papersfromtheautomaticsearchwereretained,giving88 relevant papers in total. The snowballing process on these88 selected papers provided another 102 candidates, of which12 were selected. The final total at the end of the whole pro-cess was thus 100 papers. These are listed as Selected PapersintheReferencesectionofthispaper[R01]–[R100].



B. QualityAssessmentResults

Toassessthequalityoftheselectedstudies(byansweringaseriesofquestions—seeSectionV),thescientificmeth-ods they adopted to support their arguments were collected,TableI.Thisinformationpersecanbeofinteresttothecomputerscienceeducationcommunity.

onlysixofthe100papershadscoresforatleastoneofthecrite-ria that differed by two or more points. In the histogram of thequality assessment scores, Fig. 2, the x-axis labels the upperborderofeachhistobarrange.

Most studies received a good score, which is to be expectedsince they were published in peer-reviewed scientific journalsandconferences.Afewstudies,however,obtainedsignif-icantlylowscores,andwerenotconsideredofacceptablequality to be included in the analysis. The threshold scoreadopted was 1.81, which corresponds to one of the apparentdiscontinuitiesinthehistogram.Consequently,11papers(of

TABLEI
OVERVIEW OF METHODS

Method	Selected References
Pedagogical intervention evaluated through analysis of student performance and/or feedback	R04, R08, R09, R11, R12, R17, R18, R22, R23, R24, R25, R26, R27, R29, R30, R35, R38, R40, R41, R42, R43, R44, R45, R48, R51, R53, R55, R56, R57, R58, R59, R60, R61, R62, R63, R64, R66, R74, R75, R78, R79, R81, R82, R83, R85, R89, R92, R94, R96, R98, R99, R100
Survey, questionnaires, interviews and /or focus groups	R03, R06, R07, R10, R11, R12, R15, R19, R20, R21, R28, R31, R32, R33, R34, R35, R37, R41, R42, R44, R45, R46, R47, R50, R52, R53, R54, R55, R57, R62, R65, R61, R69, R70, R71, R76, R80, R81, R83, R87, R93, R95, R97
Observation of classroom activities	R03, R04, R09, R19, R28, R44, R50, R51, R83
Analysis of students' learning styles and strategies	R13, R33, R45, R51, R67, R70, R74, R85, R88
Analysis of student performance with regard to type of assessment, pedagogical approach or contextual factors	R64, R66, R67, R70, R73, R75, R77
Analysis of student errors	R36, R65, R84
Comparison between teachers' predictions/perceptions and students' actual performance	R01, R16
Theoretical reflection based on teaching experience	R02, R49, R68, R72, R86
Literature review	R05, R14, R39, R68, R71, R72, R74, R75, R80, R81, R86, R90, R91

the 100) were excluded. The results and discussion below onlyconsidertheremaining89papers.

C. ProtocolLimitations

Thisreviewsharesthemostcommonlimitationsofthesystematic method: search coverage and possible biases intro-duced during study selection, data extraction, and analysis.Theselimitationswereaddressedfollowingthegeneralrecom-mendations for systematic reviews—using a combined manualandautomaticsearchcomplementedbyasnowballingprocess,and havingtwoormoreresearchersselectingstudies,assessingqualityandextr actingdata[12],[13].

Thespecificlimitationofthestudylies inthefactthatthestatedresearchq uestionsrequireanswersthatarenotbinary or objective. This review required a categorization ofterms used by different primary studies to characterize intro-ductory programming challenges. However, as is typical ofeducation-related issues, the categories have fuzzy boundaries,so a given lexical term is used by different authors with thesamemeaning.

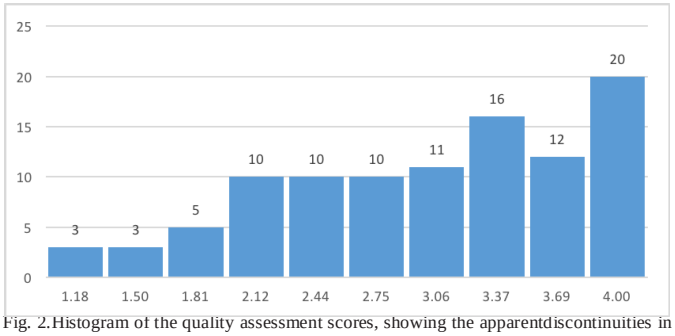


Fig. 2.Histogram of the quality assessment scores, showing the apparentdiscontinuities in the threshold score adopted. X-axis labels give the upperborderofeachhistobarrange,andthenumberofpapersinthatrange.

TABLEII
STUDENT SKILLS

Category	Skill	References
Programming -Related Skills	problem solving	R04, R09, R10, R15, R19, R27, R39, R41, R51, R67, R73, R74, R75, R77, R80, R87
	mathematical ability	R04, R07, R10, R28 R38, R44, R45, R49, R69, R73, R75, R77, R80, R85, R86
	previous knowledge in programming	R06, R15, R23, R29, R48, R67, R69, R73, R75, R77, R86
	abstraction	R07, R12, R19, R28, R81, R86
General Educational Skills	basic knowledge in English	R01, R23, R38, R75, R80
	critical thinking and discussion skills	R10, R12, R14, R28
	creativity	R04, R81 R12, R28, R87, R97
	time management	

V. RESULTS

A. SkillandBackgroundKnowledge(RQ1)

The skills extracted from the papers were grouped into twocategories:Programming-RelatedandGeneralEducational,TableII.

*1) Programming-RelatedSkills:*Problem solving,whichcan be defined as understanding the context of a problem,identifyingkeyinformation,andmakingaplantosolveit[16],was among the most cited skills for answering RQ1 (16 pub-lications).Itisconsideredaprerequisiteforlearninghowto program: "*There are researchers that emphasize that thecomputer programming requires the use of problem-solvingtechniques.Thelackofmasteryofthesetechniquesmakesitdifficult tolearnprogramming.*"[R39,p.02];andeventhemaingoalofintroductory programmingcourses:"*[thegoalis]tofamiliarizethestudentstotheartofp roblem*

solving and to teach them how to express solutions in a simpleway"[R15,p.68];and"*thelong-termgoalsfortheintroductory programming course are that the students shouldeventually develop problem-solving ability and design compe-tence*" [R41, p. 130]. It is also important that students cansolve problems efficiently when working cooperatively withpeers,teachers,andmentors[R39].

Mathematical ability is cited as a necessary skill closelyrelated to problem-solving ability. Fifteen studies argue thatdeficientmathematicalabilityisonereasonforthediffi-culty encountered by novice learners of programming: "*Whentalking with programming teachers, most of them claim thatstudents don't know how to program because they don't knowhow to solve problems and do not have enough mathematicalbackground*" [R07, p. 113]; "*Moreover many students did notshow basic mathematical skills that are expected when enter-ing higher education (...). Students had difficulties in calculus;Students did not have enough basic mathematical conceptsconcerning number theory; Students had difficulties to trans-formatextualproblemintoamathematicalformulathatsolvesit;(...)Studentshadweakabstractionlevels;Studentshadlackof logical reasoning*" [R45, pp. 01–02]. However, the correla-tion between mathematics and programming is not necessarilya cause-effect one: "*some teachers believe that mathematicalability is important for programming. However, they believethatthepositivecorrelationbetweenthemisduetomorebasiccognitive functions that would be common to both domains.Thus, when developing a "mathematical logic", the studentacquiresstructuresorcognitiveabilitiesthatfacilitateorpromotelearningtoprogram*"[R28,p.04].

Eleven publications discuss the impact of students' previousknowledgeinprogramming;thereisnoconsensusasto its consequences. For instance, [R15] and [R29] considerpreviousknowledgeinprogrammingasapredictorofsuc-cess for novice learners in higher education: "*Although a fewof the students have already learnt some programming lan-guage before getting enrolled in Computer Science degreeprogram at university level, and it is believed that prior pro-gramming has a positive impact on Computer Science studies(...)*" [R15, p. 64]; "*Students who studied programming in thepast tend to do well compared to those with no programmingexperience.*"[R29,p.296].

However,theassociatedbenefitsarenotstraightforward:"*Wenormally thinkoftheabilitytocodeasthemainbenefitofhavingpriorprogramminge xperience,butstudentinterviewssuggestthattheimpactofthisexperienceismorecomplex:itaffectsstudents'expectations,workhabits,attitudeandconfidence,andperceptionsofselfandpeers*"[R06,p.244].Finally,abstraction iscited insix studies. Wing[17]definesabstraction as "*the most important and high-level thought processincomputationalthinking(...)usedindefiningpatterns,generalizingfromspecificinstances,andparameterization(...)tocaptureessentialpropertiescommontoasetofobjects*".Indeed,"*aprogrammerneedstoapplyabstractionwhenanalyzingcomputationalproblems,aswellastoinstantiate abstractprogrammingconceptsandtechniquestosolvepar-ticular computational problems*" [R12, p. 624]; "*It is expectedthatstudentswilluseit[abstraction]topassfromtheconcrete worldtoamoresemanticandsymbolicreality,formulatingorrepresentingaprobleminamoreabstractmanner,less attached to the details and singular concrete elements.When a student faces difficulties while attempting to do thisabstraction,he/sheisnotcapableofmovingfromrealitytoamoregenericlanguagesuchasthenotationusedinprogramming*"[R28,p.04].

*2) General Educational Skills:*Critical thinking and dis-cussion skills were cited in four publications, associated withthestudents'levelofmaturityattheuniversity.Theyaffectthe interpretation and analysis of problems, and figure in thelowincidenceofclassroomdebate.

Four studies comment on the importance of creativity inprogramming, for example, "*it is important to stress that thesestudents also believe that mathematical dexterity is importantas well as creativity, even though they state that programmingisbasicallyreasoning*"[R28,p.04].

FivepublicationsaddthatlimitedknowledgeinEnglishis a barrier, as it is used in the syntax of all consolidatedprogramming languages. Reference [R23] argues that, in thecourse they analyzed, fluent English speaking students weremorelikelythannon-fluentonestobesuccessful.

Finally,timemanagementwasalsocitedasanessentialskillby [R04] and [R81], in terms of planning use of time and thestagesintheexecutionofaproject.

B. DifficultiesEncounteredbyStudents(RQ2)

ThefirstthreecategoriesinTableIIIarerelatedtothethreestagesofcomputationalthinking(problemformula-tion, solution expression and solution execution and evaluation)presentedbyRepenning,BasawapatnaandEscherle[14, pp. 268–269]. The remaining category is behavior, relatedtosocial,emotionalandself-managementaspects.

*1) ProblemFormulation:*Asexpected,problem-solvingreappeared as a challenge faced by students, with 23 studiesidentifying this as a difficulty. These echo the arguments discussedwithrespecttoRQ1inSectionV-Aabove:"*Asmaybe expected, the learners face more difficulties to establishrelationsbetweendifferentprobleminstances.Conceivably,astudentwhohasgainedalittlefamiliaritywiththisproblem-solvingapproachshouldprovidelogicallycorrectanswers*" [R33, p. 164]; "*Many of these [students] end updropping the course due to not being able to solve problemsandthereforefeelinginadequate.*"[R26,p.93].

Another problem cited, also discussed under RQ1, concernsthe abstract nature of programming, with similar arguments:"*(...) students in learning programming need to imagine andcomprehend many abstract terms that do not have equivalentsin real life: how does a variable, a data type, or a memoryaddressrelatetoareal-lifeobject?Theseconceptsaredifficulttograsp.Consequently,manystudentsstruggletocomprehendeven the most basic of programming concepts*" [R49, p. 16];*"Students face diverse problems when they are learning pro-gramming (...) mainly because programming is dynamic andabstract"*[R54,p.414].

Algorithmic reasoning, which appeared in three studies, isunderstoodhereas *"apoolofabilitiesthatareconnectedto*

TABLEIII
STUDENT DIFFICULTIES

Category	Difficulty	References
problem formulation	problem solving	R10, R14, R15, R16, R22, R26, R33, R37, R39, R41, R44, R45, R48, R49, R70, R73, R74, R81, R87, R89, R92, R94, R99
	abstract nature of programming	R10, R14, R33, R49, R54, R99
	algorithmic and logical reasoning	R28, R41, R99
solution expression	syntax of programming language	R02, R09, R15, R18, R21, R27, R33, R36, R39, R42, R44, R48, R49, R74, R82, R83, R84, R87, R90, R92
	control structures	R16, R21, R25, R27, R30, R43, R44, R48, R65, R74, R86, R90
	data structures	R21, R30, R34, R61, R90
	structure of the code	R21, R25, R34, R65
	others	R03, R21, R25, R34, R53
solution execution and evaluation	debugging	R02, R14, R18, R21, R27, R48, R55, R84, R87, R89, R90, R91, R96
	tracing the execution	R14, R16, R19, R27, R42, R53, R61, R68, R84
behavior	motivation and engagement	R03, R04, R10, R11, R12, R13, R17, R20, R23, R24, R26, R30, R44, R45, R48, R55, R56, R70, R71, R72, R73, R74, R75, R77, R79, R80, R84, R87, R88, R89, R90, R94, R96, R97, R98, R100
	time management	R01, R13, R44, R46, R50, R55, R56, R69, R80, R81, R88, R93, R94, R96, R100
	study skills	R10, R13, R46, R50, R56, R69, R75, R80, R93, R94, R97
	confidence	R10, R56, R69, R75, R85, R92, R93, R98, R100
	perception of programming as a complex discipline	R03, R12, R21, R65, R81, R93

constructing and understanding algorithms: to analyze givenproblems; to specify a problem precisely; to find basic actionsthatareadequatetothegivenproblem;toconstructacorrectalgorithm to a givenproblemusing the basic actions"[18, p. 160]. According to teachers, "*(...) it is this shift inperception that allows them [the students] to see the wholebasedonthepartsthatwillallowthestudenttoelaboratethealgorithm.Inwordsofoneoftheteachers,whenthinking*

this way, the student 'no longer thinks as a person but startsthinking as a computer, taking into account the machine'slimitedcomprehension."[R28,pp.03–04].

*2) Solution Expression:*Once the problem is formulated,students can move on to expressing a solution through pro-gramming structures. The first problem within the solutionexpressionscope,with20occurrences,concernsthesyntaxof programming languages, as reported by [R48, p. 02]: *"Theoverhead of learning the syntax and semantics of a languageat the same time, and difficulties in combining new and previ-ous knowledge and developing their general problem-solvingskills".* According to [R49, p. 02], "*even students, who haveadequate problem-solving skills and manage to phrase a solu-tiontoaprogrammingproblemintermsofapseudocode,findit difficult to turn the pseudo code into a syntactically correctcomputer program.*" In this context, it is important to considersyntaxerroraspartofthelearningprocessandhelpstu-dents learn to recover from them [R36], [R48]: "*Syntax errorsare an important area of programming pedagogy research.Experiencedprogrammersrarelymakesyntaxerrors,andwhe ntheydo,theyhaveclearstrategiestocorrectthemvery quickly. However, syntax errors are significant for noviceprogrammers; correcting them is a time-consuming processand often leads to random debugging behavior, also influ-enced by the fact that students do not understand compilermessages"*[R48,p.16].

Control structures also appeared as a challenge (13 occur-rences). Among them, conditional statements [R25], [R43],loops[R14],[R25],[R27],[R43],[R48],[R65],[R86],[R90], recursion [R18], [R27], [R90], and sequence, selection andrepetition[R21],[R30]werethemostcited.Selectingtheappropriate structures for solving a problem (if, if/else, with,for,while)wasalsocited[R21]asadifficulty.

Several papers also mention difficulties related to the use ofdata structures. References [R21] and [R34] explore this topicthe most deeply, but only on the basis of previous literature,and not on their findings. Data structures most often cited aspotentiallydifficultwerearrays[R21],[R30],[R61],[R90].

For code structure, some papers mention the use of func-tions/classes [R21], [R65], the relationship between classesandobjects[R25],andusinglanguagelibraries[R34].Finally, other aspects of solution expression cited were: point-ers[R03],[R21],[R34],[R53],[R65],references[R34],[R65], parameters [R25], [R34], [R65], variable scoping [R53] anderrorhandling[R34].

3) Solution Execution and Evaluation: Students should testand analyze their code to identify and correct problems. In thiscontext, debugging (13 publications) and code tracing (ninepublications)werethemostcitedchallenges.

Debuggingisdescribedasacomplexactivity,requiringsev-eral aspects to be mastered: *"Debugging, one of the essentialskills for successful programmers, is difficult for novices as itrequires the application of many new skills simultaneously.Students must understand the problem domain, know rudi-mentary programming concepts and understand at least oneprogramming language well enough to read and write instruc-tions, comprehend the logic of the intended program, and beabletotrackdownandfixbugs"*[R55,p.390].Itdemands

muchpracticeand shouldbe explicitlysupportedby teachers:*"debuggingshouldbeexplicitlytaught.Assumingstudentswill simply "pick up" debugging skills as a by-product oflearning to program may lead them to develop some of theineffective strategies observed in this study"* [R55, p. 395].Reference [R02] states that students rarely test their code, per-haps because they do not know how to formulate proper testcases, or because they lack the discipline to test their codesystematically.

Finally, students should also be given the opportunity fordeveloping code tracing skills, as cited by nine papers, sincethis helps them understand programming more holistically.Thisactivitymayberelatedtodebugging,butcanalsobeused to understand code passages. The authors argue that:*"code tracing activities exposed students' not viable modelsand, more importantly, provided a much clearer starting pointwithwhichtodiscusscorrectmodels"*[R53,p.562].

*4) Behavior:*This category encompasses social and emotionalaspectsthatcanhaveanimpactonstudentlearning,suchas motivation, engagement, and confidence (including the per-ception of programming as being difficult), as well as studyhabitsandtimemanagement.

The most discussed issue in this category (36 occurrences),is the relationship between students' motivation and engage-ment, and positive learning outcomes [R14]: *"For learners,engagement correlates with improvements in specific desirableoutcomes(...),suchasgeneralabilitiesandcriticalthinking, cognitive development, improved grades and persistence"*[R14,p.02].Theperceptionofprogrammingasa complex discipline, coupled with the previously discussedlearningchallenges,contributestostudents'demotivation,particularly in the first year of higher education [R10]: *"(...) anunmotivatedstudentwillhardlysucceed.Thisisaggravatedby the fact that programming courses usually gain a reputa-tion that passes from student to student of being difficult thatso many of them start already feeling defeated"* [R12, p. 02].*"The motivational dimension has a great impact on the individual'scognitivedevelopmentandisadeterminantfactorfor success in the learning process (…). It is motivation thatfosters in the student the disposition to want to progress andreachthegoalsthatwereset,maintainingan adequatelevelofvolitiontoovercomethedemandsthatarebeingdealtwith"*[R13,pp.1–2].

Students' confidence is another issue (nine occurrences).According to [R10, p. 02], *"some students may have moti-vation but the self-confidence may be blocked, as such thesetypesofstudentsneedtheteachertoworkcloserwiththemto develop self-esteem".* Similarly, [R98, p. 14] argues that*"Student programmers who lack confidence are less able tomake independent progress with coding exercises. They frequentlybecome"stuck",andwillwaitforassistancefromtheinstructor,ratherthantryanalternativeapproachontheir own*". Confidence in programming is expected to changerapidlyasthestudentgainsexperience[R98].

Fifteen publications mention students' poor time manage-ment as another challenge: *"most students indicated that theyregularly spent far less time studying for the unit than the recommendedeighthours[perweek]"*[R46,p.125].Students

TABLEIV
FACULTY CHALLENGES

Challenges	References
methods and tools for teaching programming	R13, R18, R19, R24, R28, R35, R41, R46, R47, R49, R53, R57, R58, R59, R60, R61, R63, R64, R70, R72, R73, R74, R76, R78, R79, R81, R82, R88, R89, R93, R98, R99, R100
scale problems	R03, R09, R10, R32, R44, R50, R57, R68, R72, R93, R96
keeping students' motivation, engagement and persistence	R03, R10, R12, R14, R20, R24, R27, R51, R78, R79, R80, R81, R86, R87, R89, R93, R96
teacher-student communication and feedback	R10, R14, R18, R24, R27, R29, R41, R76, R82, R83, R85, R87, R90, R91, R96, R97
programming language	R05, R15, R19, R47, R48
curriculum and instructional sequences	R03, R51, R75
addressing students' inadequate mathematical background	R28, R90, R91

themselvesacknowledgethisdifficulty:*"Anotherconstantlyrecurringtheme inthefreecommentssectionwastimemanagementissues.Manystudentsreportedthattheyhavetoomanyconcurrentcourses,allwithhighworkload.Withoutcareful time management and prioritizing, there is not enoughtime for deep-level learning in every study topic"* [R44, p. 04].Anotherissue isstudyskills (elevenoccurrences)suchasorganizationand minimalwork habits[R10], or the comprehensionofstudents'ownlearningstyles[R46],[R75],[R80]:"*Theacquisitionofwell-functioninglearningstrategiesandskillsseems,inthisspecificcontext,tobeaffectedbygroupworkstrategies,extrinsicmotivation,andsomeissuesrelatedto study habits. It is not always easy to establish a direction ofcausality,butoneimportantaspectinlearningprogrammingseemstobetheworkthatstudentsarerequiredtodoontheirowntime,outsidetheinstructedlearningsessions.Foronereasonoranother,intoomanycasesstudentsarenotableto findeffectivewaysofworking(...)"*[R80,p.306].

C. FacultyChallenges(RQ3)

Challengesfacedbyfacultywhenteachingintroductoryprogramming are listed in Table IV. Obviously, learning chal-lenges (RQ1 and RQ2) would naturally appear in RQ3 asteachingones.Toavoidredundancy,onlyteachingchal-lenges corresponding to the most-cited learning difficulties,Table III, were included in the discussion here. Other teachingchallengesthathavenotyetbeendiscussedarealsotreatedhere.

*1) Revisited Challenges:*Maintaining student motivation,engagement and persistence is fundamental (17 occurrences),butteachersstruggletofindstrategiestoattractstudents'interest[R10].Reference[R24,p.499]arguesthatthe

That "not many teachers realise that it is also their responsibility to motivate the students and arouse the students' interests" and that teachers should "be a motivator, not just a provider" are two important points. "R24," page 499. According to reference [R51], teachers should provide students a solid foundation in basic problem-solving skills and fundamental concepts before presenting more complex problem-solving plans. This will ensure that students have the necessary background knowledge to understand and benefit from the more advanced plans. As a result, less-advanced beginners should be more motivated to stay in introductory classes. [R51, page 311]. The third idea that is strongly connected to this is maintaining student involvement: "... improving the student engagement can be different from case to case and must be elaborated into practical approach and strategies." In the current educational style, not all students are engaged. Additionally, some teachers do not pay attention to how their students react to different teaching strategies in class. Lastly, students may feel overwhelmed by the amount of work they have to do, which can affect their "quality of effort." (R14, page three). Additionally, students' weak mathematical foundations are reviewed, and educators are encouraged to address this issue by developing targeted lessons and/or modifying their pedagogical strategies: We suggest including exercises that assist students build mental models appropriate to the cognitive abilities required for programming into the curriculum in the hopes that this would improve their performance in the class. Although the very concept of problem-based learning is to foster problem-solving skills, our research shows that this approach is insufficient when dealing with students who pose more challenging problems. [Page 06, R28]. 2) Approaches and Resources for Programming Education: Teaching techniques and tools are the greatest difficulty in RQ3, as mentioned by 33 articles. This overarching issue highlights many difficulties highlighted in RQ3. Learning through doing and example-based learning, problem-based learning, recorded classes, active learning exercises, live coding, canned examples, and a trace-driven approach to teaching are all proposed methods. Other approaches include extreme apprenticeship, gamification, games, team-based learning, media computing, collaborative learning, tutoring sessions, mentor support, peer instruction, and pair programming. Applying these strategies has not yet shown definitive results, albeit [11]. This apparent mismatch in teacher expectations and student behaviour could provide universities the urge to reassess the sort of educational experiences we provide for our students, as reference [R46] reinforces: "The current teaching methods need to be rethought." R46, page 128. Research on the use of certain pedagogical approaches in beginning computer science courses has shown mixed results, with some methods failing to provide desired results for reasons that remain puzzling. "Like many institutions, our introductory classes suffered from low retention rates and poor student performance," reads one account of a somewhat successful program (R53, pp. 562-563). The addition of active learning activities and the use of live coding and canned examples to demonstrate the functionality of code features were among the pedagogical adjustments we made, yet these challenges persisted still. that is not a These courses already included possibilities for students to construct accurate mental models include in-class code tracing and coding activities, laboratories, minor programming homework problems, bigger programming projects, and other similar possibilities. Some of these things seemed to harm more than they helped, according to our observations. However, an experiment utilising a trace-driven teaching strategy was reported by [R53] and it resulted in a 25.49% reduction in dropouts and an 8.51% decrease in grade fails. The need to prioritise and modify teaching techniques in order to enhance problem-solving abilities is discussed in reference [R47]. This is one of the most often mentioned talents required for learning programming (RQ2). According to reference [R49], there aren't enough resources to effectively teach the object-oriented paradigm. The selection of tools and the first programming language provide significant challenges for educators, as noted in reference [R19].

3) Issues with Scale: Factors such as student heterogeneity [R09], class size [R03], staff resource limitations [R32], [R44], [R72], [R93], [R96] and personalised instruction [R09], [R10], [R50] are all a part of the scale topic, which is linked to the number and diversity of students in classes.

Working with pupils who have varying degrees of background knowledge, levels of dedication, and preferred methods of learning makes it difficult for teachers to hone their problem-solving abilities [R10]. Additionally, it is quite challenging to attend to cognitive demands and individual challenges in big groups [R10]. The availability of staff is critical since mentors and tutors may enhance the quantity of activities and evaluations. (R03, [R32], [R44]). Engaging pupils with extrinsic motivators also requires preparation and additional work [R03]. Six articles mention the fact that students vary in terms of their knowledge, motivation, dedication, and pace of learning: "Emphasising the development of problem-solving skills and accommodating the wide range of cognitive needs, learning styles, difficulties, and motivations among a diverse group of students is a formidable challenge." Consequently, it is challenging for the educator to use a strategy that is appropriate for each and every student (R10, p. 01). In their pursuit of inclusivity, educators often construct lessons and assignments for the "typical student," a demographic that may not really exist inside the classroom. In order to meet individual requirements and obstacles, personalised help and advice are essential in improving this situation. [C09, page one]. The impact of staff resource limits on instruction is discussed in five publications: Limited staff resources and university control mechanisms were among the contextual difficulties addressed by the general problems (...). In the long term, a course revision should strive to optimise the demands and uses of resources, not the other way around. Adding more student assignments and changing the course needed more resources. (R03, page 23). Teaching would be more fluid with more individuals participating, including tutors and mentors, according to most instructors. This would lead to better classroom dynamics and less gaps in students' learning cycles. (R03, [R32], [R44]).

4) Feedback and Communication Between Instructors and Students: The feedback process and student-teacher contact can provide challenges (16 times): "Effective educational theory for

teachingprogrammingshouldfocusonstudents'learning,andeffective communication between teacher and student. Thiscould be achieved by clearly stating goals and keep the stu-dents motivated" [R14, p. 104]. The way of evaluating andprovidingfeedbacktostudentscanbeadeterminingfactorin their demotivation [R09]–[R12], [R14], [R29], particularlyfor women [R71]. However, giving quality feedback is notsimple:*"Givefeedback,notonlyjudgment.Assessmentcanbeclassifie dintotwocategories:formativeandsummative.In formative assessment we can get the feedback of the cur-rent teaching and learning condition and use the feedback toimprove them. On the other hand, the summative intention issolely to form a judgment"* [R24, p. 499]. Besides its inherentcomplexity, the feedback process is also affected by lack oftime,quantityofstudentsorformatofcourses.

Moregenerallyspeaking,therearetworeasonsthatteacher-studentcommunicationseemstofail:toomanystudentsinoneclass [R10]; and students' resistance to discussing their errorsin exercises due to lack of rapport [R24]. There is a clear rela-tionship between the communication and feedback challenges,andthecategoryofscaleproblems.

*2) Choice of Programming Language:*The most specificintroductory programming challenge is choosing the first pro-gramming language to be taught to students. Several program-ming languages can be used for introductory programmingcourses, as discussed by [R21, p. 02]: "*The available pro-gramminglanguagesarenumerousandselectingtheonethatwillbeusedis amulti-criteriadecision".*Thischoicehasadirectimpactonthedevelopmentofno vices'pro-grammingskills[R48]andcanbekeyinfacilitatingtheteaching process [R05] and shaping programming style andcoding technique [R15]. Reference [R47] discusses the choicebetweencommercialandacademiclanguages:*"Theteacheris exposed to risks when teaching programming courses fornovices in which the emphasis is on the programming lan-guage. One of the risks is that the novices focus their attentionon syntax issues and not on the computational semantic poweroftheIanguage,whichattheendofthedayiswhatmakesit possible to build solutions using computer programs. Thisapproach prevents novices from understanding that the mainrole of a programming language is to serve as a means toexpresscomputationalsolutionsproposedinthetrainingexer-cises. Moreover, choosing a standard de facto language in theindustryofferstheadvantageoftrainingthestudenttodevelopskills that the market is looking for. However, it can also gen-erate a bias of the concept of the student regarding the futilityof learning other languages and, besides, it can reduce theenvironment in which students develop the ability to learn tolearn"*[R47,p.02].

3) Curriculum and Instructional Sequences:[R03] high-lightstheideathattheintroductoryprogrammingcourseshould build more explicitly upon skills acquired in previouscourses. *"At the curriculum level, departments should developexplicitcoursesequencesthatbuilduponthesk illsacquiredinothe rcoursestoincreasethestudentintrinsicmotivationto complete the courses as planned. The departments shouldalsotracktheperformanceoftheircoursestobeabletoactonproble msearly,sincerehabilitatingacourseisalengthy process. (...) In general, a competent instructor may be able todeliver a successful course without all the proposed systemicframeworks"*[R03,p.23].

There also seems little research on instructional sequencesfor teaching programming for novices in higher education, aspointed out by [R51]: *"(...) nostudy has directly comparedthe effectiveness and efficiency of the instructional sequencesused in presenting programming material to novices. Is it bet-ter to begin with a strategic overview, to start with syntaxdetails and work upward, or to work through entire program-ming concepts one at a time? Knowing the effect of sequenceon instructional effectiveness can help instructional designersavoid sequences that make learning unnecessarily difficult inanalreadydifficultdomain"*[R51,p.292].

VI. GeneralDiscussion

Some issues for inclusion in the research roadmap on teach-ingandlearningintroductoryprogramminghaveemergedfromthisreview :

A. Better Understanding and Characterization ofProblem-SolvinginProgramming

Problem solving is a crucial concept in introductory pro-gramming. Some authors in this review [R09], [R15], [R41],[R48],[R49]considerthattheprimaryaimofanintroduc-tory programming course is to develop problem-solving skillsthrough algorithmic thinking and basic concepts of program-ming,ratherthantoteachthesyntacticparticularitiesofaspecificprogram minglanguage.Positivestudentpercep-tions have been reported from approaches aligned with thisperspective [20]. Computer Science Curricula 2013 [21] rec-ommendsthatintroductoryprogrammingcoursesshouldavoidconveying the idea to students that computer science is mainlyabout learning the specifics of a programming language, andinstead emphasize general concepts in computing within thecontextoflearninghowtoprogram.

Although problem solving appears in several articles, itsdefinitions are generic, lacking, or inconsistent across authors.The reasons for students' limitations in problem solving arenot detailed: is it because they do not understand the statementof the problem, do not know the strategy to solve it, or do notknow a step-by-step plan to implement the strategy? How isproblem-solvingrelatedtoalgorithmicthinking,computationalthinking,orabstract ion?Isproblem-solvingamonolithicissue, or does it encompass various sub-problems? And if so,how are they related? Some current definitions of computa-tional thinking [15] may help. Deeper investigations should becarried out to understand what problem-solving really meansinprogramming.

B. ImprovingBackground Knowledge

Students'inadequatebackgroundknowledge(mainlyofmathematics and English) is cited by introductory program-ming teachers as a challenge, but how can they "solve" thisproblem within the timeframe of an introductory program-mingcourse?Effortstoteachprogrammingtochildrenand

teenagersatschool [2]areexpectedtoimprovethissitua-tion for the future generations of programming students, butthisisnotcertain.Methodsandmetricsshouldbeestab-lished to evaluate the efficacy of this approach. In a broaderperspective, it could be desirable to foster a productive andsystematic exchange of information between school and uni-versity teachers, with the involvement of policymakers whodefine curricula at all levels. This would help bridge gaps instudents' competencies between school and university. Suchinteractionshouldbeencouraged.

C. Better Specific Tools and Methods for ProblemFormulationandSolutionExpression

Someteachersbelievethatlearningtoproperlyformulatea problem and express its solution in a specific programminglanguage, in the timeframe of an introductory programmingcourse, is a great cognitive load for students. The idea ofseparating problem formulation strategies and solution expres-sion into two different courses is gaining ground, but thisassumes that problem framing and expression is one of theroots of the learning problems in introductory programming.Moreempiricalevidenceisneeded.

Moremethodsandtoolsarealsoneededifthesetwophases are to be developed separately. Nowadays, block-basedprogrammingenvironmentsforbeginnershelpdevelopingcomputational thinking, and are not tied to a professional pro-gramming language as taught in university courses. However,thevastmajorityofthesearedesignedforchildren(e.g.,Scratch, Alice, and others from code.org). Although they havebeen shown to help [2], [R61], most are too childish to betransferabletotheuniversitylevel.Buildingsimilartoolshighereducationshouldbeconsidered.

D. ImprovingMotivation

Motivation appears is a significant concern for teachers,with high impact on the other learning problems. Approachesareneededtohelpteachersmaintainstudentmotivation.

Many papers discuss motivation, but only one examines thereasons for students' demotivation [R07]. A more in-depthunderstanding of this phenomenon seems necessary. Severalpromising teaching methods (such as gamification, problem-based learning, or unplugged computing) are available, butthesearestillnotcommonlyadoptedinhighereducation.The barriers to adoption could be teaching culture, insufficientknowledge of the methods, the work required to put them intopractice,orotherunknowns.

E. MoreEmpiricalBasisandStandards

Thelastpointconcernsthebasisuponwhichthepapersana-lyzed deliver their conclusions. Several authors make claimsmainly based on observations made by teachers in their dailyclassroomwork.Althoughtheimportanceoftheseobser-vations cannot be denied, empirical data from more formalexperimentalprojectswouldbewelcome.Whenempiricalresults exist, they are not comparable because the experimen-tal protocol, the number and profile of participants, and eventhegoalsaretooheterogeneous.

Therefore the creation of experimental standards for tasks,participant profiles and number, expected results, and evaluation criteria should be encouraged. Benchmarks for evaluatingmethods and tools for introductory programming would beverywelcome.

VII. ComparingResultsWithPreviousStudies

Studiescoverin gpaperspublishedbefore2010[1],[8]–[10] aresurveysratherthansystematicliteraturereviews.Theydo not follow a rigid protocol in presenting results, so thesecannotbequantitativelycomparedwiththeresultsofthisreview. Nevertheless it is possible to qualitatively comparesome issues from the 1990s or 2000s. Remarkably, the solu-tion expression problems (language syntax, control structures,anddatastructures)highlightedintheearlierstudiesarestill relevant, but are now of similar importance to problemformulation.

Severaltopicslistedasnovices'difficultiesbyRobinsetal.intheirreview from2003[9]werealsoidentifiedinthe present review, including: problem solving (strategies andskills); solution expression (abstraction, use of data structureslike variables and arrays, use of control structures like loops,conditionalstatements,andrecursions);andsolutionexecution(tracing/tracking code, dedicating time to planning and test-ing; and debugging). This 2003 review did not discuss issuesof choosing a programming language and learning syntax, orthe importance of students' mathematical background, algo-rithmic and logical reasoning—all of which were identified inthepresentreview.

Topics involving methods and tools for teaching and learningintroductoryprogrammingwereasignificantconcernin 2003, and, as discussed in the present review, still are.Robins*etal.*[9]discussedtheimportanceoffosteringlearningof core principles of programming, focusing on student learningratherthaninstructorteaching,linkedtomethodssuchas learning-by-doing, problem solving-based methods, peerand collaborative work, and lab-based learning (all of whichshouldbecombinedwithappropriateassessmentandfeedbackgiven through effective student-teacher communication). Theauthors also discussed tools for teaching programming; cur-riculum and instructional sequences, but did not discuss scaleproblems.

From the 2014 review [11, p. 25] focused on evaluatingmethods for teaching introductory programming: *"What maybe missing however, are the reports on interventions that didnot yield an improvement. Thus, educators that have tried anintervention but received poor results should also be encour-aged and supported in reporting the results to create a morestablepictureofthefield"*.

Robins*etal.*[9]discusstheimportanceofengagementand indicate motivation and confidence as potential factors forbetter understanding effective and ineffective novice learners.This confirms the finding of the present review of the growinginterest in student engagement and motivation in recent years.Further topics related to the behavior of novice programminglearners discussed here, such as time management and studyskills,werenotmentionedin[9].

VIII. Conclusion

A systematic literature review was performed on 100 papersto move towards a better understanding of introductory pro-grammingproblems.Withrespecttoprevioussystematicreviews,thisstudyhasasmaincontributions:(i)coveringmorerecent papers; (ii) proposing novel research questions, includ-ing an original discussion on previous skills and backgroundknowledge; (iii) stimulating discussion of a categorization ofintroductory programming challenges; (iv) suggesting poten-tiallykeyissuesforaresearchroadmaponintroductoryprogramminglearningandteaching.

For RQ1, on students' backgroundknowledgeandskills,the most frequently cited issues were problem-solving abil-ities and mathematical knowledge. For RQ2, on challengesfaced by students in introductory programming courses, themajor issues cited were related to motivation and engagement,problem-solving, and the syntax of programming languages.For RQ3, about the challenges faced by faculty, the need forappropriatemethodsandtoolsforteachingprogrammingattheintroductorylevelwerethemostcitedissue.

Thisreviewhasalsoshowntheneedto:(i)clarifytheconcept of problem-solving in programming; (ii) foster thedialogue between the communities of primary and higher edu-cation; (iii) create specific tools for the problem formulationstage;(iv)understandwhypromisingteachingmethodsarenot frequently adopted in higher education; (v) stimulate thecreationofexperimentalstandards.

In future work, the authors aim to pursue some of the issuesraisedinSectionVI,deepeningtheinvestigationbasedona better understanding of problem-solving in programming,followingrecentresultsonthecategorizationofcomputationalthinking [14], [15]. Another stream of future research, alreadyunderway,isananalysisofbarrierstosuccessfulmethodsin basic education on computational thinking being used inhigher education. Finally, the authors intend to create a newsystematic review devoted to the investigation of empiricalmethods,tools,andtypeofdataemployedintheidenti-ficationofproblemsinlearningandteachingintroductoryprogramming,aswellasintheevaluationofsolutions.

Acknowledgment

TheauthorswouldliketothankDr.G.CabralandDr.P.TedescofromUniversidadeFederaldePernambucoand C.E.S.A.R (Centro de Estudos e SistemasAvançados doRecife)forthesupportandassistanceinthepreparationofthispaper.

References

[1] E. Lahtinen, K. Ala-Mutka, and H. M. Järvinen, "A study of the dif-ficultiesof novice programmers," *ACMSIGCSE Bull.*, vol. 37, no. 3,pp.14–18,2005.

[2] M.Resnick*etal.*,"Scratch:Programmingforall,"*Commun.ACM*,vol.52,no.11,pp.60–67,Nov.2009.

[3] S.MohorovičićandV.Strčić,"Anoverviewofcomputerprogram-mingteachingmethods,"in*Proc.CentralEur.Conf.Inf.*,2011,pp.47–52.

[4] C. Watson and F. W. B. Li, "Failure rates in introductory programmingrevisited," in *Proc. Conf. Innov. Technol. Comput. Sci. Educ.*, Uppsala,Sweden,2014,pp.39–44.

[5] J.BennedsenandM.E.Caspersen,"Failureratesinintroductoryprogramming,"*ACMSIGCSEBull.*,vol.39,no.2,pp.32–36,2007.

[6] D.Teague,"Pedagogyofintroductorycomputerprogramming:Apeople-first approach," M.S. thesis, Dept. Inf. Syst., Queensland Univ.Technol.,Brisbane,QLD,Australia,2011.

[7] P.Kinnunen,"Challengesofteachingandstudyingprogrammingata University of Technology—Viewpoints of students, teachers and theUniversity," Ph.D. dissertation, Dept. Comput. Sci. Eng., Helsinki Univ.Technol.,Espoo,Finland,2009.

[8] T.Jenkins,"Onthedifficultyoflearningtoprogram,"in*Proc.3rdHEAConf.ICSLearn.Teach.SupportNetw.*,Loughborough,U.K.,2002,pp.1–8.

[9] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching pro-gramming: A review and discussion," *Comput. Sci. Educ.*, vol. 13, no. 2,pp.137–172,2003.

[10] A.Pears*etal.*,"Asurveyofliteratureontheteachingofintro-ductoryprogramming,"in*Proc.ITiCSE-WGR*,Dundee,U.K.,2007,pp.204–223,doi:10.1145/1345443.1345441.

[11] A. Vihavainen, J. Airaksinen, and C. Watson, "A systematic review ofapproaches for teaching introductory programming and their influenceonsuccess,"in*Proc.ICER*,Glasgow,U.K.,2014,pp.19–26.

[12] B. Kitchenham and S. Charters, "Guidelines for performing systematicliterature reviews in software engineering, version 2.3," School Comput.Sci.Math.,KeeleUniv.,Keele,U.K.,Rep.EBSE-2007-01,2007.

[13] M. Petticrew and H. Roberts, *Systematic Reviews in the Social Sciences:APracticalGuide*.Hoboken,NJ,USA:Blackwell,2008,p.352,doi:10.1002/9780470754887.

[14] A. Repenning, A. Basawapatna, and N. Escherle, "Computational think-ing tools," in *Proc. IEEE Symp. Vis. Lang. Human-Centric Comput.(VL/HCC)*,Cambridge,U.K.,2016,pp.218–222.

[15] M.Berry,*QuickStartComputing:ACPDToolkitforPrimaryTeachers*,BCS,Swindon,U.K.,2015.[Online].Available:http://primary.quickstartcomputing.org/resources/pdf/qs_handbook.pdf

[16] S. Lehoczky and R. Rusczyk, *The Art of Problem Solving, Volume 1:TheBasics*,7thed.Alpine,CA,USA:AoPS,2006,p.288.

[17] J.M.Wing,*ComputationalThinking—WhatandWhy?*LinkMag.,CarnegieMellonUniv.,Pittsburgh,PA,USA,2011,pp.20–23.[Online].Available:http://link.cs.cmu.edu/files/11-399_The_Link_Newsletter-3.pdf

[18] G.Futschek,"Algorithmicthinking:Thekeyforunderstandingcomputerscience,"in*InformaticsEducation—TheBridgeBetweenUsingandUnderstandingComputers.ISSEP2006*(LNCS4226),R.T.Mittermeir,Ed. Berlin, Germany: Springer-Verlag, 2006,pp.159–168,doi:10.1007/11915355_15.

[19] V.Trowler,*StudentEngagementLiteratureReview*,HigherEduc.Acad.,York, U.K., 2010. [Online]. Available: https://www.heacademy.ac.uk/system/files/studentengagementliteraturereview_1.pdf

[20] S. Turner and G. Hill, "Robots inproblem-solving and programming,"in *Proc. 8th Annu. Conf. Subject Centre Inf. Comput. Sci.*, Southampton,U.K.,2007,pp.82–85,doi:10.13140/2.1.2425.6800.

[21] Joint Task Force on Computing Curricula, Association for ComputingMachinery(ACM)andIEEEComputerSociety,*ComputerScienceCurricula2013:CurriculumGuidelinesforUndergraduateDegreeProgramsinComputerScience*.NewYork,NY,USA:ACM,2013,doi:10.1145/2534860.

SelectedPapers

[R01]I.Utting*etal.*,"Afreshlookatnoviceprogrammers'performanceand their teachers' expectations," in *Proc. ITiCSE-WGR*, Canterbury,U.K.,2013,pp.15–32.

[R02] A. N. Kumar, "A mid-career review of teaching computer science I,"in *Proc. SIGCSE 44th ACM Tech. Symp. Comput. Sci. Educ.*, Denver,CO,USA,2013,pp.531–536.

[R03]U.Nikula,O.Gotel,andJ.Kasurinen,"Amotivationguidedholisticrehabilitationofthefirstprogrammingcourse,"*ACMTrans.Comput. Educ.*, vol. 11, no. 4, 2011, Art. no. 24,doi:10.1145/2048931.2048935.

[R04]S. Haatainen, A.-J. Lakanen, V. Isomöttönen, and V. Lappalainen, "Apractice for providing additional support in CS1," in *Proc. Learn.Teach.Comput.Eng.(LaTiCE)*,2013,pp.178–183.

[R05]J.Sorva,V.Karavirta,andL.Malmi,"Areviewofgenericpro-gram visualization systems for introductory programming education,"*ACMTrans.Comput.Educ.*,vol.13,no.4,Nov.2013,Art.no.15,doi:10.1145/2490822.

[R06] A. Tafliovich, J. Campbell, and A. Petersen, "A student perspective on prior experience in CS 1," in *Proc. SIGCSE 44th ACM Tech. Symp. Comput. Sci. Educ.*, Denver, CO, USA, 2013, pp. 239–244.

[R07] A. J. Gomes and A. J. Mendes, "Estudyon student performance in first year CS courses," in *Proc. ITiCSE*, Ankara, Turkey, 2010, pp. 113–117.

[R08] C. Gavan and M. Anderson, "Engaging undergraduate programming students: Experiences using legomindstorms NXT," in *Proc. 13th Annu. Conf. Inf. Technol. Educ. (SIGITE)*, Calgary, AB, Canada, 2012, pp. 139–144.

[R09] A. Santos, A. Gomes, and A. Mendes, "A taxonomy of exercises to support individual learning paths in initial programming learning," in *Proc. Front. Educ. Conf. (FIE)*, Oklahoma City, OK, USA, 2013, pp. 87–93, doi: 10.1109/FIE.2013.6684794.

[R10] A. Gomes and A. Mendes, "A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations," in *Proc. Front. Educ. Conf. (FIE)*, Madrid, Spain, 2014, pp. 1–8, doi: 10.1109/FIE.2014.7044086.

[R11] S. Alhazbi, "ARCS-based tactics to improve students' motivation in computer programming course," in *Proc. Comput. Sci. Educ. (ICCSE)*, Cambridge, U.K., 2015, pp. 317–321, doi: 10.1109/ICCSE.2015.7250263.

[R12] M. A. Brito and F. de Sá-Soares, "Assessment frequency in introductory computer programming disciplines," *ACM Comput. Human Behav.*, vol. 30, pp. 623–628, Jan. 2014, doi: 10.1016/j.chb.2013.07.044.

[R13] A. P. Ambrosio, S. Martins, L. Almeida, A. Franco, and F. Georges, "Assessment of self-regulated attitudes and behaviors of introductory programming students," in *Proc. Front. Educ. Conf. (FIE)*, Seattle, WA, USA, 2012, pp. 1–6, doi: 10.1109/FIE.2012.6462314.

[R14] N. Eltegani and L. Butgereit, "Attributes of students engagement in fundamental programming learning," in *Proc. Comput. Control Netw. Electron. Embedded Syst. Eng. (ICCNEEE)*, Khartoum, Sudan, 2015, pp. 101–106, doi: 10.1109/ICCNEEE.2015.7381438.

[R15] M. Ateeq, H. Habib, A. Umer, and M. U. Rehman, "C++ or Python? Which one to begin with: A learner's perspective," in *Proc. Teach. Learn. Comput. Eng. (LaTiCE)*, Kuching, Malaysia, 2014, pp. 64–69, doi: 10.1109/LaTiCE.2014.20.

[R16] Simon *et al.*, "Can computing academics assess the difficulty of programming examination questions?" in *Proc. Koli Calling*, 2012, pp. 160–163, doi: 10.1145/2401796.2401822.

[R17] D. Horton, M. Craig, J. Campbell, P. Gries, and D. Zingaro, "Comparing outcomes in inverted and traditional CS1," in *Proc. ITiCSE*, Uppsala, Sweden, 2014, pp. 261–266, doi: 10.1145/2591708.2591752.

[R18] T. Wang, X. Su, P. Ma, Y. Wang, and K. Wang, "Ability-training-oriented automated assessment in introductory programming course," *Comput. Educ.*, vol. 56, no. 1, pp. 220–226, 2011.

[R19] J. Holvikivi, "Conditions for successful learning of programming skills," in *Key Competencies in the Knowledge Society* (IFIP Advances in Information and Communication Technology (AICT)), vol. 324, N. Reynolds and M. Turcsányi-Szabó, Eds. Berlin, Germany: Springer, 2010, pp. 155–164, doi: 10.1007/978-3-642-15378-5_15.

[R20] P. Kinnunen and B. Simon, "CS majors' self-efficacy perceptions in CS 1: Results in light of social cognitive theory," in *Proc. ICER*, Providence, RI, USA, 2011, pp. 19–26, doi: 10.1145/2016911.2016917.

[R21] S. Xinogalos, "Designing and deploying programming courses: Strategies, tools, difficulties and pedagogy," *Educ. Inf. Technol.*, vol. 21, no. 3, pp. 559–588, Jul. 2014, doi: 10.1007/s10639-014-9341-9.

[R22] O. Seppälä, P. Ihantola, E. Isohanni, J. Sorva, and A. Vihavainen, "Do we know how difficult the rainfall problem is?" in *Proc. Koli Calling*, 2015, pp. 87–96, doi: 10.1145/2828959.2828963.

[R23] D. Horton and M. Craig, "Drop, fail, pass, continue: Persistence in CS 1 and beyond in traditional and inverted delivery," in *Proc. SIGCSE*, Kansas City, MO, USA, 2015, pp. 235–240, doi: 10.1145/2676723.2677273.

[R24] B. Hartanto, "Enhancing the student engagement in an introductory programming: A holistic approach in improving the student grade in the informatics Department of the University of Surabaya," in *Intelligence in the Era of Big Data. ICSIIT 2015* (Communications in Computer and Information Science), vol. 516, R. Intan, C. H. Chi, H. Palit, and L. Santoso, Eds. Berlin, Germany: Springer, 2015, pp. 493–504, doi: 10.1007/978-3-662-46742-8_45.

[R25] T. Sirkiä and J. Sorva, "Exploring programming misconceptions: An analysis of student mistakes in visual program simulation exercises," in *Proc. Koli Calling*, 2012, pp. 19–28, doi: 10.1145/2401796.2401799.

[R26] A. Vihavainen, M. Paksula, and M. Luukkaine, "Extreme apprenticeship method in teaching programming for begin-ners," in *Proc. SIGCSE*, Dallas, TX, USA, 2011, pp. 93–98, doi: 10.1145/1953163.1953196.

[R27] L. Ma, J. Ferguson, M. Roper, and M. Wood, "Investigating and improving the models of programming concepts held by novice pro-grammers," *Comput. Sci. Educ.*, vol. 21, no. 1, pp. 57–80, 2011, doi: 10.1080/08993408.2011.554722.

[R28] A. P. Ambrósio, F. M. Costa, L. Almeida, A. Franco, and J. Macedo, "Identifying cognitive abilities to improve CS1 outcome," in *Proc. Front. Educ. Conf. (FIE)*, Rapid City, SD, USA, 2011, pp. F3G-1–F3G-7, doi: 10.1109/FIE.2011.6142824.

[R29] L. Porter and D. Zingaro, "Importance of early performance in CS1: Two conflicting assessment stories," in *Proc. SIGCSE*, Atlanta, GA, USA, 2014, pp. 295–300, doi: 10.1145/2538862.2538912.

[R30] H. Aris, "Improving student's performance in introductory programming subject: A case study," in *Proc. Comput. Sci. Educ. (ICCSE)*, Cambridge, U.K., 2015, pp. 657–662, doi: 10.1109/ICCSE.2015.7250328.

[R31] M. Konecki, N. Kadoić, and R. Piltaver, "Intelligent assistant for helping students to learn programming," in *Proc. Inf. Commun. Technol. Electron. Microelectron. (MIPRO)*, Opatija, Croatia, 2015, pp. 924–928, doi: 10.1109/MIPRO.2015.7160406.

[R32] M. Hertzand S. M. Ford, "Investigating factors of student learning in introductory courses," in *Proc. SIGCSE*, Denver, CO, USA, 2013, pp. 195–200, doi: 10.1145/2445196.2445254.

[R33] C. Mirolo, "Learning (through) recursion: A multidimensional analy-sis of the competences achieved by CS1 students," in *Proc. ITiCSE*, Ankara, Turkey, 2010, pp. 160–164, doi: 10.1145/1822090.1822136.

[R34] M. Piteira and C. Costa, "Learning computer programming: Study of difficulties in learning programming," in *Proc. ISDOC*, Lisbon, Portugal, 2013, pp. 75–80, doi: 10.1145/2503859.2503871.

[R35] C. Watson, F. W. B. Li, and R. W. H. Lau, "Learning program-minglanguages through corrective feedback and concept visuali-sation," in *Advances in Web-Based Learning—ICWL2011* (LNCS 7048), H. Leung, E. Popescu, Y. Cao, R. W. H. Lau, and W. Nejdl, Eds. Heidelberg, Germany: Springer, 2011, pp. 11–20, doi: 10.1007/978-3-642-25813-8_2.

[R36] D. McCall and M. Kölling, "Meaningful categorisation of novice pro-grammer errors," in *Proc. Front. Educ. Conf. (FIE)*, Madrid, Spain, 2014, pp. 1–8, doi: 10.1109/FIE.2014.7044420.

[R37] R. Matthews, H. S. Hin, and K. A. Choo, "Novice programming students' perception of learning object," in *Proc. Informat. Creative Multimedia (ICICM)*, Kuala Lumpur, Malaysia, 2013, pp. 292–297, doi: 10.1109/ICICM.2013.67.

[R38] M. Yousoof and M. Sapiyan, "Optimizing instruction for learning computer programming—A novel approach," in *Intelligence in the Era of Big Data. ICSIIT 2015* (Communications in Computer and Information Science), vol. 516, R. Intan, C. H. Chi, H. Palit, and L. Santoso, Eds. Berlin, Germany: Springer, 2015, pp. 128–139, doi: 10.1007/978-3-662-46742-8_12.

[R39] O. D. L. Tavares, C. S. de Menezes, and R. A. de Nevado, "Pedagogical architectures to support the process of teaching and learning of computer programming," in *Proc. Front. Educ. Conf. (FIE)*, Seattle, WA, USA, 2012, pp. 1–6, doi: 10.1109/FIE.2012.6462427.

[R40] A. Shargabi, S. A. Aljunid, M. Annamalai, S. M. Shuhidan, and A. M. Zin, "Program comprehension levels of abstraction for novices," in *Proc. Comput. Commun. Control Technol. (I4CT)*, Kuching, Malaysia, 2015, pp. 211–215, doi: 10.1109/I4CT.2015.7219568.

[R41] N. Thota, "Programming course design: Phenomenographic approach to learning and teaching," in *Proc. Teach. Learn. Comput. Eng. (LaTiCE)*, Kuching, Malaysia, 2014, pp. 125–132, doi: 10.1109/LaTiCE.2014.30.

[R42] D. Teague and R. Lister, "Programming: Reading, writing and reversing," in *Proc. ITiCSE*, Uppsala, Sweden, 2014, pp. 285–290, doi: 10.1145/2591708.2591712.

[R43] M. Yamamoto, T. Sekiya, and K. Yamaguchi, "Relationship between programming concepts underlying programming skills," in *Proc. Inf. Technol. Based Higher Educ. Train. (ITHET)*, Izmir, Turkey, 2011, pp. 1–7, doi: 10.1109/ITHET.2011.6018678.

[R44] M. Apiola, N. Moisseinen, and M. Tedre, "Results from an action research approach for designing CS 1 learning environments in Tanzania," in *Proc. Front. Educ. Conf. (FIE)*, Seattle, WA, USA, 2012, pp. 1–6, doi: 10.1109/FIE.2012.6462321.

[R45] A. Gomes and A. J. Mendes, "Studies and proposals about initial pro-gramming learning," in *Proc. Front. Educ. Conf. (FIE)*, Washington, DC, USA, 2010, pp. 1–6, doi: 10.1109/FIE.2010.5673426.

[R46] J. Sheard, A. Carbone, D. Chinn, and M.-J. Laakso, "Study habits of CS1 students: What do they say they do?" in *Proc. Learn. Teach. Comput. Eng. (LaTiCE)*, 2013, pp. 122–129, doi: 10.1109/LaTiCE.2013.46.

[R47] S. M. M. Rubiano, O. López-Cruz, and E. G. Soto, "Teaching computer programming: Practices, difficulties and opportunities," in *Proc. Front. Educ. Conf. (FIE)*, El Paso, TX, USA, 2015, pp. 1–9, doi: 10.1109/FIE.2015.7344184.

[R48] T. Koulouri, S. Lauria, and R. D. Macredie, "Teaching introduc-tory programming: A quantitative evaluation of different approaches," *ACM Trans. Comput. Educ.*, vol. 14, no. 4, 2014, Art. no. 26, doi: 10.1145/2662412.

[R49] L. M. M. Giraffa, M. C. Moraes, and L. Uden, "Teaching object-oriented programming in first-year undergraduate courses supported by virtual classrooms," in *Proc. 2nd Int. Workshop Learn. Technol. Educ. Cloud*, 2013, pp. 15–26, doi: 10.1007/978-94-007-7308-0_2.

[R50] V. Isomöttönen and V. Tirronen, "Teaching programming by empha-sizing self-direction: How did students react to the active role required of them?" *ACM Trans. Comput. Educ.*, vol. 13, no. 2, Jun. 2013, Art. no. 6, doi: 10.1145/2483710.2483711.

[R51] D. A. Kranch, "Teaching the novice programmer: A study of instruc-tional sequences and perception," *Educ. Inf. Technol.*, vol. 17, no. 3, pp. 291–313, 2011, doi: 10.1007/s10639-011-9158-8.

[R52] Á. Matthíasdóttir and H. J. Geirsson, "The novice problem in computer science," in *Proc. CompSysTech*, Vienna, Austria, 2011, pp. 570–576, doi: 10.1145/2023607.2023702.

[R53] M. Hertz and M. Jump, "Trace-based teaching in early programming courses," in *Proc. SIGCSE*, Denver, CO, USA, 2013, pp. 561–566, doi: 10.1145/2445196.2445364.

[R54] C. J. Costa, M. Aparicio, and C. Cordeiro, "Web-based graphic envi-ronment to support programming in the beginning learning process," in *Entertainment Computing—ICEC2012* (LNCS 7522), M. Herrlich, R. Malaka, and M. Masuch, Eds. Heidelberg, Germany: Springer, 2012, pp. 413–416, doi: 10.1007/978-3-642-33542-6_41.

[R55] S. Fitzgerald *et al.*, "Debugging from the student perspec-tive," *IEEE Trans.*, vol. 53, no. 3, pp. 390–396, Aug. 2010, doi: 10.1109/TE.2009.2025266.

[R56] M. Haungs, C. Clark, J. Clements, and D. Janzen, "Improving first-year success and retention through interest-based CS0 courses," in *Proc. 43rd ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, Raleigh, NC, USA, 2012, pp. 589–594.

[R57] J. Kurhila and A. Vihavainen, "Management, structures and tools to scale up personal advising in large programming courses," in *Proc. Conf. Inf. Technol. Educ. (SIGITE)*, New York, NY, USA, 2011, pp. 3–8.

[R58] P. Lasserre and C. Szostak, "Effects of team-based learning on a CS1 course," in *Proc. 6th Annu. Joint Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, Darmstadt, Germany, 2011, pp. 133–137.

[R59] L. Porter and B. Simon, "Retaining nearly one-third more majors with a trio of instructional best practices in CS1," in *Proc. SIGCSE*, Denver, CO, USA, 2013, pp. 165–170.

[R60] C. F. Reilly and N. De La Mora, "The impact of real-world topic labs on student performance in CS1," in *Proc. Front. Educ. Conf.*, Seattle, WA, USA, 2012, pp. 1–6, doi: 10.1109/FIE.2012.6462329.

[R61] M. Rizvi and T. Humphries, "A scratch-based CS0 course for at-risk computer science majors," in *Proc. Front. Educ. Conf.*, Seattle, WA, USA, 2012, pp. 1–5, doi: 10.1109/FIE.2012.6462491.

[R62] G. Rößling and M. Mühlhäuser, "An unusual CS 1 with high standards and confirming results," in *Proc. 15th Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, Ankara, Turkey, 2010, pp. 169–173.

[R63] S. C. Shaffer and M. B. Rosson, "Increasing student success by modi-fying course delivery based on student submission data," *ACM Inroads*, vol. 4, no. 4, pp. 81–86, 2013, doi: 10.1145/2537753.2537778.

[R64] B. Simon, P. Kinnunen, L. Poter, and D. Zazkis, "Experience report: CS1 for majors with media computation," in *Proc. 15th Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, Ankara, Turkey, 2010, pp. 214–218.

[R65] R. Cacceffo, S. Wolfman, K. S. Booth, and R. Azevedo, "Developing a computer science concept inventory for introductory programming," in *Proc. 47th ACM Tech. Symp. Comput. Sci. Educ.*, Memphis, TN, USA, 2016, pp. 364–369.

[R66] J. Figueiredo, N. Gomes, and F. J. García-Peñalvo, "Ne-course for learning programming," in *Proc. 4th Int. Conf. Technol. Ecosyst. Enhancing Multiculturality*, Salamanca, Spain, 2016, pp. 549–553.

[R67] A. Lishinski, A. Yadav, R. Enbody, and J. Good, "The influence of problem solving abilities on students' performance on different assess-ment tasks in CS1," in *Proc. 47th ACM Tech. Symp. Comput. Sci. Educ.*, Memphis, TN, USA, 2016, pp. 329–334.

[R68] A. Luxton-Reilly, "Learning to program is easy," in *Proc. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, Arequipa, Peru, 2016, pp. 284–289.

[R69] A. Petersen, M. Craig, J. Campbell, and A. Tafliovich, "Revisiting why students drop CS1," in *Proc. 16th Koli Calling Int. Conf. Comput. Educ. Res.*, 2016, pp. 71–80.

[R70] D. F. Shell, L.-K. Soh, A. E. Flanigan, and M. S. Peteranetz, "Students' initial course motivation and their achievement and retention in college CS1 courses," in *Proc. 47th ACM Tech. Symp. Comput. Sci. Educ.*, Memphis, TN, USA, 2016, pp. 639–644.

[R71] A. Lishinski, A. Yadav, J. Good, and R. Enbody, "Learning to program: Gender differences and interactive effects of students' moti-vation, goals, and self-efficacy on performance," in *Proc. ACM Conf. Int. Comput. Educ. Res.*, Melbourne, VI C, Australia, 2016, pp. 211–220.

[R72] C. Ott, A. Robins, and K. Shephard, "Translating principles of effective feedback for students into the CS1 context," *ACM Trans. Comput. Educ.*, vol. 16, no. 1, Jan. 2016, Art. no. 1, doi: 10.1145/2737596.

[R73] G. Silva-Maceda, P. D. Arjona-Villicaña, and F. E. Castillo-Barrera, "More time or better tools? A large-scale retrospective comparison of pedagogical approaches to teach programming," *IEEE Trans.*, vol. 59, no. 4, pp. 274–281, Nov. 2016, doi: 10.1109/TE.2016.2535207.

[R74] M. P. Uysal, "Improving first computer programming experiences: The case of adapting a Web-supported and well-structured problem-solving method to a traditional course," *Contemporary Educ. Technol.*, vol. 5, no. 3, pp. 198–217, 2014.

[R75] R. A. Alturki, "Measuring and improving student performance in an introductory programming course," *Informat. Educ.*, vol. 15, no. 2, pp. 183–204, 2014, doi: 10.15388/infedu.2016.10.

[R76] K. Hartness and L.-J. Shannon, "Peer mentors and their impact for beginning programmers," *Inf. Syst. Educ. J.*, vol. 9, no. 6, pp. 21–29, 2011.

[R77] A. Hoskey and P. S. M. Maurino, "Beyond introductory programming: Success factors for advanced programming," *Inf. Syst. Educ. J.*, vol. 9, no. 5, pp. 61–70, 2011.

[R78] B. Hosack, B. Lim, and W. P. Vogt, "Increasing student performance through the use of Web services in introductory programming class-rooms: Results from a series of quasi-experiments," *J. Inf. Syst. Educ.*, vol. 24, no. 4, pp. 373–383, 2012.

[R79] K. Sung *et al.*, "Game-themed programming assignment modules: A pathway for gradual integration of gaming context into existing introductory programming courses," *IEEE Trans. Edu.*, vol. 54, no. 3, pp. 416–427, Aug. 2011, doi: 10.1109/TE.2010.2064315.

[R80] M. Apiola and M. Tedre, "New perspectives on the pedagogy of programming in a developing country context," *Comput. Sci. Educ.*, vol. 22, no. 3, pp. 285–313, 2012.

[R81] T. B. Bati, H. Gelderblom, and J. van Biljonc, "A blended learning approach for teaching computer programming: Design for large classes in Sub-Saharan Africa," *Comput. Sci. Educ.*, vol. 24, no. 1, pp. 71–99, 2014.

[R82] B. A. Becker *et al.*, "Effective compiler error message enhancement for novice programming students," *Comput. Sci. Educ.*, vol. 26, nos. 2–3, pp. 148–175, 2016.

[R83] J. Bennedsen and M. E. Caspersen, "Persistence of elementary pro-gramming skills," *Comput. Sci. Educ.*, vol. 22, no. 2, pp. 81–107, 2012.

[R84] M. C. Hughes, M. C. Jadud, and M. M. T. Rodrigo, "String format-ting considered harmful for novice programmers," *Comput. Sci. Educ.*, vol. 20, no. 3, pp. 201–228, 2010.

[R85] C. Ott, A. Robins, P. Haden, and K. Shephard, "Illustrating per-formance indicators and course characteristics to support students' self-regulated learning in CS1," *Comput. Sci. Educ.*, vol. 25, no. 2, pp. 174–198, 2015.

[R86] A. Robins, "Learning edge momentum: A new account of out-comes in CS1," *Comput. Sci. Educ.*, vol. 20, no. 1, pp. 37–71, 2010.

[R87] S. Shuhidan, M. Hamilton, and D. D'Souza, "Instructor perspec-tives of multiple-choice questions in summative assessment for novice programmers," *Comput. Sci. Educ.*, vol. 20, no. 3, pp. 229–259, 2010.

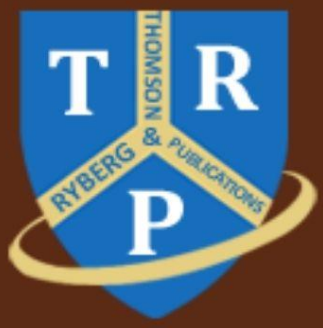
[R88] S. Willman *et al.*, "On study habits on an introductory course on programming," *Comput. Sci. Educ.*, vol. 25, no. 3, pp. 276–291, 2015, doi: 10.1080/08993408.2015.1073829.

[R89] L. Beck andA. Chizhik, “Cooperative learninginstructional meth-ods for CS1: Design, implementation, and evaluation,” *ACM Trans.Comput.Educ.*,vol.13,no.3,pp.1–21,2013.
[R90]A. Berglund and R. Lister, “Introductory programming and the didactictriangle,” in *Proc. 12th Aust. Comput. Educ. Conf. (ACE)*, Brisbane,QLD,Australia,2010,pp.35–44.
[R91]R. Cardell-Oliver, “How can software metrics help novice program-mers?” in *Proc. 13th Aust. Comput. Educ. Conf. (ACE)*, Perth, WA,Australia,2011,pp.55–62.
[R92]P.Denny,A.Luxton-Reilly,E.Tempero,andJ.Hendrickx,“Understanding the syntax barrier for novices,” in *Proc. 16th Annu.Joint Conf. Innov. Technol. Comput. Sci. Educ.*, Darmstadt, Germany,2011,pp.208–212.
[R93]N. Hawi, “Causal attributions of success and failure made by under-graduatestudentsinanintroductory-levelcomputerprogrammingcourse,”*Comput.Educ.*,vol.54,no.4,pp.1127–1136,2010.
[R94]L. J. Höök and A. Eckerdal, “On the bimodality in an introductoryprogramming course: An analysis of student performance factors,” in*Proc. Int. Conf. Learn. Teach. Comput. Eng.*, Taipei, Taiwan, 2015,pp.79–86.
[R95]R. Ibrahim, R. C. M. Yusoff, H. M. Omar, and A. Jaafar, “Studentsperceptions of using educational games to learn introductory program-ming,”*Comput.Inf.Sci.*,vol.4,no.1,pp.205–216,2011.
[R96]V. Isomöttönen and V. Lappalaine, “CS1 with games and an emphasison TDD and unit testing: Pilling a trend upon a trend,” *ACM Inroads*,vol.3,no.3,pp.62–68,2012.
[R97]D. F. Shell *et al.*, “Improving learning of computational thinking usingcomputational creativity exercises in a college CSI computer sciencecourse for engineers,” in *Proc. Front. Educ. Conf. (FIE)*, Madrid,Spain,2014,pp.1–7,doi:10.1109/FIE.2014.7044489.
[R98]K. Wood, D. Parsons, J. Gasson, and P. Haden, “It’s never too early:Pair programming in CS1,” in *Proc. 15th Aust. Comput. Educ. Conf.(ACE)*,Adelaide,SA,Australia,2013,pp.13–21.
[R99]A. Yadin, “Reducing the dropout rate in an introductory programmingcourse,”*ACMInroads*,vol.2,no.4,pp.71–76,2011.
[R100]B. Simon, J. Parris, and J. Spacco, “How we teach impacts studentlearning:Peerinstructionvs.lectureinCS0,”in*Proc.44thACMTech .Symp.Comput.Sci.Educ.*,Denver,CO,USA,2013,pp.41–46.

Rodrigo Pessoa Medeiros received the B.S. degree in Internet systems fromFaculdade Marista, Brazil, in 2005 and the M.S. degree in digital arts andtechnology from Universidade do Minho, Portugal, in 2009. He is currentlypursuing the Ph.D. degree in computer science with the Universidade FederaldePernambuco,Brazil.

Geber Lisboa Ramalho received the Ph.D. degree in computer science fromthe University of Paris VI in 1997. He is an Electronic Engineer and anAssociate Professor with the Centro de Informática, Universidade Federal dePernambuco, Brazil, where he is currently the Head of the Systems EngineerDepartment and the Chair of the Board of CESAR, a Brazilian innovationinstitute.

TacianaPontualFalcãoreceived the bachelor’s degree in computer sci-enceandthePh.D.degreefromtheUniversityofLondonin2014.Sheisan Assistant Professor with the Departamento de Computação, UniversidadeFederalRuraldePernambuco,Brazil.



THOMSON & RYBERG PUBLICATIONS

www.trpubonline.com/journals.php
trpub.online@gmail.com
or ijedtl@trpubonline.com

This article can be downloaded from https://ijetl.com/index.php/ijetl